2DMAT

CCMSハンズオン: 2DMAT講習会

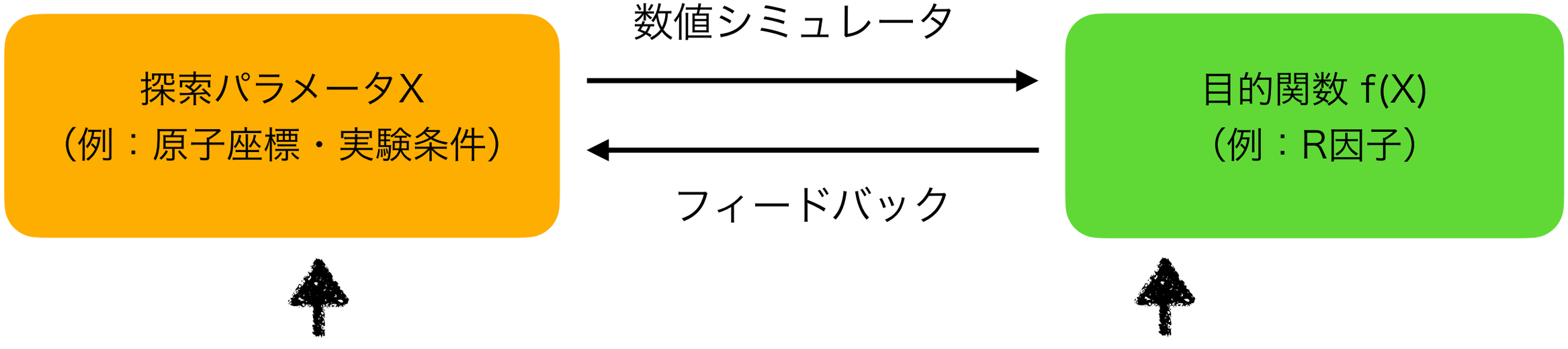
2024年12月02日 @物性研A614+Zoom

東京大学物性研究所 附属物質設計評価施設
ソフトウェア開発・高度化チーム

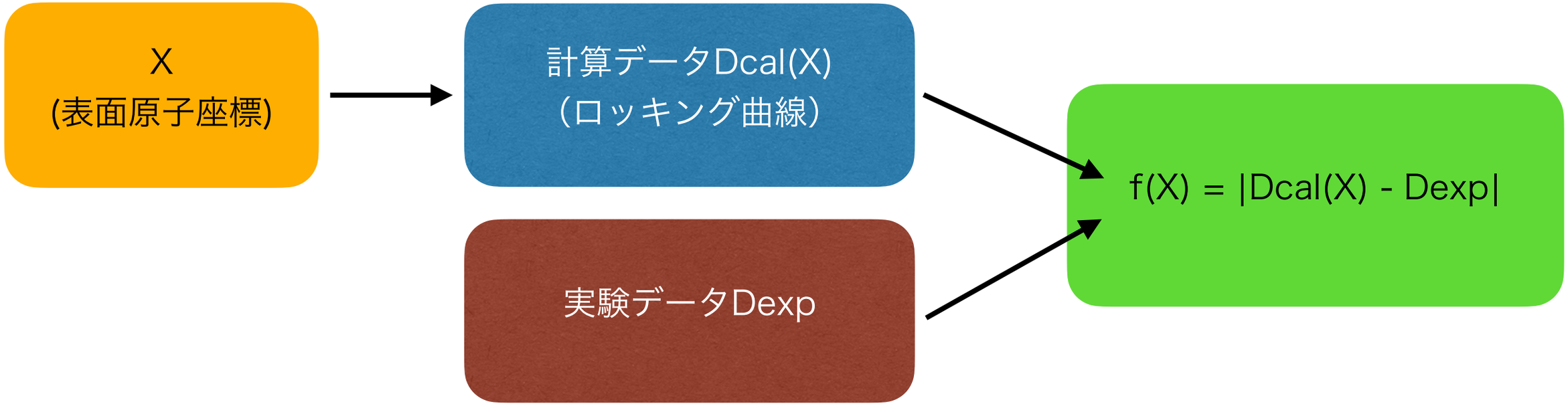
2DMATとは

- 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
- 主要なターゲット
  - ビーム回折実験による表面の原子構造解析
    - 実験で得られたデータ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)を最小化する最適化問題として定式化可能

逆問題解析



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



2DMATの構成

- 2DMAT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) は実験で得られたロッキング曲線と計算で得られるロッキング曲線との"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）
    - サンプリング（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

2DMAT → ODAT-SE

- 2DMAT version 3 から
  - 順問題ソルバーと逆問題解析アルゴリズムを分離
  - より汎用の解析プラットフォームに
  - 名称変更:

Open Data Analysis Tool for Science and Engineering (ODAT-SE)

- 順問題ソルバーは独立なモジュールとして構成
  - 解析対象となる実験やモデル計算ごとに用意する
- 逆問題解析アルゴリズムの追加・拡張が可能

ODAT-SEへの移行ガイド

(2DMATを使ったことがあるユーザ向け)

2DMATユーザの方

- py2dmat コマンド → ソルバーごとの実行コマンド (例: odatse-STR)
- 入力ファイル input.toml は、ほぼそのまま利用可能

ユーザプログラム(main.py)を作成している方

- include 文やモジュール指定を書き換え: py2dmat → odatse
  - 例: import py2dmat → import odatse

ソルバークラスを自作している方

- include 文やモジュール指定を書き換え: py2dmat → odatse
- インターフェース変更
  - prepare/run/get_result メソッドを evaluate メソッドにまとめる
  - 引数の Message クラスを置き換え

ODAT-SEのインストール

- ODAT-SEライブラリと、個別の順問題ソルバーモジュールをそれぞれインストールする
- ODAT-SE/順問題ソルバーは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

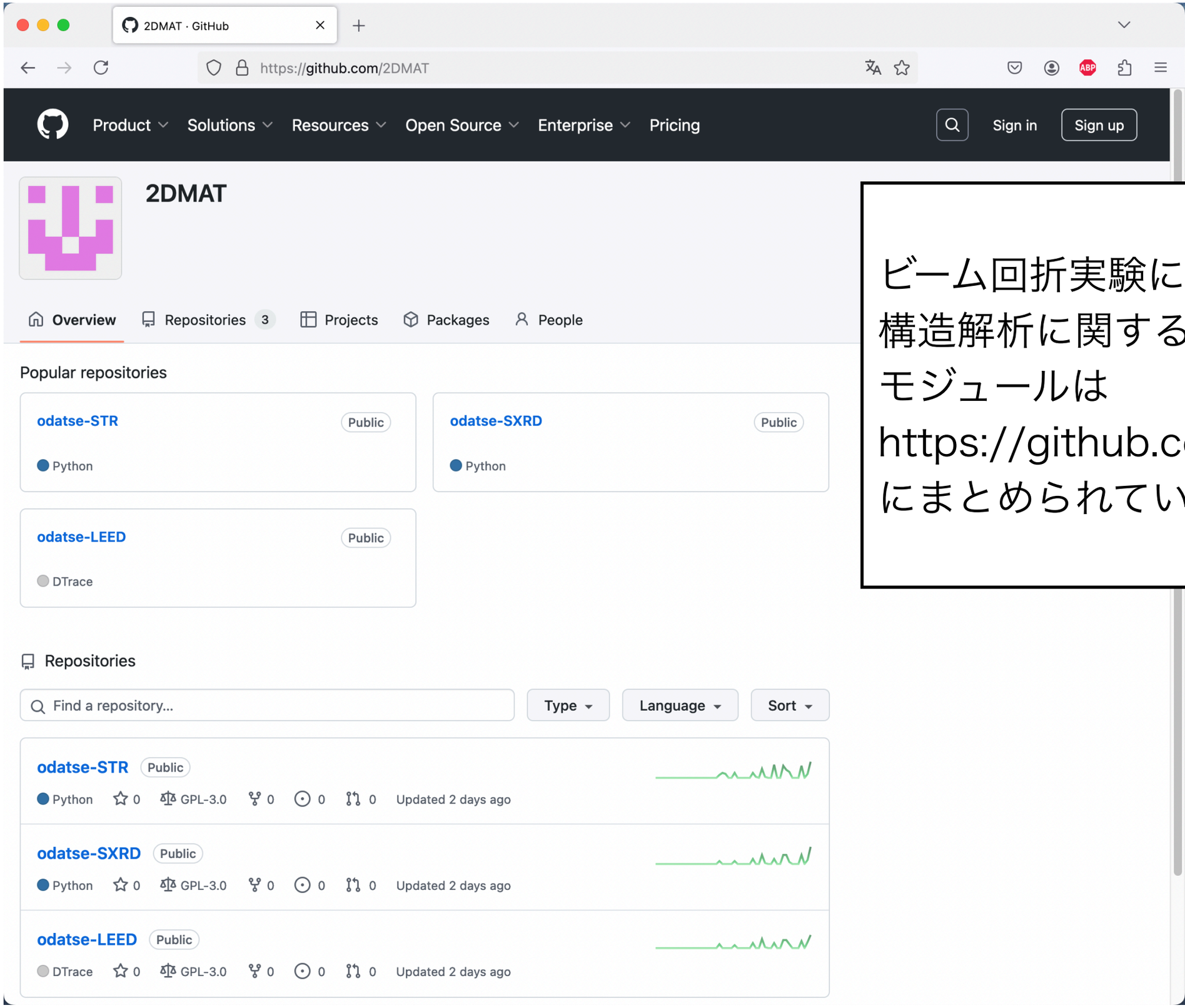
```
$ python3 -m pip install ODAT-SE
```

