

2DMAT

Web: <https://www.pasums.issp.u-tokyo.ac.jp/2DMAT>

Repository: <https://github.com/issp-center-dev/2DMAT>

Preprint: [arXiv:2204.04484](https://arxiv.org/abs/2204.04484)

質問・要望は
GitHub のissue か
メール (2dmat-dev@issp.u-tokyo.ac.jp)へ！

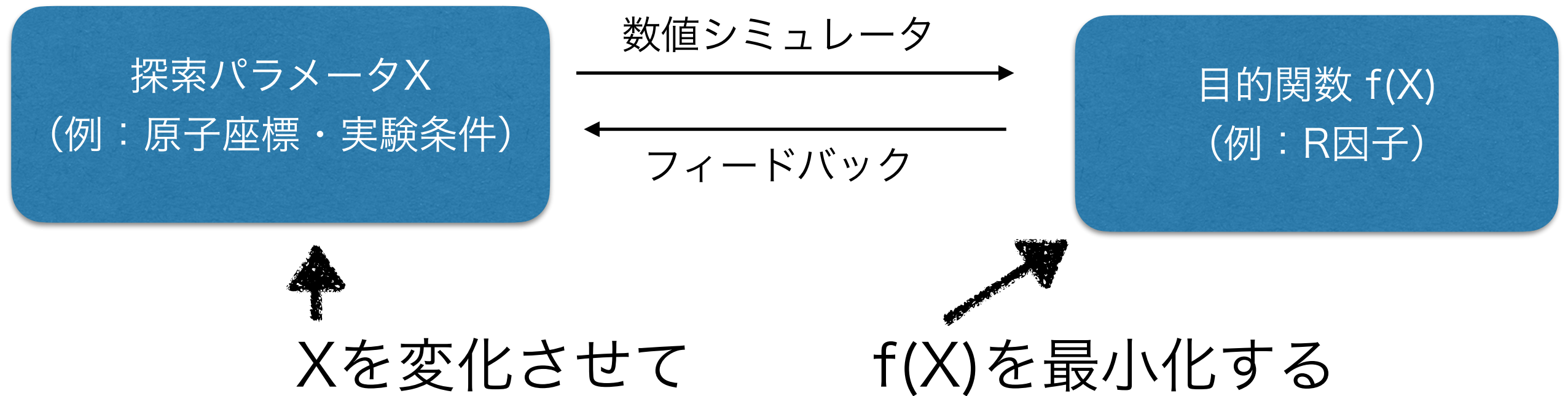
Yuichi Motoyama (ISSP)

2022-04-26
for 2DMAT ver 2.1.0

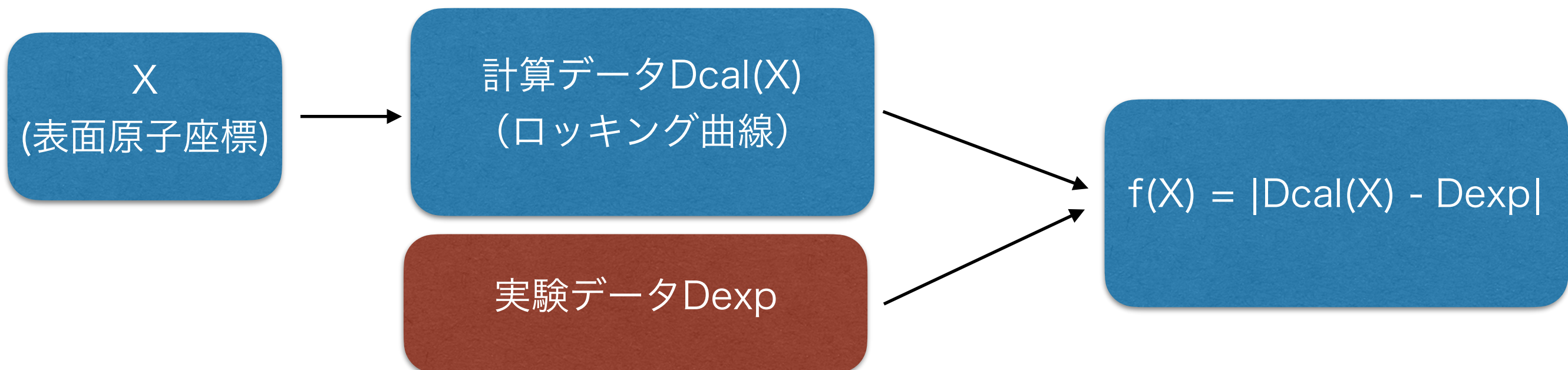
2DMAT

- メインミッション
 - 測定データの解析に対する、汎用・高速・高信頼なツールを目指す
 - 特に、スパコンなどの大規模並列計算機での利用を見据えたツール
 - もちろんクラスター計算機や個人のパソコンでも使える
- 現在のメインターゲット
 - ビーム回折実験による表面の原子構造解析
 - 実験で得られた曲線を再現するような原子構造を探索する（逆問題）
 - 逆問題は参照データ（実験データ）と順問題データ（計算データ）との距離（目的関数）を最小化するような最適化問題として定式化可能
- 最適化問題を解くためのフレームワークとして `py2dmat` を開発している
 - 表面構造探索以外にも利用可能

逆問題



数値シミュレータの例: TRHEPD シミュレータ (sim-trhepd-rheed)



py2dmat の構成

- py2dmat は大雑把に次の2つの主コンポーネントからなる
 - 目的関数 $f(x)$ を計算する Solver
 - TRHEPD による表面構造推定の場合
 - x が表面の原子座標
 - $f(x)$ は実験で得られたrocking curve と計算で得られる rocking curve との距離
 - $f(x)$ をもとに x の空間を探索する Algorithm
 - 最適化（ベイズ最適化など）
 - サンプルング（モンテカルロ法など）
- Solver を切り替えることで問題（実験手法）が変わっても同じように解析可能
- Algorithm を切り替えることで様々な数理手法で実験データを解析可能

py2dmat のインストール

- py2dmat は Python3 で書かれている
- 一番簡単なインストール方法は pip を用いて PyPI からインストールすること
 - `python3 -m pip install py2dmat`
 - `py2dmat module` と `py2dmat コマンド` がインストールされる
 - 後者への実行パスが通っているかは状況による
- チュートリアル用のサンプルファイルやスクリプトが必要ならば GitHub からダウンロードする
 - `git clone https://github.com/issp-center-dev/2DMAT`
 - `sample` ディレクトリと `script` ディレクトリに色々入っている
 - `src/py2dmat_main.py` を python スクリプトとして実行すると `py2dmat コマンド` と同じことができる

py2dmat の入力ファイル

- py2dmat コマンドはひとつの入力ファイル (input.toml) を引数にする
- 入力ファイルは TOML 形式
 - <https://toml.io/ja/>
 - 大雑把には、
 - [section] によるセクションと
 - key = value による定義
- 右の入力ファイルは
 - Rosenblock 関数を ([solver])
 - Nelder-Mead 法で最適化する ([algorithm])

実行方法

```
$ py2dmat input.toml
```

```
$ cat input.toml
```

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

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

```
[algorithm.param]
max_list = [5.0, 5.0]
min_list = [-5.0, -5.0]
```

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

Algorithm

～探索手法～

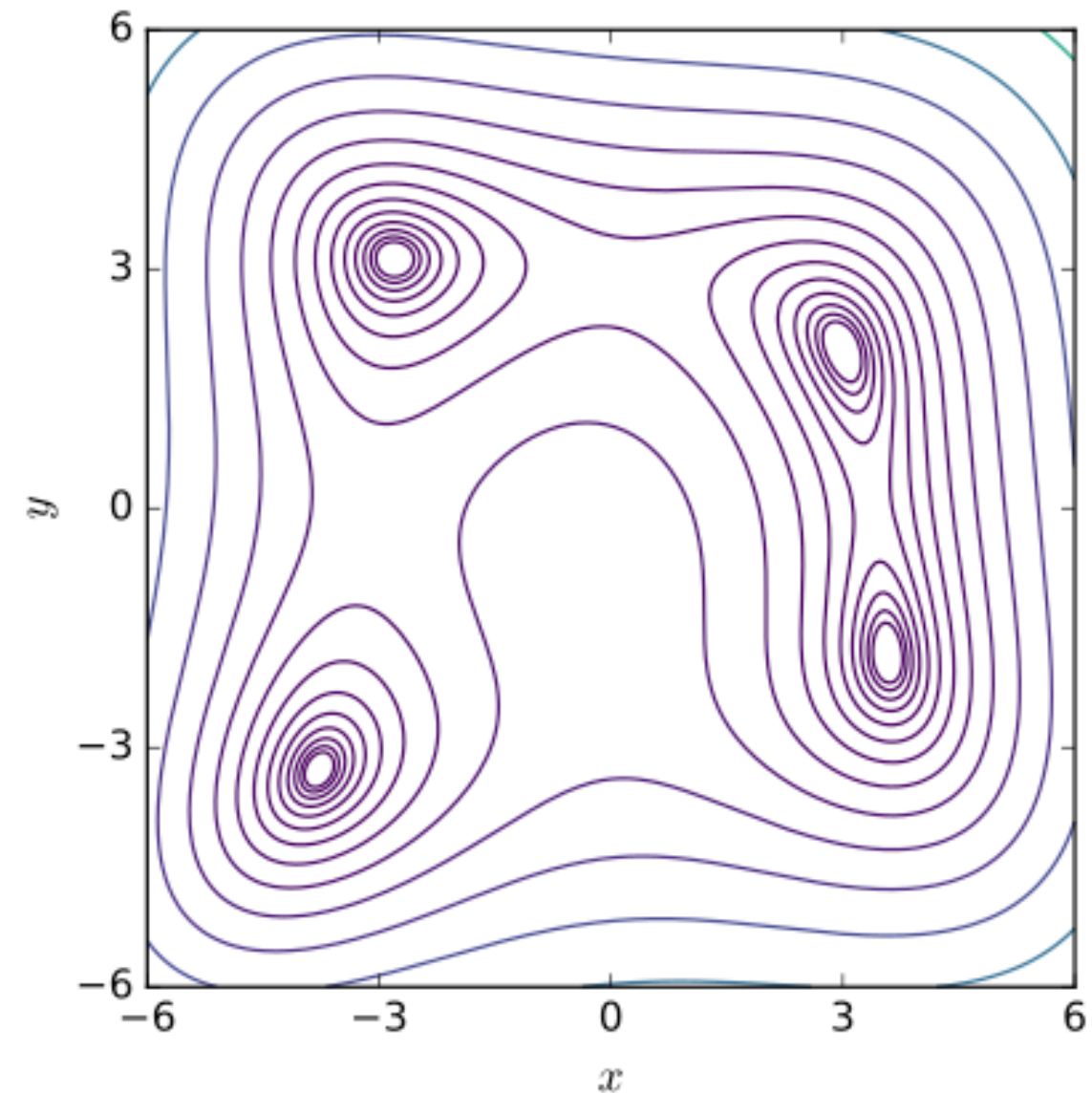
- 全探索
 - グリッドサーチ (mapper)
- 最適化
 - Nelder-Mead法 (minsearch)
 - ベイズ最適化 (bayes)
- モンテカルロサンプリング
 - レプリカ交換法 (exchange)
 - ポピュレーションアニーリング (pamc) [NEW!]

