2DMAT Documentation

リリース *2.2.0*

University of Tokyo

2024 年 03 月 28 日

Contents:

第1章　はじめに

1.1 2DMAT とは

2DMAT は，順問題ソルバーに対して探索アルゴリズムを適用して最適解を探すためのフレームワークです. 順問題ソルバーはユーザー自身で定義することが可能です. 標準的な順問題ソルバーとしては 2 次元物質構造解析向け実験データ解析ソフトウェアが用意されています. 順問題ソルバーでは原子位置などをパラメータとし得られたデータと実験データとのずれを損失関数として与えます. 探索アルゴリズムを組み合わせ, この損失関数を最小化することで, 最適なパラメータを推定します. 現バージョンでは, 順問題ソルバーとして量子ビーム回折実験の全反射高速陽電子回折実験（Total-reflection high-energy positron diffraction ,TRHEPD，トレプト）[1, 2],sxrd[3], leed[4] に対応しており, 探索アルゴリズムは Nelder-Mead 法 [5], グリッド型探索法 [6], ベイズ最適化 [7], レプリカ交換モンテカルロ法 [8], ポピュレーションアニーリングモンテカルロ法 [9, 10, 11] が実装されています. 今後は, 本フレームワークをもとにより多くの順問題ソルバーおよび探索アルゴリズムを実装していく予定です.

[1] レビューとして， Y．Fukaya, et al., J. Phys. D: Appl. Phys. 52, 013002 (2019); 兵頭俊夫,「全反射高速陽電子回折　(TRHEPD) による表面構造解析」，固体物理 53, 705 (2018).

[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 ライセンス

本ソフトウェアのプログラムパッケージおよびソースコード一式は GNU General Public License version 3（GPL v3）に準じて配布されています。

本ソフトウェアは 2020 年度 東京大学物性研究所 ソフトウェア高度化プロジェクトの支援を受け開発されました。2DMAT を引用する際には以下の文献を引用してください。

“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 バージョン履歴

- 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 主な開発者

2DMAT は以下のメンバーで開発しています.

- v2.0.0 -
  - 本山 裕一 (東京大学 物性研究所)
  - 吉見 一慶 (東京大学 物性研究所)
  - 岩本 晴道 (鳥取大学 大学院工学研究科)
  - 星 健夫 (鳥取大学 大学院工学研究科)
- v0.1.0 -

- 本山 裕一 (東京大学 物性研究所)
- 吉見 一慶 (東京大学 物性研究所)
- 星 健夫 (鳥取大学 大学院工学研究科)

第2章　py2dmat のインストール

2.1 実行環境・必要なパッケージ

- python 3.6.8 以上
  - 必要な python パッケージ
    - tomli (>= 1.2)
    - numpy (>= 1.14)
  - Optional なパッケージ
    - mpi4py (グリッド探索利用時)
    - scipy (Nelder-Mead 法利用時)
    - physbo (ベイズ最適化利用時, ver. 0.3 以上)

2.2 ダウンロード・インストール

下記に示す方法で、 `py2dmat` python パッケージと `py2dmat` コマンドがインストールできます。

- PyPI からのインストール（推奨）
  - `python3 -m pip install py2dmat`
    - `--user` オプションをつけるとローカル (`$HOME/.local`) にインストールできます
    - `py2dmat[all]` とすると Optional なパッケージも同時にインストールします
- ソースコードからのインストール
  1. `git clone https://github.com/issp-center-dev/2DMAT`
  2. `python3 -m pip install ./2DMAT`
     - `pip` のバージョンは 19 以上が必要です (`python3 -m pip install -U pip` で更新可能)
- サンプルファイルのダウンロード
  - サンプルファイルはソースコードに同梱されています。
  - `git clone https://github.com/issp-center-dev/2DMAT`

なお、 `py2dmat` で用いる順問題ソルバーのうち、

- TRHEPD 順問題ソルバー (`sim-trhepd-rheed`)

- SXRD 順問題ソルバー (sxrdcalc)
- LEED 順問題ソルバー (satleed)

については、別途インストールが必要です。インストールの詳細については各ソルバーのチュートリアルを参照してください。

2.3 実行方法

py2dmat コマンドは定義済みの最適化アルゴリズム Algorithm と順問題ソルバー Solver の組み合わせで解析を行います。:

```
$ py2dmat input.toml
```

定義済みの Algorithm については **探索アルゴリズム** を、Solver については **順問題ソルバー** を参照してください。

Algorithm や Solver をユーザーが準備する場合は、 py2dmat パッケージを利用します。詳しくは *(開発者向け) ユーザー定義アルゴリズム・ソルバー* を参照してください。

なお、プログラムを改造する場合など、 py2dmat コマンドをインストールせずに直接実行することも可能です。src/py2dmat_main.py を利用してください。:

```
$ py2dmat src/py2dmat_main.py input.toml
```

2.4 アンインストール

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

第3章　チュートリアル

順問題ソルバーとして用意されている sim_trhepd_rheed は東北大学の花田貴先生によって開発された物質反射高速（陽）電子回折 (RHEED, TRHEPD) の解析ソフトウェアをベースに作成されています。TRHEPD では原子座標を与えた場合に、回折データがシミュレーション結果として与えられます。そのため、原子座標から回折データへの順問題を取り扱っているといえます。一方、多くの場合回折データは実験で与えられ、それを再現するような原子座標などが求められます。これらは上記の順問題に対して、逆問題に相当します。

本ソフトウェアでは逆問題を解くためのアルゴリズムとして

- minsearch

  Nealder-Mead 法を用いもっともらしい原子座標を推定

- mapper_mpi

  与えられたパラメータの探索グリッドを全探索することで、もっともらしい原子座標を推定

- bayes

  ベイズ最適化を用いもっともらしい原子座標を推定

- exchange

  レプリカ交換モンテカルロ法を用いてもっともらしい原子座標をサンプリング

- pamc

  ポピュレーションアニーリング法を用いてもっともらしい原子座標をサンプリング

の 5 つのアルゴリズムが用意されています。本チュートリアルでは、最初に順問題プログラム sim_trhepd_rheed の実行方法、その後に minsearch , mapper_mpi, bayes, exchange, pamc の実行方法について順に説明します。また、制約式を用いて探索範囲を制限出来る [runner.limitation] セクションを使用した実行方法も説明しています。最後に、自分で順問題ソルバーを定義する簡単な例について説明します。

3.1 TRHEPD 順問題ソルバー

2DMAT は順問題ソルバーのひとつとして、 反射高速 (陽) 電子回折 (RHEED, TRHEPD) の強度計算 (A. Ichimiya, Jpn. J. Appl. Phys. 22, 176 (1983); 24, 1365 (1985)) を行うプログラム sim-trhepd-rheed のラッパーを提供しています。本チュートリアルでは sim-trhepd-rheed を用い、様々なアルゴリズムを利用した解析を行います。最初に、チュートリアルを行うために必要な sim-trhepd-rheed のインストールおよびテストを行います。

3.1.1 ダウンロード・インストール

まずチュートリアルの前提として、 2DMAT フォルダがある場所にいることを仮定します。:

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

GitHub の sim-trhepd-rheed リポジトリから、ソースコード一式を入手し、ビルドします。:

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

make が成功すると、 bulk.exe 及び surf.exe が作成されます。

3.1.2 計算実行

sim-trhepd-rheed では、最初に bulk.exe で表面構造のバルク部分に関する計算をします。その後、bulk.exe の計算結果 (bulkP.b ファイル) を用いて、 surf.exe 表面構造の表面部分を計算します。

このチュートリアルでは実際に、TRHEPD 計算をしてみます。サンプルとなる入力ファイルは 2DMAT の sample/sim-trhepd-rheed にあります。まず、このフォルダを適当な作業用フォルダ work にコピーします。

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

次に bulk.exe と surf.exe を work にコピーします

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

bulk.exe を実行します。

```
./bulk.exe
```

上記実行時に、以下のように出力され、バルクファイル bulkP.b が生成されます。

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

続いて、 surf.exe を実行します。

```
./surf.exe
```

上記実行時に、以下のように出力されます。

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

実行後に、ファイル surf-bulkP.md 、 surf-bulkP.s 及び SURFYYYYMMDD-HHMMSSlog.txt が生成されます。(YYYYMMDD、 HHMMSS には実行日時に対応した数字が入ります)

3.1.3 計算結果の可視化

surf-bulkP.s は以下の通りです。

```
#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,
(以下略)
```

上記ファイルより、縦軸に角度（5 行目以降の 1 列目データ）と (0,0) ピークの強度（5 行目以降の 8 列目データ）からロッキングカーブを作成します。Gnuplot 等のグラフソフトを用いる事も出来ますが、ここでは、 2DMAT/script フォルダにあるプログラム plot_bulkP.py を用います。以下のように実行して下さい。

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

以下のような plot_bulkP.png が作成されます。

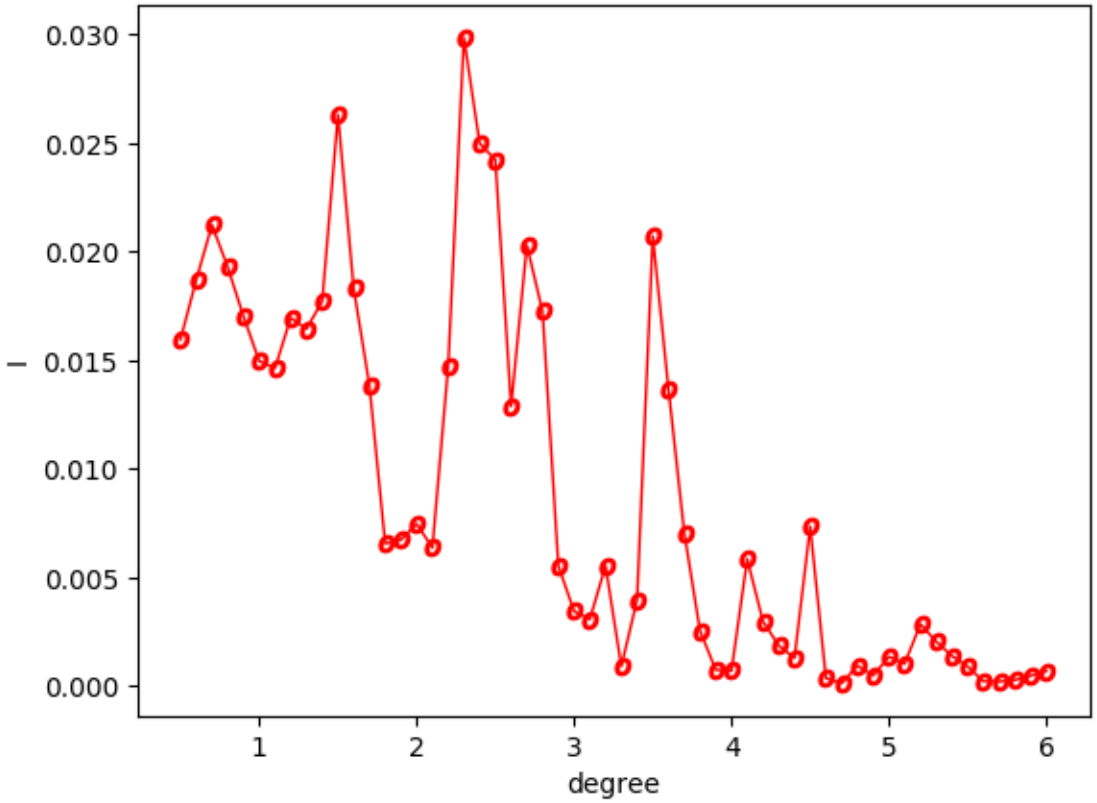


図 1: Si(001)-2x1 面のロッキングカーブ。

この 00 ピークの回折強度のデータに対し、コンボリューションを掛けたうえで規格化します。surf-bulkP.s を準備して、 make_convolution.py を実行してください。

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

実行すると、以下のようなファイル convolution.txt ができあがります。

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

1 列目が視射角、2 列目が surf-bulkP.s に書かれた 00 ピーク回折強度のデータに半値幅 0.5 のコンボリューションを付加して規格化したものです。

3.2 Nelder-Mead 法による最適化

ここでは、Nelder-Mead 法を用いて回折データから原子座標を解析する逆問題の計算を行う方法について説明します。具体的な計算手順は以下の通りです。

0. 参照ファイルの準備

   合わせたい参照ファイル (今回は後述する `experiment.txt` に相当) を準備する。

1. 表面構造のバルク部分に関する計算実行

   `bulk.exe` を `sample/sim-trhepd-rheed/minsearch` にコピーして計算を実行する。

2. メインプログラムの実行

   `src/py2dmat_main.py` を用いて計算実行し原子座標を推定する。

メインプログラムでは、Nelder-Mead 法 (scipy.optimize.fmin を使用) を用いて、ソルバー (今回は `surf.exe`) を用いて得られた強度と、参照ファイル (`experiment.txt`) に記載された強度のずれ (R 値) を最小化するパラメータを探索します。

3.2.1 サンプルファイルの場所

サンプルファイルは `sample/py2dmat/sim-threpd-rheed/single_beam/minsearch` にあります。フォルダには以下のファイルが格納されています。

- `bulk.txt`

  `bulk.exe` の入力ファイル

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

  メインプログラムでの計算を進めるための参照ファイル

- `ref.txt`

  本チュートリアルで求めたい回答が記載されたファイル

- `input.toml`

  メインプログラムの入力ファイル

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

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.2.2 参照ファイルの説明

template.txt は surf.exe の入力ファイルとほぼ同じ形式のファイルです。動かすパラメータ（求めたい原子座標などの値）を「value_*」などの適当な文字列に書き換えられています。以下が、template.txt の中身です。

```
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)
```

今回の入力ファイルでは、value_01, value_02, value_03 を用いています。サンプルフォルダには、原子位置が正しく推定できているかを知るための参照ファイルとして、ref.txt が置いてあります。ファイルの中身は

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

となっており、value_0x が z_x (x=1, 2, 3) に対応しています。fx は目的関数の最適値です。experiment.txt は、メインプログラムで参照に用いるファイルで、template.txt に ref.txt の入っているパラメータをいれ、順問題のチュートリアルと同様な手順で計算して得られる convolution.txt に相当しています (順問題のチュートリアルとは bulk.exe , suft.exe の入力ファイルが異なるのでご注意ください)。

3.2.3 入力ファイルの説明

ここでは、メインプログラム用の入力ファイル input.toml の準備をします。input.toml の詳細については入力ファイルに記載されています。ここでは、サンプルファイルにある input.toml の中身について説明します。

```
[base]
dimension = 3
```

(次のページに続く)

(前のページからの続き)

```
[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

[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]
```

最初に [base] セクションについて説明します。

- dimension は最適化したい変数の個数で、今の場合は template.txt で説明したように 3 つの変数の最適化を行うので、3 を指定します。

[solver] セクションではメインプログラムの内部で使用するソルバーとその設定を指定します。

- name は使用したいソルバーの名前で、このチュートリアルでは、 sim-trhepd-rheed を用いた解析を行うので、 sim-trhepd-rheed を指定します。

ソルバーの設定は、サブセクションの [solver.config], [solver.param], [solver.reference] で行います。

[solver.config] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイルを読み込む際のオプションを指定します。

- calculated_first_line は出力ファイルを読み込む最初の行数を指定します。
- calculated_last_line は出力ファイルを読み込む最後の行数を指定します。
- row_number は出力ファイルの何列目を読み込むかを指定します。

[solver.param] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイ

ルを読み込む際のオプションを指定します。

- string_list は、 template.txt で読み込む、動かしたい変数の名前のリストです。
- degree_max は、最大角度（度単位）の指定をします。

[solver.reference] セクションでは、実験データの置いてある場所と読みこむ範囲を指定します。

- path は実験データが置いてあるパスを指定します。
- first は実験データファイルを読み込む最初の行数を指定します。
- end は実験データファイルを読み込む最後の行数を指定します。

[algorithm] セクションでは、使用するアルゴリズムとその設定をします。

- name は使用したいアルゴリズムの名前で、このチュートリアルでは、Nelder-Mead 法 を用いた解析を行うので、 minsearch を指定します。
- label_list は、value_0x (x=1,2,3) を出力する際につけるラベル名のリストです。

[algorithm.param] セクションでは、探索するパラメータの範囲や初期値を指定します。

- min_list と max_list はそれぞれ探索範囲の最小値と最大値を指定します。
- initial_list は初期値を指定します。

ここではデフォルト値を用いるため省略しましたが、その他のパラメータ、例えば Nelder-Mead 法で使用する収束判定などについては、[algorithm] セクションで行うことが可能です。詳細については入力ファイルの章をご覧ください。

3.2.4 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

```
cd sample/sim-trhepd-rheed/single_beam/minsearch
```

順問題の時と同様に、bulk.exe と surf.exe をコピーします。

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

最初に bulk.exe を実行し、bulkP.b を作成します。

```
./bulk.exe
```

そのあとに、メインプログラムを実行します (計算時間は通常の PC で数秒程度で終わります)。

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

実行すると、以下の様な出力がされます。

```
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']
...
```

z1, z2, z3 に各ステップでの候補パラメータと、その時の ``R-factor`` が出力されます。また各ステップでの計算結果は Logxxxxx (xxxxx にステップ数) のフォルダに出力されます。最終的に推定されたパラメータは、res.dat に出力されます。今の場合、

