# 2DMAT's Documentation

***Release 2.2.0***

**2DMAT's developer team**

**Mar 28, 2024**

# 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], sxrd[3], and leed[4]. As algorithms, it offers the Nelder-Mead method[5], the grid search method[6], the Bayesian optimization method[7], the replica exchange Monte Carlo method[8], and the population annealing Monte Carlo method[9-11]. 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] T. Hanada, Y. Motoyama, K. Yoshimi, and T. Hoshi, Computer Physics Communications 277, 108371 (2022).

[3] W. Voegeli, K. Akimoto, T. Aoyama, K. Sumitani, S. Nakatani, H. Tajiri, T. Takahashi, Y. Hisada, S. Mukainakano, X. Zhang, H. Sugiyama, H. Kawata, Applied Surface Science 252 (2006) 5259.

[4] M.A. Van Hove, W. Moritz, H. Over, P.J. Rous, A. Wander, A. Barbieri, N. Materer, U. Starke, G.A. Somorjai, Automated determination of complex surface structures by LEED, Surface Science Reports, Volume 19, 191-229 (1993).

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

[6] K. Tanaka, I. Mochizuki, T. Hanada, A. Ichimiya, T. Hyodo, and T. Hoshi, JJAP Conf. Series,.

[7] Y. Motoyama, R. Tamura, K. Yoshimi, K. Terayama, T. Ueno, and K. Tsuda, Computer Physics Communications 278, 108405 (2022)

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

[9] R. M. Neal, Statistics and Computing 11, 125-139 (2001).

[10] K. Hukushima and Y. Iba, AIP Conf. Proc. 690, 200 (2003).

[11] J. Machta, Phys. Rev. E 82, 026704 (2010).

## 1.2 License

This package is distributed under GNU General Public License version 3 (GPL v3).



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:

"Data-analysis software framework 2DMAT and its application to experimental measurements for two-dimensional material structures", Y. Motoyama, K. Yoshimi, I. Mochizuki, H. Iwamoto, H. Ichinose, and T. Hoshi, Computer Physics Communications 280, 108465 (2022).

Bibtex:

@article{MOTOYAMA2022108465, title = {Data-analysis software framework 2DMAT and its application to experimental measurements for two-dimensional material structures}, journal = {Computer Physics Communications}, volume = {280}, pages = {108465}, year = {2022}, issn = {0010-4655}, doi = {https://doi.org/10.1016/j.cpc.2022.108465}, url = {https://www.sciencedirect.com/science/article/pii/S0010465522001849}, author = {Yuichi Motoyama and Kazuyoshi Yoshimi and Izumi Mochizuki and Harumichi Iwamoto and Hayato Ichinose and Takeo Hoshi} }

## 1.3 Version Information

- v2.1.0: 2022-04-08
- v2.0.0: 2022-01-17
- v1.0.1: 2021-04-15
- v1.0.0: 2021-03-12
- v0.1.0: 2021-02-08

## 1.4 Main developers

2DMAT has been developed by following members.

- v2.0.0 -
  - Y. Motoyama (The Institute for Solid State Physics, The University of Tokyo)
  - K. Yoshimi (The Institute for Solid State Physics, The University of Tokyo)
  - H. Iwamoto (Department of Applied Mathematics and Physics, Tottori University)
  - T. Hoshi (Department of Applied Mathematics and Physics, Tottori University)
- v0.1.0 - v1.0.1
  - 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 (>=3.6.8)
  - **The following Python packages are required.**
    - tomli >= 1.2
    - numpy >= 1.14
  - 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`)

- SXRD forward problem solver (`sxrdcalc`)
- LEED forward problem solver (`satleed`)

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 *Direct Problem Solver* 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

# 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.

- `pamc`

  Sampling plausible atomic coordinates using a population annealing 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`, `exchange`, and `pamc` .

## 3.1 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 show some examples which use some algorithms with sim-trhepd-rheed. First, we will install and test sim-trhepd-rheed (for details, see the official web page for sim-trhepd-rheed).

### 3.1.1 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.

### 3.1.2 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/solver 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).

### 3.1.3 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
```

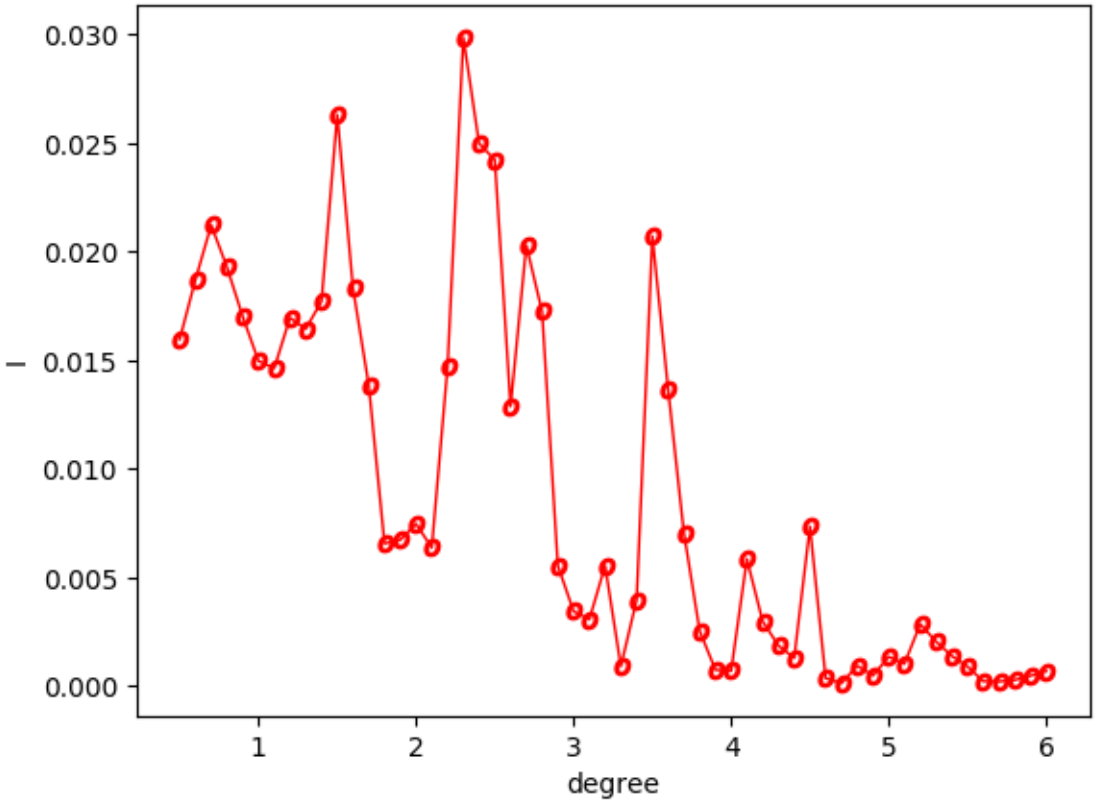



Fig. 3.1: Rocking curve of Si(001)-2x1 surface.

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.

## 3.2 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/sim-trhepd-rheed/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`).

### 3.2.1 Location of the sample files

The sample files are located in `sample/py2dmat/sim-trhepd-rheed/single_beam/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.

### 3.2.2 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.)

### 3.2.3 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.

### 3.2.4 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/sim-trhepd-rheed/single_beam/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
```

### 3.2.5 Visualization of calculation results

The data of the rocking curve at each step is stored in `Logxxxx_00000001` (where `xxxx` is the index 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_00000001/RockingCurve.txt RockingCurve_ini.txt
cp 0/Log00000062_00000001/RockingCurve.txt RockingCurve_con.txt
cp ../../../../script/draw_RC_double.py .
python draw_RC_double.py
```

Running the above will output `RC_double.png`.

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

## 3.3 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.

### 3.3.1 Location of the sample files

The sample files are located in `sample/sim-trhepd-rheed/single_beam/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.

- `ref_ColorMap.txt`

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

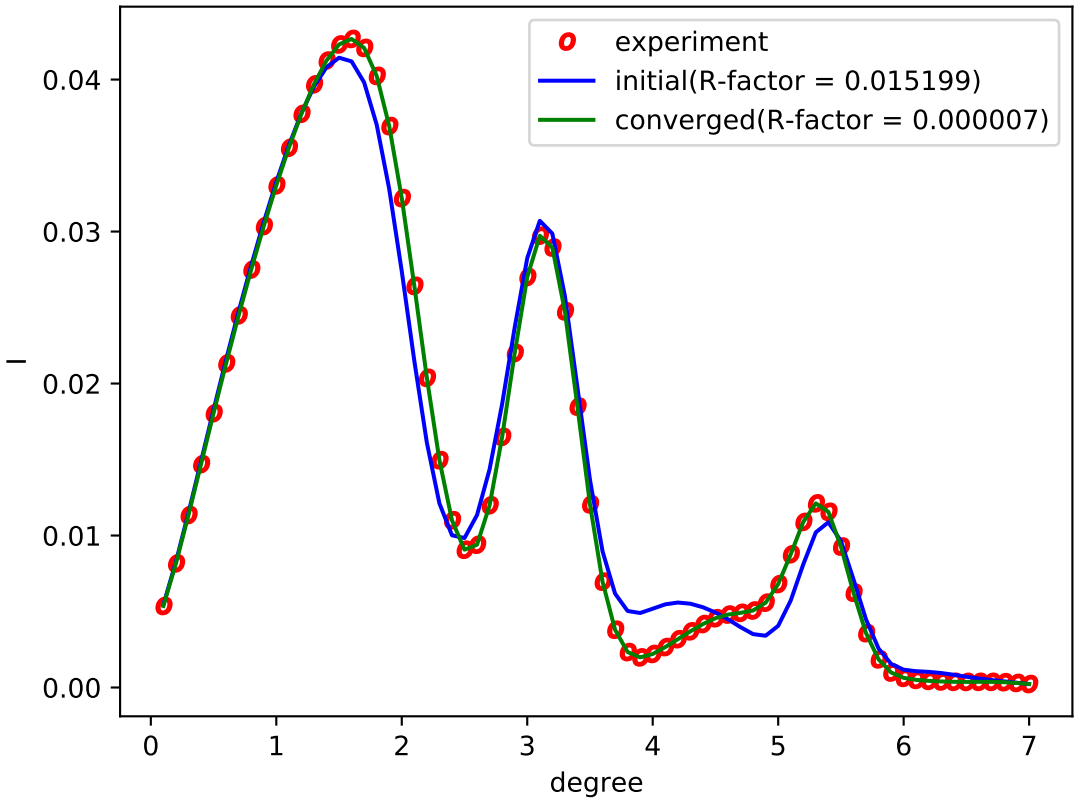



Fig. 3.2: Analysis using the Nelder-Mead method. The red circle represents the experimental value, the blue line represents the first step, and the green line represents the rocking curve obtained at the last step.

- `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.

### 3.3.2 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.

### 3.3.3 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]
path = "experiment.txt"
first = 1
last = 70
```