デモンストレーション用関数

- 以下では Himmelblau 関数をデモンストレーションに用いる

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

- 右図は $\log(f)$ の等高線プロット
- 4つの最小点 ($f=0$) を持つ
- py2dmat では Solver としてこのようなデモンストレーション関数をいくつか実装してある



By Nicoguardo - Own work, CC BY 4.0,

<https://commons.wikimedia.org/w/index.php?curid=51597574>

グリッドサーチ

- 探索空間をグリッド（格子点）に区切って、全点計算する
- 並列計算可能
 - 単純に担当箇所を分ける
- 大雑把な傾向を掴むのに便利
 - 探索空間の範囲を変える
 - Nelder-Mead の初期値を作る
- パラメータ数（＝探索空間の次元）が増えると点数（＝計算コスト）も増えていくことに注意
 - 点数を抑えるとスカスカになる

グリッドサーチのデモ

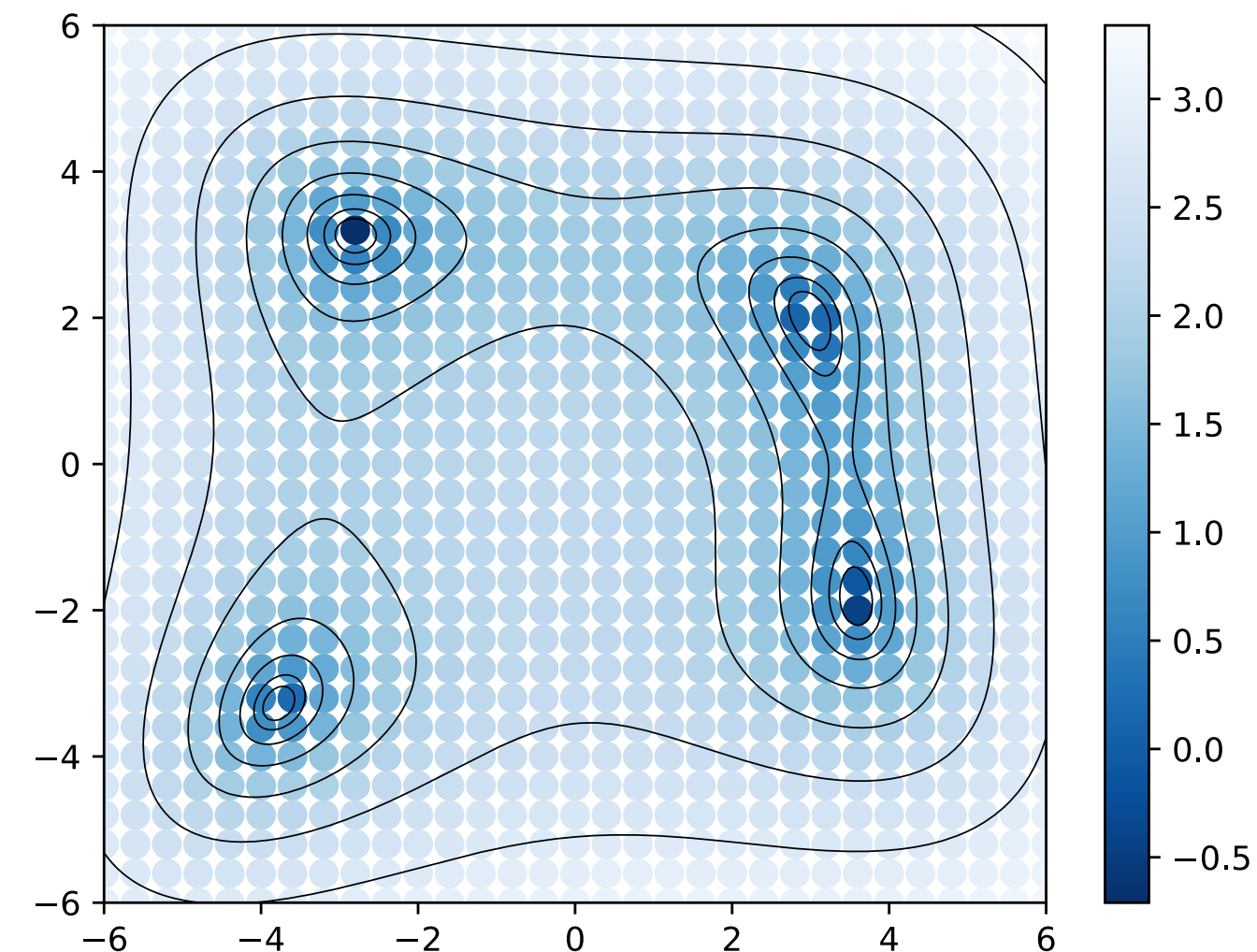
入力

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

```
[algorithm]
name = "mapper"           # グリッドサーチ
```

```
[algorithm.param]        # 探索空間
max_list = [6.0, 6.0]    # 上限
min_list = [-6.0, -6.0] # 下限
num_list = [31, 31]      # 刻み数
```

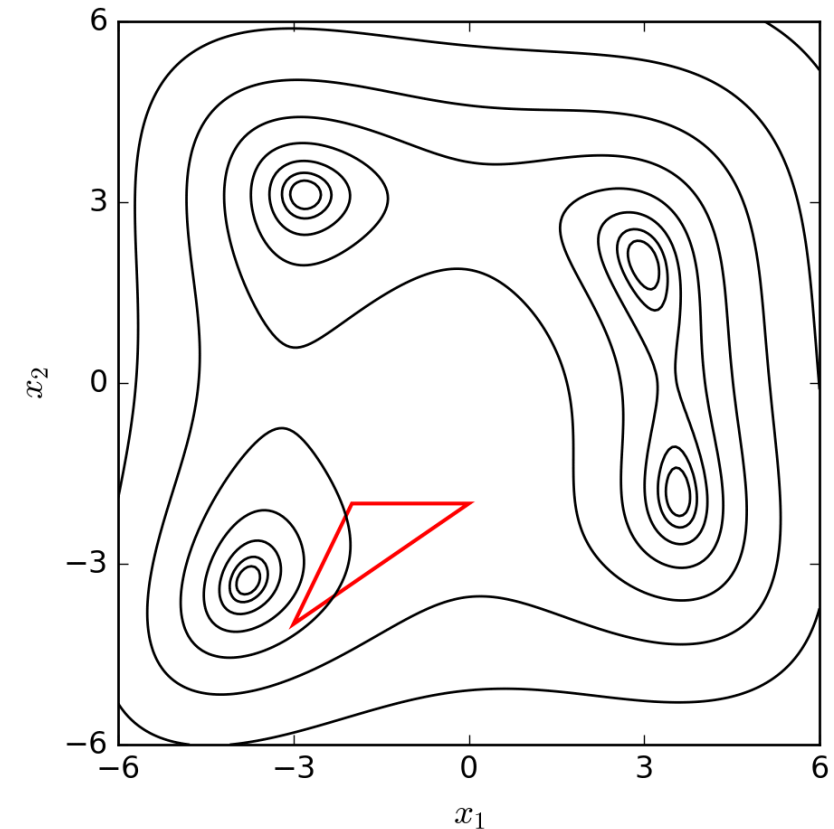
```
[solver]
name = "analytical"      # ベンチマーク関数を使う
function_name = "himmelblau" # 関数名
```



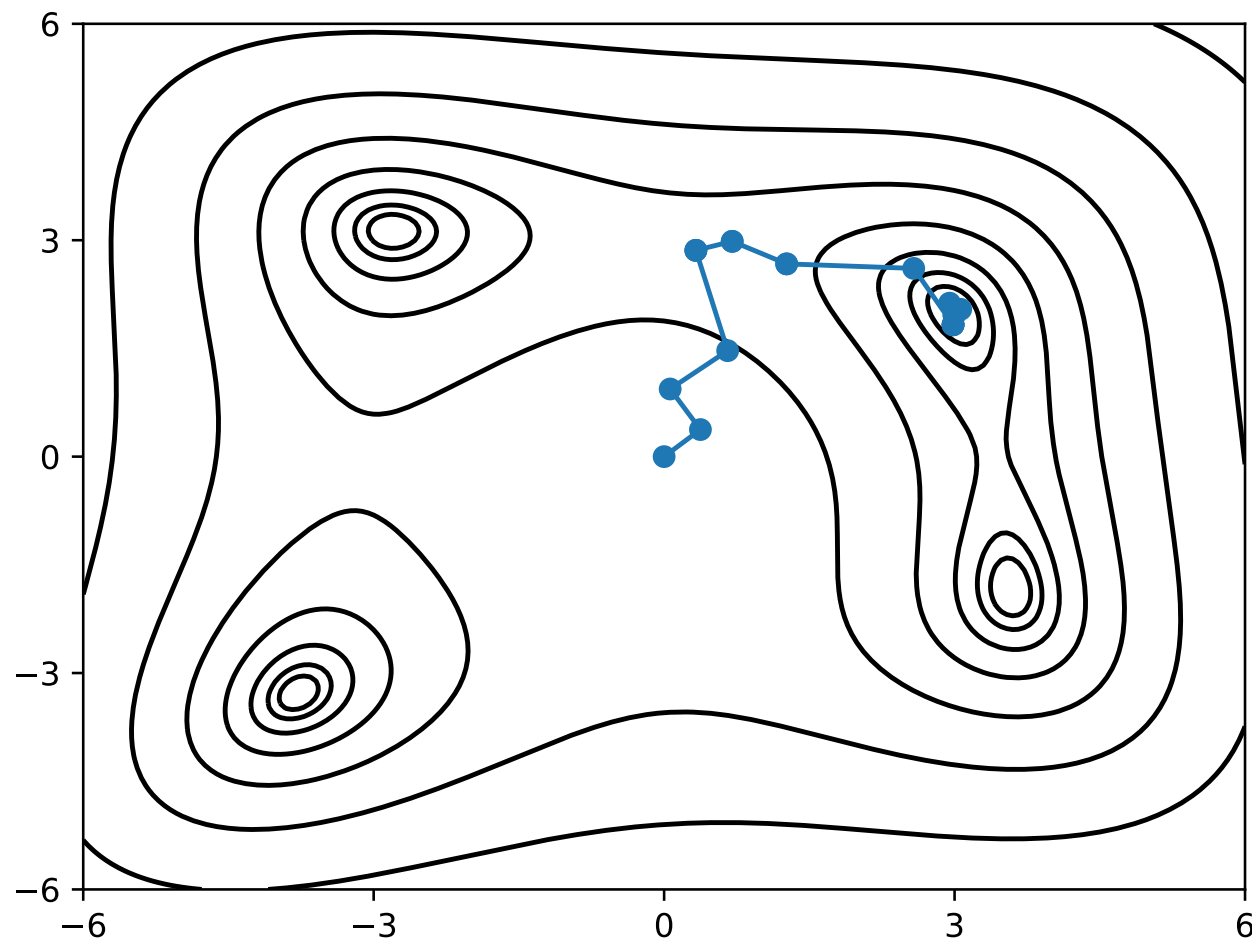
Nelder-Mead(a.k.a Simplex downhill)

https://commons.wikimedia.org/wiki/File:Nelder-Mead_Himmelblau.gif

- 連続空間の最適化手法
- 必要なのは目的関数の値 $f(x)$ のみ (導関数は不要)
- D 次元空間の最適化をする場合、 $D+1$ 個の点からなる単体 (三角形、四面体、...) を少しずつ動かして谷を下っていく
- 結果は初期配置に依存する
 - 右図(gif アニメ)にしめす Himmelblau 関数は4つの最小値を持つ
 - この例では単体 (赤三角) は一番近い最小値である左下に到達する
 - (PDF だとアニメーションしないので右上のURL を参照ください)
- いくつかの初期配置で何度かチェックする必要がある
 - 他の手法で大雑把にあたりを付けるのがよい