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

が得られ、正解のデータ ref.txt と同じ値が得られていることがわかります。なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では res.txt と ref.txt の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
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
```

(次のページに続く)

(前のページからの続き)

```
  echo Test FAILED: res.txt and ref.txt differ
  false
fi
```

3.2.5 計算結果の可視化

それぞれのステップでのロッキングカーブのデータは、LogXXX_00000001 (XXX はステップ数) フォルダに RockingCurve.txt として保存されています (LogXXX_00000000 フォルダは Nelder-Mead 法の途中での評価です)。このデータを可視化するツール draw_RC_double.py が準備されています。ここでは、このツールを利用して結果を可視化します。

```
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
```

上記を実行することで、RC_double.png が出力されます。

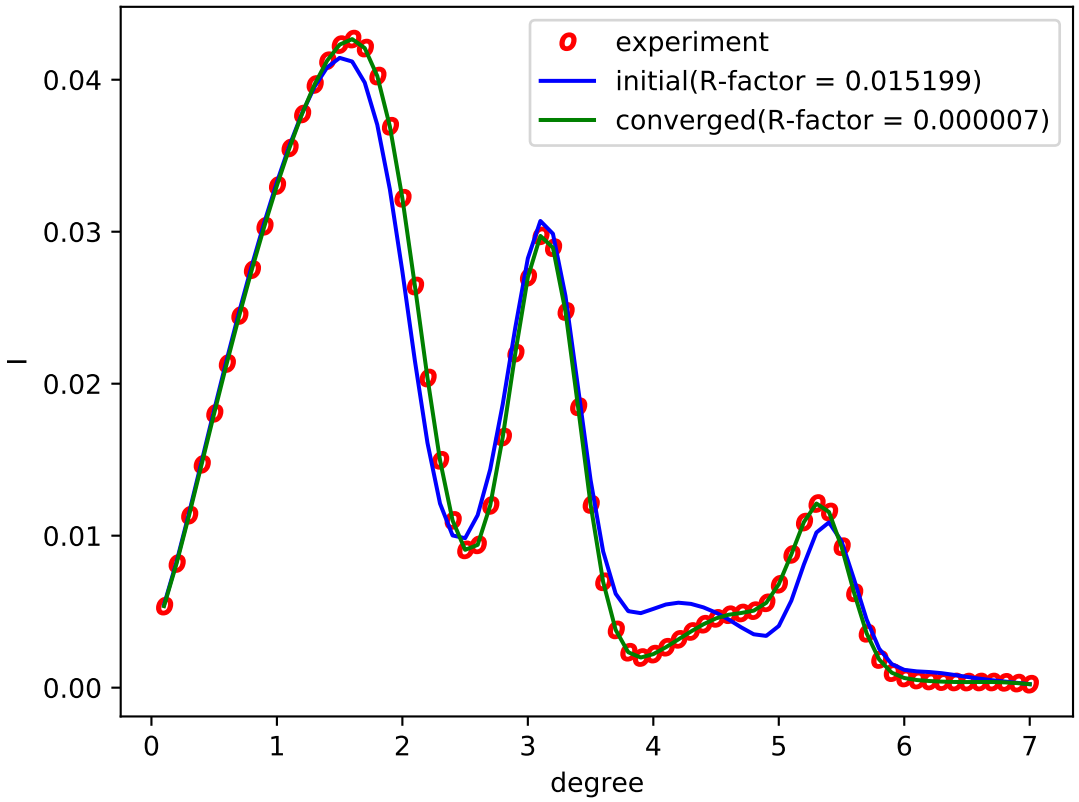


図 2: Nelder-Mead 法を用いた解析。赤丸が実験値、青線が最初のステップ、緑線が最後のステップで得られたロッキングカーブを表す。

図から最後のステップでは実験値と一致していることがわかります。

3.3 グリッド型探索

ここでは、グリッド型探索を行い、回折データから原子座標を解析する方法について説明します。グリッド型探索は MPI に対応しています。具体的な計算手順は `minsearch` の時と同様です。ただし、探索グリッドを与えるデータ `MeshData.txt` を事前に準備する必要があります。

3.3.1 サンプルファイルの場所

サンプルファイルは `sample/sim-trhepd-rheed/single_beam/mapper` にあります。フォルダには以下のファイルが格納されています。

- `bulk.txt`

  `bulk.exe` の入力ファイル

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

  メインプログラムでの計算を進めるための参照ファイル

- `ref_ColorMap.txt`

  計算が正しく実行されたか確認するためのファイル (本チュートリアルを行うことで得られる `ColorMap.txt` の回答)。

- `input.toml`

  メインプログラムの入力ファイル

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

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.3.2 参照ファイルの説明

`template.txt` , `experiment.txt` については、前のチュートリアル (Nealder-Mead 法による最適化) と同じものを使用します。ただし、計算を軽くするため `value_03` は用いずに `3.5` に固定し、2 次元のグリッド探索を行うように変更してあります。実際に探索するグリッドは `MeshData.txt` で与えます。サンプルでは `MeshData.txt` の中身は以下のようになっています。

```
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
```

(次のページに続く)

(前のページからの続き)

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

1 列目が通し番号、2 列目以降は template.txt に入る ``value_0`` , value_1 の値が順に指定されています。

3.3.3 入力ファイルの説明

ここでは、メインプログラム用の入力ファイル input.toml について説明します。input.toml の詳細については入力ファイルに記載されています。以下は、サンプルファイルにある input.toml の中身になります。

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

最初に [base] セクションについて説明します。

- dimension は最適化したい変数の個数で、今の場合は template.txt で説明したように 2 つの変数の最適化を行うので、2 を指定します。

[solver] セクションではメインプログラムの内部で使用するソルバーとその設定を指定します。

- name は使用したいソルバーの名前で、このチュートリアルでは、 sim-trhepd-rheed を用いた解析を行うので、 sim-trhepd-rheed を指定します。

ソルバーの設定は、サブセクションの [solver.config], [solver.param], [solver.reference] で行います。

[solver.config] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイルを読み込む際のオプションを指定します。

- calculated_first_line は出力ファイルを読み込む最初の行数を指定します。
- calculated_last_line は出力ファイルを読み込む最後の行数を指定します。
- row_number は出力ファイルの何列目を読み込むかを指定します。

[solver.param] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイルを読み込む際のオプションを指定します。

- string_list は、 template.txt で読み込む、動かしたい変数の名前のリストです。
- label_list は、value_0x (x=1,2) を出力する際につけるラベル名のリストです。
- degree_max は、最大角度（度単位）の指定をします。

[solver.reference] セクションでは、実験データの置いてある場所と読みこむ範囲を指定します。

- path は実験データが置いてあるパスを指定します。
- first は実験データファイルを読み込む最初の行数を指定します。
- end は実験データファイルを読み込む最後の行数を指定します。

[algorithm] セクションでは、使用するアルゴリズムとその設定をします。

- name は使用したいアルゴリズムの名前で、このチュートリアルでは、グリッド探索による解析を行うので、 mapper を指定します。
- label_list は、value_0x (x=1,2) を出力する際につけるラベル名のリストです。

その他、入力ファイルで指定可能なパラメータの詳細については入力ファイルの章をご覧ください。

3.3.4 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

```
cd sample/sim-trhepd-rheed/single_beam/mapper
```

順問題の時と同様に、bulk.exe と surf.exe をコピーします。

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

最初に bulk.exe を実行し、bulkP.b を作成します。

```
./bulk.exe
```

そのあとに、メインプログラムを実行します (計算時間は通常の PC で数秒程度で終わります)。

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

ここではプロセス数 2 の MPI 並列を用いた計算を行っています。実行すると、各ランクのフォルダが作成され、その下に更にグリッドの id がついたサブフォルダ Log%%%%% (%%%%% がグリッドの id) が作成されます (MeshData.txt に付けられた番号がグリッドの id として割り振られます)。以下の様な標準出力がされます。

```
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
...
```

z1, z2 に各メッシュでの候補パラメータと、その時の ``R-factor`` が出力されます。最終的にグリッド上の全ての点で計算された R-factor は、ColorMap.txt に出力されます。今回の場合は

```
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
...
```

のように得られます。1 列目、2 列目に value_01, value_02 の値が、3 列目に R-factor が記載されます。なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では ColorMap.dat と ref_ColorMap.dat の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
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
```

(次のページに続く)

(前のページからの続き)

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

3.3.5 計算結果の可視化

ColorMap.txt を図示することで、R-factor の小さいパラメータがどこにあるかを推定することができます。今回の場合は、以下のコマンドをうつことで 2 次元パラメータ空間の図 ColorMapFig.png が作成されます。

```
python3 plot_colormap_2d.py
```

作成された図を見ると、(5.25, 4.25) 付近に最小値を持っていることがわかります。

また、 RockingCurve.txt が各サブフォルダに格納されています。これを用いることで、前チュートリアルの手順に従い、実験値との比較も行うことが可能です。

3.4 ベイズ最適化

ここでは、ベイズ最適化を行い、回折データから原子座標を解析する方法について説明します。ベイズ最適化には PHYSBO を用いています。グリッド探索と同様に、探索グリッドを与えるデータ MeshData.txt を事前に準備する必要があります。

3.4.1 サンプルファイルの場所

サンプルファイルは sample/sim-trhepd-rheed/single_beam/bayes にあります。フォルダには以下のファイルが格納されています。

- bulk.txt

  bulk.exe の入力ファイル

- experiment.txt , template.txt

  メインプログラムでの計算を進めるための参照ファイル

- ref_BayesData.txt

  計算が正しく実行されたか確認するためのファイル (本チュートリアルを行うことで得られる ColorMap.txt の回答)。

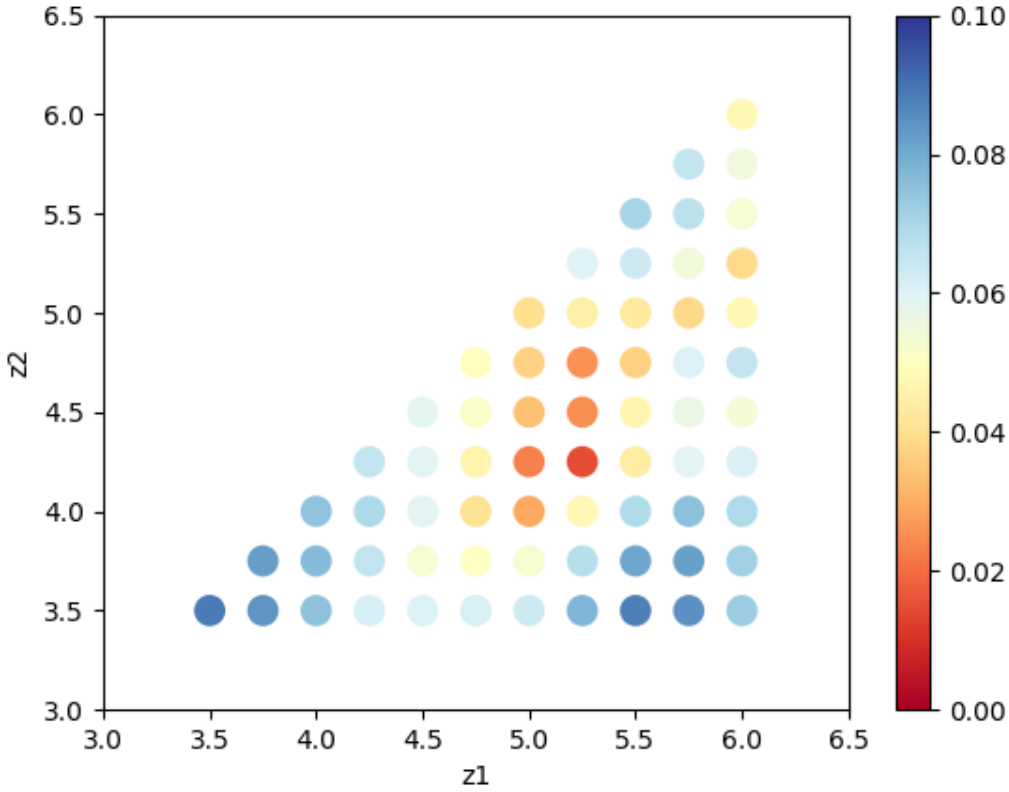


図 3: 2 次元パラメータ空間上での `R-factor` 。

- input.toml

  メインプログラムの入力ファイル

- prepare.sh , do.sh

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.4.2 参照ファイルの説明

template.txt , experiment.txt については、前のチュートリアル (Nealder-Mead 法による最適化) と同じものを使用します。ただし、計算を軽くするため value_03 は用いずに 3.5 に固定し、2 次元のグリッド探索を行うように変更してあります。実際に探索するグリッドは MeshData.txt で与えます。サンプルでは MeshData.txt の中身は以下のようになっています。

```
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
  ...
```

1 列目が通し番号、2 列目以降は template.txt に入る value_0 , value_1 の値が順に指定されています。

3.4.3 入力ファイルの説明

ここでは、メインプログラム用の入力ファイル input.toml について説明します。input.toml の詳細については入力ファイルに記載されています。以下は、サンプルファイルにある input.toml の中身になります。

```
[base]
dimension = 2

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

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
```

(次のページに続く)

(前のページからの続き)

```
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
```

最初に [base] セクションについて説明します。

- dimension は最適化したい変数の個数で、今の場合は template.txt で説明したように 2 つの変数の最適化を行うので、 2 を指定します。

[solver] セクションではメインプログラムの内部で使用するソルバーとその設定を指定します。

- name は使用したいソルバーの名前で、このチュートリアルでは、 sim-trhepd-rheed を用いた解析を行うので、 sim-trhepd-rheed を指定します。

ソルバーの設定は、サブセクションの [solver.config], [solver.param], [solver.reference] で行います。

[solver.config] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイルを読み込む際のオプションを指定します。

- calculated_first_line は出力ファイルを読み込む最初の行数を指定します。
- calculated_last_line は出力ファイルを読み込む最後の行数を指定します。
- row_number は出力ファイルの何列目を読み込むかを指定します。

[solver.param] セクションではメインプログラム内部で呼び出す surf.exe により得られた出力ファイルを読み込む際のオプションを指定します。

- string_list は、 template.txt で読み込む、動かしたい変数の名前のリストです。
- label_list は、 value_0x (x=1,2) を出力する際につけるラベル名のリストです。

- degree_max は、最大角度（度単位）の指定をします。

[solver.reference] セクションでは、実験データの置いてある場所と読みこむ範囲を指定します。

- path は実験データが置いてあるパスを指定します。
- first は実験データファイルを読み込む最初の行数を指定します。
- end は実験データファイルを読み込む最後の行数を指定します。

[algorithm] セクションでは、使用するアルゴリズムとその設定をします。

- name は使用したいアルゴリズムの名前で、このチュートリアルでは、ベイズ最適化による解析を行うので、 bayes を指定します。
- label_list は、 value_0x (x=1,2) を出力する際につけるラベル名のリストです。

[algorithm.param] セクションで、探索パラメータを設定します。

- mesh_path はメッシュファイルへのパスを設定します。

[algorithm.bayes] セクションでは、ベイズ最適化のハイパーパラメータを設定します。

- random_max_num_probes は、ベイズ最適化を行う前のランダム探索する回数を指定します。
- bayes_max_num_probes は、ベイズ探索を行う回数を指定します。

その他、入力ファイルで指定可能なパラメータの詳細については入力ファイルの章をご覧ください。

3.4.4 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

```
cd sample/sim-trhepd-rheed/bayes
```

順問題の時と同様に、 bulk.exe と surf.exe をコピーします。

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

最初に bulk.exe を実行し、 bulkP.b を作成します。

```
./bulk.exe
```

そのあとに、メインプログラムを実行します (計算時間は通常の PC で数秒程度で終わります)。

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

実行すると、各ランクのフォルダが作成されます。以下の様な標準出力がされます。

```
# 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


  ...
```

最初に設定したパラメータのリスト、そのあとに各ステップでの候補パラメータと、その時の R-factor にマイナスが乗算された f(x)``が出力されます。また、その時点での一番良いスコアを持つグリッドインデックス (``action) とその場合の f(x) と変数が出力されます。0 番の下には更にグリッドの id がついたサブフォルダ Log%%%%% (%%%%% がグリッドの id) が作成され、ソルバーの出力が保存されます (MeshData.txt に付けられた番号がグリッドの id として割り振られます)。最終的に推定されたパラメータは、BayesData.txt に出力されます。

今回の場合は

```
#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
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
```

(次のページに続く)

(前のページからの続き)

```
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
  ...
```

のように得られます。1 列目にステップ数、2 列目、3 列目、4 列目にその時点での最高スコアを与える value_01, value_02 と R-factor が記載されます。続けて、そのステップで候補となった value_01, value_02 と R-factor が記載されます。今回の場合は 21 ステップ目で正しい解が得られていることがわかります。

なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では BayesData.dat と ref_BayesData.dat の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
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 計算結果の可視化

BayesData.txt を参照することで、何ステップ目のパラメータが最小スコアを与えたかがわかります。RockingCurve.txt は各ステップ毎にサブフォルダに格納されているので、minsearch.rst の手順に従い、実験値との比較を行うことが可能です。

