
2DMAT Documentation

リリース *1.0.0*

University of Tokyo

2021年03月13日

Contents:

第 1 章	はじめに	1
1.1	2DMAT とは	1
1.2	ライセンス	2
1.3	バージョン履歴	2
1.4	主な開発者	2
第 2 章	py2dmat のインストール	3
2.1	実行環境・必要なパッケージ	3
2.2	ダウンロード・インストール	3
2.3	実行方法	4
2.4	アンインストール	4
第 3 章	チュートリアル	5
3.1	TRHEPD 順問題ソルバー	6
3.2	Nelder-Mead 法による最適化	9
3.3	グリッド型探索	14
3.4	ベイズ最適化	20
3.5	レプリカ交換モンテカルロ法による探索	26
第 4 章	入力ファイル	33
4.1	[base] セクション	33
4.2	[solver] セクション	34
4.3	[algorithm] セクション	34
4.4	[runner] セクション	35
第 5 章	出力ファイル	37
5.1	共通ファイル	37
第 6 章	探索アルゴリズム	39
6.1	Nelder-Mead 法 minsearch	39
6.2	自明並列探索 mapper	42
6.3	交換モンテカルロ法 exchange	44
6.4	ベイズ最適化 bayes	48

第 7 章	順問題ソルバー	53
7.1	analytical ソルバー	53
7.2	sim-trhepd-rheed ソルバー	54
第 8 章	関連ツール	59
8.1	to_dft.py	59
第 9 章	(開発者向け) ユーザー定義アルゴリズム・ソルバー	65
9.1	共通事項	65
9.2	Solver の定義	67
9.3	Algorithm の定義	69
9.4	実行方法	71
第 10 章	謝辞	73
第 11 章	お問い合わせ	75

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] に対応しており、探索アルゴリズムは Nelder-Mead 法 [3], グリッド型探索法 [4], ベイズ最適化 [5], レプリカ交換モンテカルロ法 [6] 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] 順問題ルーチンは, 東北大学 花田貴によって開発されたコードに基づき作成しています. T. Hanada, H. Daimon, and S. Ino, *Phys. Rev. B* 51, 13320 (1995).

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

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

[5] ベイズ最適化には, PHYSBO を用いています.

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

1.2 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。

Copyright (c) <2020-> The University of Tokyo. All rights reserved.

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。

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

1.3 バージョン履歴

- 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。

- 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 以上
 - 必要な python パッケージ
 - * toml
 - * numpy
 - 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)

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 [solver/input](#) 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。

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

の 4 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の実行方法、その後に `minsearch`, `mapper_mpi`, `bayes`, `exchange` の実行方法について順に説明します。

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` 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 計算を試みます。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 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 `SURFYYYMMDD-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)からロック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。