  - odatse moduleとodatseコマンドがインストールされる

- 現在公開している順問題ソルバー

	pip install するモジュール
全反射高速陽電子回折 TRHEPD/反射高速電子線回折 RHEED	odatse-STR
表面X線回折 SXRD	odatse-SXRD
低速電子線回折 LEED	odatse-LEED

ODAT-SEのインストール



ビーム回折実験による物質表面の構造解析に関する順問題ソルバーモジュールは
https://github.com/2DMAT
にまとめられています

ODAT-SEのインストール

- 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/ODAT-SE
```

  - sampleディレクトリとscriptディレクトリに色々入っている
  - src/odatse_main.pyをpythonスクリプトとして実行するとodatseコマンドと同じことができる

- TRHEPD/RHEED順問題ソルバー

```
$ git clone https://github.com/2DMAT/odatse-STR
```

- SXRD順問題ソルバー

```
$ git clone https://github.com/2DMAT/odatse-SXRD
```

- LEED順問題ソルバー

```
$ git clone https://github.com/2DMAT/odatse-LEED
```

ODAT-SEの入力ファイル

- odatseコマンドはひとつの入力ファイル (input.toml) を引数に取る
- 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
  - Rosenbrock 関数を ([solver])
  - Nelder-Mead 法で最適化する ([algorithm])