3.5 レプリカ交換モンテカルロ法による探索

ここでは、レプリカ交換モンテカルロ法によって、回折データから原子座標を解析する方法について説明します。具体的な計算手順は minsearch の時と同様です。

3.5.1 サンプルファイルの場所

サンプルファイルは sample/sim-threpd-rheed/single_beam/exchange にあります。フォルダには以下のファイルが格納されています。

- bulk.txt

  bulk.exe の入力ファイル

- experiment.txt , template.txt

  メインプログラムでの計算を進めるための参照ファイル

- ref.txt

  計算が正しく実行されたか確認するためのファイル (本チュートリアルを行うことで得られる best_result.txt の回答)。

- input.toml

  メインプログラムの入力ファイル

- prepare.sh , do.sh

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.5.2 参照ファイルの説明

template.txt , experiment.txt については、前のチュートリアル (Nealder-Mead 法による最適化) と同じものを使用します。

3.5.3 入力ファイルの説明

ここでは、メインプログラム用の入力ファイル input.toml について説明します。input.toml の詳細については入力ファイルに記載されています。以下は、サンプルファイルにある input.toml の中身になります。

```
[base]
dimension = 2

[algorithm]
```

(次のページに続く)

(前のページからの続き)

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

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

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

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

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

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

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

ここではこの入力ファイルを簡単に説明します。詳細は入力ファイルのレファレンスを参照してください。

[base] セクションはメインプログラム全体のパラメータです。dimension は最適化したい変数の個数で、今の場合は 2 つの変数の最適化を行うので、2 を指定します。

[algorithm] セクションは用いる探索アルゴリズムを設定します。交換モンテカルロ法を用いる場合には、name に "exchange" を指定します。label_list は、value_0x (x=1,2) を出力する際につけるラベル名のリストです。seed は擬似乱数生成器に与える種です。

[algorithm.param] サブセクションは、最適化したいパラメータの範囲などを指定します。min_list は最小値、 max_list は最大値を示します。

[algorithm.exchange] サブセクションは、交換モンテカルロ法のハイパーパラメータを指定します。

- numstep はモンテカルロ更新の回数です。

- numsteps_exchange で指定した回数のモンテカルロ更新の後に、温度交換を試みます。
- Tmin, Tmax はそれぞれ温度の下限・上限です。
- Tlogspace が true の場合、温度を対数空間で等分割します

[solver] セクションではメインプログラムの内部で使用するソルバーを指定します。minsearch のチュートリアルを参照してください。

3.5.4 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

```
cd sample/sim-trhepd-rheed/single_beam/exchange
```

順問題の時と同様に、 bulk.exe と surf.exe をコピーします。

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

最初に bulk.exe を実行し、bulkP.b を作成します。

```
./bulk.exe
```

そのあとに、メインプログラムを実行します (計算時間は通常の PC で数秒程度で終わります)。

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

ここではプロセス数 4 の MPI 並列を用いた計算を行っています。(Open MPI を用いる場合で、使えるコア数よりも要求プロセス数の方が多い時には、mpiexec コマンドに --oversubscribed オプションを追加してください。) 実行すると、各ランクのフォルダが作成され、各モンテカルロステップで評価したパラメータおよび目的関数の値を記した trial.txt ファイルと、実際に採択されたパラメータを記した result.txt ファイルが作成されます。ともに書式は同じで、最初の 2 列がステップ数とプロセス内の walker 番号、次が温度、3 列目が目的関数の値、4 列目以降がパラメータです。

```
# 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
```

また、 sim-trhepd-rheed ソルバーの場合は、 各作業フォルダの下にサブフォルダ Log%%%%% (%%%%% がグリッドの id) が作成され、ロッキングカーブの情報などが記録されます (各プロセスにおけるモンテカルロステップ数が id として割り振られます)。

最後に、 best_result.txt に、目的関数 (R-factor) が最小となったパラメータとそれを得たランク、モンテカルロステップの情報が書き込まれます。

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

なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では best_result.txt と ref.txt の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
sh prepare.sh

./bulk.exe

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

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

3.5.5 後処理

各ランクフォルダにある result.txt には、各レプリカでサンプリングされたデータが記録されていますが、Py2DMat の実装では同一レプリカが様々な温度のサンプルを保持しています。2DMat は、全レプリカの結果から温度ごとのサンプルに整列し直す script/separateT.py スクリプトを提供しています。

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

result_T%.txt に各温度点ごとにまとめなおされたデータが書き込まれます (% は温度点の index)。1 列目がステップ、2 列めがランク、3 列目が目的関数の値、4 列目以降がパラメータです。

```
# 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 計算結果の可視化

result_T%.txt を図示することで、 R-factor の小さいパラメータがどこにあるかを推定することができます。今回の場合は、以下のコマンドをうつことで 2 次元パラメータ空間の図 result.png が作成されます。

```
python3 plot_result_2d.py
```

作成された図を見ると、(5.25, 4.25) と (4.25, 5.25) 付近にサンプルが集中していることと、R-factor の値が小さいことがわかります。

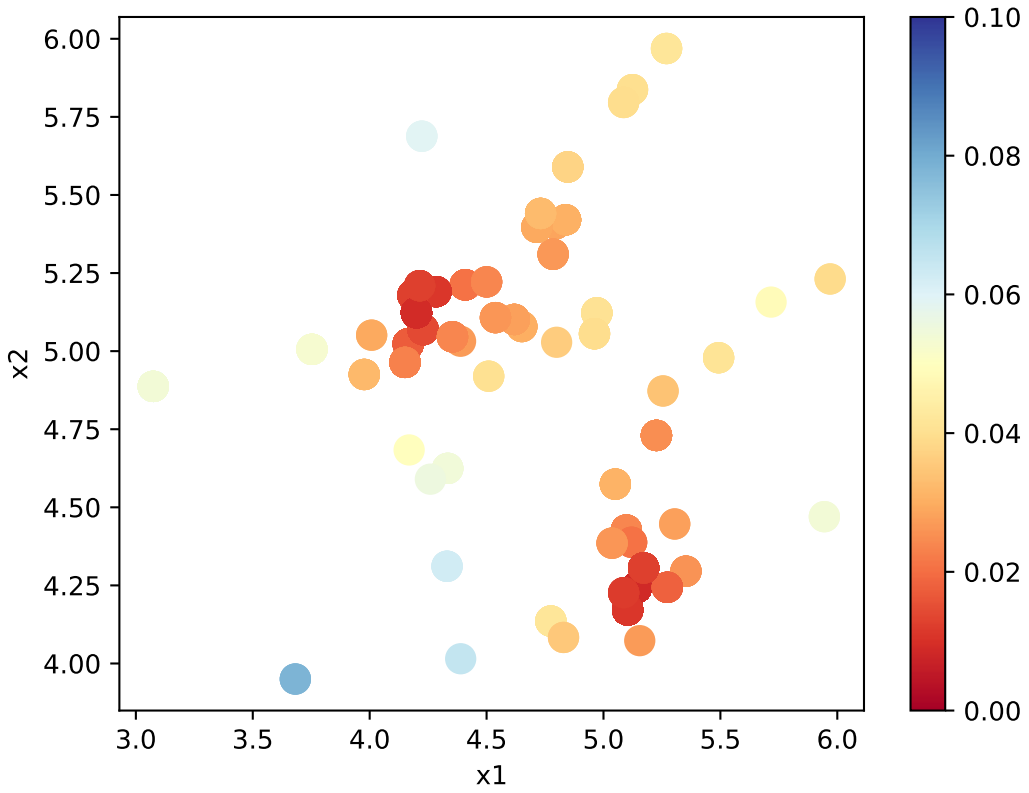


図 4: サンプルされたパラメータと R-factor 。横軸は value_01 , 縦軸は value_02 を表す。

また、 RockingCurve.txt が各サブフォルダに格納されています。LogXXX_YYY の XXX はモンテカルロのステップ数、 YYY は各 MPI プロセス内のレプリカの番号です。これを用いることで、前チュートリアルの手順に従い、実験値との比較も行うことが可能です。

3.6 制約式を適用したレプリカ交換モンテカルロ法による探索

ここでは、 [runner.limitation] セクションに設定できる制約式機能のチュートリアルを示します。例として、レプリカ交換モンテカルロ法を、Himmelblau を対象に探索する計算に制約式を適用します。

3.6.1 サンプルファイルの場所

サンプルファイルは sample/analytical/limitation にあります。フォルダには以下のファイルが格納されています。

- ref.txt

  計算が正しく実行されたか確認するためのファイル (本チュートリアルを行うことで得られる best_result.txt の回答)。

- input.toml

  メインプログラムの入力ファイル。

- do.sh

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.6.2 入力ファイルの説明

メインプログラム用の入力ファイル input.toml について説明します。input.toml の詳細については入力ファイルに記載されています。以下は、サンプルファイルにある input.toml の中身になります。

```
[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
```

(次のページに続く)

(前のページからの続き)

```
numsteps_exchange = 100

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

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

ここではこの入力ファイルを簡単に説明します。詳細は入力ファイルのレファレンスを参照してください。

[base] セクションはメインプログラム全体のパラメータです。dimension は最適化したい変数の個数で、今の場合は 2 つの変数の最適化を行うので、2 を指定します。

[algorithm] セクションは用いる探索アルゴリズムを設定します。交換モンテカルロ法を用いる場合には、name に "exchange" を指定します。seed は擬似乱数生成器に与える種です。

[algorithm.param] サブセクションは、最適化したいパラメータの範囲などを指定します。min_list は最小値、 max_list は最大値を示します。

[algorithm.exchange] サブセクションは、交換モンテカルロ法のハイパーパラメータを指定します。

- numstep はモンテカルロ更新の回数です。
- numsteps_exchange で指定した回数のモンテカルロ更新の後に、温度交換を試みます。
- Tmin, Tmax はそれぞれ温度の下限・上限です。
- Tlogspace が true の場合、温度を対数空間で等分割します。このオプションはデフォルト値が true であるため、今回の input.toml に指定は無いですが、 true になっています。

[solver] セクションではメインプログラムの内部で使用するソルバーを指定します。今回は analytical ソルバーを指定しています。 analytical ソルバーは function_name パラメータを用いて関数を設定します。今回は Himmelblau 関数を指定しています。analytical ソルバーに関してはチュートリアル「順問題ソルバーの追加」を参照してください。

[runner] セクションは [runner.limitation] サブセクションを持ち、この中に制約式を設定します。現在、制約式は N 次元のパラメータ x 、 M 行 N 列の行列 A 、M 次元の縦ベクトル b から定義される $Ax + b > 0$ の制約式が利用可能です。パラメータとしては、以下の項目が設定可能です。

- co_a は行列 A を設定します。
- co_b は縦ベクトル b を設定します。

パラメータの詳しい設定方法はマニュアル内「入力ファイル」項の「 [limitation] セクション」を参照してください。今回は

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

の制約式を課して実行しています。

3.6.3 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

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

そのあとに、メインプログラムを実行します (計算時間は通常の PC で 20 秒程度で終わります)。

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

ここではプロセス数 10 の MPI 並列を用いた計算を行っています。(Open MPI を用いる場合で、使えるコア数よりも要求プロセス数の方が多い時には、mpiexec コマンドに --oversubscribed オプションを追加してください。) 実行すると、output フォルダが生成され、その中に各ランクのフォルダが作成されます。更にその中には、各モンテカルロステップで評価したパラメータおよび目的関数の値を記した trial.txt ファイルと、実際に採択されたパラメータを記した result.txt ファイルが作成されます。ともに書式は同じで、最初の 2 列がステップ数とプロセス内の walker 番号、次が温度、3 列目が目的関数の値、4 列目以降がパラメータです。以下は、 output/0/result.txt ファイルの冒頭部分です。

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

最後に、 output/best_result.txt に、目的関数が最小となったパラメータとそれを得たランク、モンテカルロステップの情報が書き込まれます。

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

なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では best_result.txt と ref.txt の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
#!/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
```

(次のページに続く)

(前のページからの続き)

```
  true
else
  echo TEST FAILED: best_result.txt and ref.txt differ
  false
fi
```

3.6.4 計算結果の可視化

result.txt を図示して、制約式を満たした座標のみを探索しているかを確認します。今回の場合は、以下のコマンドを打つことで 2 次元パラメータ空間の図が <実行日>_histogram フォルダ内に作成されます。生成されるヒストグラムは、burn-in 期間として最初の 1000 ステップ分の探索を捨てたデータを使用しています。

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

作成された図には 2 本の直線 $x_1 - x_2 = 0, x_1 + x_2 - 1 = 0$ と探索結果 (事後確率分布のヒストグラム) を図示しています。図を見ると $x_1 - x_2 > 0, x_1 + x_2 - 1 > 0$ の範囲のみ探索をしていることが確認できます。以下に図の一部を掲載します。

3.7 ポピュレーションアニーリングによる探索

ここでは、ポピュレーションアニーリングを用いて、回折データから原子座標を解析する方法について説明します。具体的な計算手順は minsearch の時と同様です。

3.7.1 サンプルファイルの場所

サンプルファイルは sample/sim-trhepd-rheed/single_beam/pamc にあります。フォルダには以下のファイルが格納されています。

- bulk.txt

  bulk.exe の入力ファイル

- experiment.txt , template.txt

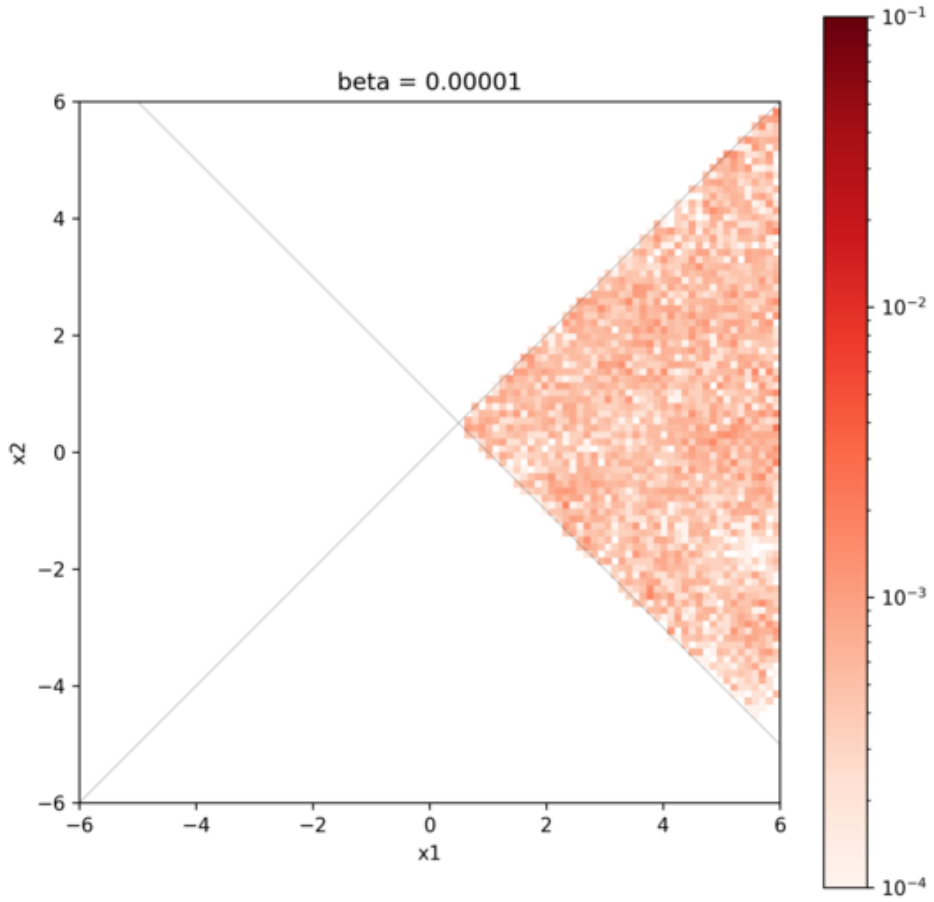
  メインプログラムでの計算を進めるための参照ファイル

- ref.txt

  計算が正しく実行されたか確認するためのファイル (本チュートリアルを行うことで得られる fx.txt の回答)。

- input.toml

  メインプログラムの入力ファイル


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

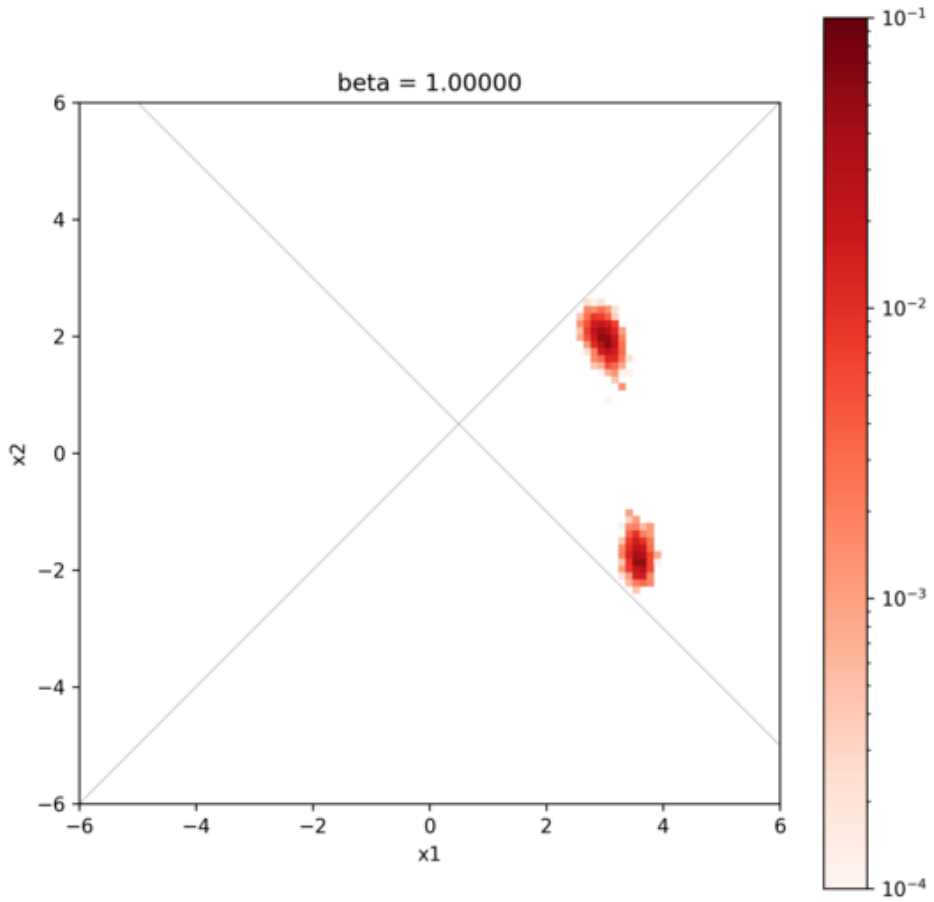


図 5: サンプルされたパラメータと確率分布。横軸は `value_01 (x1)` , 縦軸は `value_02 (x2)` を表す。

- prepare.sh , do.sh

  本チュートリアルを一括計算するために準備されたスクリプト

以下、これらのファイルについて説明したあと、実際の計算結果を紹介します。

3.7.2 参照ファイルの説明

template.txt , experiment.txt については、前のチュートリアル (Nealder-Mead 法による最適化) と同じものを使用します。

3.7.3 入力ファイルの説明

ここでは、メインプログラム用の入力ファイル input.toml について説明します。input.toml の詳細については入力ファイルに記載されています。以下は、サンプルファイルにある input.toml の中身になります。