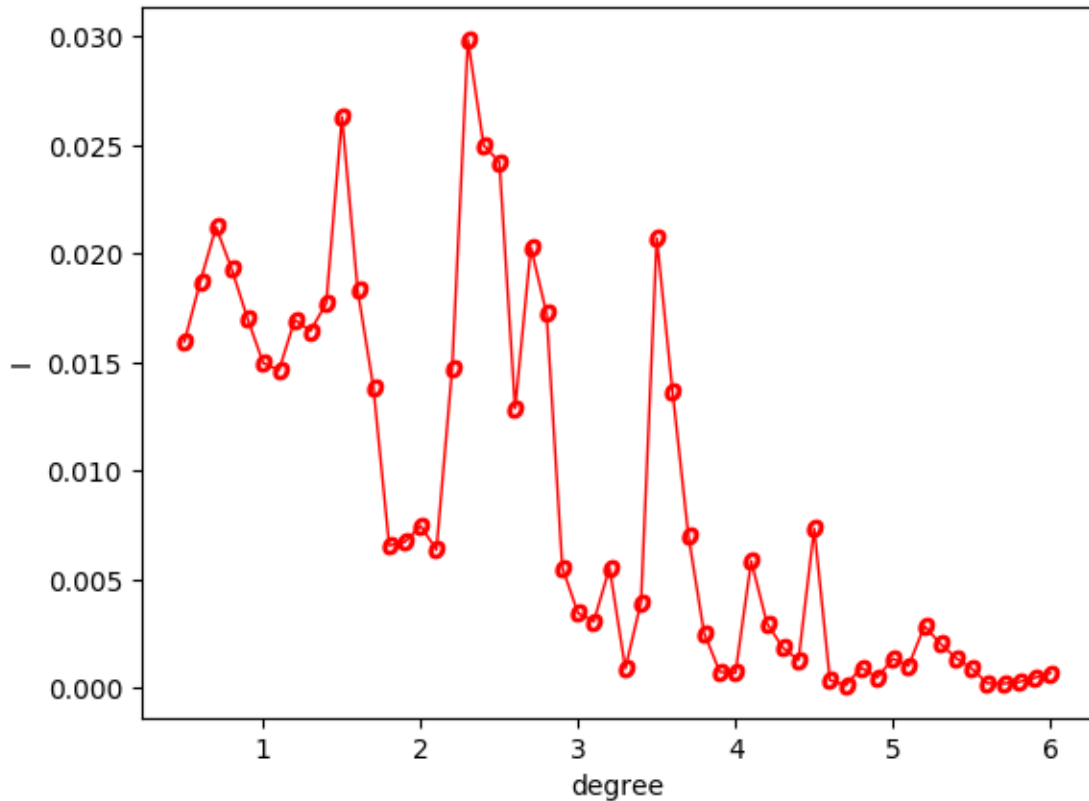


図1 Si(001)-2x1面のロックングカーブ。

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を実行して下さい。

```
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)
```

(次のページに続く)

(前のページからの続き)

```
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/py2dmat/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/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,dal,sap
0.6,0.6,0.6 ,BH(I),BK(I),BZ(I)
32,1.0,0.1 ,Ge Z,dal,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

[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/py2dmat/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 `Logxxxxxx` (`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
    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、Logxxxxx (xxxx はステップ数) に RockingCurve.txt として保存されています。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/RockingCurve.txt RockingCurve_ini.txt
cp 0/Log00000017/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_minsearch.png が出力されます。

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。

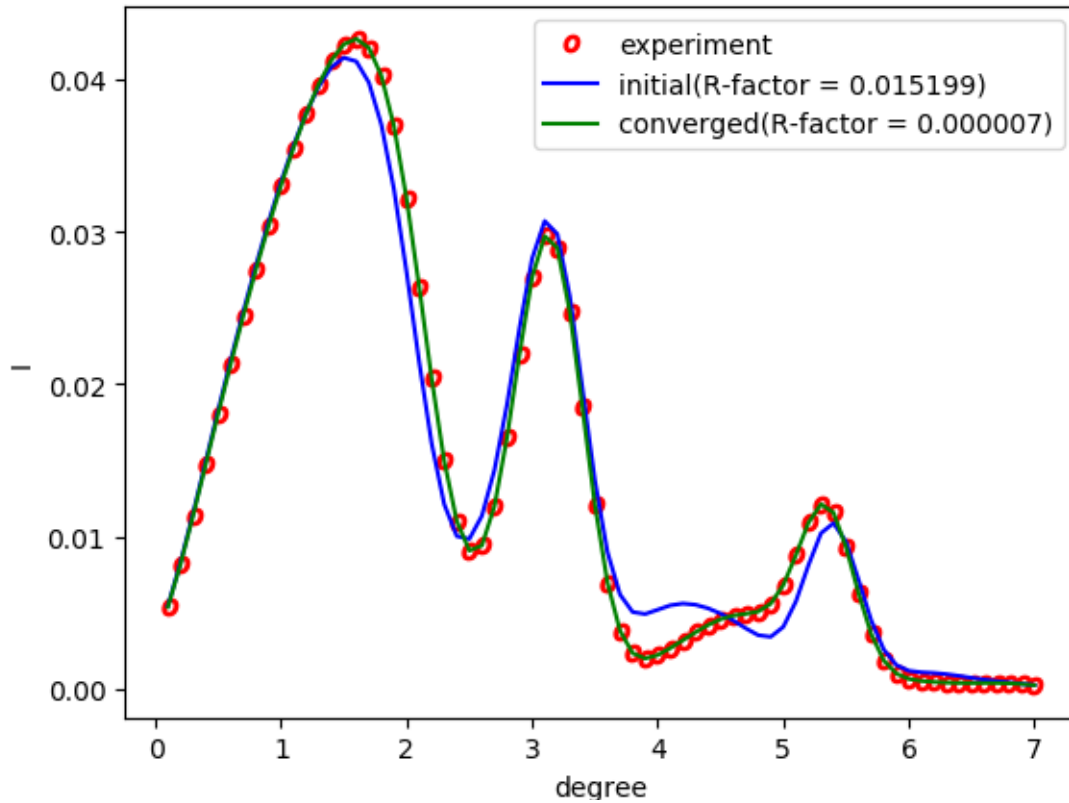


図2 Nelder-Mead 法を用いた解析。赤丸が実験値、青線が最初のステップ、緑線が最後のステップで得られたロッキングカーブを表す。

3.3.1 サンプルファイルの場所

サンプルファイルは `sample/py2dmat/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
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
```