Nelder-Mead のデモ



入力

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

```
[algorithm]
name = "minsearch" # Nelder-Mead
seed = 12345
```

```
[algorithm.param]      # 探索空間
max_list = [6.0, 6.0]  # 上限
min_list = [-6.0, -6.0] # 下限
initial_list = [0, 0]  # 初期値
```

```
[solver]
name = "analytical"    # ベンチマーク関数を使う
function_name = "himmelblau" # 関数名
```

結果

```
fx = 4.2278370361994904e-08
x1 = 2.9999669562950175
x2 = 1.9999973389336225
```

原点から始めたら右上に到達した

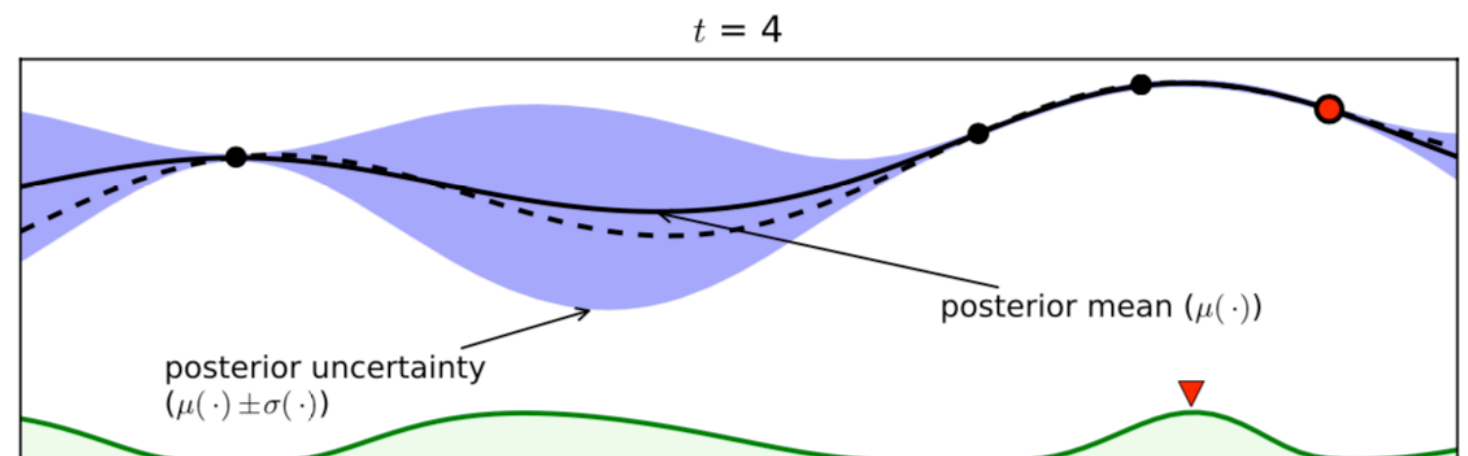
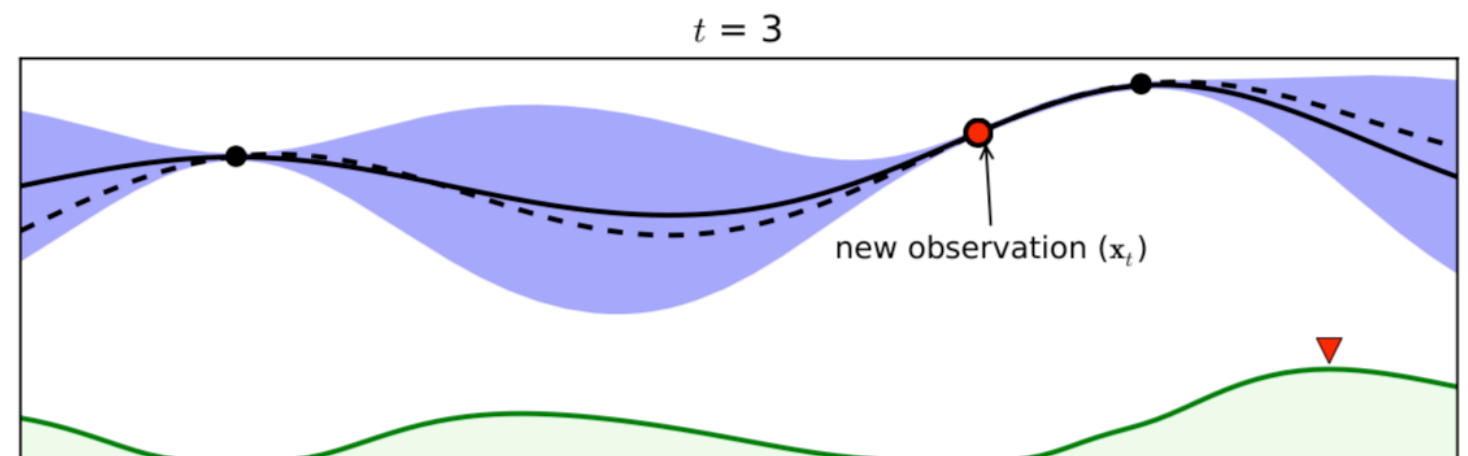
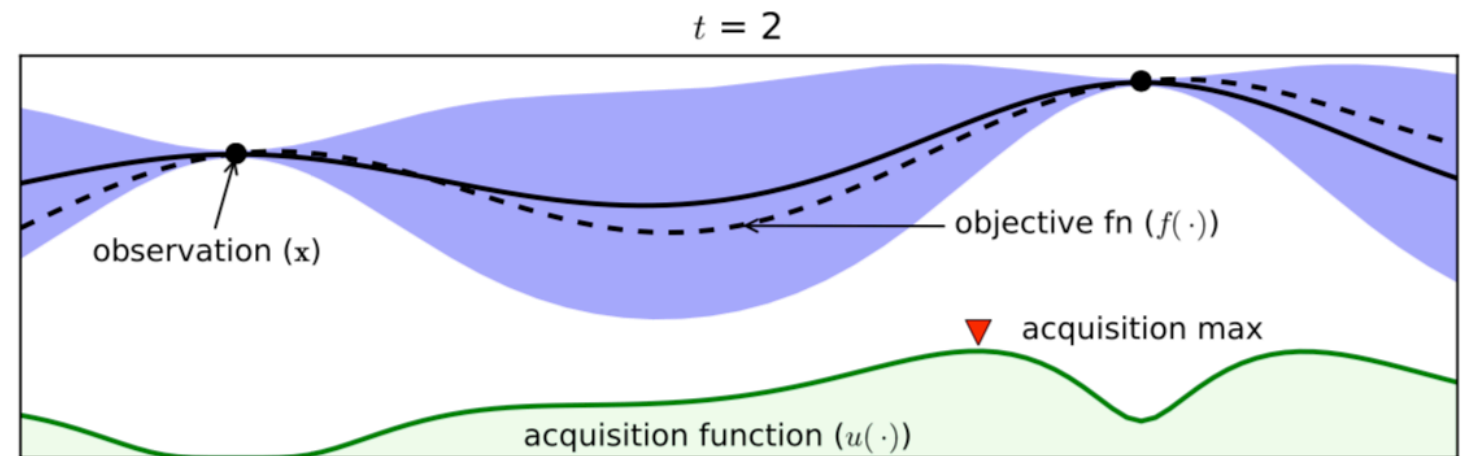
ベイズ最適化(Bayesian Optimization)

- $f(x)$ の評価が難しい（コスト高）であるときに特に有用な最適化手法
- 目的関数 $f(x)$ を、計算しやすいモデル関数 $g(x)$ で近似する
 - x_i を数点適当にサンプルし、 $y_i=f(x_i)$ を計算
 - (x_i, y_i) のセットを用いて $g(x)$ を学習する（フィッティングする）
 - $g(x)$ を最小化・最大化するような点 x' を次の観測点として、 $y'=f(x')$ を計算、学習データに追加して $g(x)$ を再学習する
 - 適当な回数繰り返す
- 一般的には $g(x)$ として、ガウス過程を事前分布とした事後分布を用いる
 - $g(x)$ という関数の分布をベイズ推定するという意味で「ベイズ」最適化
- 実際には $g(x)$ そのものではなく、 $g(x)$ の期待値・標準偏差から計算される獲得関数（スコア関数）を最小化・最大化する
 - 「利用」と「探索」のトレードオフ

ベイズ最適化の流れ

taken from E. Brochu, et al., arXiv:1012.2599

- 破線が真の目的関数（未知）
- 黒丸・赤丸が測定値
- 実線が推定の期待値
- 青が期待値の不確かさ
- 緑が獲得関数
- 三角が獲得関数の最大
- 獲得関数は、 $g(x)$ の期待値と分散とから定義される、候補点の「スコア」