```
[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

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

[solver.config]
calculated_first_line = 5
calculated_last_line = 74
```

(次のページに続く)

(前のページからの続き)

```
row_number = 2

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

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

ここではこの入力ファイルを簡単に説明します。詳細は入力ファイルのレファレンスを参照してください。

[base] セクションはメインプログラム全体のパラメータです。dimension は最適化したい変数の個数で、今の場合は 2 つの変数の最適化を行うので、2 を指定します。

[algorithm] セクションは用いる探索アルゴリズムを設定します。交換モンテカルロ法を用いる場合には、name に "exchange" を指定します。label_list は、value_0x (x=1,2) を出力する際につけるラベル名のリストです。seed は擬似乱数生成器に与える種です。

[algorithm.param] サブセクションは、最適化したいパラメータの範囲などを指定します。min_list は最小値、 max_list は最大値を示します。unit_list はモンテカルロ更新の際の変化幅 (ガウス分布の偏差) です。

[algorithm.pamc] サブセクションは、ポピュレーションアニーリングのハイパーパラメータを指定します。

- numsteps_annealing で指定した回数のモンテカルロ更新の後に、逆温度を増やします (温度を下げます)。
- bmin, bmax はそれぞれ逆温度の下限・上限です。
- Tnum は計算する温度・逆温度の点数です。
- Tlogspace が true の場合、温度を対数空間で等分割します
- nreplica_per_proc は MPI プロセスひとつが受け持つ計算レプリカの数です。

[solver] セクションではメインプログラムの内部で使用するソルバーを指定します。minsearch のチュートリアルを参照してください。

3.7.4 計算実行

最初にサンプルファイルが置いてあるフォルダへ移動します (以下、本ソフトウェアをダウンロードしたディレクトリ直下にいることを仮定します).

```
cd sample/sim-trhepd-rheed/pamc
```

順問題の時と同様に、 bulk.exe と surf.exe をコピーします。

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

最初に bulk.exe を実行し、bulkP.b を作成します。

```
./bulk.exe
```

そのあとに、メインプログラムを実行します (計算時間は通常の PC で数秒程度で終わります)。

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

ここではプロセス数 4 の MPI 並列を用いた計算を行っています。(Open MPI を用いる場合で、使えるコア数よりも要求プロセス数の方が多い時には、mpiexec コマンドに --oversubscribed オプションを追加してください。) 実行すると、各ランクのフォルダが作成され、温度ごとに、各モンテカルロステップで評価したパラメータおよび目的関数の値を記した trial_TXXX.txt ファイル (XXX は温度点の番号) と、実際に採択されたパラメータを記した result_TXXX.txt ファイル、さらにそれぞれを結合した trial.txt, result.txt ファイルが生成されます。それぞれ書式は同じで、最初の 2 列がステップ数とプロセス内の walker (replica) 番号、次が（逆）温度、3 列目が目的関数の値、4 列目以降がパラメータです。最後の 2 列は、 walker の重み (Neal-Jarzynski weight) と祖先（計算を開始したときのレプリカ）の番号です。

```
# 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
```

また、 sim-trhepd-rheed ソルバーの場合は、 各作業フォルダの下にサブフォルダ Log%%%% (%%%% がグリッドの id) が作成され、ロッキングカーブの情報などが記録されます (各プロセスにおけるモンテカルロステップ数が id として割り振られます)。

best_result.txt に、目的関数 (R-factor) が最小となったパラメータとそれを得たランク、モンテカルロステップの情報が書き込まれます。

```
nprocs = 4
rank = 0
step = 71
walker = 5
fx = 0.008186713312593607
z1 = 4.225633749839847
z2 = 5.142666117413409
```

最後に、 fx.txt には、各温度ごとの統計情報が記録されます。

```
# $1: 1/T
# $2: mean of f(x)
# $3: standard error of f(x)
```

(次のページに続く)

(前のページからの続き)

```
# $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
```

1 列目は温度・逆温度で、2・3 列目は目的関数 $f(x)$ の期待値と標準誤差、4 列目はレプリカの個数、5 列目は分配関数の比の対数 $\log(Z_n/Z_0)$ です (Z_0 は最初の温度点における分配関数)、6 列目はモンテカルロ更新の採択率です。

なお、一括計算するスクリプトとして do.sh を用意しています。do.sh では res.txt と ref.txt の差分も比較しています。以下、説明は割愛しますが、その中身を掲載します。

```
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 計算結果の可視化

result_T%.txt を図示することで、R-factor の小さいパラメータがどこにあるかを推定することができます。今回の場合は、以下のコマンドをうつことで 2 次元パラメータ空間の図 result_fx.pdf と result_T.pdf が作成されます。シンボルの色はそれぞれ R-factor と逆温度 β に対応します。

```
python3 plot_result_2d.py
```

作成された図を見ると、(5.25, 4.25) と (4.25, 5.25) 付近にサンプルが集中していることと、R-factor の値が小さいことがわかります。

また、 RockingCurve.txt が各サブフォルダに格納されています。LogXXX_YYY の XXX はモンテカルロのステップ数、 YYY は各 MPI プロセス内のレプリカの番号です。これを用いることで、前チュートリアルの手順に従い、実験値との比較も行うことが可能です。

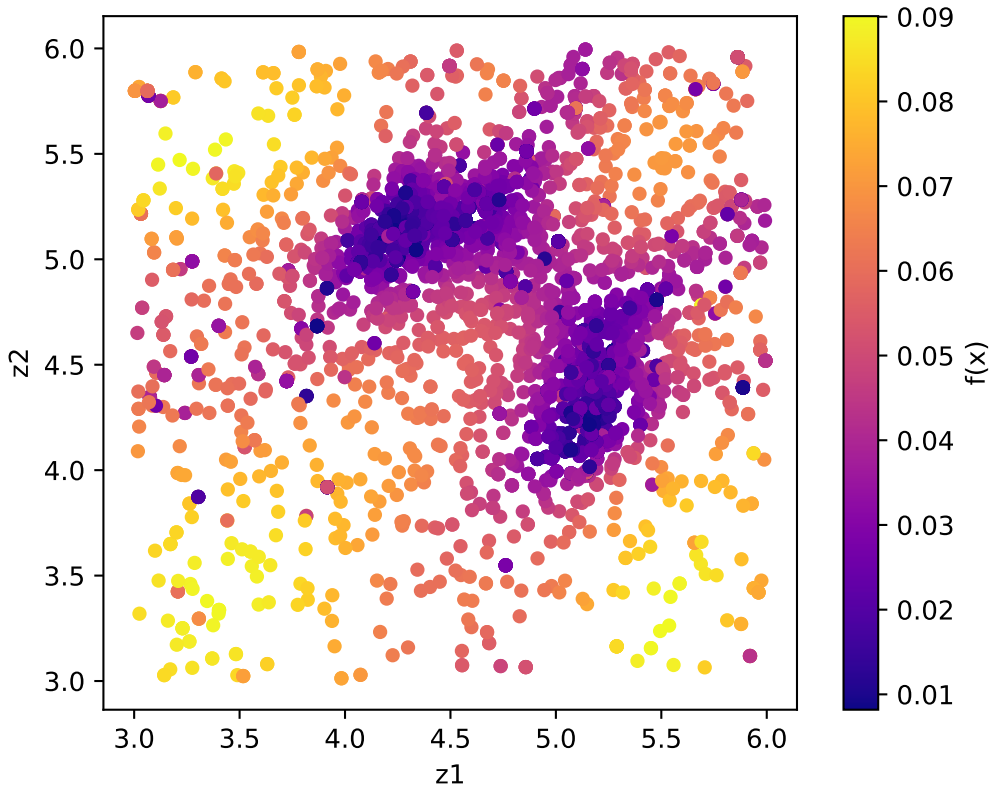


図 6: サンプルされたパラメータ。横軸は `value_01` , 縦軸は `value_02` を、色は `R-factor` を表す。

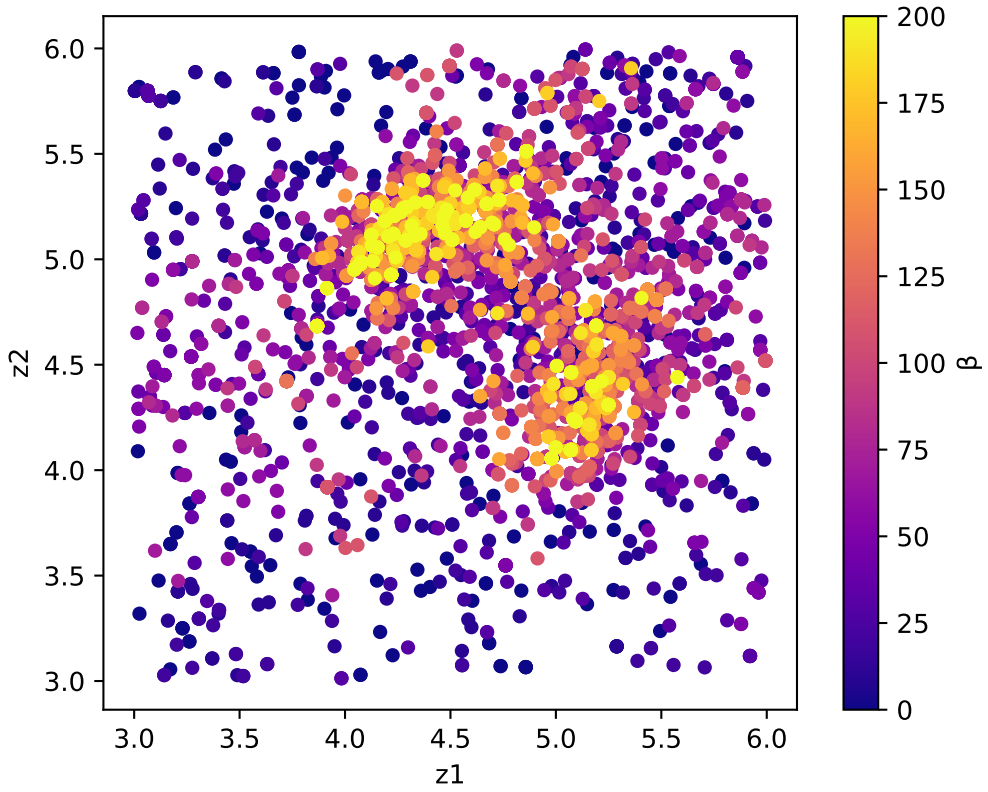


図 7: サンプルされたパラメータと逆温度。横軸は `value_01` , 縦軸は `value_02` を、色は逆温度を表す。

3.8 順問題ソルバーの追加

3.8.1 ベンチマーク関数ソルバー

py2dmat では探索アルゴリズムのテストに利用できる順問題ソルバーとして、 analytical ソルバーを準備しています。

analytical ソルバーを使うには、入力ファイルの [solver] セクションの name を "analytical" に設定します。また、 function_name パラメータを用いてベンチマーク関数 $f(x)$ を選択します。たとえば、Himmelblau 関数を用いる場合には

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

とします。利用可能な関数の詳細は *analytical* **ソルバーのリファレンス** を参照してください。

3.8.2 順問題ソルバーの追加

ユーザ定義の順問題ソルバーを定義・解析する一番簡単な方法は、 analytical ソルバーに追加することです。ここでは例として、 Booth 関数

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

を追加してみましょう（最小値は $f(1, 3) = 0$ ）。

そのためには、 py2dmat のソースコードをダウンロードし、ファイルを編集する必要があります。ダウンロード方法や、ソースコードからの実行方法などは、 **インストールページ** を参照してください。

analytical ソルバーは src/py2dmat/solver/analytical.py に定義されているので、これを編集します。

まず、 booth 関数を定義します。

```
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
```

つぎに、入力ファイルの solver.function_name パラメータで booth 関数を指定できるようにするために、Solver クラスのコンストラクタ (__init__) 中の if 分岐に以下のコードを挿入します。

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

この改造した analytical ソルバーでは、 Booth 関数の最適化が行なえます。たとえば Nelder-Mead 法による最適化は、以下の入力ファイル (input.toml) を

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

[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"
```

src/py2dmat_main.py に渡せば実行可能です:

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

... skipped ...

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

第4章　入力ファイル

py2dmat は入力ファイルの形式に TOML を採用しています。入力ファイルは次の 6 つのセクションから構成されます。

- `base`
  - `py2dmat` 全体のパラメータを指定します。
- `solver`
  - `Solver` のパラメータを指定します。
- `algorithm`
  - `Algorithm` のパラメータを指定します。
- `runner`
  - `Runner` のパラメータを指定します。
- `mapping`
  - `Algorithm` で探索しているパラメータから `Solver` で使うパラメータへの写像を定義します。
- `log`
  - solver 呼び出しの logging に関して設定します。

4.1 [base] セクション

- `dimension`

  形式: 整数型

  説明: 探索空間の次元（探索するパラメータの数）

- `root_dir`

  形式: string 型 (default: プログラム実行時のディレクトリ)

  説明: プログラムを実行する一番上のディレクトリ。
  入力ファイルなどのパスはすべて `root_dir` を起点とします。

- `output_dir`

  形式: string 型 (default: プログラム実行時のディレクトリ)

  説明: プログラムの実行結果を出力するディレクトリ名

4.2 [solver] セクション

name でソルバーの種類を決定します。各パラメータはソルバーごとに定義されています。

- name

  形式: string 型

  説明: ソルバーの名前。以下のソルバーが用意されています。

  - sim-trhepd-rheed : 反射高速 (陽) 電子回折 (RHEED, TRHEPD) の強度計算をするためのソルバー sim-trhepd-rheed
  - analytical : 解析解を与えるソルバー (主にテストに利用)

- dimension

  形式: 整数型 (default: base.dimension)

  説明: ソルバーが受け取る入力パラメータの数。

各種ソルバーの詳細および入出力ファイルは 順問題ソルバー を参照してください。

4.3 [algorithm] セクション

name でアルゴリズムの種類を決定します。各パラメータはアルゴリズムごとに定義されています。

- name

  形式: string 型

  説明: アルゴリズムの名前。以下のアルゴリズムが用意されています。

  - minsearch : Nelder-Mead 法による最小値探索
  - mapper : グリッド探索
  - exchange : レプリカ交換モンテカルロ
  - bayes : ベイズ最適化

- seed

  形式: 整数値。

  説明: 初期値のランダム生成やモンテカルロ更新などで用いる擬似乱数生成器の種を指定します。
  各 MPI プロセスに対して、 seed + mpi_rank * seed_delta の値が実際の種として用いられます。省略した場合は Numpy の規定の方法 で初期化されます。

- seed_delta

  形式: 整数値。 (default: 314159)

  説明: 疑似乱数生成器の種について、MPI プロセスごとの値を計算する際に用いられます。
  詳しくは seed を参照してください。

各種アルゴリズムの詳細および入出力ファイルは 探索アルゴリズム を参照してください。

4.4 [runner] セクション

Algorithm と Solver を橋渡しする要素である Runner の設定を記述します。サブセクションとして mapping 、 limitation 、 log を持ちます。