(次のページに続く)

(前のページからの続き)

```
[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/py2dmat/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
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。

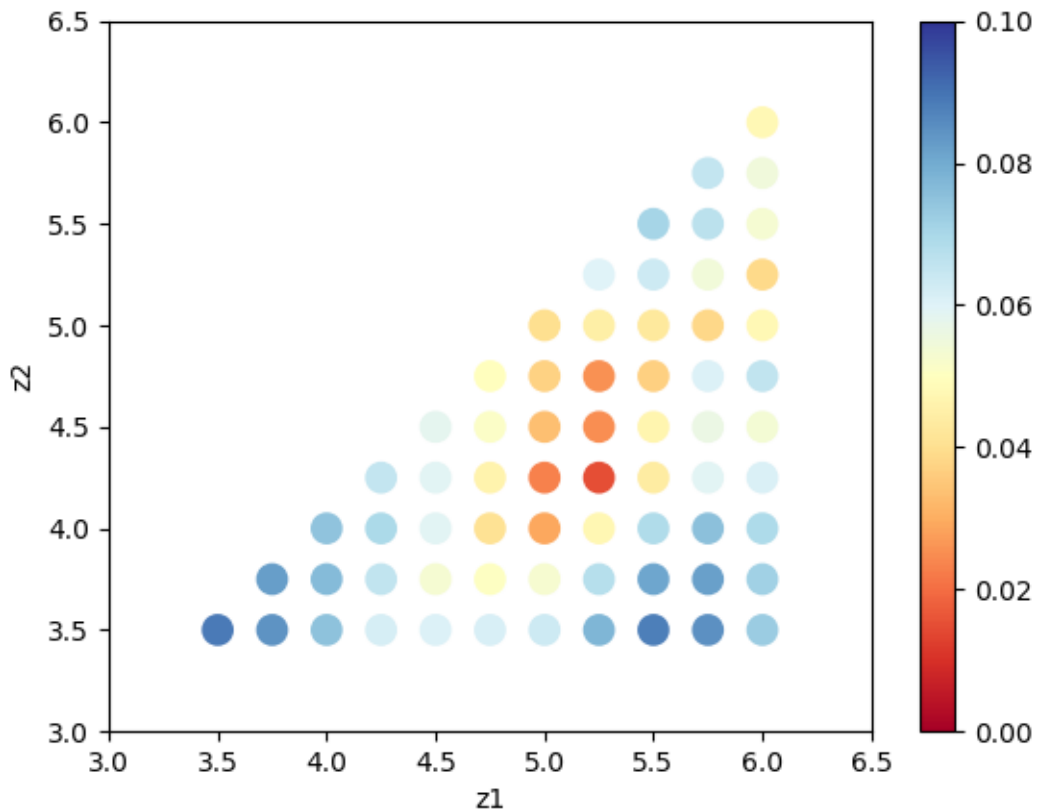


図3 2次元パラメータ空間上での R-factor。

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/py2dmat/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)。

- `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 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
...
```