```
# 実行方法
$ odatse 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"
```

ODAT-SEの機能 (advanced)

- 探索領域を制限する
  - 線形不等式制約 $Ax + b \geq 0$ を満たす x のみに制限する
  - 制約条件を表すクラスを定義することで任意の制約が可能

- 目的関数 f(x) を定義する空間 x と、探索空間 y をマップする
  - アフィン写像による変換
  - 任意の写像を定義して変換も可能

- 探索の中断・再開
  - チェックポイントを定期的に設けて、計算途中の状態を保存
  - 実行が中断した箇所(直前のチェックポイント)から再開(resume)
  - 反復計算をもう少し続けたい場合の継続実行(continue)も対応

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

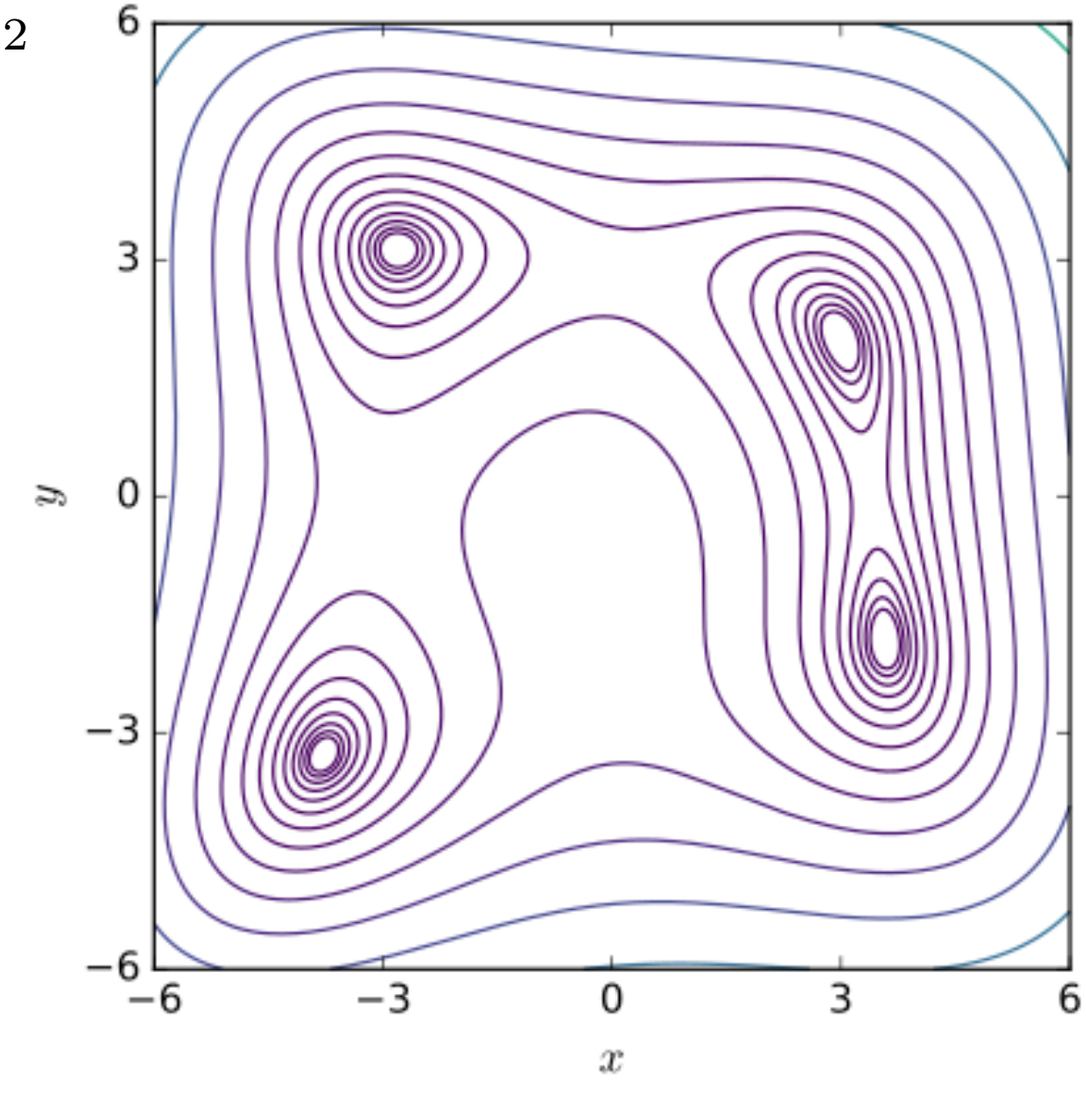
- レプリカ交換法 (exchange)
- ポピュレーションアニーリング (pamc)

解析関数の最小化問題を例に

- 以下ではHimmelblau関数をデモンストレーションに用いる

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

- 右図は$\log f$の等高線プロット
- 4つの最小点$f = 0$を持つ
- ODAT-SE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

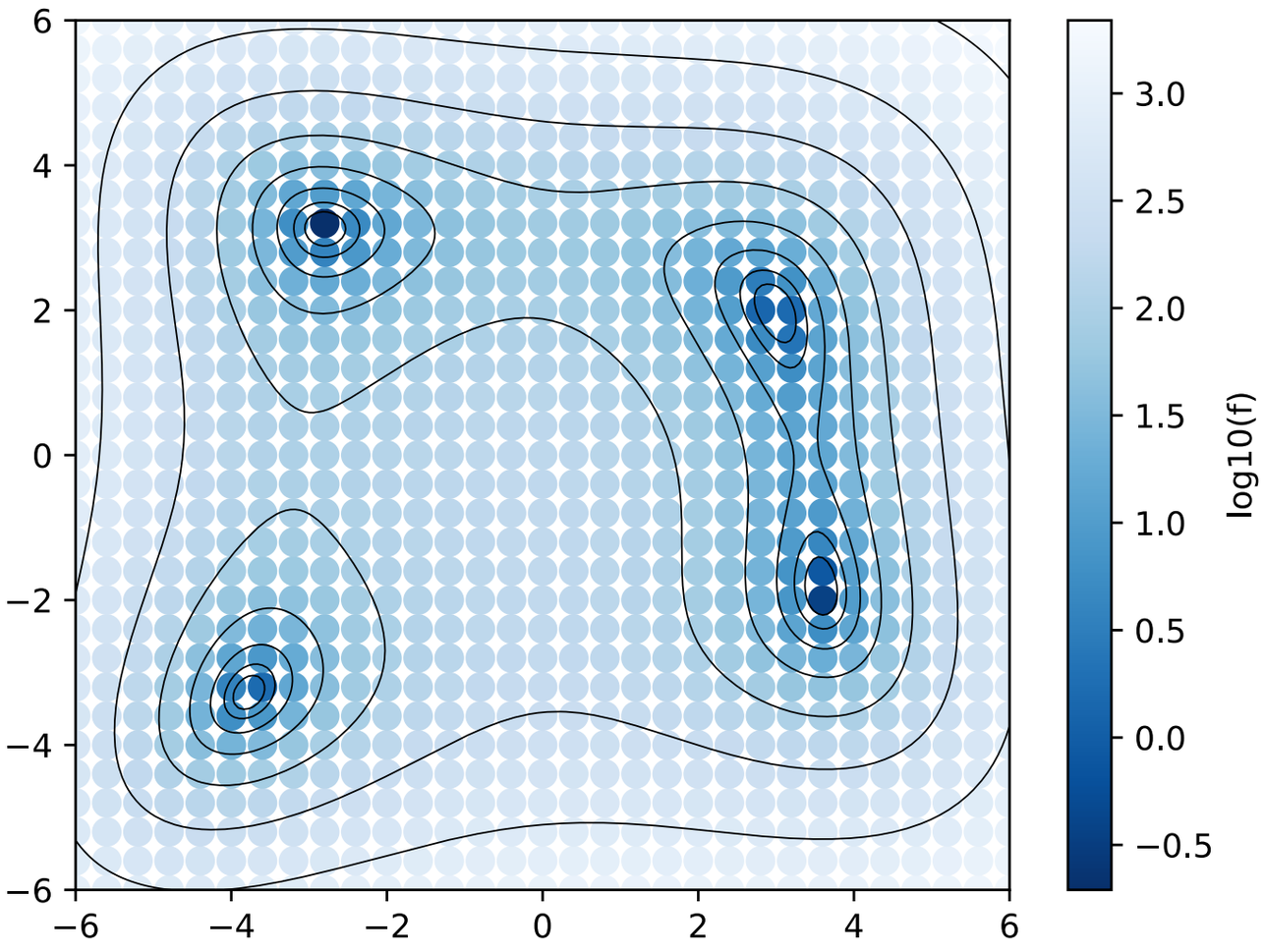


By Nicoguaro - Own work, CC BY 4.0,
https://commons.wikimedia.org/w/index.php?curid=51597574

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（後述）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）も指数的に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

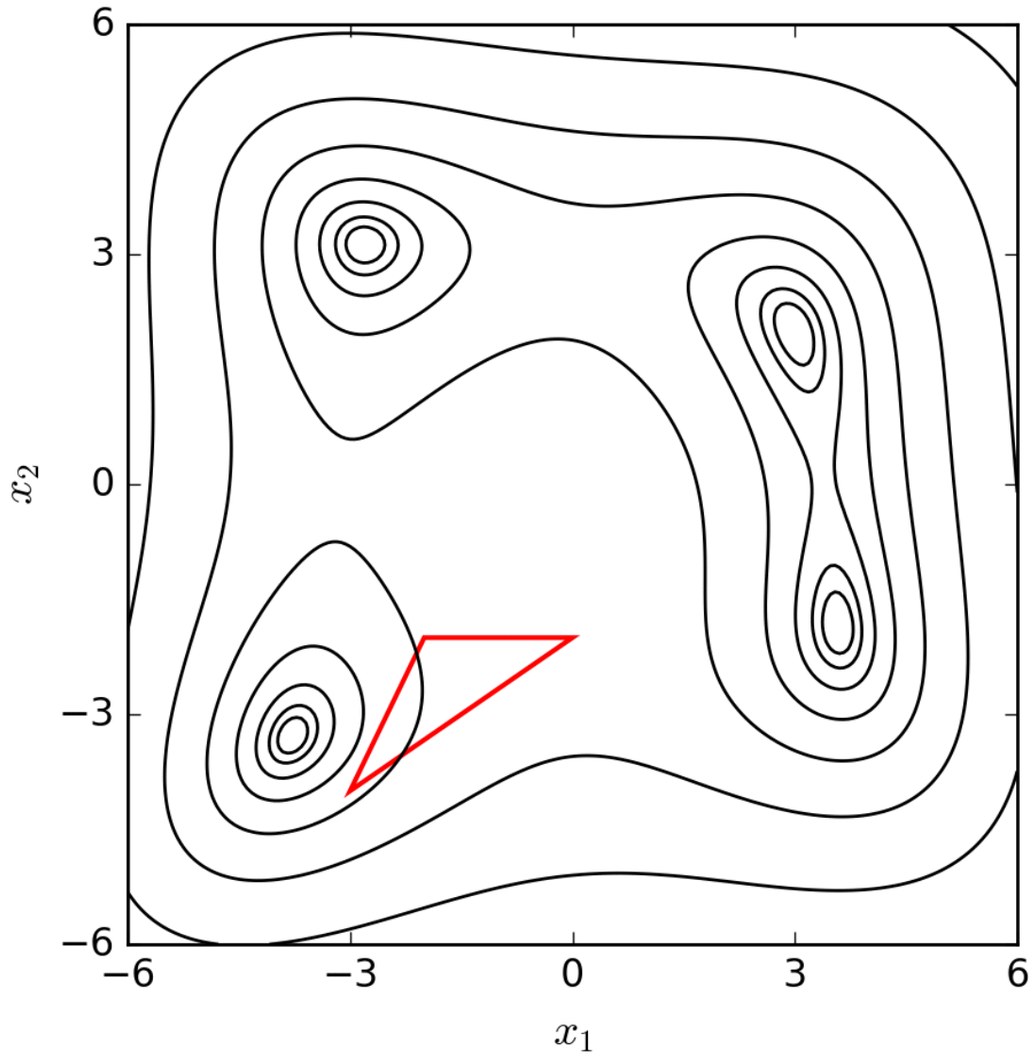


入力