4.5 [mapping] セクション

Algorithm で探索している N 次元のパラメータ x から Solver で使う M 次元のパラメータ y への写像を定義します。$N \neq M$ となる場合には、 solver セクションにも dimension パラメータを指定してください。

現在はアフィン写像（線形写像+平行移動） $y = Ax + b$ が利用可能です。

- A

  形式: リストのリスト、あるいは文字列 (default: [])

  説明: $N \times M$ の変換行列 A 。空のリストを渡した場合、単位行列とみなされます。
  文字列として与える場合はそのまま行列の要素を空白および改行で区切って並べてください。

- b

  形式: リスト、あるいは文字列 (default: [])

  説明: M 次元の並進移動ベクトル b 。空のリストを渡した場合、ゼロベクトルとみなされます。
  文字列として与える場合はそのままベクトルの要素を空白区切りで並べてください。

行列の指定方法について、例えば、

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

と

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

はともに

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

を表します。

4.6 [limitation] セクション

Algorithm で探索している N 次元のパラメータ x に、制約条件を課すことが出来ます。Algorithm ごとに定義する探索範囲（例：exchange の min_list や max_list ） に加えて課すことが出来ます。現在は M 行 N 列の行列:math:*A* と M 次元の縦ベクトル:math:*b* から定義される $Ax + b > 0$ の制約式が利用可能です。具体的に

$$A_{1,1}x_1 + A_{1,2}x_2 + ... + A_{1,N}x_N + b_1 > 0$$
$$A_{2,1}x_1 + A_{2,2}x_2 + ... + A_{2,N}x_N + b_2 > 0$$
$$...$$
$$A_{M,1}x_1 + A_{M,2}x_2 + ... + A_{M,N}x_N + b_M > 0$$

という制約をかけることが出来ます。ここで M は制約式の個数（任意）となります。

- co_a

  形式: リストのリスト、あるいは文字列 (default: [])

  説明: 制約式の行列 A を設定します。

  行数は制約式数 M 列数は探索変数の数 N である必要があります。

  co_b を同時に定義する必要があります。

- co_b

  形式: リストのリスト、あるいは文字列 (default: [])

  説明: 制約式の縦ベクトル b を設定します。

  次元数が制約式数 M の縦ベクトルを設定する必要があります。

  co_a を同時に定義する必要があります。

行列の指定方法について、[mapping] セクションと同様で、例えば、

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

と

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

はともに

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

を表します。また、

```
co_b = [[0], [-1]]
```

と

```
co_b = """0 -1"""
```

と

```
co_b = """
0
-1
"""
```

はともに

$$b = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

を表します。co_a と co_b のどちらも定義しない場合、制約式を課さずに探索します。

4.7 [log] セクション

solver 呼び出しの logging に関する設定です。

- `filename`

  形式: 文字列 (default: "runner.log")

  説明: ログファイルの名前。

- `interval`

  形式: 整数 (default: 0)

  説明: solver を interval 回呼ぶ毎にログが書き出されます。0 以下の場合、ログ書き出しは行われません。

- `write_result`

  形式: 真偽値 (default: false)

  説明: solver からの出力を記録するかどうか。

- `write_input`

  形式: 真偽値 (default: false)

  説明: solver への入力を記録するかどうか。

第5章　出力ファイル

各種 Solver, Algorithm が出力するファイルについては、 順問題ソルバー および 探索アルゴリズム を参照してください。

5.1 共通ファイル

5.1.1 time.log

MPI のランク毎に計算にかかった総時間を出力します。各ランクのサブフォルダ以下に出力されます。計算の前処理にかかった時間、計算にかかった時間、計算の後処理にかかった時間について、prepare , run , post のセクションごとに記載されます。

以下、出力例です。

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

5.1.2 runner.log

MPI のランクごとに、ソルバー呼び出しに関するログ情報を出力します。各ランクのサブフォルダ以下に出力されます。入力で runner.log.interval パラメータが正の整数の時のみ出力されます。

- 1 列目がソルバー呼び出しの通し番号、
- 2 列目が前回呼び出しからの経過時間、
- 3 列目が計算開始からの経過時間

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

```

(次のページに続く)

(前のページからの続き)

```
1 0.0010826379999999691 0.0010826379999999691
2 6.96760000000185e-05 0.0011523139999999876
3 9.67080000000009e-05 0.0012490219999999885
4 0.00011765699999999324 0.0013666789999999818
5 4.965899999997969e-05 0.0014163379999999615
6 8.666900000003919e-05 0.0015030070000000006
  ...
```

第6章　探索アルゴリズム

探索アルゴリズム `Algorithm` は `Solver` の結果 $f(x)$ を用いてパラメータ空間 $\mathbf{X} \ni x$ を探索します。

6.1 Nelder-Mead 法 `minsearch`

`minsearch` は Nelder-Mead 法 (a.k.a. downhill simplex 法) によって最適化を行います。Nelder-Mead 法では、パラメータ空間の次元を D として、 $D+1$ 個の座標点の組を、各点での目的関数の値に応じて系統的に動かすことで最適解を探索します。

重要なハイパーパラメータとして、座標の初期値があります。局所最適解にトラップされるという問題があるので、初期値を変えた計算を何回か繰り返して結果を確認することをおすすめします。

2DMAT は、SciPy の `scipy.optimize.minimize(method="Nelder-Mead")` 関数を用いています。詳しくは 公式ドキュメント をご参照ください。

6.1.1 前準備

あらかじめ scipy をインストールしておく必要があります。

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

6.1.2 入力パラメータ

サブセクション `param` と `minimize` を持ちます。

[param] セクション

- `initial_list`

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータの初期値。 定義しなかった場合は一様ランダムに初期化されます。

- `unit_list`

  形式: 実数型のリスト。長さは dimension の値と一致させます。

説明: 各パラメータの単位。
探索アルゴリズム中では、各パラメータをそれぞれこれらの値で割ることで、簡易的な無次元化・正規化を行います。定義しなかった場合にはすべての次元で 1.0 となります。

- `min_list`

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最小値。
  最適化中にこの値を下回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

- `max_list`

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最大値。
  最適化中にこの値を上回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

[minimize] セクション

Nelder-Mead 法のハイパーパラメータを設定します。詳細は scipy.optimize.minimize のドキュメントを参照してください。

- `initial_scale_list`

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: Nelder-Mead 法の初期 simplex を作るために、初期値からずらす差分。`initial_list` と、`initial_list` に `initial_scale_list` の成分ひとつを足してできる dimension 個の点を 合わせたものが `initial_simplex` として使われます。定義しなかった場合、各次元に 0.25 が設定されます。

- `xatol`

  形式: 実数型 (default: 1e-4)

  説明: Nelder-Mead 法の収束判定に使うパラメータ

- `fatol`

  形式: 実数型 (default: 1e-4)

  説明: Nelder-Mead 法の収束判定に使うパラメータ

- `maxiter`

  形式: 整数 (default: 10000)

  説明: Nelder-Mead 法の反復回数の最大値

- `maxfev`

  形式: 整数 (default: 100000)

  説明: 目的関数を評価する回数の最大値

6.1.3 出力ファイル

SimplexData.txt

最小値を求める途中経過に関する情報を出力します。1 行目はヘッダー、2 行目以降に step, 入力ファイルの [solver] - [param] セクションにある、string_list で定義された変数の値、最後に関数の値が出力されます。

以下、出力例です。

```
#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

最終的に得られた目的関数の値とその時のパラメータの値を記載しています。最初に目的関数が、その後は入力ファイルの [solver] - [param] セクションにある、string_list で定義された変数の値が順に記載されます。

以下、出力例です。

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

6.2 自明並列探索 mapper

mapper_mpi はパラメータ空間中の候補点をあらかじめ用意しておいて、そのすべてで $f(x)$ を計算することで最小値を探索するアルゴリズムです。MPI 実行した場合、候補点の集合を等分割して各プロセスに自動的に割り振ることで自明並列計算を行います。

6.2.1 前準備

MPI 並列を行う場合は、 mpi4py をインストールしておく必要があります。:

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

6.2.2 入力パラメータ

[param] セクション

探索パラメータ空間を定義します。

mesh_path が定義されている場合はメッシュファイルから読み込みます。メッシュファイルは 1 行がパラメータ空間中の 1 点を意味しており、1 列目がデータ番号で、2 列目以降が各次元の座標です。

mesh_path が定義されていない場合は、 min_list, max_list, num_list を用いて、各パラメータについて等間隔なグリッドを作成します。

- mesh_path

  形式: string 型

  説明: メッシュ定義ファイルへのパス。

- min_list

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最小値。

- max_list

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最大値。

- num_list

  形式: 整数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる数。

6.2.3 アルゴリズム補助ファイル

メッシュ定義ファイル

本ファイルで探索するグリッド空間を定義します。1 + dimension 列のテキストファイルで、1 列目にメッシュのインデックス、2 列目以降は探索パラメータ x に対応する値を記載します。また、 # から始まる行はコメントとして無視されます。

以下、2 次元パラメータ空間探索のサンプルを記載します。

```
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 出力ファイル

ColorMap.txt

各メッシュでの候補パラメータと、その時の R-factor が記載されたファイルです。入力ファイルの [solver] - [param] セクションにある、string_list で定義された変数の順番でメッシュデータは記載され、最後に R-factor の値が記載されます。

以下、出力例です。

```
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 交換モンテカルロ法 exchange

exchange はレプリカ交換モンテカルロ法を用いてパラメータ探索を行う Algorithm です。

6.3.1 前準備

あらかじめ mpi4py をインストールしておく必要があります。:

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

6.3.2 入力パラメータ

サブセクション `param` と `exchange` を持ちます。

[param] セクション

探索空間を定義します。`mesh_path` キーが存在する場合は離散空間を、そうでない場合は連続空間を探索します。

- 連続空間
  - `initial_list`

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータの初期値。 定義しなかった場合は一様ランダムに初期化されます。

  - `unit_list`

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: 各パラメータの単位。
    探索アルゴリズム中では、各パラメータをそれぞれこれらの値で割ることで、簡易的な無次元化・正規化を行います。定義しなかった場合にはすべての次元で 1.0 となります。

  - `min_list`

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータが取りうる最小値。
    モンテカルロ探索中にこの値を下回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

  - `max_list`

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータが取りうる最大値。
    モンテカルロ探索中にこの値を上回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

- 離散空間

- mesh_path

  形式: ファイルパス

  説明: メッシュ定義ファイル。

- neighborlist_path

  形式: ファイルパス

  説明: 近傍リスト定義ファイル。

[exchange] セクション

- numsteps

  形式: 整数値。

  説明: モンテカルロ更新を行う回数。

- numsteps_exchange

  形式: 整数値。

  説明: 「温度」のレプリカ交換を行う頻度。この回数だけモンテカルロ更新を行ったらレプリカ交換を実行します。

- Tmin

  形式: 実数値。

  説明: 「温度」(T) の最小値。

- Tmax

  形式: 実数値。

  説明: 「温度」(T) の最大値。

- bmin

  形式: 実数値。

  説明: 「逆温度」($\beta = 1/T$) の最小値。温度と逆温度はどちらか片方だけを指定する必要があります。

- bmax

  形式: 実数値。

  説明: 「逆温度」($\beta = 1/T$) の最大値。温度と逆温度はどちらか片方だけを指定する必要があります。

- Tlogspace

  形式: 真偽値。 (default: true)

  説明: 「温度」を各レプリカに割り当てる際に、対数空間で等分割するか否かを指定します。
  true のときは対数空間で等分割します。

- nreplica_per_proc

  形式: 整数。 (default: 1)

  説明: ひとつの MPI プロセスが担当するレプリカの数。

6.3.3 アルゴリズム補助ファイル

メッシュ定義ファイル

本ファイルで探索するグリッド空間を定義します。1 列目にメッシュのインデックス（実際には使用されません）、2 列目以降は探索空間の座標を指定します。

以下、サンプルを記載します。

```
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
...
```

近傍リスト定義ファイル

離散空間をモンテカルロ法で探索する場合、各点 i ごとに次に移動できる点 j を定めておく必要があります。そのために必要なのが近傍リスト定義ファイルです。

1 列目に始点の番号 i を記載し、2 列目以降に i から移動できる終点 j を列挙します。

近傍リスト定義ファイルをメッシュ定義ファイルから生成するツール py2dmat_neighborlist が提供されています。詳細は 関連ツール を参照してください。

```
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 出力ファイル

RANK/trial.txt

各レプリカについて、モンテカルロサンプリングで提案されたパラメータと、対応する目的関数の値です。1 列目にステップ数、2 列目にプロセス内の walker 番号、3 列目にレプリカの温度、4 列目に目的関数の値、5 列目以降にパラメータが記載されます。

```
# 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

各レプリカについて、モンテカルロサンプリングで生成されたパラメータと、対応する目的関数の値です。trial.txt と同一の書式です。

```
# 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

サンプリングされた全データのうち、目的関数の値が最小となったパラメータと、対応する目的関数の値です。

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

result_T#.txt

サンプリング結果を温度ごとにまとめ直したものです。# は温度点の番号です。ファイルの 1 列目はステップ数、2 列目は全体での walker 番号、3 列目は目的関数の値、 4 列目以降は探索パラメータの値です。