(continues on next page)

(continued from previous page)

```
[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.

### 3.3.4 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/sim-trhepd-rheed/single_beam/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
```

(continues on next page)

(continued from previous page)

```
  echo TEST FAILED: ColorMap.txt and ref_ColorMap.txt differ
  false
fi
```

### 3.3.5 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).

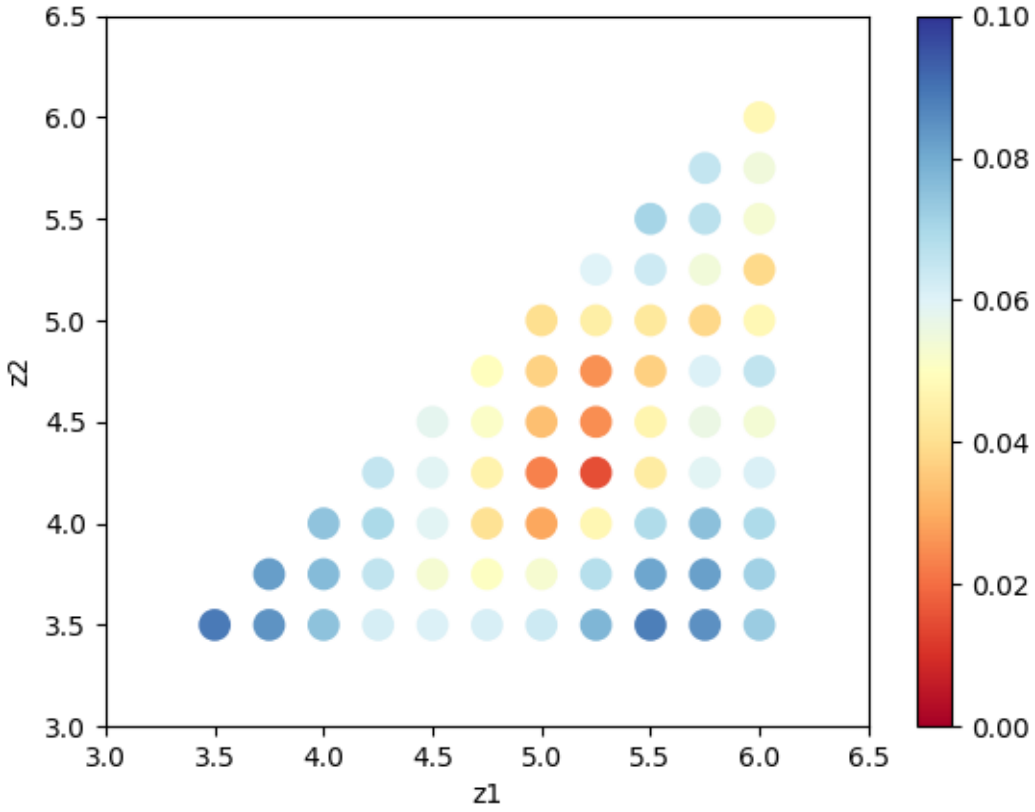



Fig. 3.3: `R-factor` on a two-dimensional parameter space.

`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.

## 3.4 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.

### 3.4.1 Sample files

Sample files are available from `sample/sim-trhepd-rheed/single_beam/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

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

### 3.4.2 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 3.5 3.5
2 3.6 3.5
3 3.6 3.6
4 3.7 3.5
5 3.7 3.6
6 3.7 3.7
7 3.8 3.5
8 3.8 3.6
9 3.8 3.7
10 3.8 3.8
  ...
```

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`.

### 3.4.3 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]
random_max_num_probes = 10
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`.

### 3.4.4 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/sim-trhepd-rheed/single_beam/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 = 10
bayes_max_num_probes = 20
score = TS
interval = 5
num_rand_basis = 5000
value_01 =  5.10000
value_02 =  4.90000
R-factor = 0.037237314010261195
0001-th step: f(x) = -0.037237 (action=150)
   current best f(x) = -0.037237 (best action=150)

value_01 =  4.30000
value_02 =  3.50000
```

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 fx z1_action z2_action fx_action
0 5.1 4.9 0.037237314010261195 5.1 4.9 0.037237314010261195
1 5.1 4.9 0.037237314010261195 4.3 3.5 0.06050786306685965
2 5.1 4.9 0.037237314010261195 5.3 3.9 0.06215778000834068
3 5.1 4.9 0.037237314010261195 4.7 4.2 0.049210767760634364
4 5.1 4.9 0.037237314010261195 5.7 3.7 0.08394457854191653
5 5.1 4.9 0.037237314010261195 5.2 5.2 0.05556857782716691
6 5.1 4.9 0.037237314010261195 5.7 4.0 0.0754639895013157
7 5.1 4.9 0.037237314010261195 6.0 4.4 0.054757310814479355
8 5.1 4.9 0.037237314010261195 6.0 4.2 0.06339787375966344
9 5.1 4.9 0.037237314010261195 5.7 5.2 0.05348404677676544
10 5.1 4.7 0.03002813055356341 5.1 4.7 0.03002813055356341
```

(continues on next page)

(continued from previous page)

```
11 5.1 4.7 0.03002813055356341 5.0 4.4 0.03019977423448576
12 5.3 4.5 0.02887504880071686 5.3 4.5 0.02887504880071686
13 5.1 4.5 0.025865346123665988 5.1 4.5 0.025865346123665988
14 5.2 4.4 0.02031077875240244 5.2 4.4 0.02031077875240244
15 5.2 4.4 0.02031077875240244 5.2 4.6 0.023291891689059388
16 5.2 4.4 0.02031077875240244 5.2 4.5 0.02345999725278686
17 5.2 4.4 0.02031077875240244 5.1 4.4 0.022561543431398066
18 5.2 4.4 0.02031077875240244 5.3 4.4 0.02544527153306051
19 5.2 4.4 0.02031077875240244 5.1 4.6 0.02778877135528466
20 5.2 4.3 0.012576357659158034 5.2 4.3 0.012576357659158034
21 5.1 4.2 0.010217361468113488 5.1 4.2 0.010217361468113488
22 5.1 4.2 0.010217361468113488 5.2 4.2 0.013178053637167673
  ...
```

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 21th 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
else
  echo TEST FAILED: BayesData.txt.txt and ref_BayesData.txt.txt differ
  false
fi
```

### 3.4.5 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`.

## 3.5 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).

### 3.5.1 Sample files

Sample files are available from `sample/sim-trhepd-rheed/single_beam/exchange` . 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.

### 3.5.2 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.

### 3.5.3 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
```

(continues on next page)

(continued from previous page)

```
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.

### 3.5.4 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/sim-trhepd-rheed/single_beam/exchange
```

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 two columns are time (step) and the index of walker in the process, the third is the temperature, the fourth column is the value of the objective function, and the fifth and subsequent columns are the parameters.

```
# step walker T fx x1 x2
0 0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
2 0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
3 0 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
```

(continues on next page)

(continued from previous page)

```
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
```

### 3.5.5 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
```

### 3.5.6 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.

Also, `RockingCurve.txt` is stored in each subfolder, `LogXXX_YYY` (`XXX` is an index of MC step and `YYY` is an index of a replica in the MPI process). By using this, it is possible to compare with the experimental value according to the procedure of the previous tutorial.

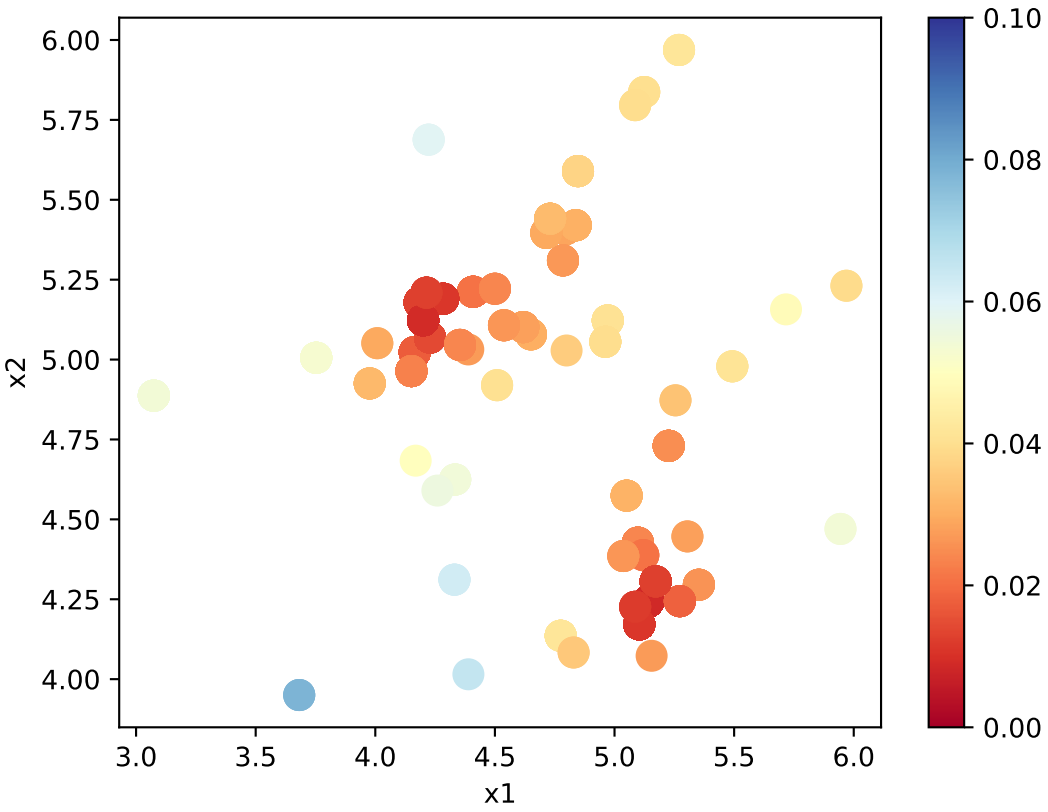



Fig. 3.4: Sampled parameters and `R-factor`. The horizontal axes is `value_01` and the vertical axes is `value_02` .

## 3.6 Replica Exchange Monte Carlo search with limitation

This tutorial describes the constraint expression function that can be set in the `[runner.limitation]` section. As an example, the replica exchange Monte Carlo method is applied to the calculation of Himmelblau with constraints.

### 3.6.1 Sample files location

Sample files are available in the `sample/analytical/limitation` directory. This directory contains the following files.

- `ref.txt`

  File to check if the calculation is executed correctly (answer to obtain by performing this tutorial).

- `input.toml`

  Input file for the main program.

- `do.sh`

  Script prepared to calculate this tutorial at once.

In the following, we will explain these files, and then introduce the actual calculation results.