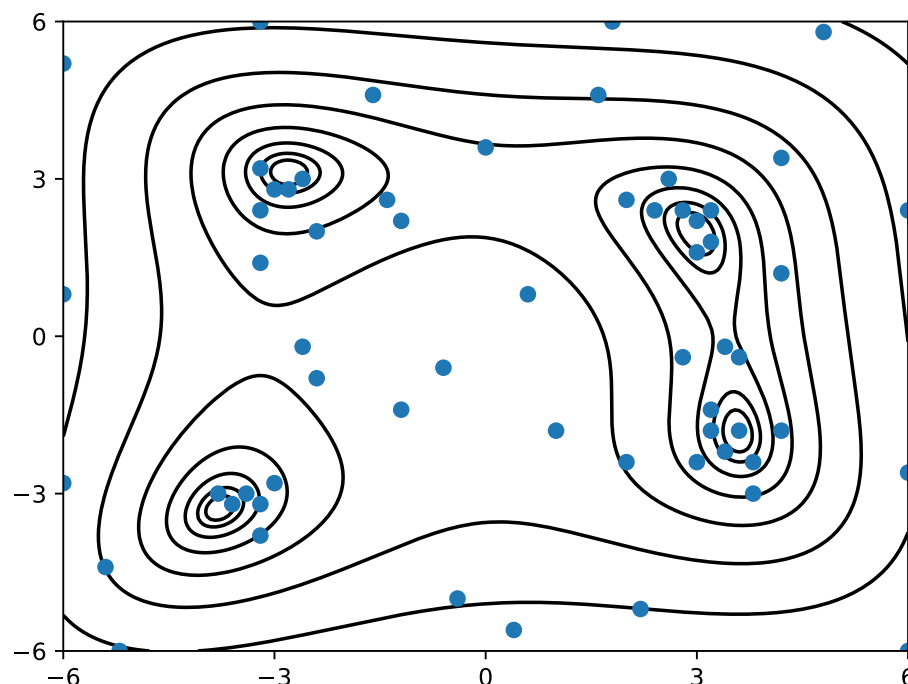


ベイズ最適化ライブラリPHYSBO

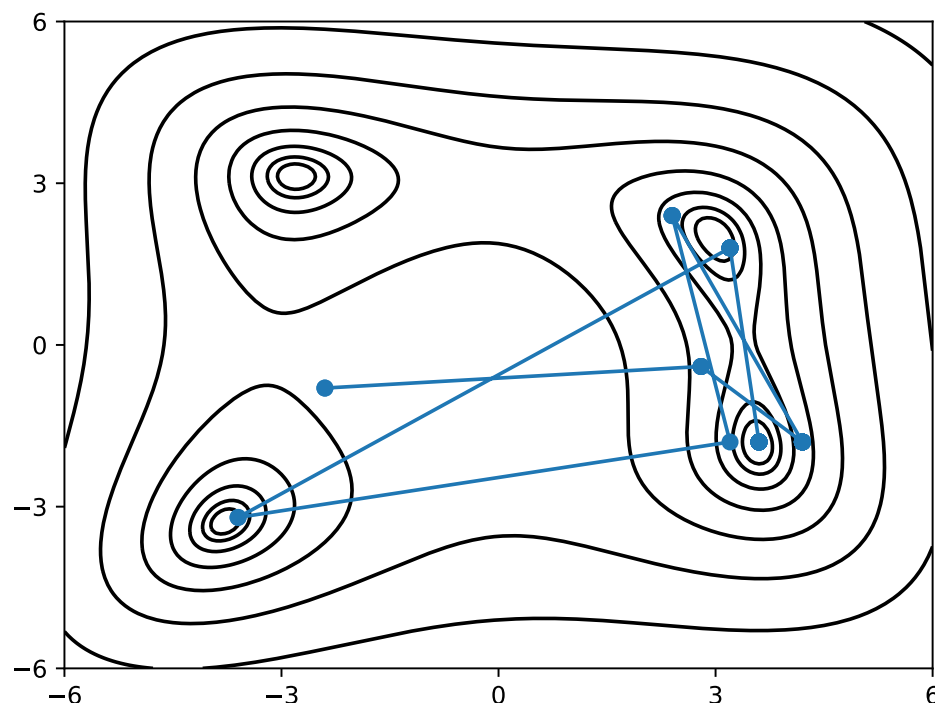
- py2dmat はベイズ最適化に PHYSBO ライブラリを利用している
 - <https://www.pasums.issp.u-tokyo.ac.jp/physbo/>
- 東大津田研で開発されているCOMBO というライブラリの派生
- py2dmat と同じく物性研ソフトウェア高度化プロジェクトで支援された
- 探索空間は離散化されている（最初に離散化した点の集合を与える）
 - 解像度や探索コストをコントロール可能
 - 気になる場合は最適化結果を初期値にして Nelder-Mead をする
 - 獲得関数の計算で並列計算可能
 - 探索空間中で担当箇所を分ける

ベイズ最適化のデモ

探索した点



最小値の変遷



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

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

```
[algorithm.param]
max_list = [6.0, 6.0]
min_list = [-6.0, -6.0]
num_list = [61, 61]
```

```
[algorithm.bayes]
# 初期ランダムデータの数
random_max_num_probes = 20
# ベイズ最適化で探すデータの数
bayes_max_num_probes = 40
```

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

3600 個の候補から、たかだか数十回でかなり良い解を選択

モンテカルロサンプリング

- $f(x)$ をできるだけ小さくしたい（大きくしたい場合は $-f(x)$ を考える）
- $f(x)$ の増加に対して単調増加する関数 $w(x)$ を導入し、 頻度分布が $w(x)$ に従うように x の系列を確率的に生成する
 - つまり、 $f(x)$ が小さい x ほど出やすい
 - 出てきた x のばらつきから結果の「不確かさ」を見積もれる
 - 手法としてはマルコフ連鎖モンテカルロ法を用いる（詳細はスキップ）
- w として具体的にはカノニカル分布（ボルツマン分布）を用いる
 - すなわち、目的関数 $f(x)$ を「エネルギー」とみなし、「温度」 T のもとで $w(x) = \exp[-f(x)/T]$ とする
 - 温度 T のもとでは、高さ T ぐらいの山なら乗り越えられる
 - T は f に対する精度・解像度
 - 温度 T をどうやって設定するかが問題（→レプリカ交換法・ポピュレーションアニーリング）

レプリカ交換モンテカルロ法

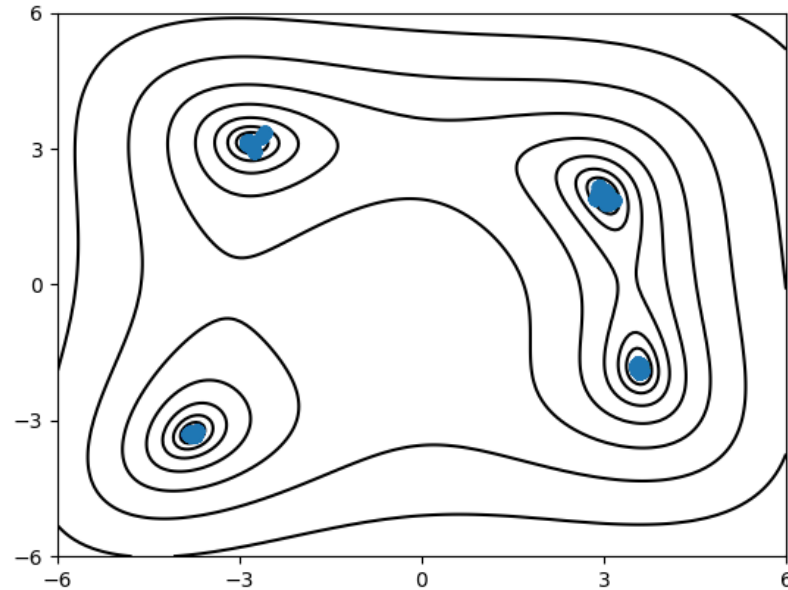
K. Hukushima and K. Nemoto, J. Phys. Soc. Jan.. **65**, 1604 (1996)

- マルコフ連鎖モンテカルロ法では、現在の x から確率的に次の x' を生成する
 - 普通は x の近くに x' を配置する
 - 温度 T のもとでは、高さ T ぐらいの山なら乗り越えられる
- 温度が高すぎると構造（山とか谷）が見えなくなる
 - 温度 ∞ 極限では重み w が一定になる
- 温度が低すぎると局所解（穴）から出てこれなくなる
- レプリカ交換モンテカルロ法
 - 複数の「レプリカ」を用意
 - 各レプリカは異なる温度で並列してモンテカルロサンプリングを行う
 - 時々温度を交換する

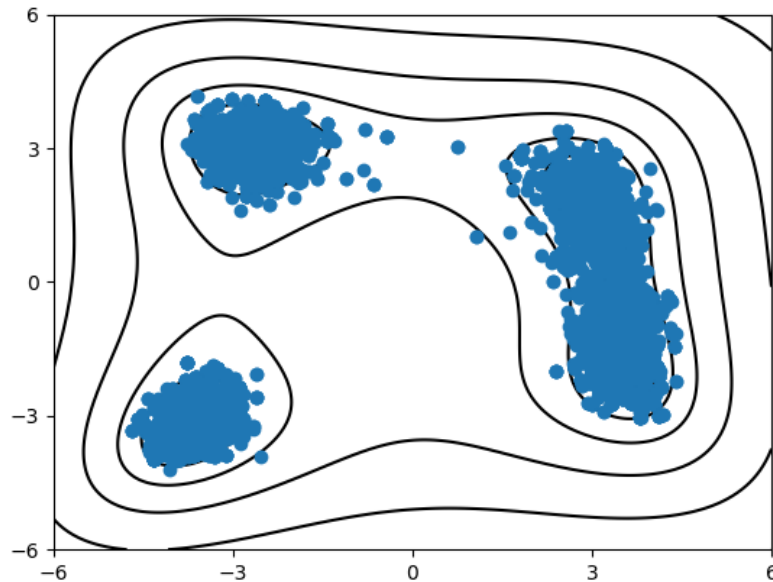
交換MC のデモ

```
$ mpiexec -np 4 --oversubscribe py2dmat input.toml
```

T=0.1



T=10



最初の20点は除外してある（初期緩和）

f が小さいところを重点的にサンプリングできている

```
[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]
initial_list = [0.0, 0.0]
```

```
[algorithm.exchange]
Tmin = 0.1          # 温度の下限
Tmax = 10.0         # 温度の上限
numsteps = 10000    # 生成するサンプル数
numsteps_exchange = 100 # 交換間隔
# 100 個生成するたびに交換を試みる
```

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

ポピュレーションアニーリング (PA)

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

- 交換モンテカルロ法では複数の温度を並列に計算していた
 - 並列化は容易だが、あまり並列度をあげられない
 - プロセス数が少ないと温度点を取りづらい
- 大量のサンプルを並列計算して、同時に温度を下げていく（アニーリング）のがポピュレーションアニーリング
 - 低温で局所解にトラップされても数の暴力で押し切る
 - 望み薄なサンプルを消去し、うまくいっているものを分裂させる
 - 原理的には好きなだけ並列数を増やせる
 - 並列数を増やせばサンプル数が増え、ひいては統計精度が増す

PA のデモ

```
$ mpiexec -np 24 --oversubscribe py2dmat input.toml
```

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

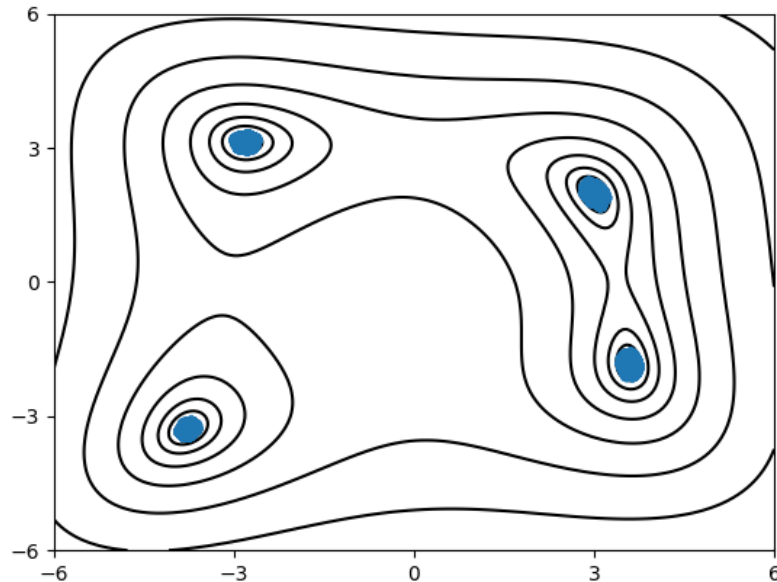
```
[algorithm]
name = "pamc"
seed = 12345
```

```
[algorithm.param]
max_list = [6.0, 6.0]
min_list = [-6.0, -6.0]
initial_list = [0.0, 0.0]
```