```
# 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
```

アルゴリズム解説

6.3.5 マルコフ連鎖モンテカルロ法

モンテカルロ法（モンテカルロサンプリング）では、パラメータ空間中を動き回る walker $\vec{x}$ を重み $W(\vec{x})$ に従って確率的に動かすことで目的関数の最適化を行います。重み $W(\vec{x})$ として、「温度」 $T > 0$ を導入して $W(\vec{x}) = e^{-f(\vec{x})/T}$ とすることが一般的です（ボルツマン重み)。ほとんどの場合において、W に基づいて直接サンプリングする (walker を生成する) のは不可能なので、walker を確率的に少しずつ動かすことで、頻度分布が W に従うように時系列 $\{\vec{x}_t\}$ を生成します (マルコフ連鎖モンテカルロ法, MCMC)。$\vec{x}$ から $\vec{x}'$ へ遷移する確率を $p(\vec{x}'|\vec{x})$ とすると、

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

となるように p を定めれば時系列 $\{\vec{x}_t\}$ の頻度分布が $W(\vec{x})$ に収束することが示されます（釣り合い条件）[1]。実際の計算では、より強い制約である詳細釣り合い条件

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

を課すことがほとんどです。 両辺で $vecx$ についての和を取ると釣り合い条件に帰着します。

p を求めるアルゴリズムはいくつか提案されていますが、2DMAT では Metropolis-Hasting 法 (MH 法) を用います。MH 法では、遷移プロセスを提案プロセスと採択プロセスとに分割します。

1. 提案確率 $P(\vec{x}|\vec{x}_t)$ で候補点 $\vec{x}$ を生成します
   - 提案確率 P としては $\vec{x}_t$ を中心とした一様分布やガウス関数などの扱いやすいものを利用します
2. 提案された候補点 $\vec{x}$ を採択確率 $Q(\vec{x}, |\vec{x}_t)$ で受け入れ、 $\vec{x}_{t+1} = \vec{x}$ とします- 受け入れなかった場合は $\vec{x}_{t+1} = \vec{x}_t$ とします

採択確率 $Q(\vec{x}|\vec{x}_t)$ は

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

とします。この定義が詳細釣り合い条件を満たすことは、詳細釣り合いの式に代入することで簡単に確かめられます。特に、重みとしてボルツマン因子を、提案確率として対称なもの $P(\vec{x}|\vec{x}_t) = P(\vec{x}_t|\vec{x})$ を用いたときには、

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

[1] 正確には、収束のためには非周期性とエルゴード性も必要です。

という更に簡単な形になります。

$\Delta f = f(\vec{x}) - f(\vec{x}_t)$ とおいて、$\Delta f \leq 0$ のときに $Q = 1$ となることを踏まえると、MH 法による MCMC は次のようになります。

1. 現在地点の近くからランダムに次の座標の候補を選び、目的関数 f の値を調べる
2. $\Delta f \leq 0$ ならば（山を下る方向ならば）移動する
3. $\Delta f > 0$ ならば採択確率 $Q = e^{-\Delta f/T}$ で移動する
4. 1-3 を適当な回数繰り返す

得られた時系列のうち、目的関数の値が一番小さいものを最適解とします。3 番のプロセスのおかげで、$\Delta f \sim T$ ぐらいの山を乗り越えられるので、局所最適解にトラップされた場合にも脱出可能です。

6.3.6 レプリカ交換モンテカルロ法

モンテカルロ法による最適化では、温度 T は非常に重要なハイパーパラメータとなっています。モンテカルロ法では、温度 T 程度の山を乗り越えられますが、逆にそれ以上の深さの谷からは容易に脱出できません。そのため、局所解へのトラップを防ぐためには温度を上げる必要があります。一方で、T よりも小さい谷は谷として見えなくなるため、得られる $\min f(\vec{x})$ の精度も T 程度になり、精度を上げるためには温度を下げる必要があります。ここから、最適解を探すためには温度 T を注意深く決める必要があることがわかります。

この問題を解決する方法として、温度 T を固定せずに更新していくというものがあります。たとえば、焼きなまし法 (simulated annealing) では、温度をステップごとに徐々に下げていきます。焼戻し法 (simulated tempering) は、温度をハイパーパラメータではなく、サンプリングすべきパラメータとして扱い、（詳細）釣り合い条件を満たすように更新することで、加熱と冷却を実現します。温度を下げることで谷の詳細を調べ、温度を上げることで谷から脱出します。レプリカ交換モンテカルロ法 (replica exchange Monte Carlo) は焼戻し法を更に発展させた手法で、並列焼戻し法 (parallel tempering) とも呼ばれます。レプリカ交換モンテカルロ法では、レプリカと呼ばれる複数の系を、それぞれ異なる温度で並列にモンテカルロシミュレーションします。そして、ある一定間隔で、（詳細）釣り合い条件を満たすように他のレプリカと温度を交換します。焼戻し法と同様に、温度を上下することで谷を調べたり脱出したりするのですが、各温度点について、かならずレプリカのどれかが対応しているため、全体として特定の温度に偏ることがなくなります。また、複数の MPI プロセスを用意してそれぞれレプリカを担当させることで簡単に並列化可能です。数多くのレプリカを用意することで温度間隔が狭まると、温度交換の採択率も上がるため、大規模並列計算に特に向いたアルゴリズムです。有限温度由来の「ぼやけ」がどうしても生まれるので、モンテカルロ法の結果を初期値として `minsearch` をするのがおすすめです。

6.4 ポピュレーションアニーリングモンテカルロ法 pamc

pamc はポピュレーションアニーリングモンテカルロ法を用いてパラメータ探索を行う Algorithm です。

6.4.1 前準備

MPI 並列をする場合にはあらかじめ mpi4py をインストールしておく必要があります。:

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

6.4.2 入力パラメータ

サブセクション param と pamc を持ちます。

[param] セクション

探索空間を定義します。mesh_path キーが存在する場合は離散空間を、そうでない場合は連続空間を探索します。

- 連続空間
  - initial_list

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータの初期値。 定義しなかった場合は一様ランダムに初期化されます。

  - unit_list

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: 各パラメータの単位。
    探索アルゴリズム中では、各パラメータをそれぞれこれらの値で割ることで、簡易的な無次元化・正規化を行います。定義しなかった場合にはすべての次元で 1.0 となります。

  - min_list

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータが取りうる最小値。
    モンテカルロ探索中にこの値を下回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

  - max_list

    形式: 実数型のリスト。長さは dimension の値と一致させます。

    説明: パラメータが取りうる最大値。
    モンテカルロ探索中にこの値を上回るパラメータが出現した場合、ソルバーは評価されずに、値が無限大だとみなされます。

- 離散空間
  - mesh_path

    形式: ファイルパス

    説明: メッシュ定義ファイル。
  - neighborlist_path

    形式: ファイルパス

    説明: 近傍リスト定義ファイル。

[pamc] セクション

- numsteps

  形式: 整数値。

  説明: モンテカルロ更新を行う総回数。
- numsteps_annealing

  形式: 整数値。

  説明: 「温度」を下げる頻度。この回数モンテカルロ更新を行った後に温度が下がります。
- numT

  形式: 整数値。

  説明: 「温度」点の数。
- Tmin

  形式: 実数値。

  説明: 「温度」(T) の最小値。
- Tmax

  形式: 実数値。

  説明: 「温度」(T) の最大値。
- bmin

  形式: 実数値。

  説明: 「逆温度」($\beta = 1/T$) の最小値。温度と逆温度はどちらか片方だけを指定する必要があります。
- bmax

  形式: 実数値。

  説明: 「逆温度」($\beta = 1/T$) の最大値。温度と逆温度はどちらか片方だけを指定する必要があります。

- `Tlogspace`

  形式: 真偽値。 (default: true)

  説明: 「温度」を各レプリカに割り当てる際に、対数空間で等分割するか否かを指定します。
  true のときは対数空間で等分割します。

- `nreplica_per_proc`

  形式: 整数。 (default: 1)

  説明: ひとつの MPI プロセスが担当するレプリカの数。

- `resampling_interval`

  形式: 整数。 (default: 1)

  説明: レプリカのリサンプリングを行う頻度。この回数だけ温度降下を行った後にリサンプリングが行われます。

- `fix_num_replicas`

  形式: 真偽値。 (default: true)

  説明: リサンプリングの際、レプリカ数を固定するかどうか。

ステップ数について

`numsteps`, `numsteps_annealing`, `numT` の 3 つのうち、どれか 2 つを同時に指定してください。残りの 1 つは自動的に決定されます。

6.4.3 アルゴリズム補助ファイル

メッシュ定義ファイル

本ファイルで探索するグリッド空間を定義します。1 列目にメッシュのインデックス（実際には使用されません）、2 列目以降は探索空間の座標を指定します。

以下、サンプルを記載します。

```
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
...
```

近傍リスト定義ファイル

離散空間をモンテカルロ法で探索する場合、各点 i ごとに次に移動できる点 j を定めておく必要があります。そのために必要なのが近傍リスト定義ファイルです。

1 列目に始点の番号 i を記載し、2 列目以降に i から移動できる終点 j を列挙します。

近傍リスト定義ファイルをメッシュ定義ファイルから生成するツール `py2dmat_neighborlist` が提供されています。詳細は 関連ツール を参照してください。

```
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 出力ファイル

RANK/trial_T#.txt

各温度点 (#) ごと、モンテカルロサンプリングで提案されたパラメータと、対応する目的関数の値です。1 列目にステップ数、2 列目にプロセス内の walker 番号、3 列目にレプリカの逆温度、4 列目に目的関数の値、5 列目からパラメータが記載されます。最後の 2 列はそれぞれレプリカの重み (Neal-Jarzynski weight) と祖先（計算開始時のレプリカ番号）です。

```
# 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

`trial_T#.txt` をすべてまとめたものです。

RANK/result_T#.txt

各温度点、モンテカルロサンプリングで生成されたパラメータと、対応する目的関数の値です。trial.txt と同一の書式です。

```
# 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

result_T#.txt をすべてまとめたものです。

best_result.txt

サンプリングされた全データのうち、目的関数の値が最小となったパラメータと、対応する目的関数の値です。

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

fx.txt

各温度ごとに、全レプリカの情報をまとめたものです。1 列目は逆温度が、2 列目と 3 列目には目的関数の期待値およびその標準誤差が、4 列目にはレプリカの総数が、5 列目には規格化因子（分配関数）の比の対数

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

が、6 列目にはモンテカルロ更新の採択率が出力されます。ここで β_0 は計算している β の最小値です。

```
# $1: 1/T
# $2: mean of f(x)
# $3: standard error of f(x)
# $4: number of replicas
```

(次のページに続く)

(前のページからの続き)

```
# $5: log(Z/Z0)
# $6: acceptance ratio
0.0 33.36426034198166 3.0193077565358273 100 0.0 0.9804
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 アルゴリズム解説

問題と目的

分布パラメータ β_i のもとでの配位 x の重みを $f_i(x)$ と書くと（例えばボルツマン因子 $f_i(x) = \exp[-\beta_i E(x)]$）、$A$ の期待値は

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

とかけます。ここで $Z = \int \mathrm{d}x f_i(x)$ は規格化因子（分配関数）で、 $\tilde{f}(x) = f(x)/Z$ は配位 x の確率密度です。

目的は複数の分布パラメータについてこの期待値および規格化因子（の比）を数値的に求めることです。

Annealed Importance Sampling (AIS) [1]

次の同時確率分布

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

を満たす点列 $\{x_i\}$ を考えます。ここで

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

であり、 $T_i(x, x')$ は β_i のもとでの配位 x から x' への遷移確率で、釣り合い条件

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

を満たすようにとります（つまりは普通の MCMC における遷移確率行列）。

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

となるので、 $\tilde{f}_n(x_n)$ は $\tilde{f}(x_0, x_1, \ldots, x_n)$ の周辺分布

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

です。これを利用すると、 $\tilde{f}_n$ における平均値 $\langle A\rangle_n$ は拡張した配位の重み付き平均として

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

と表せます。

さて、残念ながら $\tilde{f}(x_0, x_1, \ldots, x_n)$ に従うような点列を直接生成することは困難です。そこでもっと簡単に、

1. 確率 $\tilde{f}_0(x)$ に従う x_0 を生成する
   - 例えば MCMC を利用する
2. x_i から $T_{i+1}(x_i, x_{i+1})$ によって x_{i+1} を生成する
   - T_{i+1} は釣り合い条件を満たすような遷移確率行列なので、普通に MCMC を行えば良い

という流れに従って点列 $\{x_i\}$ を生成すると、これは同時確率分布

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

に従います。これを利用すると期待値 $\langle A\rangle_n$ は

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

と評価できます (reweighting method)。$\tilde{f}$ と $\tilde{g}$ との比は、

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

とかけるので、期待値は

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

となります。規格化因子の比 Z_n/Z_0 は $\langle 1\rangle_n = 1$ を用いると

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

と評価できるので、 A の期待値は

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

という、重み付き平均の形で評価できます。この重み w_n を Neal-Jarzynski 重みと呼びます。

population annealing (PA) [2]

AIS を使うと各 β に対する期待値を重み付き平均という形で計算できますが、β の幅が大きくなると重み w の分散が大きくなってしまいます。そのため、適当な周期で確率 $p^{(k)} = w^{(k)} / \sum_k w^{(k)}$ に従いレプリカをリサンプリングし、レプリカに割当られた重みをリセット $(w = 1)$ します。

PAMC のアルゴリズムは次の擬似コードで示されます:

```
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,:])
```

リサンプリング手法として、レプリカ数を固定する方法 [2] と固定しない方法 [3] の 2 通りがあります。

参考文献

[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 ベイズ最適化 bayes

`bayes` はベイズ最適化を用いてパラメータ探索を行う `Algorithm` です。

実装には PHYSBO を用いています。

6.5.1 前準備

あらかじめ PHYSBO をインストールしておく必要があります。:

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

mpi4py がインストールされている場合、MPI 並列計算が可能です。

6.5.2 入力パラメータ

[algorithmparam] セクション

探索パラメータ空間を定義します。

mesh_path が定義されている場合はメッシュファイルから読み込みます。メッシュファイルは 1 行がパラメータ空間中の 1 点を意味しており、1 列目がデータ番号で、2 列目以降が各次元の座標です。

mesh_path が定義されていない場合は、 min_list, max_list, num_list を用いて、各パラメータについて等間隔なグリッドを作成します。

- mesh_path

  形式: string 型

  説明: メッシュデータの情報が記載された参照用ファイルへのパス。

- min_list

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最小値。

- max_list

  形式: 実数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる最大値。

- num_list

  形式: 整数型のリスト。長さは dimension の値と一致させます。

  説明: パラメータが取りうる数。

[algorithm.bayes] セクション

手法のハイパーパラメータを定義します。

- random_max_num_probes

  形式： int 型 (default: 20)

  説明: ベイズ最適化を行う前に行うランダムサンプリングの回数 (パラメータとスコアが初めに無い場合にはランダムサンプリングは必須)。

- bayes_max_num_probes

  形式： int 型 (default: 40)

  説明: ベイズ最適化を行う回数。

- score

  形式: string 型 (default: TS)

説明: スコア関数を指定するパラメータ。EI, PI, TS より選択可能で、それぞれ "expected improvement", "probability of improvement", "Thompson sampling" を行う。

- interval

  形式： int 型 (default: 5)

  説明: 指定したインターバルごとに、ハイパーパラメータを学習します。負の値を指定すると、ハイパーパラメータの学習は行われません。0 を指定すると、ハイパーパラメータの学習は最初のステップでのみ行われます。

- num_rand_basis

  形式： int 型 (default: 5000)

  説明： 基底関数の数。0 を指定した場合、Bayesian linear model を利用しない通常のガウシアンプロセスが行われます。

6.5.3 アルゴリズム補助ファイル

メッシュ定義ファイル

本ファイルで探索するグリッド空間を定義します。1 列目にメッシュのインデックス、2 列目以降は [solver.param] セクションにある、string_list で定義された変数に入る値が入ります。

以下、サンプルを記載します。

```
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 出力ファイル

BayesData.txt

最適化過程の各ステップにおいて、パラメータと対応する目的関数の値が、これまでの最適パラメータとそのステップで探索したパラメータの順に記載されます。

```
#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 アルゴリズム解説

ベイズ最適化 (Bayesian optimization, BO) は、機械学習を援用した最適化アルゴリズムであり、特に目的関数の評価に時間がかかるときに強力な手法です。

BO では目的関数 $f(\vec{x})$ を、評価が早く最適化のしやすいモデル関数（多くの場合ガウス過程） $g(\vec{x})$ で近似します。g は、あらかじめ適当に決められたいくつかの点（訓練データセット） $\{\vec{x}_i\}_{i=1}^N$ での目的関数の値 $\{f(\vec{x}_i)\}_{i=1}^N$ をよく再現するように訓練されます。パラメータ空間の各点において、訓練された $g(\vec{x})$ の値の期待値およびその誤差から求められる「スコア」 (acquition function) が最適になるような点 $\vec{x}_{N+1}$ を次の計算候補点として提案します。$f(\vec{x}_{N+1})$ を評価し、訓練データセットに追加、g を再訓練します。こうした探索を適当な回数繰り返した後、目的関数の値が最も良かったものを最適解として返します。

少ない誤差でより良い期待値を与える点は、正解である可能性こそ高いですが、すでに十分な情報があると考えられるので、モデル関数の精度向上にはあまり寄与しません。逆に、誤差の大きな点は正解ではないかもしれませんが、情報の少ない場所であり、モデル関数の更新には有益だと考えられます。前者を選ぶことを「活用」、後者を選ぶことを「探索」とよび、両者をバランス良く行うのが重要です。「スコア」の定義はこれらをどう選ぶかを定めます。

2DMAT では、ベイズ最適化のライブラリとして、PHYSBO を用います。PHYSBO は `mapper_mpi` のように、あらかじめ決めておいた候補点の集合に対して「スコア」を計算して、最適解を提案します。候補点の集合を分割することで MPI 並列実行が可能です。また、訓練データの点数 N に対して線形の計算量でモデル関数の評価、ひいては「スコア」の計算が可能となるようなカーネルを用いています。PHYSBO では「スコア」関数として "expected improvement (EI)", "probability of improvement (PI)", "Thompson sampling (TS)" が利用できます。

第7章　順問題ソルバー

順問題ソルバー `Solver` は探索パラメータ x から最適化したい $f(x)$ を計算します。

7.1 analytical ソルバー

`analytical` は探索アルゴリズムの性能評価を目的とした、定義済みのベンチマーク関数 $f(x)$ を計算する `Solver` です。

7.1.1 入力パラメータ

`solver` セクション以下の `funtion_name` パラメータで用いる関数を指定します。

- `function_name`

  形式: string 型

  説明: 関数名。以下の関数が選べます。

  - `quadratics`
    - 二次形式

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

    - 最適値は $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 0)$
  - `rosenbrock`
    - Rosenbrock 関数

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

    - 最適値は $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 1)$
  - `ackley`
    - Ackley 関数

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

    - 最適値は $f(\vec{x}^*) = 0 \quad (\forall_i x_i^* = 0)$

- himmerblau
  - Himmerblau 関数

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

  - 最適値は $f(3,2) = f(-2.805118, 3.131312) = f(-3.779310, -3.283186) = f(3.584428, -1.848126) = 0$

7.2 sim-trhepd-rheed ソルバー

sim-trhepd-rheed は sim-trhepd-rheed を用いて原子位置 x から回折 rocking curve を計算し、実験で得られた rocking curve からの誤差を $f(x)$ として返す Solver です。

7.2.1 前準備

あらかじめ sim-trhepd-rheed をインストールしておく必要があります。

1. sim-trhepd-rheed の公式サイトからソースコードをダウンロード
2. sim-trhepd-rheed/src に移動し、 make で bulk.exe と surf.exe を作成

py2dmat を実行する前にあらかじめ bulk.exe を実行してバルクデータを作成しておきます。surf.exe は py2dmat から呼び出されます。

7.2.2 入力パラメータ

solver セクションとセクション中のサブセクション config, post, param, reference を利用します。

[solver] セクション

- generate_rocking_curve

  形式: 真偽値 (default: false)

  説明: RockingCurve_calculated.txt の生成をするかどうか。 RockingCurve_calculated.txt は作業ディレクトリ Log%%%_### の中に保存されます。このオプションが"true"であっても、remove_work_dir （[post] セクション）が"true"であれば Log%%%_### は削除されてしまうため、注意してください。

[config] セクション（サブセクション）

- surface_exec_file

  形式: string 型 (default: "surf.exe")

  説明: sim-trhepd-rheed の表面反射ソルバー surf.exe へのパス

- surface_input_file

  形式: string 型 (default: "surf.txt")

  説明: 表面構造のインプットファイル。

- bulk_output_file

  形式: string 型 (default: "bulkP.b")

  説明: バルク構造のアウトプットファイル。

- surface_output_file

  形式: string 型 (default: "surf-bulkP.s")

  説明: 表面構造のアウトプットファイル。

- calculated_first_line

  形式: 整数型 (default: 5)

  説明: surface_output_file ファイル中, D(x) として読み込む最初の行。 なお、最終行は実験データとして読み込んだ行数から自動計算されます。

- calculated_info_line

  形式: 整数型 (default: 2)

  説明: surface_output_file ファイル中, glancing angle 数や beam 数が記してある行番号。

- cal_number

  形式: 整数型のリスト (設定必須)

  説明: surface_output_file ファイル中, データとして読み出す列番号。複数指定が可能。設定した値は、後述の exp_number （[reference] セクション）の値とリストの長さを一致させる必要があります。

[post] セクション（サブセクション）

実験と計算の「ズレ」の大きさを示し、最小化すべき関数である「目的関数」の定義や、ロッキングカーブを描くためのコンボリューション半値幅などを指定するセクションです。

- Rfactor_type

  形式: string 型。"A"、"A2"または"B"のいずれかをとります。 (default: "A")

説明: 目的関数値の計算方法の指定。視写角のインデックスを $i = 1, 2, \ldots, m$, ビームのインデックスを $j = 1, 2, \ldots, n$ として、実験・計算のデータをそれぞれ $u_i^{(j)}$ と $v_i^{(j)}$ で表し、各ビームの重みを $w^{(j)}$ と書くとき

- "A"タイプ:

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

- "A2"タイプ:

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

- "B"タイプ:

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

  * "B"タイプはデータ列が 1 つの実験・計算データを用いた実行 ($n = 1$) のみ対応しています。

- `normalization`

  形式: string 型。"TOTAL"または"MANY_BEAM"のいずれかをとります。 (設定必須)

  説明: 実験・計算のデータベクトルの規格化の方法。

  - "TOTAL"
    * 全体の和で規格化をします。
    * 計算に使用するデータ列が 1 つのみ適用できる規格化方法であり、`cal_number`（[config] セクション）および `exp_number`（[reference] セクション）で設定したリストの長さが 1 である必要が有ります。
  - "MANY_BEAM"
    * "MANY_BEAM"はデータ列が 2 つ以上であるときに利用できる規格化方法です。後述の `weight_type` によって規格化方法が変わります。

  なお、 normalization="MAX" は廃止となりました。

- `weight_type`

  形式: string 型または `None` 。"calc"または"manual"のいずれかを設定する必要があります。 (default: `None` 、 `normalization = "MANY_BEAM"` としたとき設定必須)

  説明: 目的関数値を計算するときの、ビームごとの重み $w^{(j)}$ の計算方法を指定します。"calc"とした場合、データ列ごとの重み $w^{(n)}$ は次の式で与えられます。

$$w^{(j)} = \left(\frac{\sum_{i=1}^m v_i^{(j)}}{\sum_{k=1}^n \sum_{i=1}^m v_i^{(j)}} \right)^2$$

  "manual"とした場合、オプション `spot_weight` を用いることで、ユーザーが重みを指定可能です。

- spot_weight

  形式: float 型のリスト。 (default: [] 、 weight_type = "manual" としたとき設定必須)

  説明: 目的関数値を計算するときの、データ列ごとの重みを設定します。総和が 1 になるように自動的に規格化されます。
  例えば、[3,2,1] を指定すると、 $w^{(1)} = 1/2, w^{(2)} = 1/3, w^{(3)} = 1/6$ となります。

- omega

  形式: 実数型 (default: 0.5)

  説明: コンボリューションの半値幅の指定。

- remove_work_dir

  形式: 真偽値 (default: false)

  説明: R-factor を読み取った後に作業ディレクトリ Log%%%_### を削除するかどうか。なお、generate_rocking_curve ([solver] セクション) が"true"であっても、本オプションが"true"ならば Log%%%_### を削除します。

[param] セクション（サブセクション）

- string_list

  形式: string 型のリスト。長さは dimension の値と一致させます。 (default: ["value_01", "value_02"])

  説明: ソルバーの入力ファイルを作成するための参照用テンプレートファイルで利用するプレースホルダーのリスト。これらの文字列が探索中のパラメータの値に置換されます。

[reference] セクション（サブセクション）

- path

  形式: string 型 (default: experiment.txt)

  説明: 実験データファイルへのパス。

- reference_first_line

  形式: 整数型

  説明: 実験データファイル中、実験データを読み出す最初の行の番号。省略時は 1, すなわち先頭行から読み出します。

- reference_last_line

  形式: 整数型 (default: 実験データファイルの最後の行の行数)

  説明: 実験データファイル中、実験データを読み出す最後の行の番号。省略時は最終行まで読み出します。

- exp_number

  形式: 整数型のリスト

  説明: 実験データファイル中、実験データとして読み出す列番号。複数指定が可能。設定した値は、前述の cal_number （[config] セクション）の値とリストの長さを一致させる必要があります。

7.2.3 ソルバー用補助ファイル

入力テンプレートファイル

入力テンプレートファイル template.txt は surf.exe の入力ファイルを作成するためのテンプレートです。動かすパラメータ（求めたい原子座標などの値）を「 value_* 」などの適当な文字列に置き換えます。使用する文字列は入力ファイルの [solver] - [param] セクションにある、string_list で指定します。以下、テンプレートの例を記載します。