### 3.6.2 Input files

The following `input.toml` is an input file for the main program.

```
[base]
dimension = 2
output_dir = "output"

[algorithm]
name = "exchange"
seed = 12345

[algorithm.param]
max_list = [6.0, 6.0]
min_list = [-6.0, -6.0]
unit_list = [0.3, 0.3]

[algorithm.exchange]
Tmin = 1.0
Tmax = 100000.0
numsteps = 10000
numsteps_exchange = 100

[solver]
name = "analytical"
function_name = "himmelblau"

[runner]
[runner.limitation]
co_a = [[1, -1],[1, 1]]
co_b = [[0], [-1]]
```

`[base]` section is the parameter of the main program. `dimension` is the number of variables to be optimized, and in this case, it is 2.

`[algorithm]` section is the section to set the search algorithm. `name` is the name of the search algorithm. In this case, specify `"exchange"` for the replica exchange Monte Carlo method. `seed` is the seed given to the pseudo-random number generator.

`[algorithm.param]` sub-section specifies the range of parameters to be optimized. `min_list` indicates the minimum value, and `max_list` indicates the maximum value.

`[algorithm.exchange]` sub-section specifies the hyperparameters of the replica exchange Monte Carlo method.

- `numstep` is the number of Monte Carlo updates.
- `numsteps_exchange` specifies the number of times to attempt temperature exchange.
- `Tmin` and `Tmax` are the lower and upper limits of the temperature, respectively.
- If `Tlogspace` is `true`, the temperature is divided equally in log space. This option is not specified in this `input.toml` because the default value is `true`.

`[solver]` section specifies the solver used internally in the main program. In this case, the `analytical` solver is specified. The `analytical` solver sets the function using the `function_name` parameter, and in this case, the Himmelblau function is specified.

`[runner]` section has a sub-section `[runner.limitation]`, and in this section, the constraint expression is set. In the current version, the constraint expression is defined as $Ax + b > 0$ where $x$ is $N$ dimensional input parameter, $A$ is a $M \times N$ matrix, and $b$ is a $M$ dimensional vector. $A$ and $b$ are set by `co_a` and `co_b`, respectively. For details, see the `[limitation]` section in the input file in the manual.

In this case, the following constraint is imposed:

$$x_1 - x_2 > 0$$
$$x_1 + x_2 - 1 > 0$$

### 3.6.3 Calculation

First, move to the folder where the sample file is located (assuming that you are directly under the directory where you downloaded this software).

```
cd sample/analytical/limitation
```

Then, execute the main program as follows (the calculation time will end in about 20 seconds on a normal PC).

```
mpiexec -np 10 python3 ../../../src/py2dmat_main.py input.toml | tee log.txt
```

In this case, a calculation with 10 MPI parallel processes is performed. (When using OpenMPI, if the number of processes to be used is greater than the number of available cores, add the `--oversubscribed` option to the `mpiexec` command.) After executed, the `output` folder is generated, and in it, a subfolder for each rank is created. Each subfolder contains the results of the calculation. `trial.txt` file, which contains the parameters and objective function values evaluated at each Monte Carlo step, and `result.txt` file, which contains the parameters actually adopted, are created. Both files have the same format, with the first two columns being the step number and the walker number within the process, the next being the temperature, the third being the value of the objective function, and the fourth and subsequent being the parameters. The following is the beginning of the `output/0/result.txt` file:

```
# step walker T fx x1 x2
0 0 1.0 187.94429125133564 5.155393113805774 -2.203493345018569
1 0 1.0 148.23606736778044 4.9995614992887525 -2.370212436322816
```

(continues on next page)

(continued from previous page)

```
2 0 1.0 148.23606736778044 4.9995614992887525 -2.370212436322816
3 0 1.0 148.23606736778044 4.9995614992887525 -2.370212436322816
```

Finally, the best parameter and the rank and Monte Carlo step at which the objective function is minimized are written to `output/best_result.txt`.

```
nprocs = 10
rank = 2
step = 4523
walker = 0
fx = 0.00010188398524402734
x1 = 3.584944906595298
x2 = -1.8506985826548874
```

`do.sh` is available as a script to calculate all at once. Additionally, in `do.sh`, the difference between `best_result.txt` and `ref.txt` is also compared.

```
#!/bin/bash
mpiexec -np 10 --oversubscribe python3 ../../../src/py2dmat_main.py input.toml

echo diff output/best_result.txt ref.txt
res=0
diff output/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
```

### 3.6.4 Visualization of the calculation result

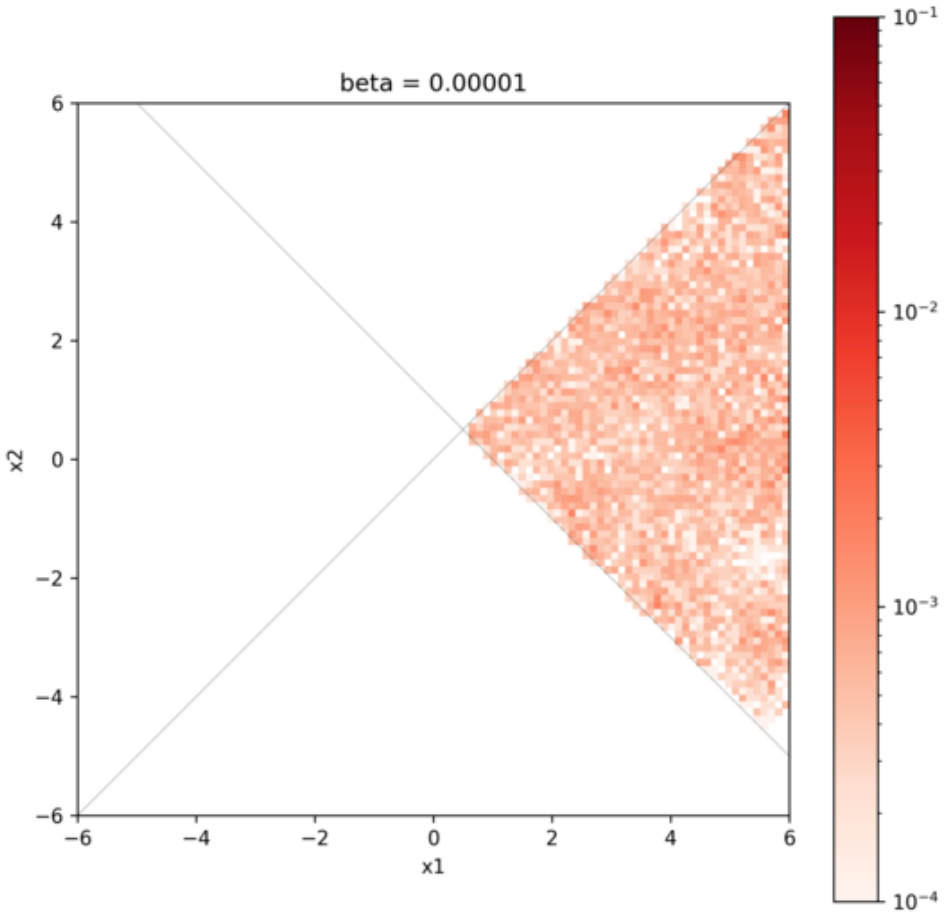
By visualizing the `result.txt` file, we can confirm that the search is only for coordinates that satisfy the constraint expression. `hist2d_limitation_sample.py` is prepared to visualize the 2D parameter space. This generates a histogram of the posterior probability distribution in the `<execution date>_histogram` folder. The histogram is generated using the data obtained by discarding the first 1000 steps of the search as a burn-in period.

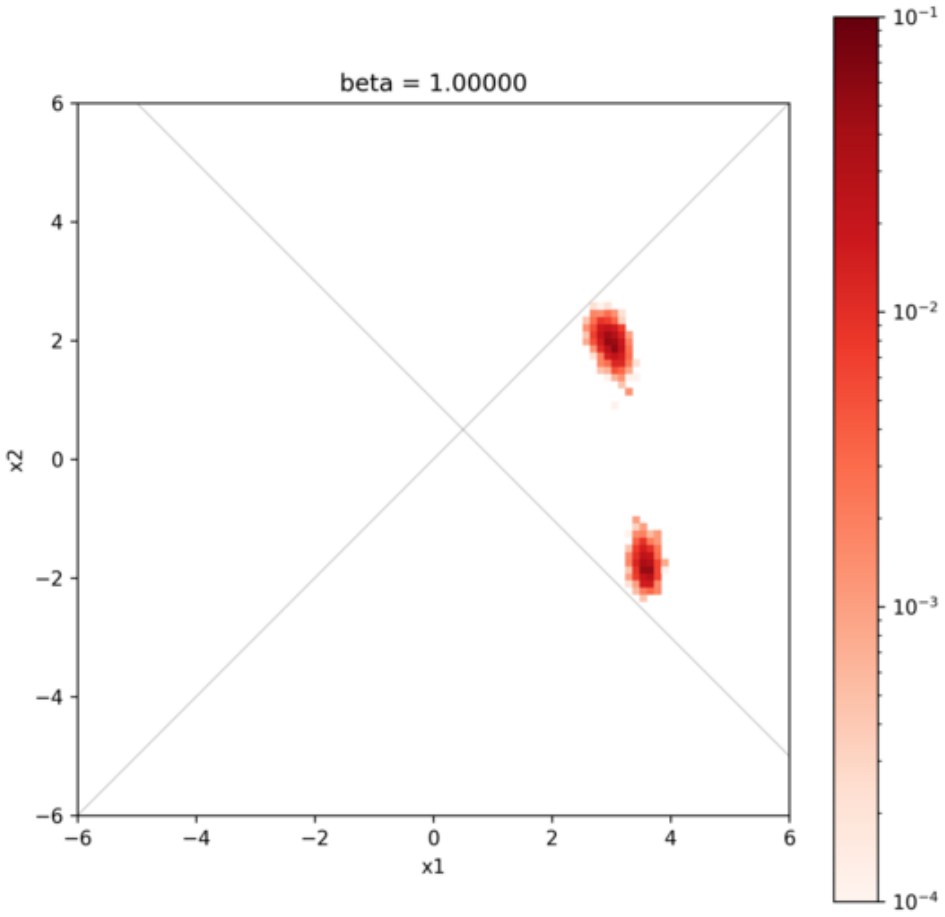
```
python3 hist2d_limitation_sample.py -p 10 -i input.toml -b 0.1
```

The figure shows the posterior probability distribution and the two lines $x_1 - x_2 = 0, x_1 + x_2 - 1 = 0$, and it is confirmed that the search is only for the range where $x_1 - x_2 > 0, x_1 + x_2 - 1 > 0$.

## 3.7 Optimization by population annealing

This tutorial subscribes how to estimate atomic positions from the experimental diffraction data by using the population annealing Monte Carlo method (PAMC).


beta = 0.00001
x1
x2
6
4
2
0
−2
−4
−6
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$


beta = 1.00000
x1
x2
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$

### 3.7.1 Sample files

Sample files are available from `sample/sim-trhepd-rheed/single_beam/pamc` . 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 (reference for `fx.txt`)

- `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.