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
```

(次のページに続く)

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

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

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

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

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

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

[algorithm.bayes]
random_max_num_probes = 5
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/py2dmat/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 = 5
bayes_max_num_probes = 20
score = TS
interval = 5
num_rand_basis = 5000
Read MeshData.txt
value_01 = 4.75000
value_02 = 4.50000
WARNING : degree in lastline = 7.0, but 6.0 expected
PASS : len(calculated_list) 70 == len(convolution_I_calculated_list)70
R-factor = 0.05141906746102885
0001-th step: f(x) = -0.051419 (action=46)
    current best f(x) = -0.051419 (best action=46)

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

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 R-factor z1_action z2_action R-factor_action
0 4.75 4.5 0.05141906746102885 4.75 4.5 0.05141906746102885
1 4.75 4.5 0.05141906746102885 6.0 4.75 0.06591878368102033
2 5.5 4.25 0.04380131351780189 5.5 4.25 0.04380131351780189
3 5.0 4.25 0.02312528177606794 5.0 4.25 0.02312528177606794
4 5.0 4.25 0.02312528177606794 6.0 5.75 0.05501069117756031
5 5.0 4.25 0.02312528177606794 5.0 4.75 0.037158316568603085
6 5.0 4.25 0.02312528177606794 5.75 4.75 0.06061194437867895
7 5.0 4.25 0.02312528177606794 4.25 3.5 0.062098618649988294
```

(次のページに続く)

(前のページからの続き)

```

8 5.0 4.25 0.02312528177606794 6.0 6.0 0.04785241875354398
9 5.0 4.25 0.02312528177606794 4.5 4.0 0.05912332368374844
10 5.0 4.25 0.02312528177606794 4.75 4.25 0.04646333628698967
11 5.0 4.25 0.02312528177606794 5.5 4.5 0.0466682914488051
12 5.0 4.25 0.02312528177606794 5.0 4.5 0.033464998538380517
13 5.25 4.25 0.015199251773721183 5.25 4.25 0.015199251773721183
14 5.25 4.25 0.015199251773721183 5.25 4.0 0.0475246576904707
...

```

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 13 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/py2dmat/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 (Nelder-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]
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/py2dmat/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、最初の列がステップ数、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 T fx x1 x2
0 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
1 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
2 0.004999999999999999 0.07830821484593968 3.682008067401509 3.9502750191292586
3 0.004999999999999999 0.06273922648753057 4.330900869594549 4.311333132184154
```

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を書き込まれます。