```
2                                   ,NELMS,  -------- Ge(001)-c4x2
32,1.2,0.15                         ,Ge Z,da1,sap
0.6,0.6,0.6                         ,BH(I),BK(I),BZ(I)
32,1.2,0.15                         ,Ge Z,da1,sap
0.4,0.4,0.4                         ,BH(I),BK(I),BZ(I)
9,4,0,0,2,1.7,-0.5,0.5              ,NSGS,msa,msb,nsa,nsb,dthick,DXS,DYS
8                                   ,NATM
1, 1.0,  value_01,  1.00000,  5.231000   ,IELM(I),ocr(I),X(I),Y(I),Z(I
1, 1.0,  value_02,  1.00000,  4.371000
2, 1.0,  1.50000,  1.50000,  3.596000
2, 1.0,  2.00000,  1.49751,  2.100000
2, 1.0,  1.00000,  1.50000,  2.000000
2, 1.0,  0.00000,  1.00000,  0.849425
2, 1.0,  2.00000,  1.00000,  0.809425
2, 1.0,  1.00997,  1.00000,  0.599425
1,1                                 ,(WDOM,I=1,NDOM)
```

この場合、 value_01, value_02 が動かすパラメータとなります。

ターゲット参照ファイル

ターゲットにするデータが格納されたファイル experiment.txt を指定します。第一列に角度、第二列以降に反射強度にコンボリューションを計算した値が入ってます。以下、ファイルの例を示します。

```
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 ...
```

(次のページに続く)

(前のページからの続き)

```
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 出力ファイル

sim-trhepd-rheed では、 surf.exe で出力されるファイルが、ランクの番号が記載されたフォルダ下にある Log%%%%%_##### フォルダに一式出力されます。%%%%% はアルゴリズムの反復回数 step (例：モンテカルロステップ数) で、##### はアルゴリズムにおけるグループの番号 set (例：モンテカルロにおけるレプリカ番号) です。大規模計算ではこれらのフォルダの数が多くなり、時には計算機のストレージの制限に引っかかることがあります。そのような場合には、 solver.post.remove_work_dir パラメータを true にして、計算が終了した作業フォルダを削除してください。以下では、 py2dmat で独自に出力するファイルについて説明します。

stdout

surf.exe が出力する標準出力が記載されています。

以下、出力例です。

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

RockingCurve_calculated.txt

generate_rocking_curve ([solver] セクション) が"true"の場合のみ Log%%%%%_##### フォルダに出力されます。

ファイル冒頭、 # で始まる行はヘッダーです。ヘッダーには探索変数の値、目的関数値 f(x) オプションで指定した Rfactor_type, normalization`, ``weight_type, cal_number, オプションで指定またはプログラムが計算したデータ列ごとの重み spot_weight, データ部分のどの列に何が記されているか (例： ## 0 glanceing_angle など) が記されています。

が付いていない部分はデータ表記部分になります。1 列目は視写角、2 列目以降はデータ列ごとに強度が記しています。どのデータ列が記されているかはヘッダーの表記で確認できます。例えば

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

との記載があれば、1 列目は視写角、2 列目は計算データファイルの 1 列目に相当する反射強度、3 列目は計算データファイルの 2 列目に相当する反射強度、4 列目は計算データファイルの 4 列目に相当する反射強度が記されていることがわかります。

また、各列の反射強度は各列の総和が 1 になるように規格化されています。目的関数値（R-factor 及び R-factor の二乗）を算出する際は、データ列ごとの重み `spot_weight` を加味して計算されています。

以下、出力例です。

```
#value_01 =  0.00000 value_02 =  0.00000
#Rfactor_type = A
#normalization = MANY_BEAM
#weight_type = manual
#fx(x) = 0.036861804623405O5
#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 ソルバー

sxrd は sxrdcalc を用いて原子位置 x や原子の占有率、デバイワラー因子を与えることで Rocking curve を計算し、実験で得られた Rocking curve からの誤差を $f(x)$ として返す Solver です。

7.3.1 前準備

sxrdcalc は py2dmat から呼び出されます。そのため、あらかじめ sxrdcalc をインストールしておく必要があります。sxrdcalc は GitHub の以下の URL で公開されています。

https://github.com/sxrdcalc/sxrdcalc

サイトにアクセスの上、「Code」-「Download zip」よりソースコード一式をダウンロードします。zip ファイル解凍後に、Makefile を自身の計算環境に合うように編集したあとに、make コマンドを打つことで sxrdcalc の実行ファイルができます。

なお、 py2dmat を実行する前にあらかじめバルクデータを作成する必要があります (フォーマットについては、後述のソルバー用補助ファイルをご覧ください)。

7.3.2 入力パラメータ

solver セクション中のサブセクション config, post, param, reference を利用します。

[config] セクション

- sxrd_exec_file

  形式: string 型

  説明: ソルバー sxrdcalc へのパス

- bulk_struc_in_file

  形式: string 型

  説明: バルク構造のインプットファイル。

以下、入力例を記載します。

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

[param] セクション

- scale_factor

  形式: float 型 (default: 1.0)

  説明: ターゲットの Rocking Curve とシミュレーションで得られる Rocking Curve のスケールの値。

- opt_scale_factor

  形式: bool 型 (default: false)

  説明: scale_factor を最適化するかどうかのフラグ。

- type_vector

  形式: list 型

  説明: 最適化する変数の種類を正の数の list 形式で指定します。[param.atom] サブセクションで指定される type の種類と、本リストが対応します。type が同じ場合は、同じ変数として取り扱われます。

[param.domain] サブセクション

本セクションではドメインを作成します。作成したいドメイン分、定義する必要があります。[param.domain.atom] サブサブセクションでドメイン内の情報を指定します。

- domain_occupancy

  形式: float 型

  説明: ドメイン全体の占有率。

[param.domain.atom] サブセクション

本セクションはドメインに所属する最適化したい原子の個数分定義を行う必要があります。なお、変数の種類を表す type は、正の数で入力します。

- name

  形式: string 型 (重複可)

  説明: 最適化する原子の名前。

- pos_center

  形式: list 型

  説明: 原子の中心座標。$[x_0, y_0, z_0]$ の形式で記載します (x_0, y_0, z_0 は float)。

- DWfactor

  形式: float 型

  説明: デバイワラー因子 (単位は $Å^2$)。

- occupancy

  形式: float 型 (default: 1.0)

  説明: 原子の占有率。

- displace_vector (省略可)

  形式: list の list 型

  説明: 原子を動かす方向を定義するベクトル。最大 3 方向まで指定可能。
  各リストで変位ベクトルと初期値を [type, D_{i1}, D_{i2}, D_{i3}] のようにして定義する (type は int、D_{i1}, D_{i2}, D_{i3} は float)。与えられた情報に従い、$dr_i = (D_{i1}\vec{a} + D_{i2}\vec{b} + D_{i3}\vec{c}) * l_{type}$ のように l_{type} を変位させます ($\vec{a}, \vec{b}, \vec{c}$ は bulk_struc_in_file もしくは struc_in_file で指定された入力ファイルに記載された単位格子ベクトルを表します)。

- opt_DW (省略可)

  形式: list 型

  説明: デバイワラー因子を変化させる場合のスケールを設定します。
  [type, scale] のようにして定義されます。

- opt_occupancy

  形式: int 型

  説明: 定義した場合、占有率が変化する。指定された変数が type を表します。

以下、入力例を記載します。

```
[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]]
   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]
```

(次のページに続く)

(前のページからの続き)

```
  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] セクション

- f_in_file

  形式: string 型

  説明: ターゲットとするロッキングカーブのインプットファイルへのパス。

7.3.3 ソルバー用補助ファイル

ターゲット参照ファイル

ターゲットにするデータが格納されたファイル。 [reference] セクションの f_in_file でパスを指定します。1 行ごとに h k l F sigma が出力されます。ここで、 h, k, l は波数, F は強度、 sigma は F の不確かさをそれぞれ表します。以下、ファイル例を記載します。

```
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
...
```

バルク構造ファイル

バルク構造のデータが格納されたファイル。 [config] セクションの bulk_struc_in_file でパスを指定します。1 行目がコメント, 2 行目が a b c alpha beta gamma を表します。ここで、 a , b, c はユニットセルの格子定数、 alpha, beta, gamma はそれらのなす角です。3 行目以降は atomsymbol r1 r2 r3 DWfactor occupancy を指定します。ここで、 atomsymbol は原子種、 r1, r2, r3 は原子の位置座標、 DWfactor はデバイワラー因子、 occupancy は占有率をそれぞれ表します。以下、ファイル例を記載します。

```
# 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 出力ファイル

sxrd では、 計算時に出力されるファイルが、ランクの番号が記載されたフォルダ下に一式出力されます。ここでは、 py2dmat で独自に出力するファイルについて説明します。

stdout

sxrd が出力する標準出力が記載されています。sxrd の Least square fitting に対して、初期パラメータとして変数を与え、1 ショット計算 (iteration 数=0) をした際の Rfactor を計算します。Rfactor は Fit results 以下の R に記載されます。以下、出力例です。

```
----------------------------------------
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
```

(次のページに続く)

(前のページからの続き)

```
 1  20107160.3315223120 8968086799556725335654 4.0000000000 -2690426039867018271784 96.
↪0000000000
 2  -60321480.9945669472 -269042603986701827178496.0000000000␣
↪807127811960105615753216.0000000000
```

7.4 leed ソルバー

leed は M.A. Van Hove 氏により作成された SATLEED を用いて、原子位置などから Rocking curve を計算し、実験で得られた Rocking curve からの誤差を $f(x)$ として返す Solver です。 SATLEED に関する詳細については [SATLEED] をご覧ください。

7.4.1 前準備

最初に SATLEED をインストールします。http://www.icts.hkbu.edu.hk/VanHove_files/leed/leedsatl.zip へアクセスし、zip ファイルをダウンロードします。zip ファイル展開後に、所定の手続きに従いコンパイルすることで、 stal1.exe, satl2.exe などの実行ファイルができます。SATLED は計算したい系の詳細によって、ソースコードのパラメータを適宜書き換える必要があります。sample/py2dmat/leed にあるサンプルを実行する場合には、 SATLEED のダウンロードから、サンプル向けのソースコードの書き換え、コンパイルまでを自動で行うスクリプト setup.sh が用意されています。:

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

setup.sh を実行すると、 leedsatl ディレクトリに satl1.exe と satl2.exe が生成されます。

py2dmat から SATLEED を利用するにあたっては、あらかじめ satl1.exe まで実行していることが前提となります。そのため、以下のファイルが生成されている必要があります。

- satl1.exe の入力ファイル: exp.d, rfac.d, tleed4.i, tleed5.i
- satl1.exe の出力ファイル: tleed.o , short.t

py2dmat はこれらをもちいて satl2.exe を実行します。

7.4.2 入力パラメータ

solver セクション中のサブセクション config, reference を利用します。

[config] セクション

- path_to_solver

  形式: string 型

  説明: ソルバー satl2.exe へのパス

[reference] セクション

- path_to_base_dir

  形式: string 型

  説明: - exp.d, rfac.d, tleed4.i, tleed5.i , tleed.o , short.t が格納されたディレクトリへのパス。

7.4.3 ソルバー用補助ファイル

ターゲット参照ファイル

ターゲットにするデータが格納されたファイル。 [reference] セクションの path_to_base_dir 中にある tleed4.i を編集します。最適化したい数値を optxxx (xxx は 000, 001, 002, ... の形式で指定する三桁の整数) として指定します。なお、xxx の数字と py2dmat の最適化する値を入れる変数のリストの順番・個数は一致させる必要があります。なお、IFLAG, LSFLAG を 0 にしない場合は satleed 側での最適化も行われます。

以下、ファイル例を記載します。

```
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 出力ファイル

leed では、 計算時に出力されるファイルが、ランクの番号が記載されたフォルダ下に一式出力されます。

第8章　関連ツール

8.1 py2dmat_neighborlist

離散空間をモンテカルロ探索する場合に使用する近傍リスト定義ファイルをメッシュ定義ファイルから生成するツールです。

pip でインストールした場合は py2dmat と同様に bin 以下に py2dmat_neighborlist という名前でインストールされます。もしくは、ディレクトリ中の src/py2dmat_neighborlist.py を直接実行することも可能です。