### 3.7.2 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.

### 3.7.3 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
output_dir = "output"

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

[algorithm.param]
min_list = [3.0, 3.0]
max_list = [6.0, 6.0]
unit_list = [0.3, 0.3]

[algorithm.pamc]
numsteps_annealing = 5
bmin = 0.0
bmax = 200.0
Tnum = 21
Tlogspace = false
nreplica_per_proc = 10
```

(continues on next page)

(continued from previous page)

```
[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.
    - `unit_list` is step length in one MC update (deviation of Gaussian)
  - The `[algorithm.pamc]` section sets the parameters for PAMC.
    - `numsteps_annealing` is the number of interval steps between temperature decreasing.
    - `bmin`, `bmax` are the minimum and the maximum of inversed temperature, respectively.
    - `Tnum` is the number of (inversed) temperature points.
    - When `Tlogspace` is `true`, the temperature points are distributed uniformly in the logarithmic space.
    - `nreplica_per_proc` is the number of replicas (MC walkers) in one MPI process.
- The `[solver]` section specifies the solver to use inside the main program and its settings.
  - See the *Optimization by Nelder-Mead method* tutorial.

### 3.7.4 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/sim-trhepd-rheed/single_beam/pamc
```

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 `trial_TXXX.txt` files containing the parameters evaluated in each Monte Carlo step and the value of the objective function at each temperature (`XXX` is the index of points), and `result_TXXX.txt` files containing the parameters actually adopted will be created. These files are concatnated into `result.txt` and `trial.txt`.

These files have the same format: the first two columns are time (step) and the index of walker in the process, the third is the (inversed) temperature, the fourth column is the value of the objective function, and the fifth and subsequent columns are the parameters. The final two columns are the weight of walker (Neal-Jarzynski weight) and the index of the grand ancestor (the replica index at the beginning of the calculation).

```
# step walker beta fx z1 z2 weight ancestor
0 0 0.0 0.07702743614780189 5.788848278451443 3.949126663745358 1.0 0
0 1 0.0 0.08737730661436376 3.551756435031283 3.6136808356591192 1.0 1
0 2 0.0 0.04954470587051104 4.70317508724506 4.786634108937754 1.0 2
0 3 0.0 0.04671675601156148 5.893543559206865 4.959531290614713 1.0 3
0 4 0.0 0.04142014655238446 5.246719912601735 4.960709612555206 1.0 4
```

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 = 0
step = 71
walker = 5
fx = 0.008186713312593607
z1 = 4.225633749839847
z2 = 5.142666117413409
```

Finally, `fx.txt` stores the statistics at each temperature point:

```
# $1: 1/T
# $2: mean of f(x)
# $3: standard error of f(x)
# $4: number of replicas
# $5: log(Z/Z0)
# $6: acceptance ratio
0.0 0.06428002079611472 0.002703413400677839 40 0.0 0.795
10.0 0.061399304916174735 0.002649424392996749 40 -0.6280819199879947 0.85
20.0 0.05904248889111052 0.0031622711212952034 40 -1.2283060742855603 0.74
30.0 0.04956921148431115 0.0028298565759159633 40 -1.7991035905899855 0.67
```

The first column is (inversed) temperature, and the second/third ones are the mean and standard error of $f(x)$, respectively. The fourth column is the number of replicas and the fifth one is the logarithm of the ratio of the partition functions, $\log(Z_n/Z_0)$, where $Z_0$ is the partition function at the first temperature. The sixth column is the acceptance ratio of MC updates.

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

### 3.7.5 Visualization

By illustrating `result_T.txt`, you can estimate regions where the parameters with small R-factor are. In this case, the figure `result_fx.pdf` and `result_T.pdf` of the 2D parameter space is created by using the following command. The color of symbols of `result_fx.pdf` and `result_T.pdf` mean `R-factor` and $\beta$, respectively.

```
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.

Also, `RockingCurve.txt` is stored in each subfolder, `LogXXX_YYY` (`XXX` is an index of MC step and `YYY` is an index of a replica in the MPI process). By using this, it is possible to compare with the experimental value according to the procedure of the previous tutorial.

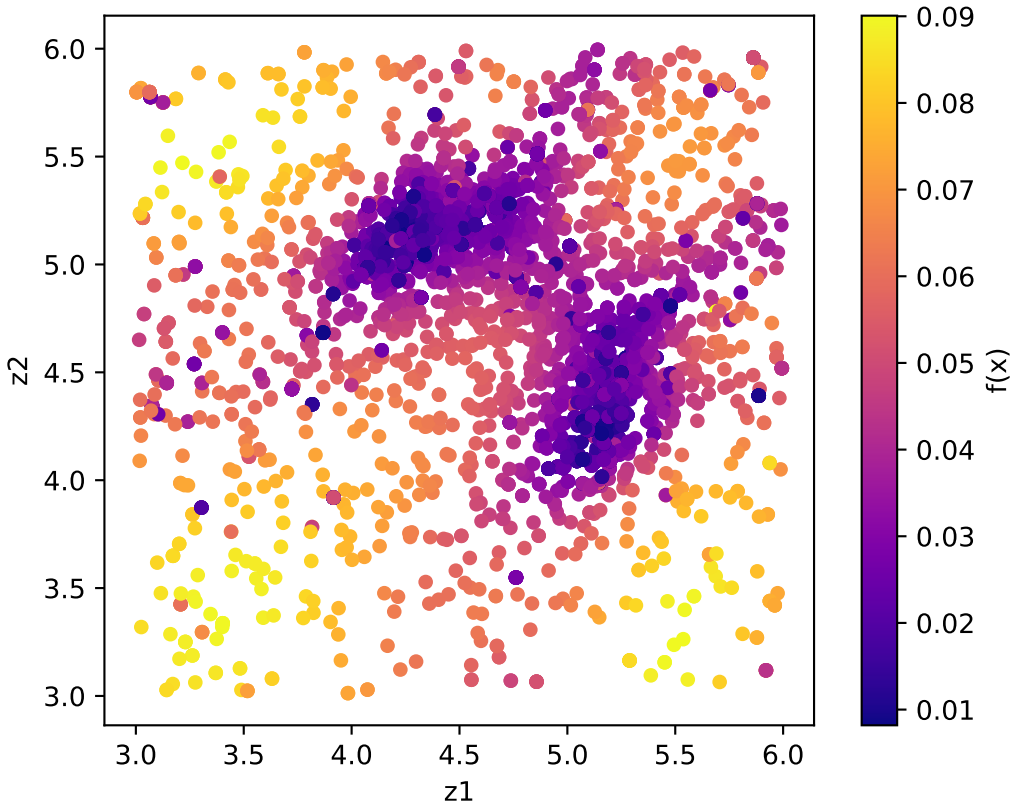



Fig. 3.5: Sampled parameters and `R-factor`. The horizontal axes is `value_01` and the vertical axes is `value_02` .

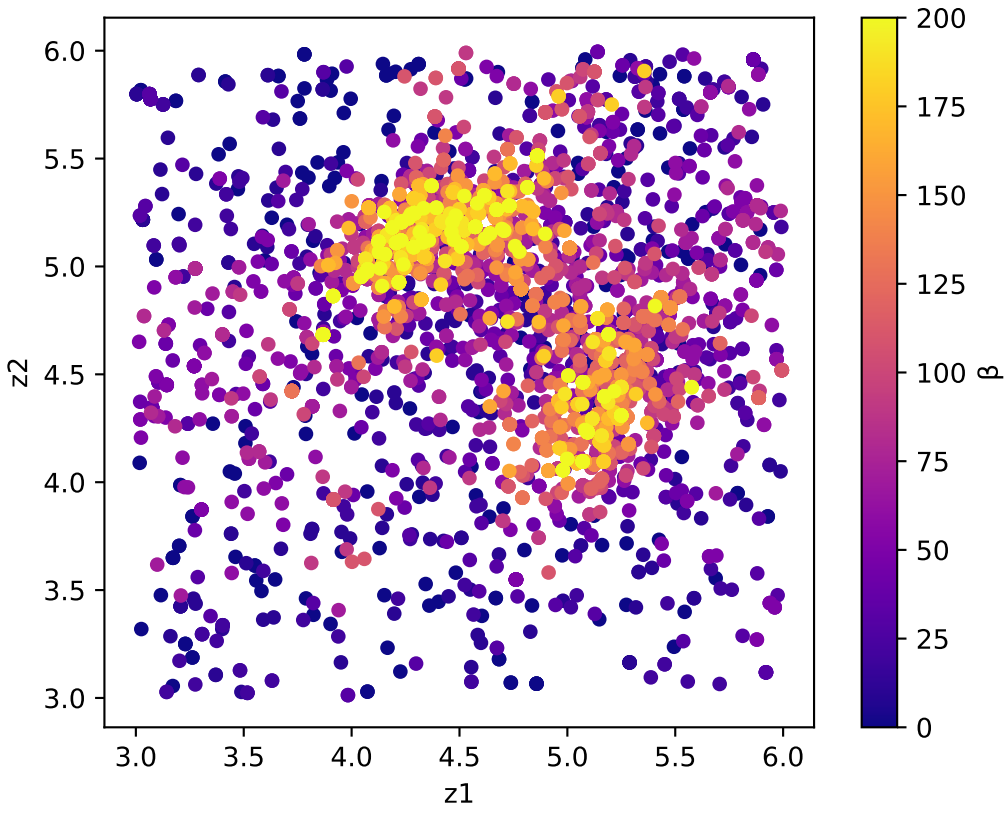



Fig. 3.6: Sampled parameters and $\beta$. The horizontal axes is `value_01` and the vertical axes is `value_02` .

## 3.8 Addition of a direct problem solver

### 3.8.1 Solver for benchmarking, `analytical`

`py2dmat` provides an `analytical` solver as a direct problem solver that can be used to test search algorithms.

To use the `analytical` solver, set `name` to `"analytical"` in the `[solver]` section of the input file. You can also use the `function_name` parameter to select the benchmark function $f(x)$.

For example, to use Himmelblau function, make an input file including the following:

```
[solver]
name = "analytical"
function_name = "himmelblau"
```

For details of `analytical` solver, please check the *the reference of the analytical solver*.

### 3.8.2 Addition of a direct problem solver

The easiest way to define and analyze a user-defined direct problem solver is to add it to the `analytical` solver. As an example, we will explain the case of adding the Booth function (the minimum point is $f(1,3)=0$):

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2.$$

To do so, we need to download the source code for `py2dmat` and edit the file for `analytical` solver. For instructions on how to download the source code and run `py2dmat` from the source code, see *how to install*. `analytical` solver is defined in the `src/py2dmat/solver/analytical.py`, so edit it.

First, define the booth function as follows:

```
def booth(xs: np.ndarray) -> float:
    """Booth function

    it has one global minimum f(xs) = 0 at xs = [1,3].
    """

    if xs.shape[0] != 2:
        raise RuntimeError(
            f"ERROR: booth expects d=2 input, but receives d={xs.shape[0]} one"
        )
    return (xs[0] + 2 * xs[1] - 7.0) ** 2 + (2 * xs[0] + xs[1] - 5.0) ** 2
```

Next, insert the following code in the if branch of the `Solver` class's constructor (`__init__`) to allow users to choose the `booth` function by the `solver.function_name` parameter of the input file.