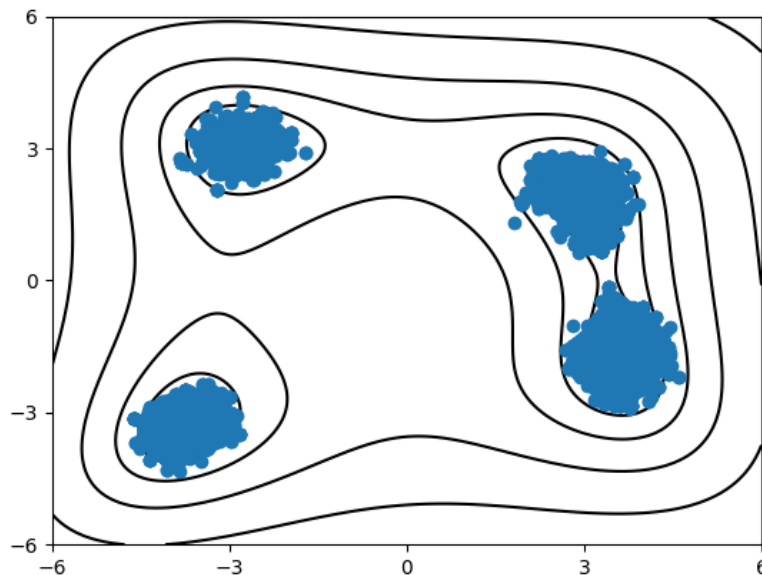
```
[algorithm.pamc]
bmin = 0.0          # 逆温度の下限
bmax = 10.0         # 逆温度の上限
Tnum = 11           # 温度点の数
Tlogspace = false  # 温度を対数で刻まない
numsteps_annealing = 100 # 温度降下の頻度
nreplica_per_proc = 100 # プロセスごとのレプリカ数
```

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

T=0.1



T=1.0



Solver

～順問題・物理モデル～

- Python 関数
 - デモンストレーション関数 (analytic)
 - Python 関数ラッパー (function)
- 表面ビーム実験
 - TRHEPD/RHEED (sim-trhepd-rheed)
 - 表面X線回折 (sxrd) [NEW!]
 - 低速電子線回折 (leed) [NEW!]

自作のAlgorithm, Solver を使う

- 独自のAlgorithm, Solver を定義することでpy2dmat を拡張可能
- Solver については、numpy.ndarray から float を返す関数を作り、solver.function ソルバーにわたすとよい
 - 将来的には（次のバージョンには？）入力ファイルの中で目的関数を定義できるようにしたい
 - function_solver ブランチで利用可能
- 現バージョン (2.1) では analytical ソルバーに追記するのが楽かも
- 複雑なSolver やAlgorithm についてはマニュアルや既存コードを参照

```
[solver]
name = "function"
function = """
def _f(x):
    return np.sum(x**2)
"""
```


sim-trhepd-rheed (STR) との接続

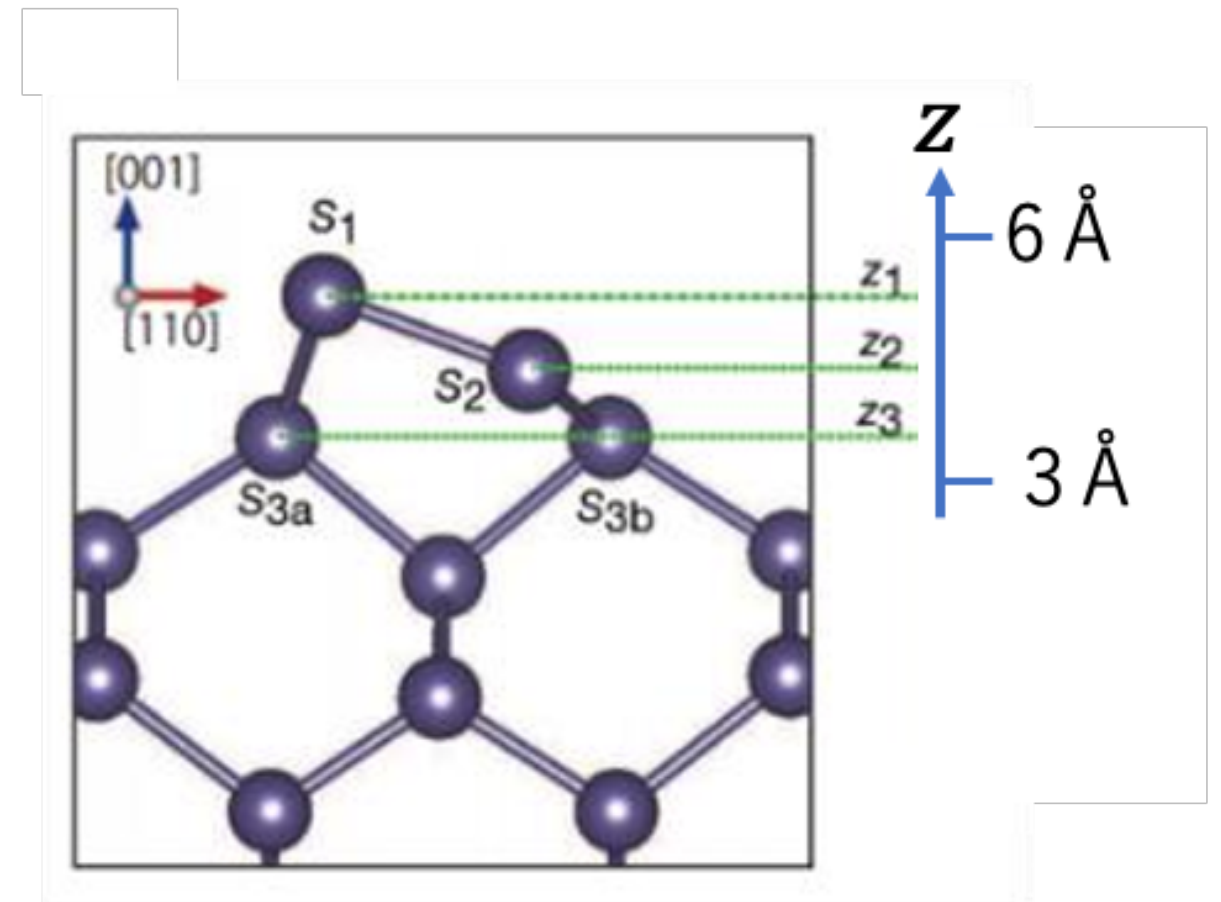
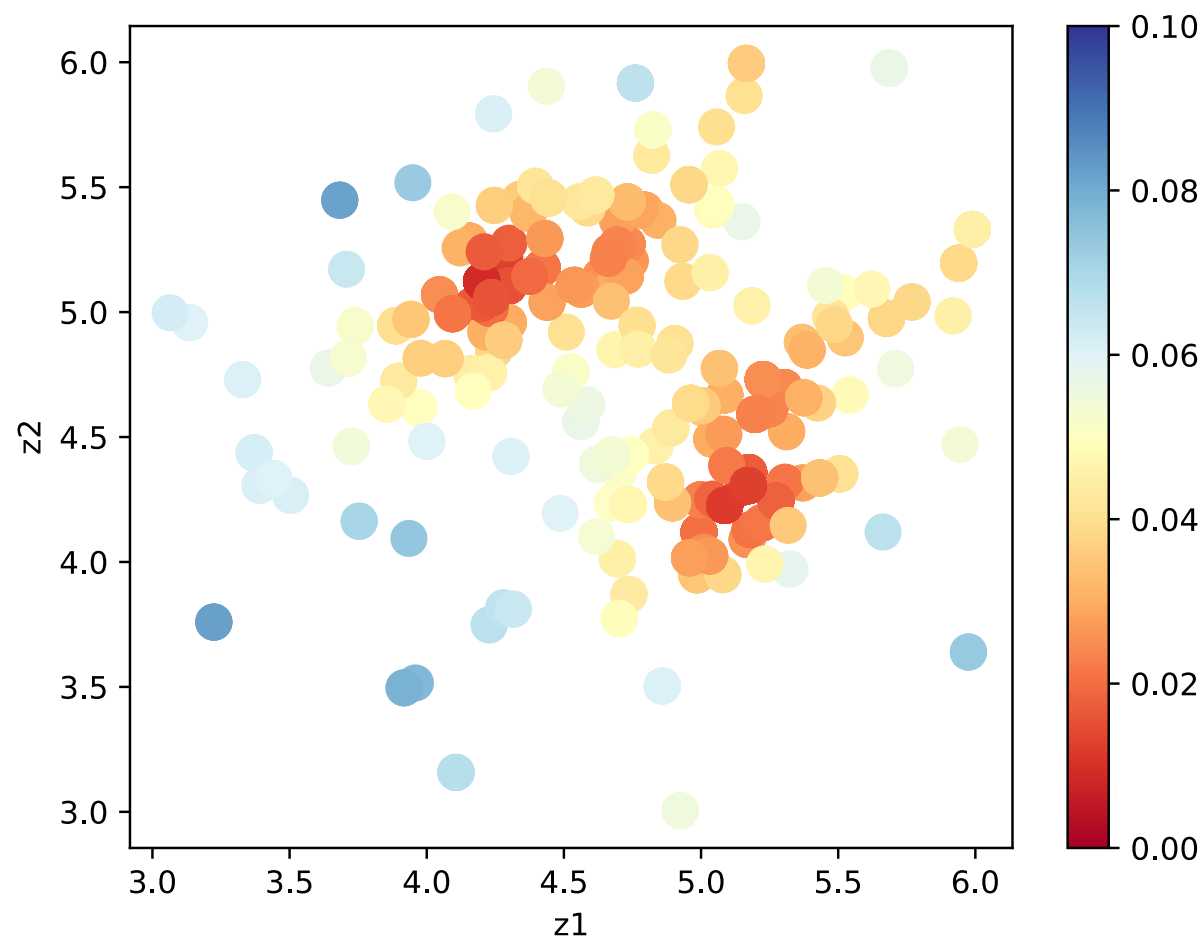
- sim-trhepd-rheed (STR)は TRHEPD と RHEED において、表面原子の座標を与えるとビーム入射角と反射強度の組 (rocking curve) を返すプログラム
 - <https://github.com/sim-trhepd-rheed/sim-trhepd-rheed>
 - by 花田貴 氏@東北大金研
- py2dmat は STR を用いた Solver を提供している
 - 入力 x は 原子座標
 - 目的関数 $f(x)$ は、別に用意した実験曲線 (D_{exp}) と 理論曲線 ($D(x)$) との距離 $f(x) = \sqrt{\sum (D_{\text{exp}} - D(x))^2}$
- ベイズ最適化や Nelder-Mead などを用いて $f(x)$ を最小化することで、表面原子構造の推定を行える (逆問題)