```
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 列目がランク、3 列目が目的関数の値、4 列目以降がパラメータです。

```
# T = 0.0049999999999999999
# 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。

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。

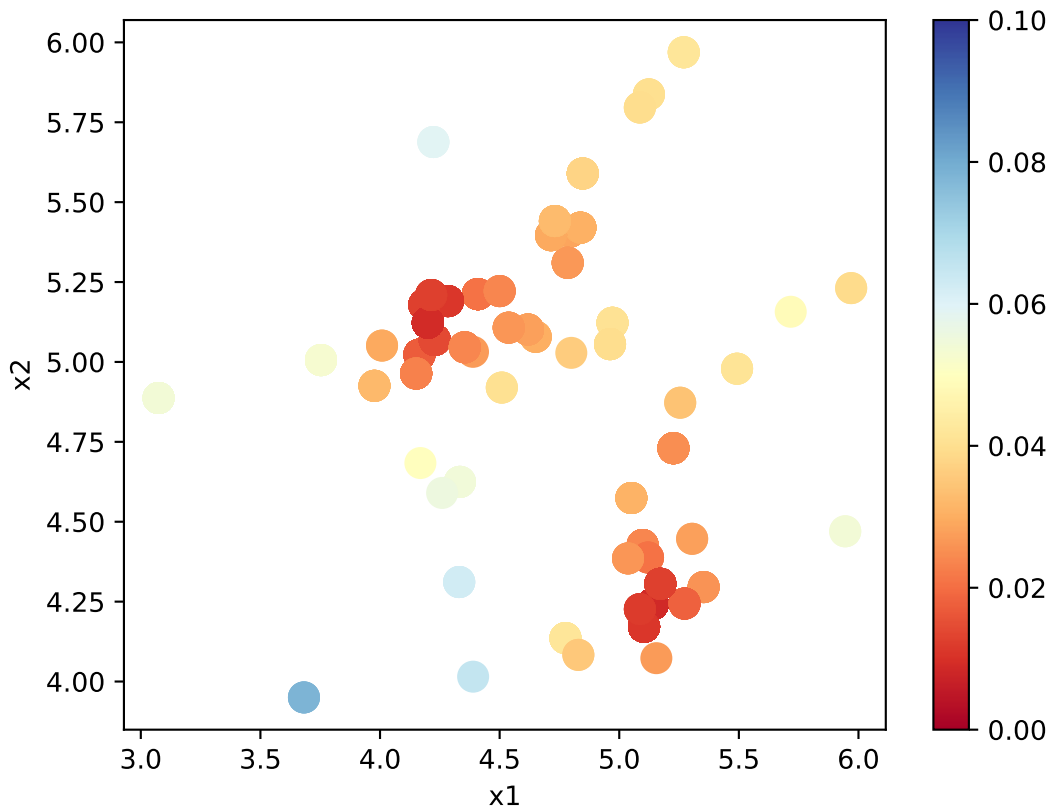


図4 サンプルされたパラメータと R-factor。横軸は value_01 , 縦軸は value_02 を表す。

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 4 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。

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)
- 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)

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: 擬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 log を持ちます。

4.4.1 [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

1 0.0010826379999999691 0.0010826379999999691
2 6.96760000000185e-05 0.0011523139999999876
3 9.670800000000009e-05 0.0012490219999999885
4 0.0001176569999999324 0.0013666789999999818
5 4.96589999997969e-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。

- `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。

[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 3.545138888888888 0.012635279378225261
4 5.179976851851851 4.348958333333334 3.594328703703703 0.006001660077530159
5 5.179976851851851 4.348958333333334 3.594328703703703 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 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
...
```

6.2.4 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

- `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。

- `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。

[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。 (default: 0.1)

説明: 「温度」の最小値。

- Tmax

形式: 実数値。 (default: 10.0)

説明: 「温度」の最大値。

- 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。

6.3.3 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 列目にレプリカの温度、3 列目に目的変数の値、4 列目以降にパラメータが記載されます。

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

RANK/result.txt

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

best_result.txt

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
```

アルゴリズム解説

6.3.4 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 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。

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、

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

という更に簡単な形になります。

$\Delta f = f(\vec{x}) - f(\vec{x}_t)$ において、 $\Delta f \leq 0$ 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。

6.3.5 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、一つの MPI プロセスに一つの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 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。

6.4.1 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。

6.4.2 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。

6.4.3 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、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
...
```

6.4.4 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
...
```

6.4.5 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」 (acquisition 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 `function_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 $f(\vec{x}^*) = 0$ ($\forall_i x_i^* = 0$)

- `rosenbrock`

- * `Rosenbrock` 関数

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

* 最適値は $f(\vec{x}^*) = 0$ ($\forall_i x_i^* = 1$)

- ackley

* Ackley 関数

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

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

- 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

- `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: ソルバーにより計算された出力ファイルを読み込む範囲を指定するパラメータ。読み込む最初の行を指定。
- `calculated_last_line`
形式: 整数型 (default: 60)
説明: ソルバーにより計算された出力ファイルを読み込む範囲を指定するパラメータ。読み込む最後の行を指定。
- `row_number`
形式: 整数型 (default: 8)
説明: ソルバーにより計算された出力ファイルを読み込む範囲を指定するパラメータ。読み込む列を指定。

[post] セクション

- `normalization`

形式: `string` 型。"TOTAL"または"MAX"のいずれかをとります。(default: "TOTAL")

説明: R 値の規格化の方法。全体の和で規格化するか最大値で規格化するかを指定します。

- `Rfactor_type`

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

説明: R ファクターの計算方法の指定。"A"だと通常の方法、"B"だと Pendry の R ファクターを計算します。

- `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。

[param] 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。

- `degree_max`

形式: 実数型 (default: 6.0)

説明: 最大角度 (度単位) の指定

[reference] 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。

- `first`

形式: 整数型 (default: 1)

説明: 実験データファイルを読み込む範囲を指定するパラメータ。実験ファイルを読み込む最初の行を指定。

- `last`

形式: 整数型 (default: 56)

説明: 実験データファイルを読み込む範囲を指定するパラメータ。実験ファイルを読み込む最後の行を指定。

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
32,1.0,0.1                       ,Ge Z,dal,sap
0.6,0.6,0.6                       ,BH(I),BK(I),BZ(I)
32,1.0,0.1                       ,Ge Z,dal,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)