```
[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
function_name = "himmelblau" # 関数名
```

Nelder-Mead法

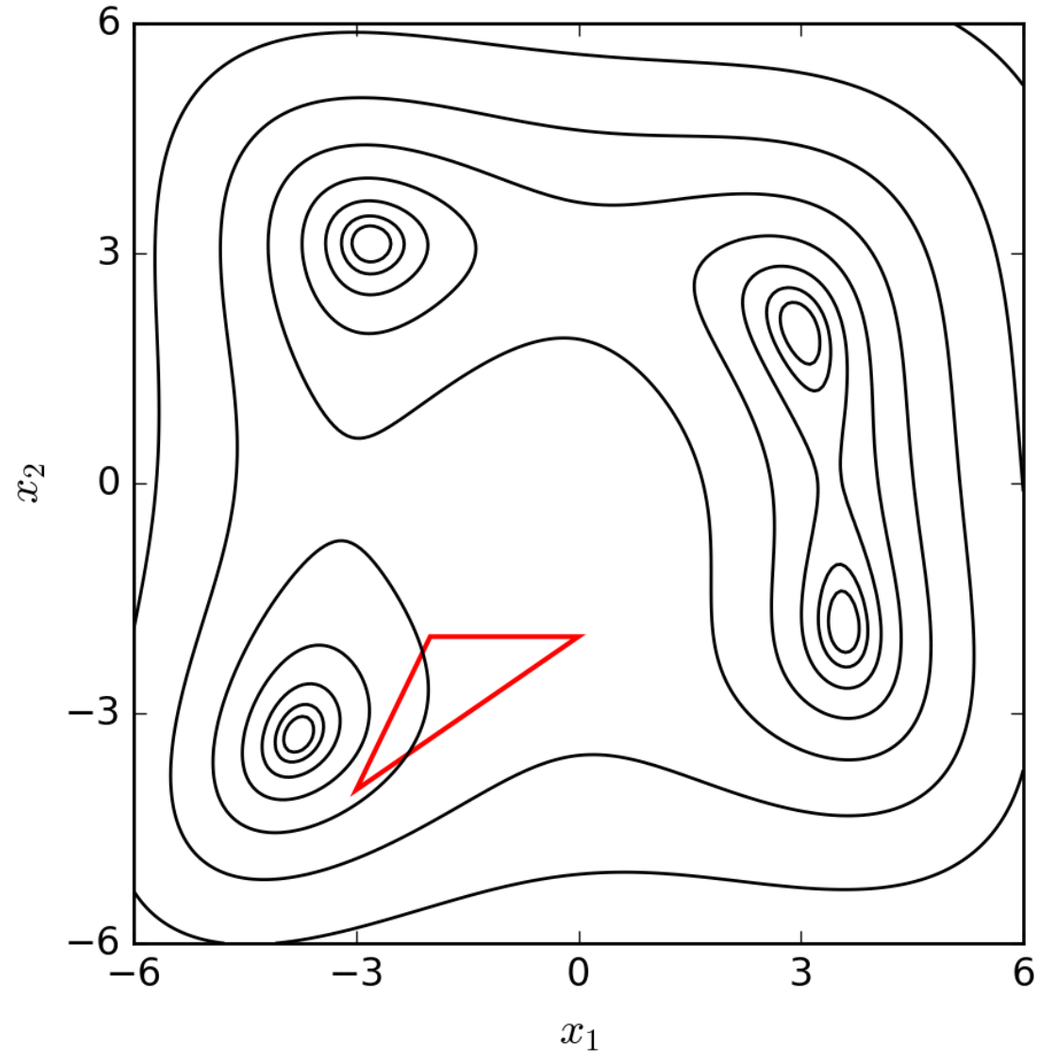
- Downhill Simplex法、アメーバ法などとも呼ばれる
- 連続空間の最適化手法
- 必要なのは目的関数の値 f(x)のみ (導関数は不要)
- D次元空間の最適化をする場合、D+1個の点からなる単体 (三角形、四面体、...) を少しずつ動かして谷を下っていく



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

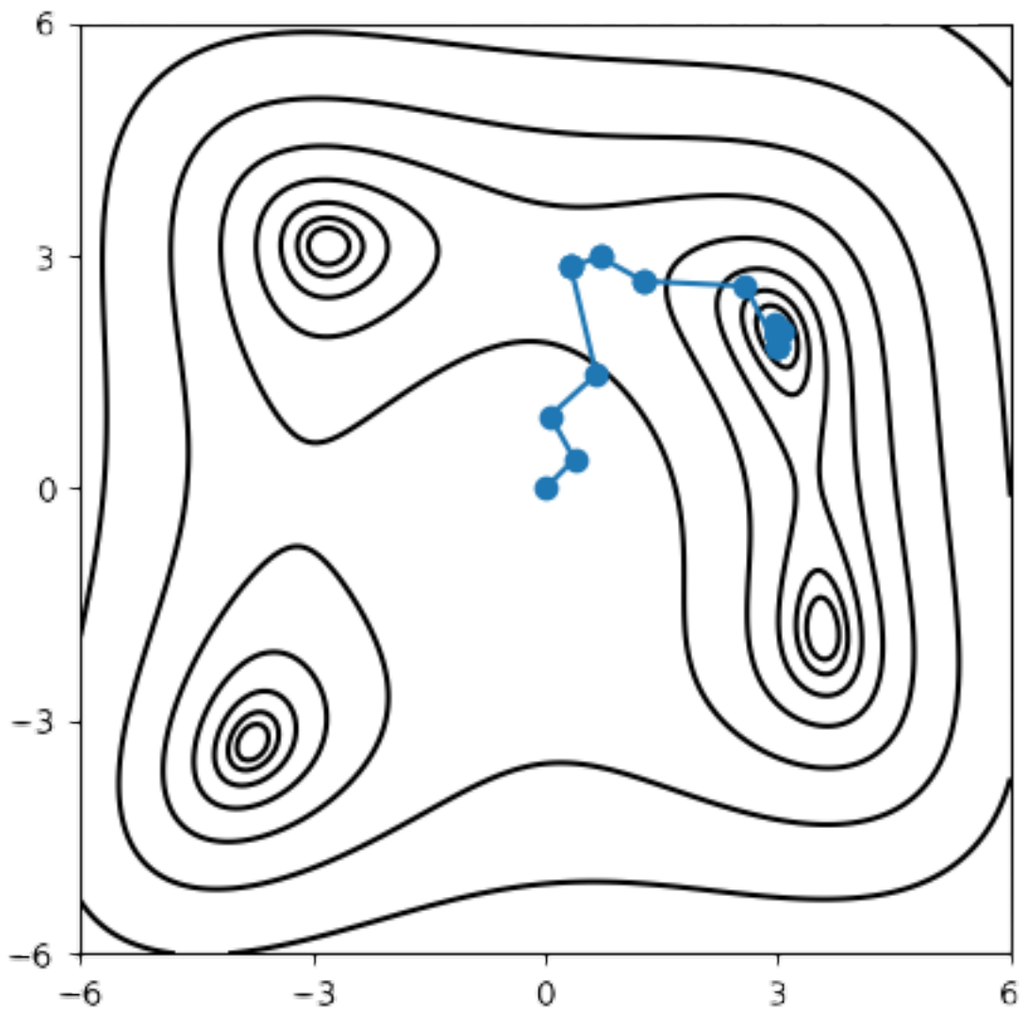
Nelder-Mead法

- 結果は初期配置に依存する
  - 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下のURLを参照してください
- 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



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

Nelder-Mead法のデモ



原点から始めたら右上に到達した

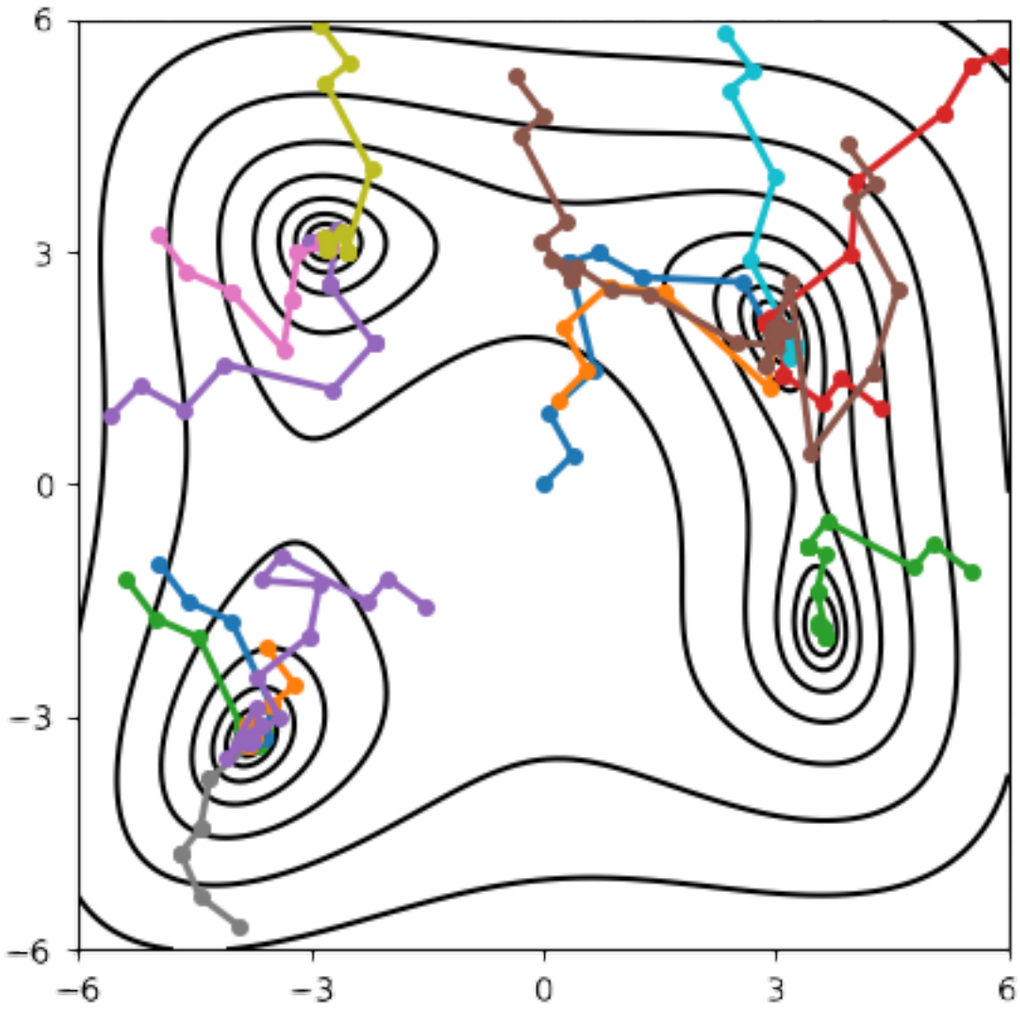
入力