STR との接続

- STR はバルクの寄与を計算する `bulk.exe` と、実際に反射強度を計算する `surf.exe` との2つの主プログラムを含む
- `bulk.exe` は最初に一度実行しておけばよい
 - `py2dmat` を実行する前にやっておく
 - バルクの原子配置やビーム条件など、表面状態を調べるにあたって不変な情報を入力に渡す
- パラメータX（原子配置など）ごとに `surf.exe` が実行され、Rocking 曲線が計算され、実験の曲線との差 (R-factor) が目的関数 $f(X)$ として返される
 - `surf.exe` の入力ファイルを自動生成する
 - 正確には、テンプレートファイルの一部文字列をパラメータの値で置き換える
 - 詳細はSTR と 2dmat のマニュアルや、本スライドの付録を参照

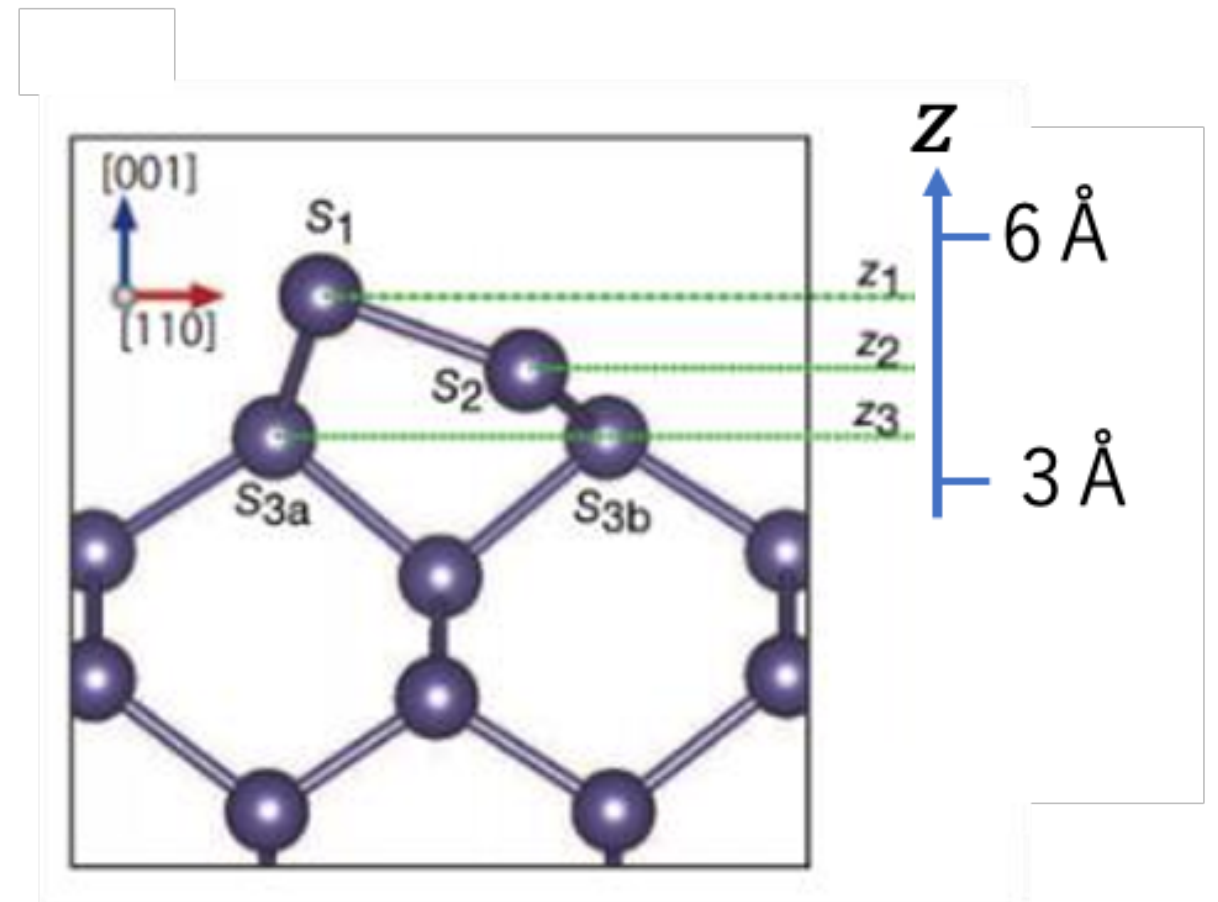
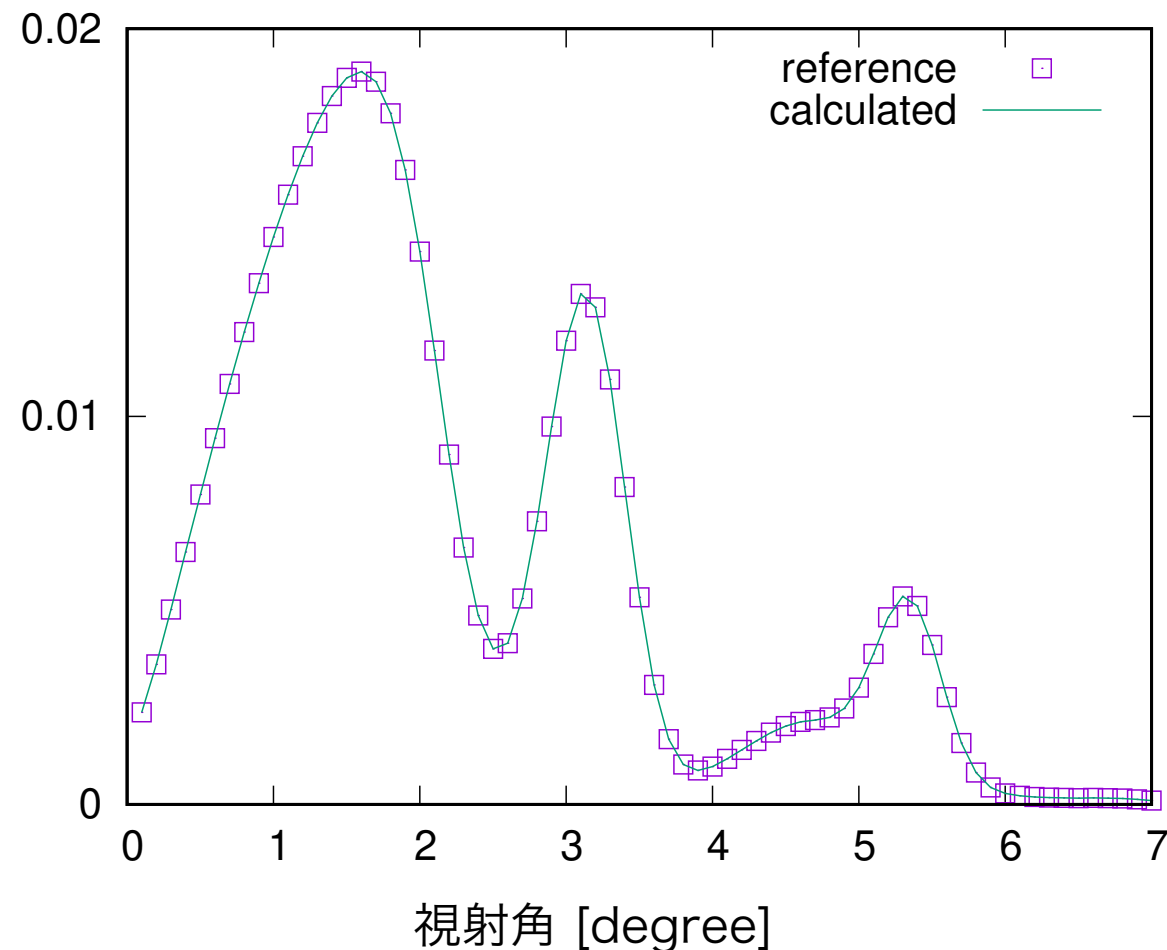
py2dmat(exchange) + STRの例

- Ge(001)-c4x2 表面の2つの Ge 原子のz 座標 (z_1, z_2)をサンプリングする例
 - 「実験データ」としては適当な原子座標に対してSTR で計算した人工データを使用
- この例では z_1, z_2 は交換に対して対称



py2dmat(minsearch) + STRの例

- Ge(001)-c4x2 表面の2つの Ge 原子のz 座標 (z_1, z_2) の最適化
 - 「実験データ」としては適当な原子座標に対してSTR で計算した人工データを使用
- Nelder-Mead で最適化した座標における rocking 曲線