```

この場合、`value_01`, `value_02`, `value_03` 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、第二列に反射強度に重みをかけて計算した値が入ってます。以下、ファイルの例を指定します。

```

0.100000 0.002374995
0.200000 0.003614789
0.300000 0.005023215
0.400000 0.006504978
0.500000 0.007990674
0.600000 0.009441623
0.700000 0.010839445
0.800000 0.012174578

```

(次のページに続く)

(前のページからの続き)

```
0.900000 0.013439485
1.000000 0.014625579
...
```

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。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

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.txt

Log%%%%%%%% フォルダに出力されます。1行目にヘッダ、2行目以降は角度、コンボリユーションされた計算値・実験値、規格化された計算値・実験値と、生の計算値が順に出力されます。以下、出力例です。

```
#degree convolution_I_calculated I_experiment convolution_I_calculated(normalized) I_
↪experiment(normalized) I_calculated
0.1 0.0023816127859192407 0.002374995 0.004354402952499057 0.005364578226620574 0.
↪001722
0.2 0.003626530149456865 0.003614789 0.006630537795012198 0.008164993342397588 0.003397
0.3 0.00504226607469267 0.005023215 0.009218987407498791 0.011346310125551366 0.005026
0.4 0.006533558304296079 0.006504978 0.011945579793136154 0.01469327865677437 0.006607
0.5 0.00803056955158873 0.007990674 0.014682628499657693 0.018049130948243314 0.008139
0.6 0.009493271317558538 0.009441623 0.017356947736613827 0.021326497600946535 0.00962
0.7 0.010899633015118851 0.010839445 0.019928258053867838 0.024483862338931763 0.01105
...
```

第 8 章

関連ツール

8.1 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。

8.1.1 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

8.1.2 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。

8.1.3 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', ...
```

(次のページに続く)

(前のページからの続き)

```

pseudo_dir='./'      # Pseudopotential directory
[Solver.system]
ecutwfc = 20.0      # Cut-off energy in Ry
nbands=33           # # of bands (only used in band structure calc)
[Solver.pseudo]
Si = 'Si.pbe-mt_fhi.UPF'
H = 'H.pbe-mt_fhi.UPF'

```

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、 $z_{\min} - z_{\text{margin}} \leq z \leq z_{\min} + z_{\text{margin}}$ の中にあ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 $z_{\max} - z_{\min} + \text{slab_margin}$ 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一構成で記載します。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、適用範囲を広げる事が可能です。本章では、Solver や Algorithm の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。

なお、v1.0 の公式リリースまでに変わる可能性があることに注意してください。

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 を渡して初期化されます。

- 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)
```

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
- `default_run_scheme(self) -> str`
 - ソルバーのデフォルトの実行方法 (`run_scheme`) を返してください。現在の選択肢は次のとおりです
 - * `subprocess`: `subprocess.run` を用いてサブプロセス実行する
 - * `function`: `python` の関数として実行する
 - 将来的に、ひとつのソルバーが実行方法をサポートできるようになる予定です
- `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
- `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

また、次のメソッドのうち、どちらか必要な方を定義してください。

- `command(self) -> List[str]`
 - ソルバーを実行するためのコマンド
 - `run_scheme == "subprocess"` の場合に必要で、そのまま `subprocess.run` に渡されます
- `function(self) -> Callable[[], None]`
 - ソルバーを実行するための `python` 関数
 - `run_scheme == "function"` の場合に必要で、そのまま実行されます

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.mpi_size: 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.mpi_rank: 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.mpi_rank)`
 - ・ 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 (Nx D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 toml
import py2dmat

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

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

file_name = sys.argv[1]

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

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

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

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 年度)の支援を受け開発されました。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)