```
[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 = 4.227837036199490 4e-08
x1 = 2.999966956295017 5
x2 = 1.999997338933622 5
```

Nelder-Mead法のデモ



ランダムな初期値からスタート

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

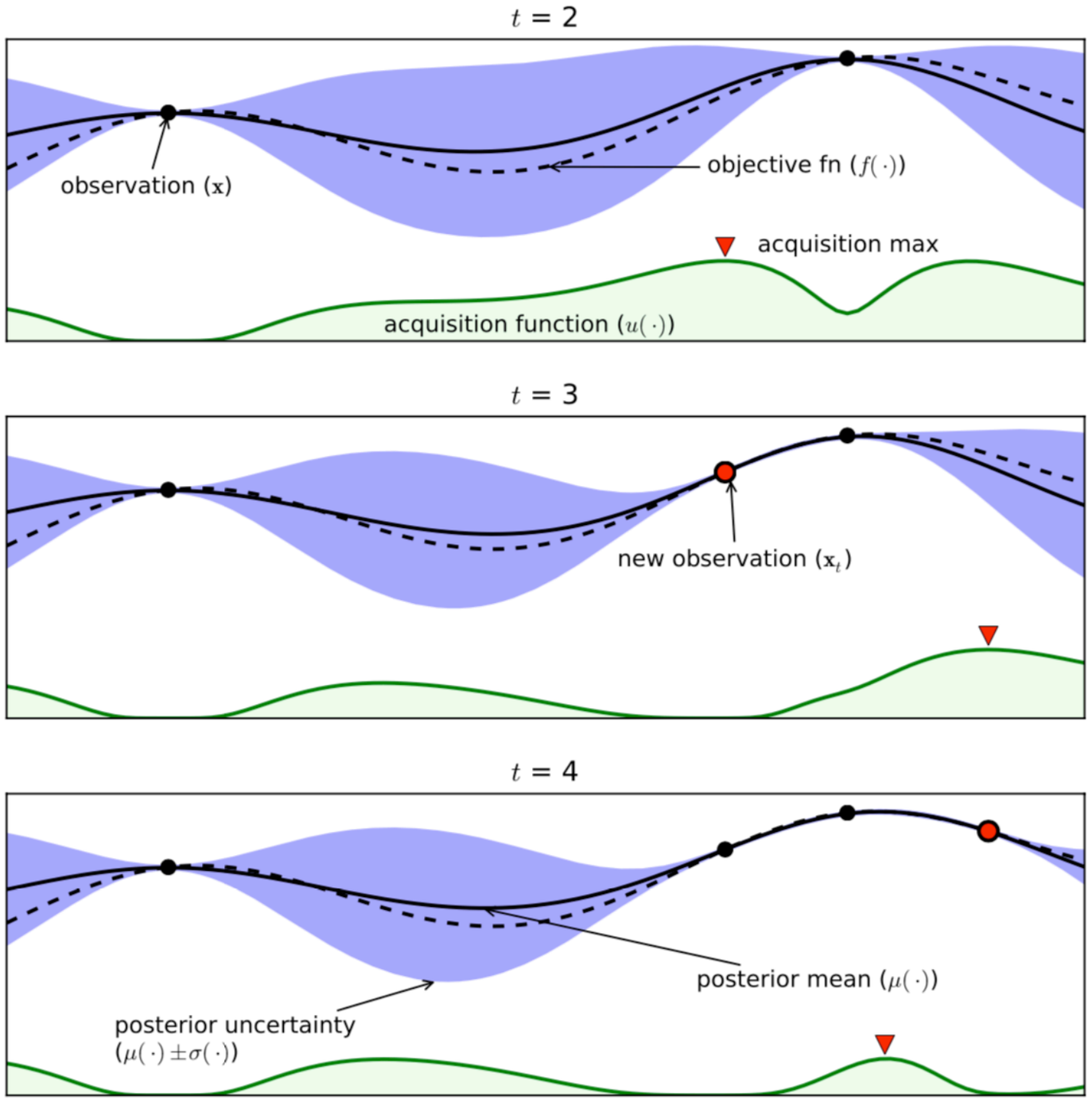
```
fx = 4.227837036199490 4e-08
x1 = 2.999966956295017 5
x2 = 1.999997338933622 5
```

ベイズ最適化

- $f(x)$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

- 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」

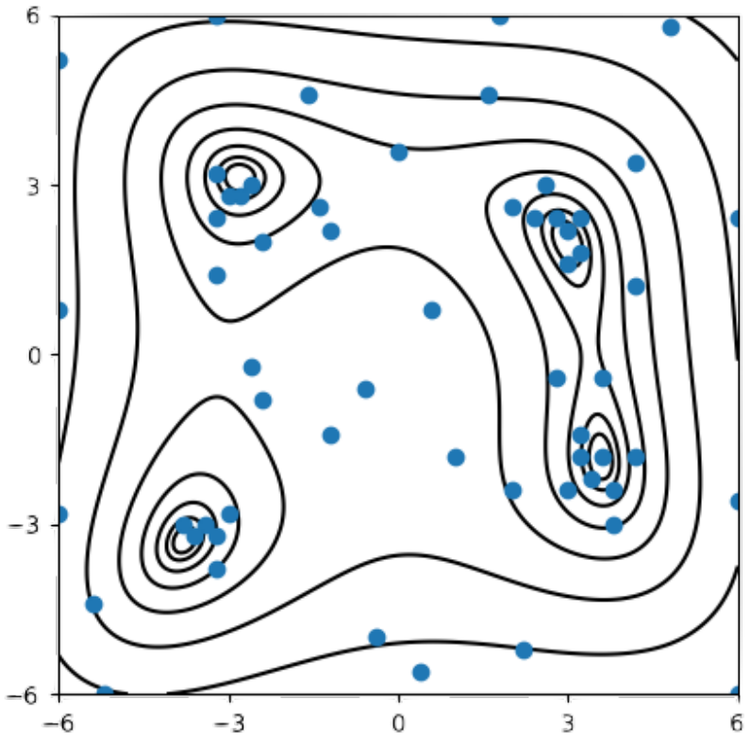


taken from E. Brochu, et al., arXiv:1012.2599

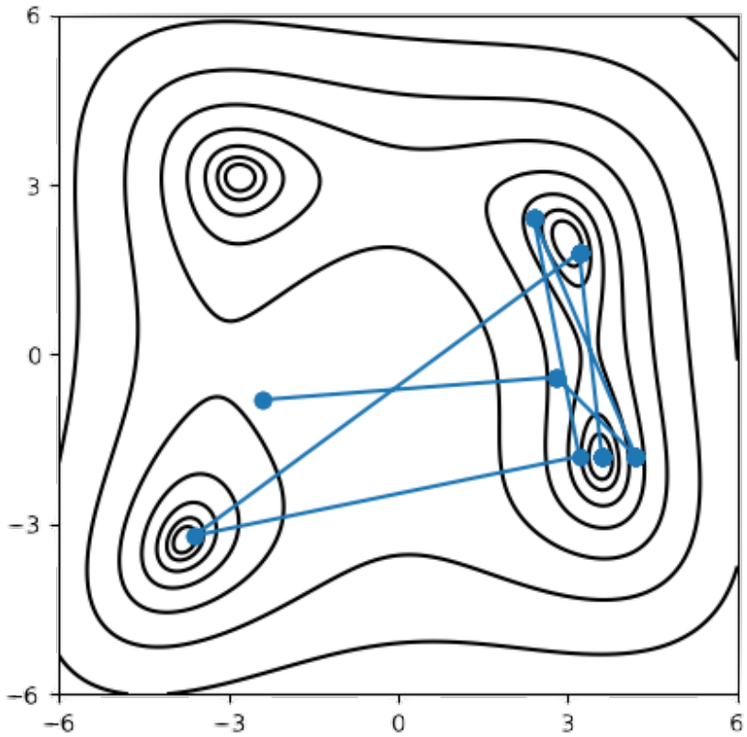
ベイズ最適化ライブラリ PHYSBO

- 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
  - 物性研ソフトウェア開発・高度化プロジェクトで支援
- 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

3600 個の候補から、たかだか数十回の探索でかなり良い解を選択