sxrdcalc との接続

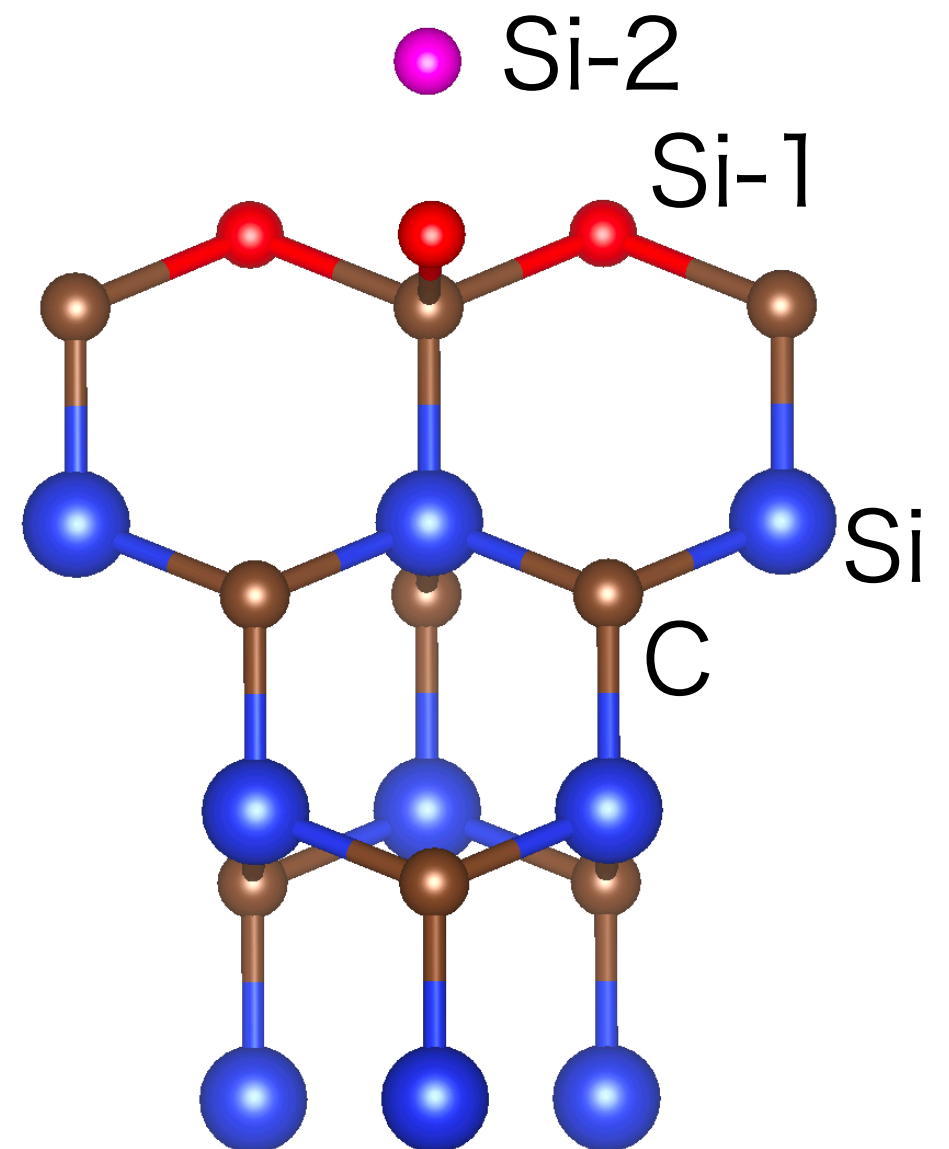
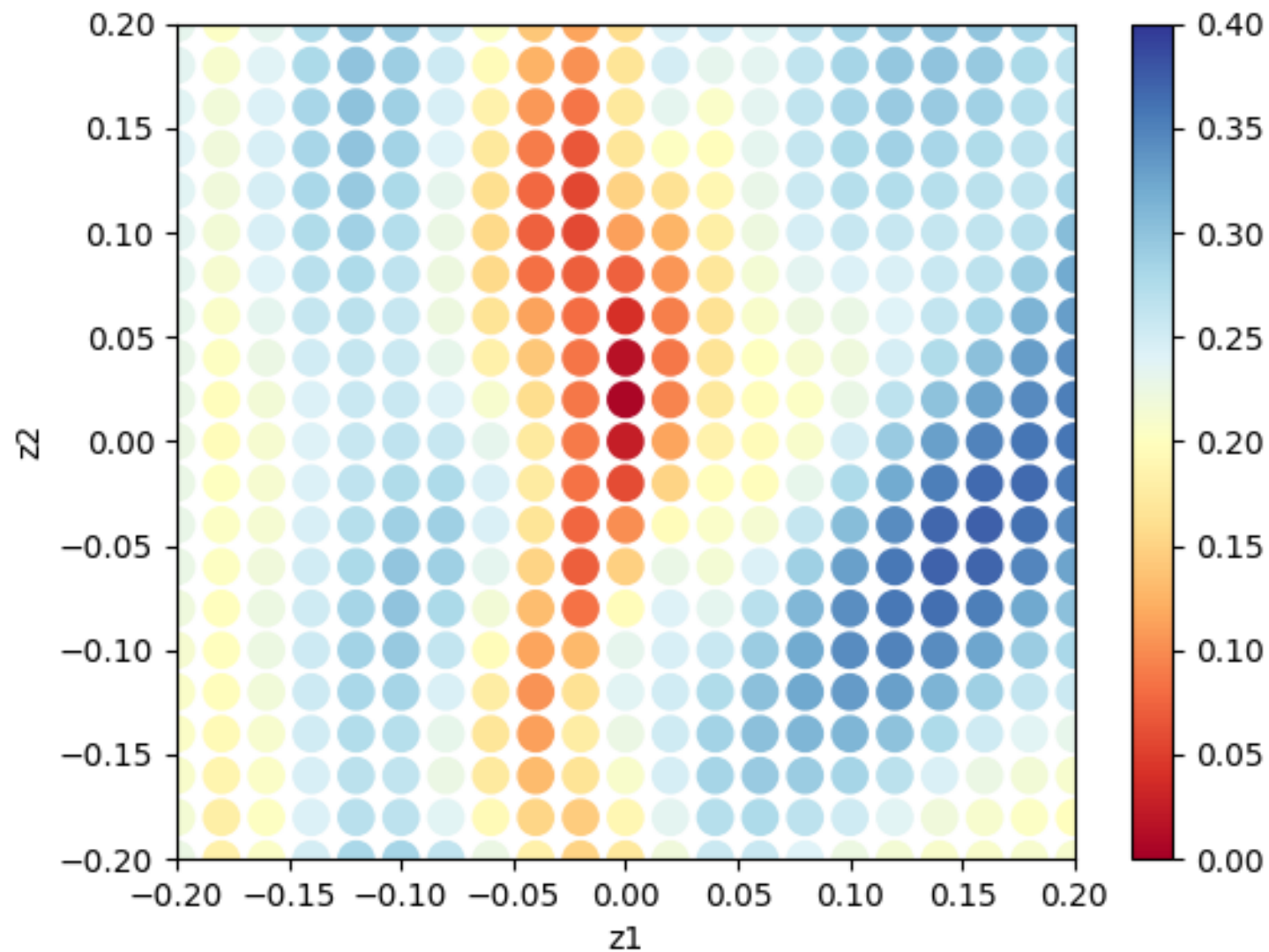
- sxrdcalc は 表面X線散乱(SXRD) において、原子座標から構造因子を計算したり、散乱データにあうように原子座標を最適化したりするソフトウェア
 - <https://github.com/sxrdcalc/sxrdcalc>
 - by Wolfgang Voegeli 氏@学芸大
- py2dmat は sxrdcalc を用いた Solver を提供している
 - 入力 x は 原子座標
 - 目的関数 $f(x)$ は、（実験などで）別に用意した構造因子 (D_{exp}) と 理論計算によるもの ($D(x)$) との距離
- ベイズ最適化や Nelder-Mead などを用いて $f(x)$ を最小化することで、表面原子構造の推定を行える（逆問題）

sxrdcalc との接続

- sxrdcalc では表面X線回折の構造因子やR-factor を計算できる
- py2dmat との接続では、バルクファイル（バルクの原子座標）が別途必要
- 表面ドメインの数や表面原子の基準位置など、sxrdcalc が要求する他のパラメータは py2mat 自体の入力ファイルで指定できる
 - 探索パラメータ毎に sxrdcalc の入力ファイルを動的に作成する
- 詳細はsxrdcalc と2dmat のマニュアルや、本スライドの付録を参照

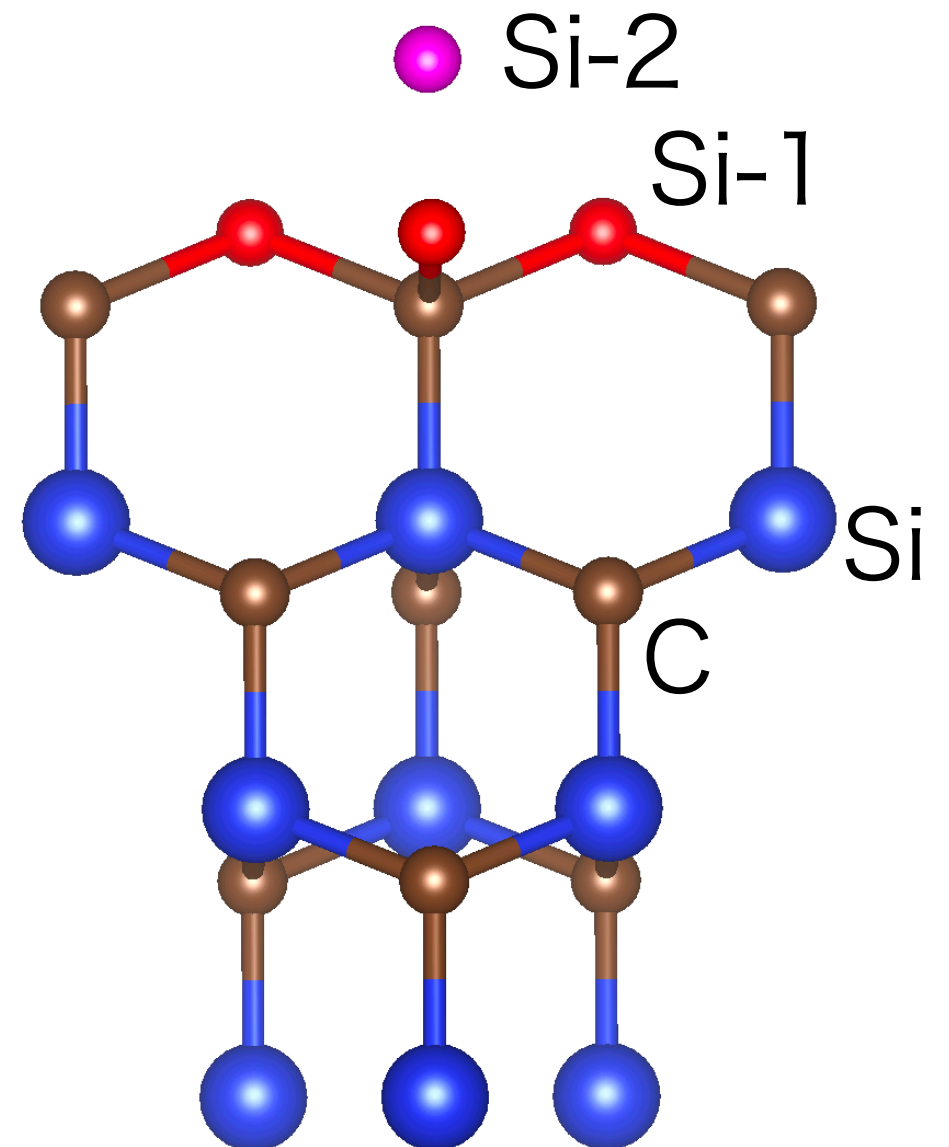
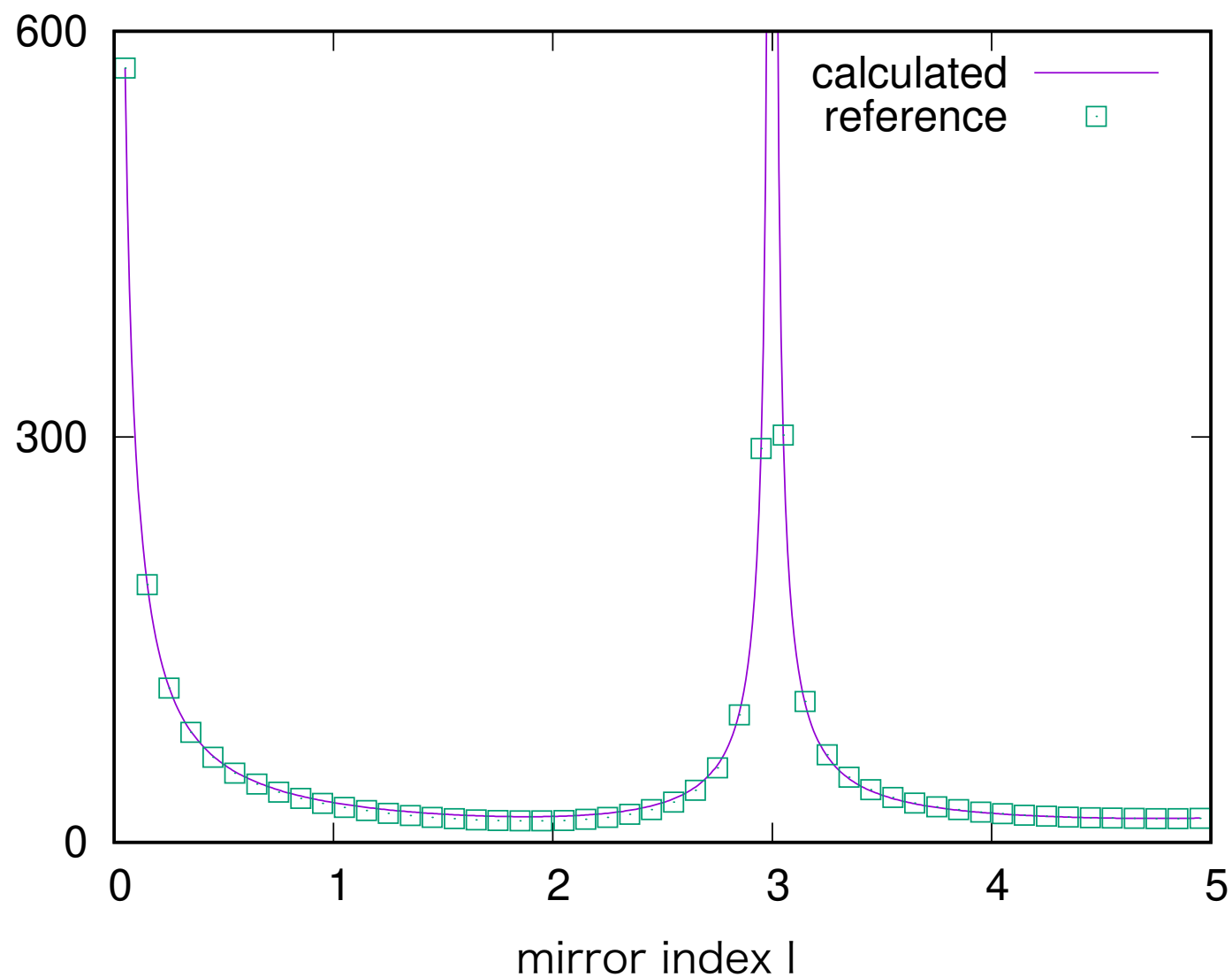
py2dmat(mapper) + sxrdcalcの例