8.1.1 使い方

引数としてメッシュ定義ファイルを渡します。生成される近傍リスト定義ファイルの名前は -o オプションで指定可能です。

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

もしくは

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

次のようなオプションが利用できます。

- -o output or --output output
  - 出力ファイル名 (default: neighborlist.txt)
- -u "unit1 unit2..." or --unit "unit1 unit2"
  - 各次元の長さスケール (default: すべて 1.0)
    - 空間次元の数だけ値を空白区切りで並べ、全体を引用符でくくってください
  - 各座標はあらかじめこれらの長さスケールで除算されます
- -r radius or --radius radius
  - 近傍とみなされるユークリッド距離 (default: 1.0)
  - 距離は -u で除されたあとの座標に対して計算されます
- -q or --quiet
  - 進捗バーを表示しません

  - なお、進捗バーの表示には `tqdm` python パッケージが必要です

- `--allow-selfloop`
  - 自分自身を隣接リストに含めます（自己ループ）
- `--check-allpairs`
  - すべての点対に対して距離を計算します
  - デバッグ用のオプションです

なお、 MPI を用いて計算を高速化できます。

8.2 tool/to_dft/to_dft.py

本ツールでは、Si 等四面体構造のボンドネットワークを有する系の (001) および (111) 表面系モデルについて、その原子構造から第一原理電子状態計算ソフトウェア Quantum Espresso (QE) 用の入力データを作成します。これにより、得られた構造の妥当性検証や、電子状態等の微視的な情報を取得します。なお、注目する表面と反対の表面から生じるダングリングボンド由来の電子の影響を排除するため、最下層のダングリングボンドの位置に水素原子を置く水素終端というテクニックを用いています。

8.2.1 必要な環境

- Python3 >= 3.6

以下のパッケージが必要です。

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

8.2.2 スクリプトの概要

XYZ 形式で記載された構造ファイルのファイル名および、2 次元的な周期構造を表すための格子ベクトルの情報などが記載された入力ファイルを読み込み、得られた座標データから、最下層と、その次の層にあたる原子の座標を抽出します。最下層の原子は取り除き、対応する位置に H 原子を置いて次層の原子との距離を四面体構造となる距離 (例えば、Si の場合はシラン分子となる距離) に調整したモデルを作成します。水素終端を行ったモデルは XYZ 形式で保存され、cif ファイルと Quantum Espresso (QE) の入力ファイルも作成します。なお、QE をインストールしている場合には、そのまま計算実行することもできます。

8.2.3 チュートリアル

1. 参照用の XYZ ファイルを作成します。

以下では、 tool/todft/sample/111 フォルダにある surf_bulk_new111.xyz を用います。ファイルの中身は以下の通りです。

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

2. 次に各種パラメータを設定するための入力ファイルを作成します。

入力ファイルのファイル形式は toml を採用しています。以下、 tool/todft/sample/111 フォルダにある input.toml を用いて、その内容について説明します。ファイルの中身は以下の通りです。

```
[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
```

(次のページに続く)

(前のページからの続き)

```
nbands=33           # # of bands (only used in band structure calc
[Solver.pseudo]
Si = 'Si.pbe-mt_fhi.UPF'
H = 'H.pbe-mt_fhi.UPF'
```

入力ファイルは、 Main , ASE , Solver の 3 セクションから構成されます。以下、各セクション毎に変数の説明を簡単に記載します。

Main セクション

このセクションでは、水素終端を行う際に必要なパラメータに関する設定を行います。

- input_xyz_file

  形式: string 型

  説明：入力する xyz ファイルの名前

- output_file_head

  形式: string 型

  説明：出力ファイル (xyz ファイルおよび cif ファイル) につくヘッダ

Main.Param セクション

- z_margin

  形式: float 型

  説明: 最下層および下から 2 番目の原子を抽出する際に用いられるマージン。例えば、最下層にいる原子の z 座標を z_min とした場合に、 z_min - z_margin <= z <= z_min + z_margin の中にいる原子が抽出されます。

- slab_margin

  形式: float 型

  説明: スラブの大きさに下駄をはかせるためのマージン。最下層および一番上の層にいる原子の z 座標を z_min , z_max とした場合に、スラブの大きさは z_max-z_min+slab_margin で与えられます。

- r_SiH

  形式： float 型

  説明： 四面体構造の頂点 (例えば Si) と H 間の距離を与えます (単位は Å)。

- theta

  形式： float 型

  説明： 四面体構造の頂点と H 間の角度 (例えば Si-H-Si の間の角) を与えます。

Main.lattice セクション

- unit_vec

  形式： list 型

  説明：2 次元平面を形成するユニットベクトル を指定します (ex. unit_vec = [[7.67932, 0.00000, 0.00000], [0.00000, 3.83966, 0.00000]])。

ASE セクション

このセクションでは、 ASE に関連したパラメータを設定します。

- solver_name

  形式： string 型

  説明： ソルバーの名前を与えます。現状では qe のみ。

- kpts

  形式： list 型

  説明： サンプリングする k ポイントを指定します (Monkhorst-Pack grid)。

- command

  形式： string 型

  説明： ソルバーを実行するときのコマンドを記載します。

Solver セクション

このセクションでは、 Solver に関連したパラメータを設定します。ASE の機能を用いてそのまま第一原理計算を実行する場合に指定が必要となります。基本的には各ソルバーの入力ファイルで指定したものと同じ構成で記載します。例えば、QE の場合には Solver.control に、QE の control セクションで設定するパラメータを記載します。

3. 以下のコマンドを実行します。

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

これを実行すると、

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

が生成されます。QE および擬ポテンシャルへのパス設定が行われている場合には、第一原理計算がそのまま行われます。行われていない場合には、第一原理計算が実行されないため、 `Calculation of get_potential_energy is not normally finished.` というメッセージが最後にでますが、上記ファイルの出力は行われています。

以下、出力ファイルについて説明します。

- `surf_bulk_new111_ext.xyz`

最下層原子の H への置換と四面体構造を形成するための H の追加が行われた結果が出力されます。実際の出力内容は以下の通りです。

```
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
```

このファイルは通常の XYZ 形式の座標データとして適当な可視化ソフト等に読ませることができますが、通常コメントを書く場所に周期構造の格子ベクトルの情報が書き込まれています。出力されたファイルの 3 行目以降の「元素名+3 次元座標」のデータをそのまま QE の入力ファイルにコピーして使用する事もできます。

`espresso.pwi` は QE の scf 計算用の入力ファイルで、構造最適化やバンド計算は本ファイルを適宜修正することで行うことができます。 各種設定については QE のオンラインマニュアル を参考にしてください。

第9章　(開発者向け)ユーザー定義アルゴリズム・ソルバー

本ソフトウェアは、順問題のためのソルバー Solver と最適化のためのアルゴリズム Algorithm を組み合わせることで全体の逆問題を解きます。Solver と Algorithm はすでに定義済みのものがいくつかありますが、これらを自分で定義することで、適用範囲を広げられます。本章では、 Solver や Algorithm を定義する方法およびこれらを使用する方法を解説します。

9.1 共通事項

Solver と Algorithm に共通する事柄について説明します。

9.1.1 py2dmat.Info

入力パラメータを扱うためのクラスです。インスタンス変数として次の 4 つの dict を持ちます。

- base
  - ディレクトリ情報など、プログラム全体で共通するパラメータ
- solver
  - Solver が用いる入力パラメータ
- algorithm
  - Algorithm が用いる入力パラメータ
- runner
  - Runner が用いる入力パラメータ

Info は base, solver, algorithm, runner の 4 つのキーを持つような dict を渡して初期化出来ます。

- base について
  - 要素として計算のルートディレクトリ root_dir と出力のルートディレクトリ output_dir が自動で設定されます
  - ルートディレクトリ root_dir
    - 絶対パスに変換されたものがあらためて root_dir に設定されます
    - 先頭の ~ はホームディレクトリに展開されます

        * デフォルトはカレントディレクトリ "." です
        * 具体的には次のコードが実行されます

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

    - 出力ディレクトリ output_dir
        * 先頭の ~ はホームディレクトリに展開されます
        * 絶対パスが設定されていた場合はそのまま設定されます
        * 相対パスが設定されていた場合、 root_dir を起点とした相対パスとして解釈されます
        * デフォルトは "." 、つまり root_dir と同一ディレクトリです
        * 具体的には次のコードが実行されます

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

9.1.2 py2dmat.Message

Algorithm から Runner を介して Solver に渡されるクラスです。次の 3 つのインスタンス変数を持ちます。

- x: np.ndarray
    - 探索中のパラメータ座標
- step: int
    - 何個目のパラメータであるか
    - 例えば exchange ではステップ数で、 mapper ではパラメータの通し番号。
- set: int
    - 何巡目のパラメータであるか
    - 例えば min_search では最適化（1 巡目）後にステップごとの最適値を再計算します（2 巡目）。

9.1.3 py2dmat.Runner

Algorithm と Solver とをつなげるためのクラスです。コンストラクタ引数として Solver のインスタンスと Info のインスタンス、パラメータの変換ルーチン mapping : Callable[[np.ndarray], np.ndarray] を取ります。

submit(self, message: py2dmat.Message) -> float メソッドでソルバーを実行し、結果を返します。探索パラメータを x として、 目的関数 fx = f(x) を得たい場合は以下のようにします

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

submit メソッドは mapping を用いて、探索アルゴリズムのパラメータ x から実際にソルバーが使う入力 y = mapping(x) を得ます。mapping を省略した場合 (None を渡した場合)、変換ルーチンとしてアフィン写像 $y = Ax + b$ が用いられます (py2dmat.util.mapping.Affine(A,b))。A, b の要素は info で与えられます。その他、 Runner で使われる info の詳細は 入力ファイル を参照してください。

9.2 Solver の定義

Solver クラスは py2dmat.solver.SolverBase を継承したクラスとして定義します。:

```
import py2dmat

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

このクラスは少なくとも次のメソッドを定義しなければなりません。

- __init__(self, info: py2dmat.Info)
  - 必ず基底クラスのコンストラクタを呼び出してください
    - super().__init__(info)
  - 基底クラスのコンストラクタでは次のインスタンス変数が設定されます
    - self.root_dir: pathlib.Path : ルートディレクトリ
      - info.base["root_dir"]
    - self.output_dir: pathlib.Path : 出力ファイルを書き出すディレクトリ
      - info.base["output_dir"]
    - self.proc_dir: pathlib.Path : プロセスごとの作業用ディレクトリ
      - self.output_dir / str(self.mpirank) で初期化されます
    - self.work_dir: pathlib.Path : ソルバーが実行されるディレクトリ
      - self.proc_dir で初期化されます
  - 入力パラメータである info から必要な設定を読み取り、保存してください
- prepare(self, message: py2dmat.Message) -> None
  - ソルバーが実行される前によびだされます
  - message には入力パラメータが含まれるので、ソルバーが利用できる形に変換してください
    - 例：ソルバーの入力ファイルを生成する
- run(self, nprocs: int = 1, nthreads: int = 1) -> None

  - ソルバーを実行します
  - 後ほど get_results で目的関数の値を読み取れるようにしておいてください
    * 例：出力結果をインスタンス変数に保存しておく
    * 例：実行結果をファイルに保存しておく
- get_results(self) -> float
  - ソルバーが実行されたあとに呼び出されます
  - ソルバーの実行結果を返却してください
    * 例：ソルバーの出力ファイルから実行結果を読み取る

9.3 Algorithm の定義

Algorithm クラスは py2dmat.algorithm.AlgorithmBase を継承したクラスとして定義します。

```
import py2dmat

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

9.3.1 AlgorithmBase

AlgorithmBase クラスは次のメソッドを提供します。

- __init__(self, info: py2dmat.Info, runner: py2dmat.Runner = None)
  - info から Algorithm 共通の入力パラメータを読み取り、次のインスタンス変数を設定します。
    * self.mpicomm: Optional[MPI.Comm] : MPI.COMM_WORLD
      · mpi4py の import に失敗した場合、 None が設定されます
    * self.mpisize: int : MPI プロセス数
      · mpi4py の import に失敗した場合、 1 が設定されます
    * self.mpirank: int : MPI ランク
      · mpi4py の import に失敗した場合、 0 が設定されます
    * self.rng: np.random.Generator : 擬似乱数生成器
      · 擬似乱数の種について、詳細は **入力パラメータの *[algorithm]* セクション** を参照してください
    * self.dimension: int : 探索パラメータ空間の次元
    * self.label_list: List[str] : 各パラメータの名前

        * self.root_dir: pathlib.Path : ルートディレクトリ
            · info.base["root_dir"]
        * self.output_dir: pathlib.Path : 出力ファイルを書き出すディレクトリ
            · info.base["root_dir"]
        * self.proc_dir: pathlib.Path : プロセスごとの作業用ディレクトリ
            · self.output_dir / str(self.mpirank)
            · ディレクトリが存在しない場合、自動的に作成されます
            · 各プロセスで最適化アルゴリズムはこのディレクトリで実行されます
        * self.timer: Dict[str, Dict] : 実行時間を保存するための辞書
            · 空の辞書が 3 つ、 "prepare", "run", "post" という名前で定義されます

- prepare(self) -> None
  - 最適化アルゴリズムの前処理をします
  - self.run() の前に実行する必要があります
- run(self) -> None
  - 最適化アルゴリズムを実行します
  - self.proc_dir に移動し、 self._run() を実行した後、元のディレクトリに戻ります
- post(self) -> None
  - 最適化結果のファイル出力など、後処理を行います
  - self.output_dir に移動し、 self._post() を実行した後、元のディレクトリに戻ります
  - self.run() のあとに実行する必要があります
- main(self) -> None
  - prepare, run, post を順番に実行します
  - それぞれの関数でかかった時間を計測し、結果をファイル出力します
- _read_param(self, info: py2dmat.Info) -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]
  - 連続なパラメータ空間を定義するためのヘルパーメソッドです
  - info.algorithm["param"] から探索パラメータの初期値や最小値、最大値、単位を取得します
  - 詳細は *min_search* **の入力ファイル** を参照してください
- _mesh_grid(self, info: py2dmat.Info, split: bool = False) -> Tuple[np.ndarray, np.ndarray]
  - 離散的なパラメータ空間を定義するためのヘルパーメソッドです
  - info.algorithm["param"] を読み取り次を返します:

        * D 次元の候補点 N 個からなる集合 (NxD 次元の行列として)
        * N 個の候補点の ID(index)
    - split が True の場合、候補点集合は分割され各 MPI ランクに配られます
    - 詳細は *mapper* の入力ファイル を参照してください

9.3.2 Algorithm

Algorithm は少なくとも次のメソッドを定義しなければなりません。

- __init__(self, info: py2dmat.Info, runner: py2dmat.Runner = None)
    - 引数はそのまま基底クラスのコンストラクタに転送してください
        * super().__init__(info=info, runner=runner)
    - 入力パラメータである info から必要な設定を読み取り、保存してください
- _prepare(self) -> None
    - 最適化アルゴリズムの前処理を記述します
- _run(self) -> None
    - 最適化アルゴリズムを記述します
    - 探索パラメータ x から対応する目的関数の値 f(x) を得る方法

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

- _post(self) -> None
    - 最適化アルゴリズムの後処理を記述します

9.4 実行方法

次のようなフローで最適化問題を実行できます。プログラム例にあるコメントの番号はフローの番号に対応しています。

1. ユーザ定義クラスを作成する
    - もちろん、 py2dmat で定義済みのクラスも利用可能です
2. 入力パラメータ info: py2dmat.Info を作成する
    - プログラム例では入力ファイルとして TOML を利用していますが、辞書をつくれれば何でも構いません
3. solver: Solver, runner: py2dmat.Runner, algorithm: Algorithm を作成する
4. algorithm.main() を実行する

プログラム例

```
import sys
import tomli
import py2dmat

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

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

# (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()
```

第10章　謝辞

本ソフトウェアの開発は、科研費 (2019-2021 年度)「超並列マシンを用いた計算統計と測定技術の融合」および東京大学物性研究所 ソフトウェア高度化プロジェクト (2020, 2021 年度) の支援を受け開発されました。本ソフトウェアの設計・変数命名の一部は abics を参考にしています。また、順問題ソルバーの実装にあたり、花田貴氏 (東北大学)、望月出海氏 (KEK)、W. Voegeli 氏 (東京学芸大学)、白澤徹郎氏 (AIST)、R. Ahmed 氏 (九州大学)、和田健氏 (KEK) にお世話になりました。ソルバーの追加方法について、塚本恭平氏 (物性研) にチュートリアルを追加していただきました。この場を借りて感謝します。

第11章　お問い合わせ

2DMAT に関するお問い合わせはこちらにお寄せください。

- バグ報告

  2DMAT のバグ関連の報告は GitHub の Issues で受け付けています。

  バグを早期に解決するため、報告時には次のガイドラインに従ってください。

  - 使用している 2DMAT のバージョンを指定してください。
  - インストールに問題がある場合には、使用しているオペレーティングシステムとコンパイラの情報についてお知らせください。
  - 実行に問題が生じた場合は、実行に使用した入力ファイルとその出力を記載してください。

- その他

  研究に関連するトピックなど GitHub の Issues で相談しづらいことを問い合わせる際には、以下の連絡先にコンタクトをしてください。

  E-mail: `2dmat-dev__at__issp.u-tokyo.ac.jp` (_at_を@に変更してください)

関連図書

[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