```
[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"
```

モンテカルロサンプリング

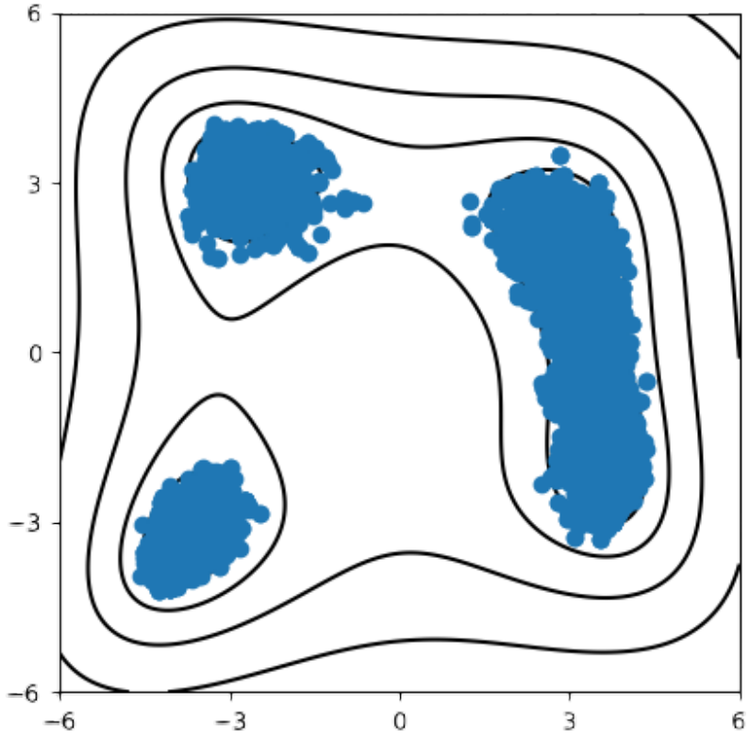
- $f(x)$をできるだけ小さくしたい（大きくしたい場合は$-f(x)$を考える）
  - $f(x)$の増加に対して単調増加する関数$w(x)$を導入し、頻度分布が$w(x)$に従うようにxの系列を確率的に生成する
  - つまり、$f(x)$が小さいxほど出やすい
    - 出てきたxのばらつきから結果の「不確かさ」を見積もれる
    - 手法としてはマルコフ連鎖モンテカルロ法を用いる（詳細はスキップ）
- $w(x)$として具体的にはカノニカル分布（ボルツマン分布）を用いる
  - すなわち、目的関数 $f(x)$ を「エネルギー」とみなし、「温度」T のもとで $w(x) = \exp[-f(x)/T]$ 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

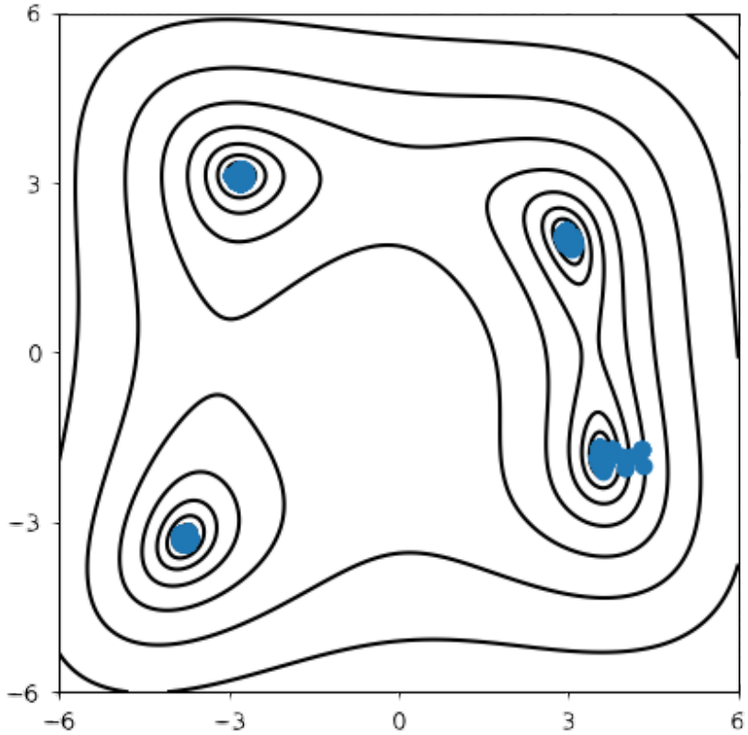
K. Hukushima and K. Nemoto, J. Phys. Soc. Jan.. **65**, 1604 (1996)

- 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
    - 高温極限では重みwが一定になる
  - 温度が低すぎると局所解 (穴) から出てこられなくなる
- 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

レプリカ交換モンテカルロ法のデモ



T=10



T=0.1

f が小さいところを重点的に
サンプリングできている

最初の20点は除外してある（初期緩和）