```
elif function_name == "booth":
        self.set_function(booth)
```

With this modified `analytical` solver, you can optimize the Booth function. For example, to optimize it by the Nelder-Mead method, pass the following input file (`input.toml`)

```
[base]
dimension = 2
output_dir = "output"
```

(continues on next page)

(continued from previous page)

```
[algorithm]
name = "minsearch"
seed = 12345

[algorithm.param]
max_list = [6.0, 6.0]
min_list = [-6.0, -6.0]
initial_list = [0, 0]

[solver]
name = "analytical"
function_name = "booth"
```

to `src/py2dmat_main.py` script as follows:

```
$ python3 src/py2dmat_main.py input.toml

... skipped ...

Iterations: 38
Function evaluations: 75
Solution:
x1 = 1.0000128043523089
x2 = 2.9999832920260863
```

CHAPTER

FOUR

# INPUT FILE

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

- `base`
  - Specify the basic parameters about `py2dmat` .
- `solver`
  - Specify the parameters about `Solver` .
- `algorithm`
  - Specify the parameters about `Algorithm` .
- `runner`
  - Specify the parameters about `Runner` .
- `mapping`
  - Define the mapping from a parameter searched by `Algorithm` .
- `limitation`
  - Define the limitation (constration) of parameter searched by `Algorithm` .
- `log`
  - Specify parameters related to logging of solver calls.

## 4.1 [base] section

- `dimension`

  Format: Integer

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

- `root_dir`

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

  Description: Name of the root directory. The origin of the relative paths to input files.

- `output_dir`

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

  Description: Name of the directory to output the results.

## 4.2 [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.

## 4.3 [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 values, Monte Carlo updates, etc.**
  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.

## 4.4 [runner] section

This section sets the configuration of `Runner`, which bridges `Algorithm` and `Solver`. It has two subsections, `mapping` and `log` .

## 4.5 [mapping] section

This section defines the mapping from an $N$ dimensional parameter searched by `Algorithm`, $x$, to an $M$ dimensional parameter used in `Solver`, $y$ . In the case of $N \neq M$, the parameter `dimension` in `[solver]` section should be specified.

In the current version, the affine mapping (linear mapping + translation) $y = Ax + b$ is available.

- `A`

  Format: List of list of float, or a string (default: `[]`)

  **Description: $N \times M$ matrix $A$. An empty list `[]` is a shorthand of an identity matrix.**
  If you want to set it by a string, arrange the elements of the matrix separated with spaces and newlines (see the example).

- `b`

  Format: List of float, or a string (default: `[]`)

  **Description: $M$ dimensional vector $b$. An empty list `[]` is a shorthand of a zero vector.**
  If you want to set it by a string, arrange the elements of the vector separated with spaces.

For example, both

```
A = [[1,1], [0,1]]
```

and

```
A = """
1 1
0 1
"""
```

mean

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

## 4.6 [limitation] section

This section defines the limitation (constraint) in an $N$ dimensional parameter searched by `Algorithm`, $x$, in addition of `min_list` and `max_list`.

In the current version, a linear inequation with the form $Ax + b > 0$ is available.

- `co_a`

  Format: List of list of float, or a string (default: `[]`)

**Description: $N \times M$ matrix $A$. An empty list `[]` is a shorthand of an identity matrix.**
If you want to set it by a string, arrange the elements of the matrix separated with spaces and newlines (see the example).

- `co_b`

  Format: List of float, or a string (default: `[]`)

  **Description: $M$ dimensional vector $b$. An empty list `[]` is a shorthand of a zero vector.**
  If you want to set it by a string, arrange the elements of the vector separated with spaces.

For example, both

```
A = [[1,1], [0,1]]
```

and

```
A = """
1 1
0 1
"""
```

mean

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

## 4.7 [log] section

Setting parametrs 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

FIVE

# OUTPUT FILES

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

## 5.1 Common file

### 5.1.1 `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
```

### 5.1.2 `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
3 9.67080000000009e-05 0.0012490219999999885
```

(continues on next page)

(continued from previous page)

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

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. Length should be equal to `dimension`.

  **Description: Minimum value of each parameter.**
  When a parameter falls below this value during the Nelson-Mead method, the solver is not evaluated and the value is considered infinite.

- `max_list`

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

  **Description: Maximum value of each parameter.**
  When a parameter exceeds this value during the Nelson-Mead method, the solver is not evaluated and the value is considered infinite.

### `[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. 1 + `dimension` columns are required. The first column is the index of the mesh, and the second and subsequent columns are the values of parameter.

A sample file for two dimensions is shown below.

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

## 6.3 Replica exchange Monte Carlo `exchange`

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

### 6.3.1 Preparation

mpi4py should be installed.

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

### 6.3.2 Input parameters

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

#### `[algorithm.param]`

This defines a space to be explored. When `mesh_path` key is defined the discrete space is used. Otherwise, continuous space is used.

- Continuous space
  - `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.**
  When a parameter falls below this value during the Monte Carlo search, the solver is not evaluated and the value is considered infinite.

- `max_list`

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

  **Description: Maximum value of each parameter.**
  When a parameter exceeds this value during the Monte Carlo search, the solver is not evaluated and the value is considered infinite.

- Discrete space
  - `mesh_path`

    Format: string

    Description: Path to the mesh definition file.

  - `neighborlist_path`

    Format: string

    Description: Path to the neighborhood-list file.

### `[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

  Description: The minimum value of the "temperature" ($T$).

- `Tmax`

  Format: Float

  Description: The maximum value of the "temperature" ($T$).

- `bmin`

  Format: Float

  Description: The minimum value of the "inverse temperature" ($\beta = 1/T$). One of the "temperature" and "inverse temperature" should be defined.

- `bmax`

  Format: Float

  Description: The maximum value of the "inverse temperature" ($\beta = 1/T$). One of the "temperature" and "inverse temperature" should be defined.

- `Tlogspace`

  Format: Boolean (default: true)

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

- `nreplica_per_proc`

  Format: Integer (default: 1)

  Description: The number of replicas in a MPI process.

### 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. Note that the index of the mesh will be ignored for this "algorithm".

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
...
```

#### Neighborhood-list file

Before searching in the discrete space by Markov Chain Monte Carlo method, we should define "neighborhoods" for each point $i$, which are points that a walker can move from $i$ A neighborhood-list file defines the list of neighborhoods. In this file, the index of an initial point $i$ is specified by the first column, and the indices of final points $j$ are specified by the second and successive columns.

An utility tool, `py2dmat_neighborlist` is available for generating a neighborhood-list file from a mesh file. For details, please see *Related Tools*.

```
0 1 2 3
1 0 2 3 4
2 0 1 3 4 5
3 0 1 2 4 5 6 7
4 1 2 3 5 6 7 8
5 2 3 4 7 8 9
...
```

### 6.3.4 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 index of the walker in the process. The third column is the temperature of the replica. The fourth column is the value of the solver. The remaining columns are the coordinates.

Example:

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

#### result_T#.txt

This file stores samples for each temperature ( # is replaced with the index of temperature ). The first column is the index of the MC step. The second column is the index of the walker. The third column is the value of the solver. The remaining columns are the coordinates.

```
# T = 1.0
0 15 28.70157662892569 3.3139009347685118 -4.20946994566609
1 15 28.70157662892569 3.3139009347685118 -4.20946994566609
2 15 28.70157662892569 3.3139009347685118 -4.20946994566609
3 15 28.98676409223712 3.7442621319489637 -3.868754990884034
```

#### Algorithm

### 6.3.5 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

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

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 $\Delta f$.
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.

### 6.3.6 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.

## 6.4 Population Annealing Monte Carlo `pamc`

`pamc` explores the parameter space by using the Population Annealing Monte Carlo (PAMC) method.

### 6.4.1 Preparation

mpi4py should be installed for the MPI parallelization

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

### 6.4.2 Input parameters

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

#### [algorithm.param]

This defines a space to be explored. When `mesh_path` key is defined the discrete space is used. Otherwise, continuous space is used.

- Continuous space
  - `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.**
  When a parameter falls below this value during the Monte Carlo search, the solver is not evaluated and the value is considered infinite.

- `max_list`

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

  **Description: Maximum value of each parameter.**
  When a parameter exceeds this value during the Monte Carlo search, the solver is not evaluated and the value is considered infinite.

- Discrete space
  - `mesh_path`

    Format: string

    Description: Path to the mesh definition file.

  - `neighborlist_path`

    Format: string

    Description: Path to the neighborhood-list file.

### `[algorithm.pamc]`

- `numsteps`

  Format: Integer

  Description: The number of Monte Carlo steps.

- `numsteps_annealing`

  Format: Integer

  Description: The number of interval Monte Carlo steps between lowering "temperature".

- `numT`

  Format: Integer

  Description: The number of "temperature" points.

- `Tmin`

  Format: Float

  Description: The minimum value of the "temperature" ($T$).

- `Tmax`

  Format: Float

  Description: The maximum value of the "temperature" ($T$).

- `bmin`

  Format: Float

  Description: The minimum value of the "inverse temperature" ($\beta = 1/T$). One of the "temperature" and "inverse temperature" should be defined.

- `bmax`

  Format: Float

  Description: The maximum value of the "inverse temperature" ($\beta = 1/T$). One of the "temperature" and "inverse temperature" should be defined.

- `Tlogspace`

  Format: Boolean (default: true)

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

- `nreplica_per_proc`

  Format: Integer (default: 1)

  Description: The number of replicas in a MPI process.

- `resampling_interval`

  Format: Integer (default: 1)

  Description: The number of annealing processes between resampling of the replicas.

- `fix_num_replicas`

  Format: Boolean (default: true)

  Description: Whether to fix the number of replicas or not on resampling.

#### About the number of steps

Specify just two of `numstep`, `numsteps_annealing`, and `numT`. The value of the remaining one will be determined automatically.

### 6.4.3 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. Note that the index of the mesh will be ignored for this "algorithm".

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
```

(continues on next page)

(continued from previous page)

```
9 6.000000 4.000000
...
```

#### Neighborhood-list file

Before searching in the discrete space by Markov Chain Monte Carlo method, we should define "neighborhoods" for each point $i$, which are points that a walker can move from $i$ A neighborhood-list file defines the list of neighborhoods. In this file, the index of an initial point $i$ is specified by the first column, and the indices of final points $j$ are specified by the second and successive columns.

An utility tool, `py2dmat_neighborlist` is available for generating a neighborhood-list file from a mesh file. For details, please see *Related Tools*.