- SiC(111)-r3xr3 表面の2種類の Si 原子 (Si-1, Si-2)のz 座標 (z1,z2)に対するグリッドサーチ



py2dmat(minsearch) + sxrdcalcの例

- SiC(111)-r3xr3 表面の2種類の Si 原子 (Si-1, Si-2)のz 座標 (z1,z2)の最適化
- Nelder-Mead で得られた座標に対する構造因子



satleed との接続

- satleed は 低速電子線回折(LEED) の解析ソフトウェア
 - https://www.icts.hkbu.edu.hk/VanHove_files/leed/leedpack.html
 - by Prof. M. A. Van Hove
- py2dmat は satleed を用いた Solver を提供している
 - 入力 x は 原子座標
 - 目的関数 $f(x)$ は、（実験などで）別に用意した rocking curve (D_{exp}) と 理論計算によるもの ($D(x)$) との距離
- ベイズ最適化や Nelder-Mead などを用いて $f(x)$ を最小化することで、表面原子構造の推定を行える（逆問題）
- satleed 自体は系のパラメータごとにコードを書き換えてコンパイルし直す必要があり、2DMAT はそのサポートまでカバーしていない
 - satleed に慣れている人向け

まとめ・展望

- py2dmat
 - 一般的な最適化・サンプリングのためのパッケージです
 - 手元のパソコンから大規模計算機まで使えます
 - 表面ビーム解析のシミュレータと接続可能です
 - [arXiv:2204:04484](https://arxiv.org/abs/2204.04484)
 - py2dmat+sim-trhepd-rheed によるもう少し細かい解析の例を書いています
- バージョンアップなど
 - 今後、既存の入力パラメータ名などが変わる可能性もあります
 - 古い物をいきなり消すことはありませんが、警告を表示します
 - バージョンが進めばいずれ消します
- 要望やバグ報告などの問い合わせはお気軽にどうぞ
 - 2dmat-dev@issp.u-tokyo.ac.jp

付録A

sim-trhepd-rheed

STR との接続

- STR はバルクの寄与を計算する bulk.exe と、実際に反射強度を計算する surf.exe との2つの主プログラムを含む
- bulk.exe は最初に一度実行しておけば良い
 - py2dmat を実行する前にやっておく
- bulk.exe の入力ファイルは bulk.txt
 - バルクの情報（原子構造）
 - 原子座標や空間群
 - 原子の原子数や散乱因子
 - などなど
 - ビームの情報
 - 視射角の初期値・最終値・刻み幅
 - ビームの強度
 - などなど
- bulk.exe の実行結果は bulkP.b として保存される（バイナリファイル）

表面の解析にあたって不変な情報を指定

STR との接続

- surf.exe は入力ファイルとして bulkP.b と surf.txt を用いる
 - surf.txt は表面構造の情報を持つ
 - 原子座標や原子種など (bulk.txt と同様)
 - py2dmat は探索パラメータ（原子座標など）から surf.txt を生成し、surf.exe を実行する
 - surf.txt の「テンプレート」ファイル template.txt を用意する
 - ある原子のz 座標など、 surf.txt の一部を（例えば） value_01 などの文字列に置き換えたもの
 - py2dmat ではこの文字列を実際の数値に置き換えて surf.txt を生成する
- ```
サンプルより一部抜粋
ある原子の z 座標をそれぞれ value_01 と呼んでいる
1, 1.0, 1.34502591 1 value_01 ,IELM(I),ocr(I),X(I),Y(I),Z(I)
```
- surf.exe が出力する surf-bulkP.s から rocking curve を読み取る

# STR との接続 (py2dmat の入力ファイル)

```
[base]
dimension = 2 # 探索空間の次元 = 動かすパラメータの数

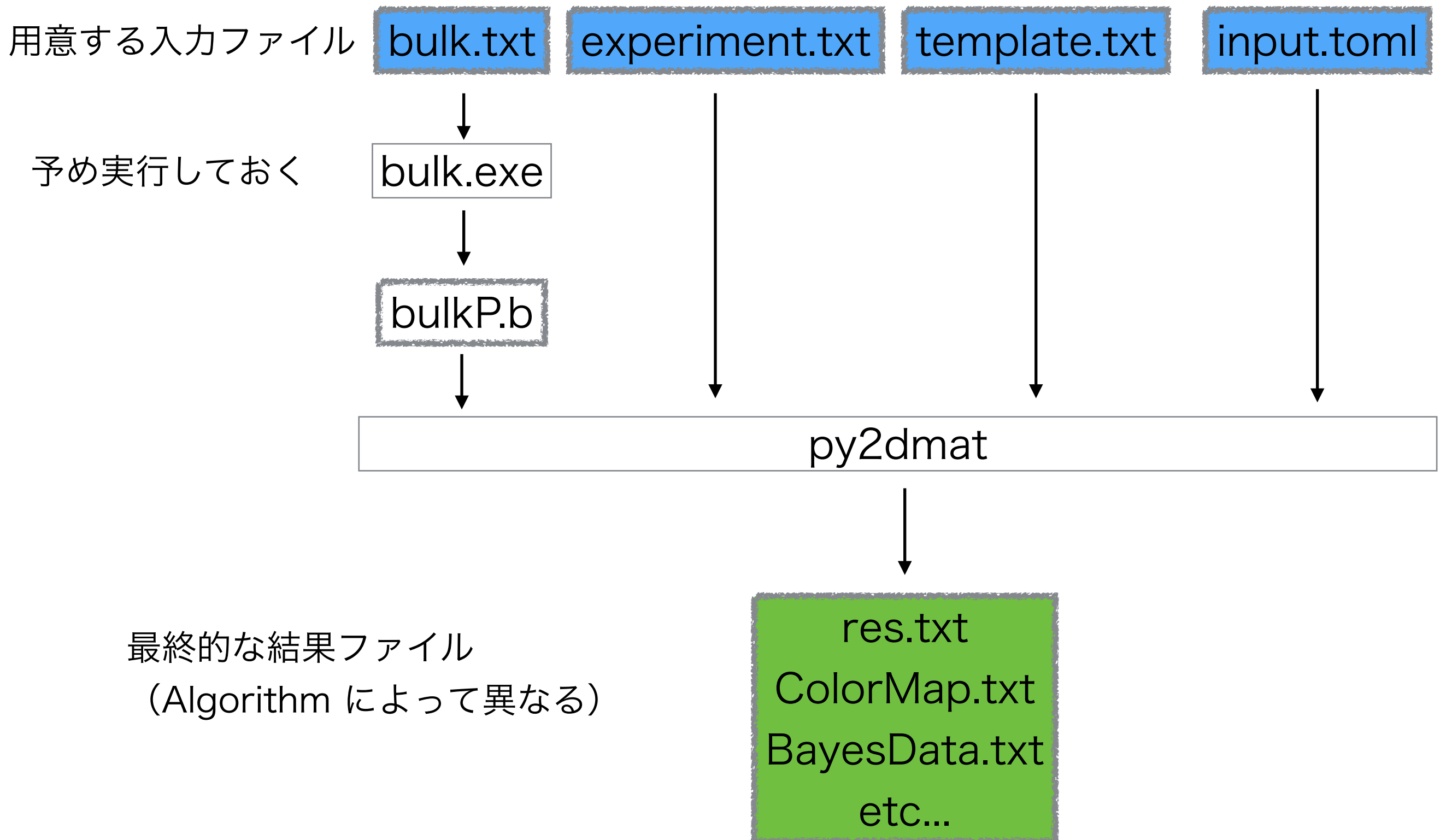
[solver]
name = "sim-trhepd-rheed" # 順問題ソルバーとして sim-trhepd-rheed を使う

[solver.config]
surface_template_file = "template.txt" # テンプレートファイル (default: template.txt)
calculated_first_line = 5 # surf-bulkP.s 中、D(x) の最初の行番号
calculated_last_line = 74 # D(x) の最後の行番号
row_number = 2 # D(x) の値が入っている列番号

[solver.param]
string_list = ["value_01", "value_02"] # テンプレートファイルのどの文字列を置き換えるか
degree_max = 7.0 # 視射角の最大値 (読み取ったものと異なっても警告が出るだけ)

[solver.reference]
path = "experiment.txt" # 実験データ Dexp が書いてあるファイル
first = 1 # Dexp として使う部分の最初の行番号
last = 70 # Dexp の最後の行番号
```

# py2dmat + STRのフロー



付録B

sxrdcalc



# STR との接続

- sxdcalc で表面原子座標を最適化するには、基準点と(1-3 本の)変位ベクトルの組み合わせで原子座標を指定することで対称性などの束縛条件を考慮する
  - たとえば次の例で示されるSi 原子の座標 $\vec{r}$  は
$$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \alpha_1 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

```
pos Si 0.0 0.0 1.0 0.0 1.0 # 基準座標 デバイワラー因子と重み
displ1 1 0.0 0.0 1.0 # パラメータ番号 変位ベクトル
```
  - $\alpha_1$  が探索パラメータとなる (他の原子の変位とも共通化可能)
- py2dmat との接続でもこれを踏襲する

```
[[solver.param.domain.atom]]
name = "Si"
pos_center = [0.0, 0.0, 1.0] # 基準座標
DWfactor = 0.0 # デバイワラー因子
occupancy = 1.0 # 重み
displace_vector = [[1, 0.0, 0.0, 1.0]] # パラメータ番号と変位ベクトル
```



# py2dmat + sxrdcalcのフロー