```
[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"
```

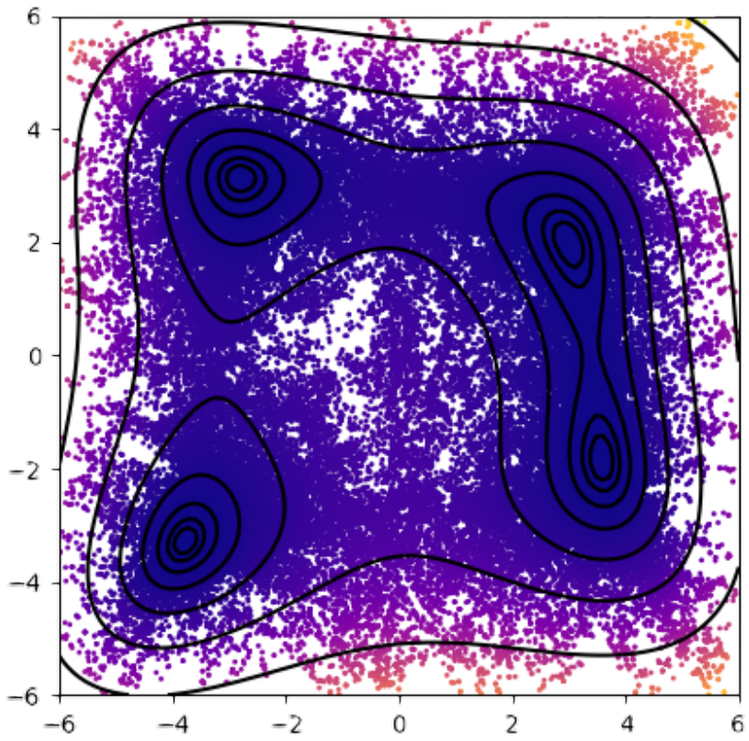
```
$ mpiexec -np 4 odatse input.toml
```

ポピュレーションアニーリング法

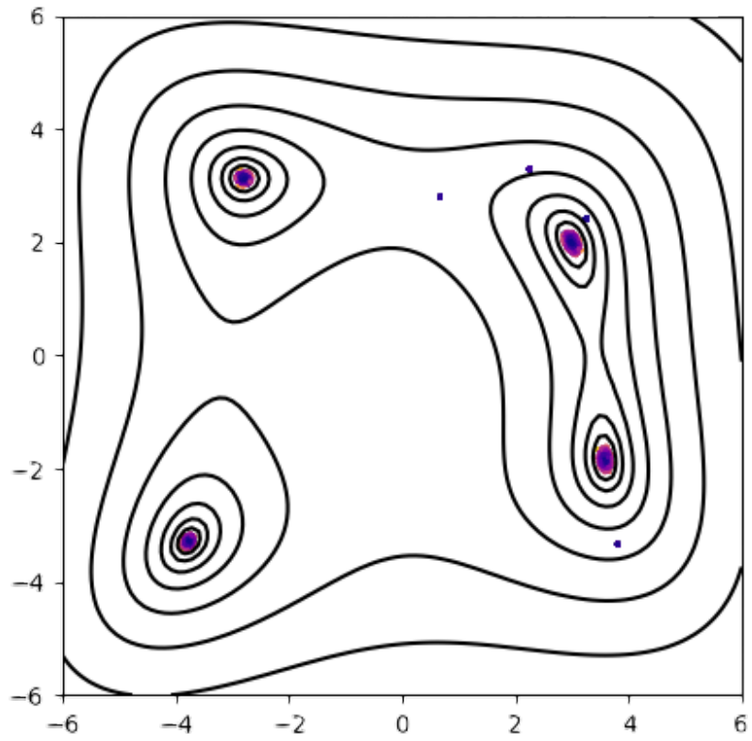
K. Hukushima and Y. Iba, AIP Conf. Proc. **690**, 200 (2003)

- 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

ポピュレーションアニーリング法のデモ



T=10.0



T=0.1

```
[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"
```

```
$ mpiexec -np 24 odatse input.toml
```

Solver

順問題・物理モデル

Python 関数

- デモンストレーション関数 (analytical)
- Python関数ラッパー (function)

表面ビーム実験解析

- 全反射高速陽電子回折 TRHEPD / 反射高速電子線回折 RHEED (sim-trhepd-rheed)
- 表面X線回折 (sxrd)
- 低速電子線回折 (leed)

自作のSolverを使う

- ODAT-SE では、独自に順問題ソルバーを用意して解析を行えるようになっています
  - 目的関数 f(x) が既にある場合:
    - ODAT-SE の function ソルバーを利用するのが簡単です
    - f(x) は numpy.ndarray 型の引数 x をとり、float 値を返す関数です
  - 複雑なソルバーの場合:
    - ソルバークラスを作成して利用します
    - マニュアルや既存のコードを参照してください
    - モジュールのテンプレートを公開予定

よくあるパターン:

- 外部プログラムを実行してモデル計算を行う
  1. 探索パラメータを含む入力ファイルを生成
  2. プログラムを実行
  3. 結果を出力ファイルから取得
  4. 参照データと比較して目的関数を計算

自作のSolverを使う

- 目的関数 f(x) を組み込む

```
import numpy as np
import odatse
from odatse.algorithm.min_search import Algorithm
from odatse.solver.function import Solver

def booth_function(xs: np.ndarray):
    assert xs.shape[0] == 2
    x, y = xs
    fx = (x + 2 * y - 7)**2 + (2 * x + y - 5)**2
    return fx

info = odatse.Info.from_file("input.toml")

solver = Solver(info)
solver.set_function(booth_function)

runner = odatse.Runner(solver, info)