```
0 1 2 3
1 0 2 3 4
2 0 1 3 4 5
3 0 1 2 4 5 6 7
4 1 2 3 5 6 7 8
5 2 3 4 7 8 9
...
```

### 6.4.4 Output files

**`RANK/trial_T#.txt`**

This file stores the suggested parameters and the corresponding value returned from the solver for each temperature point (specified by `#`). The first column (`step`) is the index of the MC step. The second column (`walker`) is the index of the walker in the process. The third column (`beta`) is the inverse temperature of the replica. The fourth column (`fx`) is the value of the solver. The fifth - (4+dimension)-th columns are the coordinates. The last two columns (`weight` and `ancestor`) are the Neal-Jarzynsky weight and the grand-ancestor of the replica.

Example:

```
# step walker beta fx x1 weight ancestor
0 0 0.0 73.82799488298886 8.592321856342956 1.0 0
0 1 0.0 13.487174782058675 -3.672488908364282 1.0 1
0 2 0.0 39.96292704464803 -6.321623766458111 1.0 2
0 3 0.0 34.913851603463 -5.908794428939206 1.0 3
0 4 0.0 1.834671825646121 1.354500581633733 1.0 4
0 5 0.0 3.65151610695736 1.910894059585031 1.0 5
...
```

#### RANK/trial.txt

This is a combination of all the `trial_T#.txt` in one.

#### RANK/result_T#.txt

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

```
# step walker beta fx x1 weight ancestor
0 0 0.0 73.82799488298886 8.592321856342956 1.0 0
0 1 0.0 13.487174782058675 -3.672488908364282 1.0 1
0 2 0.0 39.96292704464803 -6.321623766458111 1.0 2
0 3 0.0 34.913851603463 -5.908794428939206 1.0 3
0 4 0.0 1.834671825646121 1.354500581633733 1.0 4
0 5 0.0 3.65151610695736 1.910894059585031 1.0 5
...
```

#### RANK/result.txt

This is a combination of all the `result_T#.txt` in one.

#### 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
```

#### fx.txt

This file stores statistical metrics over the all replicas for each temperature. The first column is inverse temperature. The second and third column are the expectation value and the standard error of the solver's output ($f(x)$), respectively. The fourth column is the number of replicas. The fifth column is the logarithmic of the ratio between the normalization factors (partition functions)

$$\log \frac{Z}{Z_0} = \log \int \mathrm{d}x e^{-\beta f(x)} - \log \int \mathrm{d}x e^{-\beta_0 f(x)},$$

where $\beta_0$ is the minimum value of $\beta$ used in the calculation. The sixth column is the acceptance ratio of MC updates.

```
# $1: 1/T
# $2: mean of f(x)
# $3: standard error of f(x)
# $4: number of replicas
# $5: log(Z/Z0)
# $6: acceptance ratio
0.0 33.36426034198166 3.0193077565358273 100 0.0 0.9804
```

(continues on next page)

(continued from previous page)

```
0.1 4.518006242920819 0.9535301415484388 100 -1.2134775491597027 0.9058
0.2 1.5919146358616842 0.2770369776964151 100 -1.538611313376179 0.9004
...
```

### 6.4.5 Algorithm

#### Goal

When the weight of the configuration $x$ under some parameter $\beta_i$ is given as $f_i(x)$ (e.g., the Bolzmann factor $f_i(x) = \exp[-\beta_i E(x)]$ ), the expectation value of $A$ is defined as

$$\langle A \rangle_i = \frac{\int \mathrm{d}x A(x) f_i(x)}{\int \mathrm{d}x f_i(x)} = \frac{1}{Z} \int \mathrm{d}x A(x) f_i(x) = \int \mathrm{d}x A(x) \tilde{f}_i(x),$$

where $Z = \int \mathrm{d}x f_i(x)$ is the normalization factor (partition function) and $\tilde{f}(x) = f(x)/Z$ is the probability of $x$.

Our goal is to numerically calculate the expectation value for each $\beta_i$ and the (ratio of) the normalization factor.

#### Annealed Importance Sampling (AIS) [1]

First, we introduce a series of configurations $\{x_i\}$ obeying the following joint probability

$$\tilde{f}(x_0, x_1, \ldots, x_n) = \tilde{f}_n(x_n) \tilde{T}_n(x_n, x_{n-1}) \tilde{T}_{n-1}(x_{n-1}, x_{n-2}) \cdots \tilde{T}_1(x_1, x_0),$$

with

$$\tilde{T}_i(x_i, x_{i-1}) = T_i(x_{i-1}, x_i) \frac{\tilde{f}_i(x_{i-1})}{\tilde{f}_i(x_i)},$$

where $T_i(x, x')$ is a transition probability from $x$ to $x'$ under $\beta_i$ holding the balance condition,

$$\int \mathrm{d}x \tilde{f}_i(x) T_i(x, x') = \tilde{f}_i(x').$$

It turns out that $\tilde{f}_n(x_n)$ is the marginal distribution of $\tilde{f}(x_0, x_1, \ldots, x_n)$, that is,

$$\tilde{f}_n(x_n) = \int \prod_{i=0}^{n-1} \mathrm{d}x_i \tilde{f}(x_0, x_1, \ldots, x_n),$$

from

$$\int \mathrm{d}x_{i-1} \tilde{T}_i(x_i, x_{i-1}) = \int \mathrm{d}x_{i-1} \tilde{f}_i(x_{i-1}) T_i(x_{i-1}, x_i) / \tilde{f}_i(x_i) = 1.$$

Consequently, $\langle A \rangle_n$ is represented by using the extended configuration $\{x_i\}$ as

$$\begin{aligned} \langle A \rangle_n &\equiv \int \mathrm{d}x_n A(x_n) \tilde{f}_n(x_n) \\ &= \int \prod_i \mathrm{d}x_i A(x_n) \tilde{f}(x_0, x_1, \ldots, x_n). \end{aligned}$$

Unfortunately, it is difficult to generate directly a series of configurations $\{x_i\}$ following the distribution $\tilde{f}(x_0, x_1, \ldots, x_n)$. Then, instead of $\tilde{f}(x_0, x_1, \ldots, x_n)$, we consider $\{x_i\}$ obeying the joint distribution

$$\tilde{g}(x_0, x_1, \ldots, x_n) = \tilde{f}_0(x_0) T_1(x_0, x_1) T_2(x_1, x_2) \ldots T_n(x_{n-1}, x_n),$$

by using the following the following scheme:

1. Generete $x_0$ from the initial distribution $\tilde{f}_0(x)$
2. Generate $x_{i+1}$ from $x_i$ through $T_{i+1}(x_i, x_{i+1})$

By using the reweighting method (or importance sampling method), $\langle A \rangle_n$ is rewritten as

$$\begin{aligned}
\langle A \rangle_n &= \int \prod_i \mathrm{d}x_i A(x_n) \tilde{f}(x_0, x_1, \ldots, x_n) \\
&= \int \prod_i \mathrm{d}x_i A(x_n) \frac{\tilde{f}(x_0, x_1, \ldots, x_n)}{\tilde{g}(x_0, x_1, \ldots, x_n)} \tilde{g}(x_0, x_1, \ldots, x_n) \cdot \\
&= \left\langle A\tilde{f}/\tilde{g} \right\rangle_{g,n}
\end{aligned}$$

Because the ratio between $\tilde{f}$ and $\tilde{g}$ is

$$\begin{aligned}
\frac{\tilde{f}(x_0, \ldots, x_n)}{\tilde{g}(x_0, \ldots, x_n)} &= \frac{\tilde{f}_n(x_n)}{\tilde{f}_0(x_0)} \prod_{i=1}^{n} \frac{\tilde{T}_i(x_i, x_{i-1})}{T(x_{i-1}, x_i)} \\
&= \frac{\tilde{f}_n(x_n)}{\tilde{f}_0(x_0)} \prod_{i=1}^{n} \frac{\tilde{f}_i(x_{i-1})}{\tilde{f}_i(x_i)} \\
&= \frac{Z_0}{Z_n} \frac{f_n(x_n)}{f_0(x_0)} \prod_{i=1}^{n} \frac{f_i(x_{i-1})}{f_i(x_i)} \\
&= \frac{Z_0}{Z_n} \prod_{i=0}^{n-1} \frac{f_{i+1}(x_i)}{f_i(x_i)} \\
&\equiv \frac{Z_0}{Z_n} w_n(x_0, x_1, \ldots, x_n),
\end{aligned}$$

the form of the expectation value will be

$$\langle A \rangle_n = \left\langle A\tilde{f}/\tilde{g} \right\rangle_{g,n} = \frac{Z_0}{Z_n} \langle A w_n \rangle_{g,n}.$$

Finally, the ratio between the normalization factors $Z_n/Z_0$ can be evaluated as

$$\frac{Z_n}{Z_0} = \langle w_n \rangle_{g,n},$$

and therefore the expectation value of $A$ can be evaluated as a weighted arithmetic mean:

$$\langle A \rangle_n = \frac{\langle A w_n \rangle_{g,n}}{\langle w_n \rangle_{g,n}}.$$

This weight $w_n$ is called as the Neal-Jarzynski weight.

### population annealing (PA) [2]

Although the AIS method can estimate the expectation values of $A$ for each parameter $\beta$ as the form of weighted arithmetic mean, the variance of weights $w$ is generally large and then the accuracy of the result gets worse. In order to overcome this problem, the population annealing Monte Carlo (PAMC) method resamples all the replicas according to the probability $p^{(k)} = w^{(k)} / \sum_k w^{(k)}$ at some periods and resets all the weights to unity.

The following pseudo code describes the scheme of PAMC:

```
for k in range(K):
    w[0, k] = 1.0
    x[0, k] = draw_from(β[0])
for i in range(1, N):
    for k in range(K):
        w[i, k] = w[i-1, k] * ( f(x[i-1,k], β[i]) / f(x[i-1,k], β[i-1]) )
    if i % interval == 0:
        x[i, :] = resample(x[i, :], w[i, :])
        w[i, :] = 1.0
    for k in range(K):
        x[i, k] = transfer(x[i-1, k], β[i])
    a[i] = sum(A(x[i,:]) * w[i,:]) / sum(w[i,:])
```

There are two resampling methods: one with a fixed number of replicas[2] and one without[3].

#### References

[1] R. M. Neal, Statistics and Computing **11**, 125-139 (2001).

[2] K. Hukushima and Y. Iba, AIP Conf. Proc. **690**, 200 (2003).

[3] J. Machta, PRE **82**, 026704 (2010).

## 6.5 Bayse optimization `bayes`

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

### 6.5.1 Preparation

You will need to install PHYSBO beforehand.:

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

If mpi4py is installed, MPI parallel computing is possible.

### 6.5.2 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.

### 6.5.3 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
...
```

### 6.5.4 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
...
```

### 6.5.5 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 "exploition," 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.

CHAPTER

SEVEN

# DIRECT PROBLEM SOLVER

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

## 7.1 `analytical` solver

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

### 7.1.1 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 $f(3, 2) = f(-2.805118, 3.131312) = f(-3.779310, -3.283186) = f(3.584428, -1.848126) = 0$

## 7.2 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 $f(x)$.

### 7.2.1 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.

### 7.2.2 Input parameters

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

#### [solver] section

- generate_rocking_curve

  Format: boolean (default: false)

  Description: Whether to generate RockingCurve_calculated.txt. If true, RockingCurve_calculated.txt will be generated in the working directory Log%%%_###. Note that if remove_work_dir (in [post] subsection) is true, Log%%%_### will be removed.

#### [solver.config] subsection

- 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: In the output file, the first line to be read as D(x). The last line is automatically calculated from the number of the reference data.

- `calculated_info_line`

  Format: integer (default: 2)

  Description: In the output file, the line contains the information of the calculated data – the number of glancing angles (second column) and the number of beams (third column).

- `cal_number`

  Format: Integer or List of integers

  Description: In the output file, the columns to be read as D(x). Multiple columns can be specified (many-beam condition).

### `[solver.post]` subsection

This subsection is used to the postprocess – to specify how to calculate the objective function, that is, the deviation between the experimental and computational data, and to draw the rocking curve.

- `Rfactor_type`

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

  Description: This parameter specifies how to calculate the R-factor to be minimized. Let $n$ be the number of dataset, $m$ be the number of glancing angles, and $v^{(n)} = (v_1^{(n)}, v_2^{(n)}, \ldots, v_m^{(n)})$ be the calculated data. With the weights of the beams, $w^{(j)}$, R-factors is defined as follows:

  - "A" type:

$$R = \sqrt{\sum_j^n w^{(j)} \sum_i^m \left(u_i^{(j)} - v_i^{(j)}\right)^2}$$

  - "A2" type:

$$R^2 = \sum_j^n w^{(j)} \sum_i^m \left(u_i^{(j)} - v_i^{(j)}\right)^2$$

  - "B" type:

$$R = \frac{\sum_i^m \left(u_i^{(1)} - v_i^{(1)}\right)^2}{\sum_i^m \left(u_i^{(1)}\right)^2 + \sum_i^m (v_i^{(1)})^2}$$

    * "B" type is available only for the single dataset ($n = 1$).

- `normalization`

  Format: string ("TOTAL" or "MANY_BEAM")

  Description: This parameter specifies how to normalize the experimental and computational data vectors.

  - "TOTAL"
    - To normalize the data as the summation is 1.
    - The number of dataset should be one (the number of `cal_number` should be one).
  - "MANY_BEAM"
    - To normalize with weights as specified by `weight_type`.

  **NOTE: "MAX" is no longer available**

- `weight_type`

  Format: string or `None`. "calc" or "manual" (default: `None`)

  Description: The weights of the datasets for the "MANY_BEAM" normalization.

  - "calc"

$$w^{(n)} = \left( \frac{\sum_i^m v_i^{(n)}}{\sum_j^n \sum_i^m v_i^{(j)}} \right)^2$$

  - "manual"

  $w^{(n)}$ is specified by `spot_weight`.

- `spot_weight`

  Format: list of float (mandatory when `weight_type` is "manual")

  Description: The weights of the beams in the calculation of R-factor. The weights are automatically normalized as the sum be 1. For example, [3,2,1] means $w^{(1)} = 1/2, w^{(2)} = 1/3, w^{(3)} = 1/6$.

- `omega`

  Format: float (default: 0.5)

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

- `remove_work_dir`

  Format: boolean (default: false)

  Description: Whether to remove working directories `Log%%%_###` after reading R-factor or not

### [`solver.param`] subsection

- `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.

#### [solver.reference] subsection

- path

  Format: string (default: experiment.txt)

  Description: Path to the reference data file.

- reference_first_line

  Format: integer

  Description: In the reference data file, the first line to be read as Dexp. The default value is 1, that is, the first line of the file.

- reference_last_line

  Format: integer

  Description: In the reference data file, the last line to be read as Dexp. If omitted, all lines from the first line to the end of the file will be read.

- exp_number

  Format: Integer or List of integers

  Description: In the reference data file, the column numbers to be read. Multiple columns can be specified (many-beam condition).

### 7.2.3 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 and following columns contain the calculated value of the reflection intensity multiplied by the weight. An example of the file is shown below.

```
3.00000e-01 8.17149e-03 1.03057e-05 8.88164e-15 ...
4.00000e-01 1.13871e-02 4.01611e-05 2.23952e-13 ...
5.00000e-01 1.44044e-02 1.29668e-04 4.53633e-12 ...
6.00000e-01 1.68659e-02 3.49471e-04 7.38656e-11 ...
7.00000e-01 1.85375e-02 7.93037e-04 9.67719e-10 ...
8.00000e-01 1.93113e-02 1.52987e-03 1.02117e-08 ...
9.00000e-01 1.92590e-02 2.53448e-03 8.69136e-08 ...
1.00000e+00 1.86780e-02 3.64176e-03 5.97661e-07 ...
1.10000e+00 1.80255e-02 4.57932e-03 3.32760e-06 ...
1.20000e+00 1.77339e-02 5.07634e-03 1.50410e-05 ...
1.30000e+00 1.80264e-02 4.99008e-03 5.53791e-05 ...
...
```

### 7.2.4 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. `%%%%%` means an index of iteration in `Algorithm` (e.g., steps in Monte Carlo), and `#####` means an index of group (e.g., replica index in Monte Carlo). In large calculation, the number of these folders becomes too large to be written in the storage of the system. For such a case, let `solver.post.remove_work_dir` parameter be `true` in order to remove these folders. 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_calculated.txt

This file is located in the `Log%%%%%_#####` folder. At the beginning of the file, the lines beginning with `#` are headers. The header contains the values of the input variables, the objective function value `f(x)`, the parameters `Rfactor_type`, `normalization`, `weight_type`, `cal_number`, `spot_weight`, and what is marked in the data portion columns (e.g. `# #0 glancing_angle`).

The header is followed by the data. The first column shows the glancing angle, and the second and subsequent columns show the intensity of each data column. You can see which data columns are marked in the header. For example,

```
# #0 glancing_angle
# #1 cal_number=1
# #2 cal_number=2
# #3 cal_number=4
```

shows that the first column is the glancing angle, and the second, third, and fourth columns are the calculated data corresponding to the first, second, and fourth columns of the calculated data file, respectively.

Intencities in each column are normalized so that the sum of the intensity is 1. To calculate the objective function value (R-factor and R-factor squared), the data columns are weighted by `spot_weight` and normalized by `normalization`.

```
#value_01 =  0.00000 value_02 =  0.00000
#Rfactor_type = A
#normalization = MANY_BEAM
#weight_type = manual
#fx(x) = 0.03686180462340505
#cal_number = [1, 2, 4, 6, 8]
#spot_weight = [0.933 0.026 0.036 0.003 0.002]
#NOTICE : Intensities are NOT multiplied by spot_weight.
#The intensity I_(spot) for each spot is normalized as in the following equation.
#sum( I_(spot) ) = 1
#
# #0 glancing_angle
# #1 cal_number=1
# #2 cal_number=2
# #3 cal_number=4
# #4 cal_number=6
# #5 cal_number=8
0.30000 1.278160358686800e-02 1.378767858296659e-04 8.396046839668212e-14 1.
↪342648818357391e-30 6.697979700048016e-53
0.40000 1.778953628930054e-02 5.281839702773564e-04 2.108814173486245e-12 2.
↪467220122612354e-28 7.252675318478533e-50
0.50000 2.247181148723425e-02 1.671115124520428e-03 4.250758278908295e-11 3.
↪632860054842994e-26 6.291667506376419e-47
...
```

## 7.3 `sxrd` solver

`sxrd` is a `Solver` that uses `sxrdcalc` to calculate the Rocking curve by giving atomic positions $x$ , atomic occupancies, and Debye-Waller factor and finally returnes the error $f(x)$ from the experimental Rocking curve.

### 7.3.1 Preparation

The `sxrdcalc` is called from `py2dmat`. You will need to install `sxrdcalc` beforehand. `sxrdcalc` is available on GitHub at the following URL:

https://github.com/sxrdcalc/sxrdcalc

Access the site and download the source code from “Code” - “Download zip”. After unzipping the zip file, edit the Makefile to fit your computing environment, and then type `make` to create the `sxrdcalc` executable.

It is noted that the bulk data must be prepared in advance before running `py2dmat` (see the auxiliary file for solvers below for the format).

### 7.3.2 Input parameters

Input parameters are specified in subsections in the `solver` section (`config`, `post`, `param`, and `reference`).

#### [config] section

- `sxrd_exec_file`

  Format: string

  Description: Path to the solver `sxrdcalc`.

- `bulk_struc_in_file`

  Format: string

  Description: Input file name of the bulk structure.

  An example of the input input is given as follows:

```
[config]
sxrd_exec_file = "../../sxrdcalc"
bulk_struc_in_file = "sic111-r3xr3.blk"
```

#### [param] section

- `scale_factor`

  Format: float (default: 1.0)

  Description: The value of the target Rocking Curve and the scale of the Rocking Curve obtained from the simulation.

- `opt_scale_factor`

  Format: bool (default: false)

  Description: Flag whether `scale_factor` should be optimized or not.

- `type_vector`

  Format: list

  Description: A list of positive numbers which specifies the type of variables to be optimized. This list corresponds to the types specified in the [`param.atom`] subsection. If the type is the same, they are treated as the same variable.

#### [param.domain] subsection

In this section, parameters to create domains are specified. You will need to define the domains you want to create. In the [`param.domain.atom`] sub-subsection, paramters of the information in the domain are specified.

- `domain_occupancy`

  Format: float

  Description: Occupancy of the whole domain.

### [param.domain.atom] subsection

This section needs to be defined as many times as the number of atoms you want to optimize belonging to the domain. Note that the type, which represents the type of variable, must be a positive number.

- `name`

  Format: string (can be duplicated)

  Description: The name of the atom to be optimized.

- `pos_center`

  Format: list

  Description: Center coordinates of the atom. Describe in format of $[x_0, y_0, z_0]$($x_0, y_0, z_0$ float).

- `DWfactor`

  Format: float

  Description: Debye-Waller factor (in the unit of $Å^2$ ).

- `occupancy`

  Format: float (default: 1.0)

  Description: Atom occupancy.

- `displace_vector` (can be omitted)

  Format: list of lists

  Description: A vector that defines the direction in which the atoms are moved. A maximum of three directions can be specified. Define displacement vectors and initial values in each list as [type, $D_{i1}, D_{i2}, D_{i3}$ ](type is int, $D_{i1}, D_{i2}, D_{i3}$ is float type). Follwing the specified information, $l_{type}$ is varied as $dr_i = (D_{i1}\vec{a}+D_{i2}\vec{b}+D_{i3}\vec{c})*l_{type}$ ( $\vec{a}, \vec{b}, \vec{c}$ is a unit lattice vector defined in `bulk_struc_in_file` or `struc_in_file`).

- `opt_DW` (can be omitted)

  Format: list

  Description: Sets the scale at which the Debye-Waller coefficient is varied. It is defined as [type, scale].

- `opt_occupancy`

  Format: int

  Description: If defined, the occupancy changes. The specified variable represents the type.

An example of an input file is given as follows:

```
[param]
scale_factor = 1.0
type_vector = [1, 2]

[[param.domain]]
domain_occupancy = 1.0
 [[param.domain.atom]]
   name = "Si"
   pos_center = [0.00000000, 0.00000000, 1.00000000]
   DWfactor = 0.0
   occupancy = 1.0
   displace_vector = [[1, 0.0, 0.0, 1.0]]
 [[param.domain.atom]]
```

(continues on next page)

(continued from previous page)

```
  name = "Si"
  pos_center = [0.33333333, 0.66666667, 1.00000000]
  DWfactor = 0.0
  occupancy = 1.0
  displace_vector = [[1, 0.0, 0.0, 1.0]]
[[param.domain.atom]]
  name = "Si"
  pos_center = [0.66666667, 0.33333333, 1.00000000]
  DWfactor = 0.0
  occupancy = 1.0
  displace_vector = [[1, 0.0, 0.0, 1.0]]
[[param.domain.atom]]
  name = "Si"
  pos_center = [0.33333333, 0.33333333, 1.00000000]
  DWfactor = 0.0
  occupancy = 1.0
  displace_vector = [[2, 0.0, 0.0, 1.0]]
```

#### [reference] section

- f_in_file

  Format: string

  Description: Path to the input file for the target locking curve.

### 7.3.3 Reference file for Solver

#### Target reference file

The file containing the data to be targeted to fit. The path is specified by f_in_file in the [reference] section. For each line, h k l F sigma is written. Here, h, k, l are the wavenumbers, F is the intensity, and sigma is the uncertainty of F. An example file is shown below.

```
0.000000 0.000000 0.050000 572.805262 0.1
0.000000 0.000000 0.150000 190.712559 0.1
0.000000 0.000000 0.250000 114.163340 0.1
0.000000 0.000000 0.350000 81.267319 0.1
0.000000 0.000000 0.450000 62.927325 0.1
...
```

#### Bulk structure file

The file containing the bulk structure data. The path is specified by bulk_struc_in_file in the [config] section. The first line is a comment, the second line is a b c alpha beta gamma. Here, a , b, and c are the lattice constants of the unit cells, and alpha, beta , and gamma are their angles. The third and subsequent lines specify the atomsymbol r1 r2 r3 DWfactor occupancy. Here, atomsymbol is the atom species, r1, r2, and r3 are the position coordinates of the atom, DWfactor is the Debye-Waller factor, and occupancy is the occupancy rate. An example file is given below.

```
# SiC(111) bulk
5.33940 5.33940  7.5510487  90.000000 90.000000 120.000000
Si 0.00000000   0.00000000   0.00000000 0.0 1.0
Si 0.33333333   0.66666667   0.00000000 0.0 1.0
Si 0.66666667   0.33333333   0.00000000 0.0 1.0
C  0.00000000   0.00000000   0.25000000 0.0 1.0
...
```

### 7.3.4 Output files

In `sxrd`, the output files are stored in the folder with the rank number. Here is a description of the files that are output by `py2dmat`.

#### stdout

The standard output by `sxrd` is described. For sxrd's Least square fitting, we give variables as initial parameters and calculate the Rfactor for a 1-shot calculation (number of iterations = 0). The Rfactor is written in R under Fit results. Here is an example of the output.

```
----------------------------------------
Program py2dmat/mapper_sxrd/sxrdcalc for surface x-ray diffraction calculations.
Version 3.3.3 - August 2019


 Inputfile: lsfit.in
Least-squares fit of model to experimental structure factors.

...

Fit results:
Fit not converged after 0 iterations.
Consider increasing the maximum number of iterations or find better starting values.
chi^2 = 10493110.323318, chi^2 / (degree of freedom) = 223257.666454 (Intensities)
chi^2 = 3707027.897897, chi^2 / (degree of freedom) = 78872.933998 (Structure factors)
R = 0.378801

Scale factor:   1.00000000000000 +/- 0.000196
Parameter Nr. 1:   3.500000 +/- 299467640982.406067
Parameter Nr. 2:   3.500000 +/- 898402922947.218384

Covariance matrix:
          0            1            2
 0  0.0000000383 20107160.3315223120 -60321480.9945669472
 1  20107160.3315223120 896808679955672533565440.0000000000 -269042603986701827178496.
↪0000000000
 2  -60321480.9945669472 -269042603986701827178496.0000000000␣
↪807127811960105615753216.0000000000
```

## 7.4 leed solver

leed is a Solver made by M.A. Van Hove, which calculates the Rocking curve from atomic positions etc., using SATLEED, and returns the error from the experimental Rocking curve as $f(x)$. For more information on SATLEED, see [SATLEED].

### 7.4.1 Preparation

First, install SATLEED . Access to the following URL http://www.icts.hkbu.edu.hk/VanHove_files/leed/leedsatl.zip and download a zip file. Depending on the details of the system you want to calculate, it is necessary to change the parameters in the source code for SATLEED. After changing parameters, compile programs to generate the executable files such as stal1.exe, satl2.exe .

For trying the example at sample/py2dmat/leed, a utility script file setup.sh for downloading SATLEED, rewriting source codes, and compiling the program is available.:

```
$ cd sample/py2dmat/leed
$ sh ./setup.sh
```

After running setup.sh, executable files satl1.exe and satl2.exe are generated in leedsatl directory.

Note that it is assumed that you have already executed satl1.exe before using py2dmat . Therefore, the following files must be generated.

- Input files of satl1.exe : exp.d, rfac.d, tleed4.i, tleed5.i
- Output files of satl1.exe : tleed.o , short.t

py2dmat will run satl2.exe based on the above files.

### 7.4.2 Input parameters

Input parameters are specified in subsections in the solver section (config and reference).

#### [config] section

- path_to_solver

  Format: string

  Description: Path to the solver satl2.exe .

#### [reference] section

- path_to_base_dir

  Format: string

  Description: Path to the directory which stores exp.d, rfac.d, tleed4.i, tleed5.i , tleed.o , short.t .

### 7.4.3 Reference file for Solver

#### Target reference file

The file containing the data to be targeted to fit. Edit `tleed4.i` in `path_to_base_dir` in the [`reference`] section. Add the number you want to optimize to `optxxx` (where xxx is a three-digit number in the format 000, 001, 002, …). (where xxx is a three-digit integer in the form 000, 001, 002, …). Note that the number of xxx must match the order and number of variables in the list of `py2dmat` variables to be optimized. Note that if IFLAG and LSFLAG are not set to 0, the satleed side is also optimized.

An example file is shown below.

```
1  0  0                          IPR ISTART LRFLAG
1 10  0.02  0.2                  NSYM  NSYMS ASTEP VSTEP
5  1  2  2                       NT0  NSET LSMAX LLCUT
5                                NINSET
1.0000 0.0000                 1      PQEX
1.0000 2.0000                 2      PQEX
1.0000 1.0000                 3      PQEX
2.0000 2.0000                 4      PQEX
2.0000 0.0000                 5      PQEX
3                                 NDIM
opt000 0.0000 0.0000  0           DISP(1,j)  j=1,3
0.0000 opt001 0.0000  0           DISP(2,j)  j=1,3
0.0000 0.0000 0.0000  1           DISP(3,j)  j=1,3
0.0000 0.0000 0.0000  0           DISP(4,j)  j=1,3
0.0000  0                         DVOPT  LSFLAG
3  0  0                          MFLAG NGRID NIV
...
```

### 7.4.4 Output file

In `leed`, the output files are stored in the folder with the rank number.

CHAPTER

# EIGHT

# RELATED TOOLS

## 8.1 py2dmat_neighborlist

This tool generates a neighborhood-list file from the mesh file.

When you install py2dmat via `pip` command, `py2dmat_neighborlist` is also installed under the `bin`. A python script `src/py2dmat_neighborlist.py` is also available.

### 8.1.1 Usage

Pass a path to the mesh file as an argument. The filename of the generated neighborhood-list file is specified by `-o` option.

```
$ py2dmat_neighborlist -o neighborlist.txt MeshData.txt

Or

$ python3 src/py2dmat_neighborlist.py -o MeshData.txt
```

The following command-line options are available.

- `-o output` or `--output output`
  - The filename of output (default: `neighborlist.txt`)
- `-u "unit1 unit2..."` or `--unit "unit1 unit2..."`
  - Length scale for each dimension of coordinate (default: 1.0 for all dims)
    - Put values splitted by whitespaces and quote the whole
      - e.g.) `-u "1.0 0.5"`
  - Each dimension of coordinate is divided by the corresponding `unit`.
- `-r radius` or `--radius radius`
  - A pair of nodes where the Euclidean distance is less than `radius` is considered a neighborhood (default: 1.0)
  - Distances are calculated in the space after coordinates are divided by `-u`
- `-q` or `--quiet`
  - Do not show a progress bar
  - Showing a progress bar requires `tqdm` python package
- `--allow-selfloop`

  - Include $i$ in the neighborhoods of $i$ itself
- `--check-allpairs`
  - Calculate distances of all pairs
  - This is for debug

MPI parallelization is available.

## 8.2 `tool/to_dft/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.

### 8.2.1 Prerequisites

- Python3 >= 3.6

The following packages are required:

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

### 8.2.2 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.

### 8.2.3 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
```

(continues on next page)

(continued from previous page)

```
 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
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: Solver`, `info: Info`, and `mapping: Callable[[np.ndarray], np.ndarray]`.

`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)
```

`submit` internally uses `mapping` for generating a parameter used in `Solver`, $y$, from a parameter searched by `Algorithm`, $x$, as `y = mapping(x)`. When `mapping` is omitted in the constructor (or `None` is passed), an affine mapping (`py2dmat.util.mapping.Affine(A,b)`) $y = Ax + b$ is used as `mapping`. The elements of `A` and `b` are defined in `info`. See *Input file* for details how/which components of `info` `Runner` uses.

## 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.
- `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
- `run(self, nprocs: int = 1, nthreads: int = 1) -> None`
  - Run the solver
  - Result should be saved to somewhere in order to be read by `get_results` later
    - e.g., save f(x) as an instance variable
- `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

## 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 tomli
import py2dmat

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

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


# (2)
with open(sys.argv[1]) as f:
    inp = tomli.load(f)
info = py2dmat.Info(inp)

# (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. A part of the design of and naming in software is inspired by abics For the implementation of the forward problem solver, we thank to T. Hanada (Tohoku Univ.), I. Mochizuki (KEK), W. Voegeli (Tokyo Gakugei Univ.), T. Shirasawa(AIST), R. Ahmed (Kyushu Univ.), and K. Wada(KEK). For adding a tutorial for customizing solver, we thank to K. Tsukamoto (ISSP).

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 @)

# BIBLIOGRAPHY

[SATLEED] M.A. Van Hove, W. Moritz, H. Over, P.J. Rous, A. Wander, A. Barbieri, N. Materer, U. Starke, G.A. Somorjai, Automated determination of complex surface structures by LEED, Surface Science Reports, Volume 19, 191-229 (1993). https://doi.org/10.1016/0167-5729(93)90011-D