alg = Algorithm(info, runner)
alg.main()
```

目的関数の例
(Booth関数)

function Solver を利用
set_functionで目的関数をセット

sim-trhepd-rheed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
  - T. Hanada, et al., CPC 277, 108371 (2022)
- odatse-STR 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_{sim}) との距離

$$f(x) = \sqrt{\sum_i \left(D_{\mathrm{exp}}(x_i) - D_{\mathrm{sim}}(x_i) \right)^2}$$

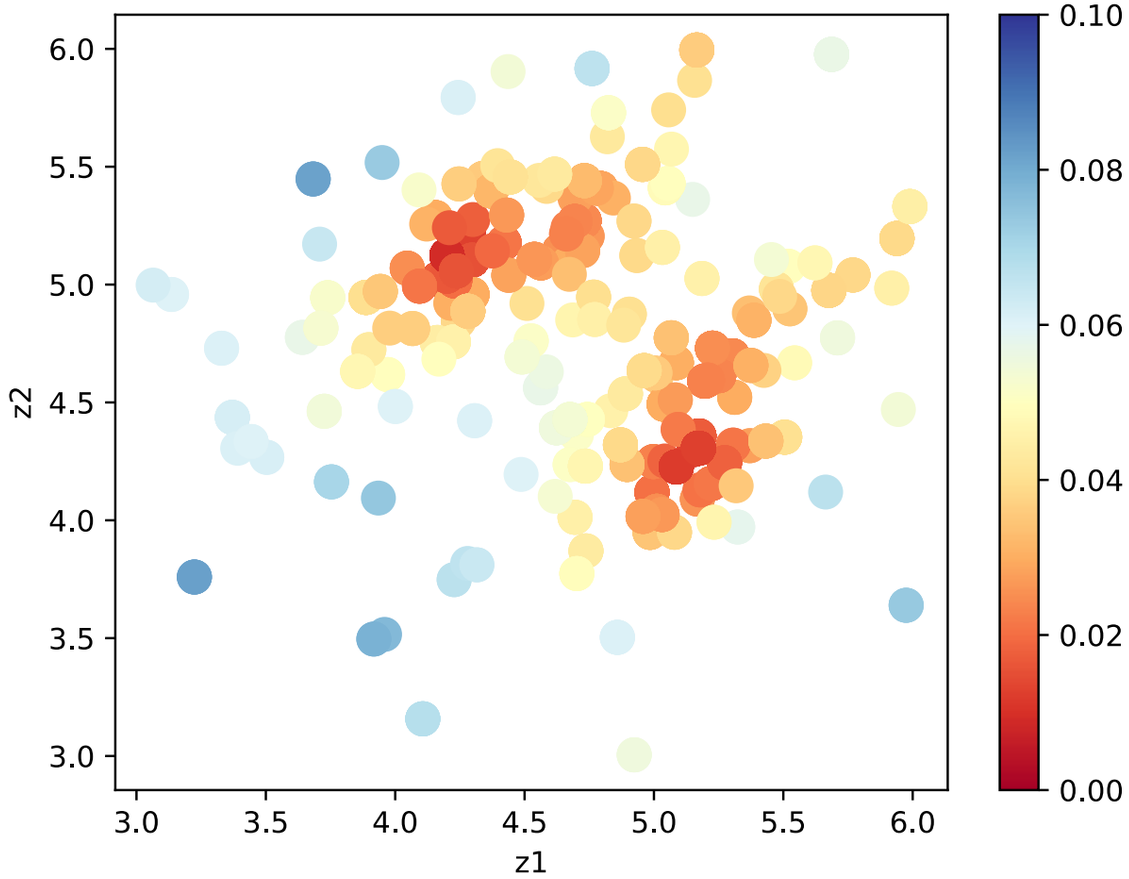
  - ミラー指数の異なる複数の回折ビームを同時にフィッティング可能
- ベイズ最適化や Nelder-Mead などを用いて $f(x)$ を最小化することで、表面原子構造の推定を行える（逆問題）

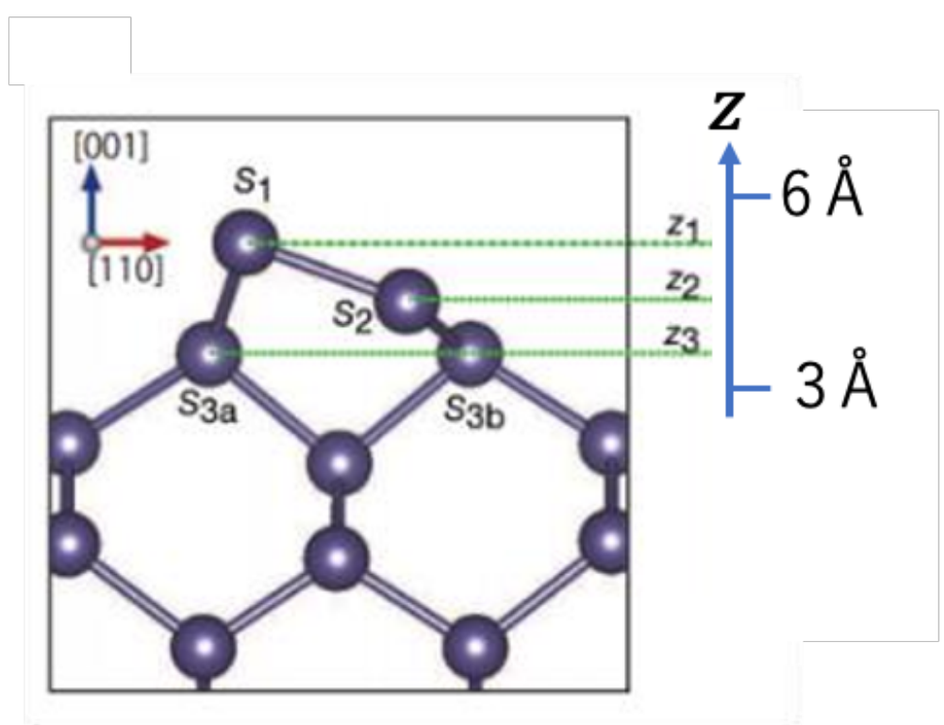
sim-trhepd-rheedとの接続

- sim-trhepd-rheed 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
  - odatse-STR 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
- STRソルバーモジュール内では、
  - ① あるパラメータXの値 (原子配置など) に対して surf.exe を実行し、Rocking曲線を計算する
    - surf.exe の入力ファイルはテンプレートをもとに自動生成
  - ② 計算で得られたRocking曲線と実験データのRocking曲線との差 (R-factor) を求めて、目的関数 f(X) として返す
- 詳細はSTR と odatse-STR のマニュアルや、本スライドの付録を参照

解析の実行例 (レプリカ交換モンテカルロ)

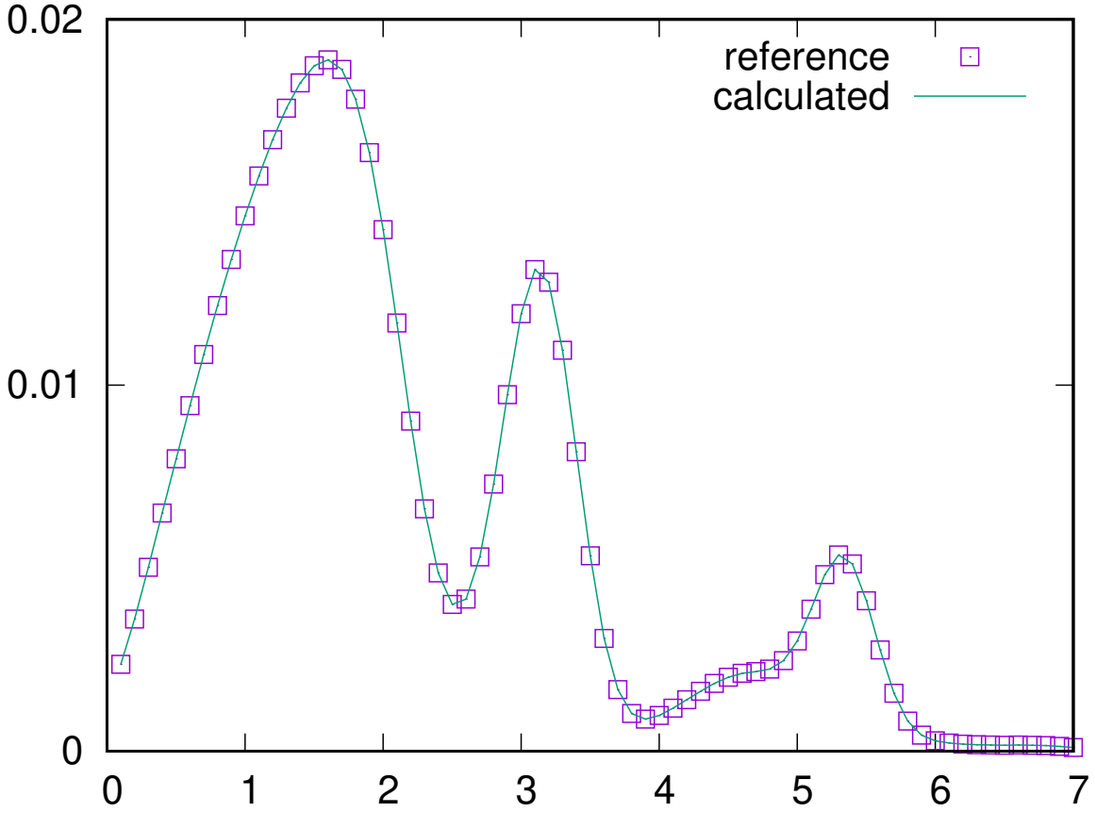
- Ge(001)-c4x2 表面の2つの Ge 原子の z 座標 (z1,z2)をサンプリングする例
  - 「実験データ」としては適当な原子座標に対してSTRで計算した人工データを使用
  - この例では z1, z2 は交換に対して対称

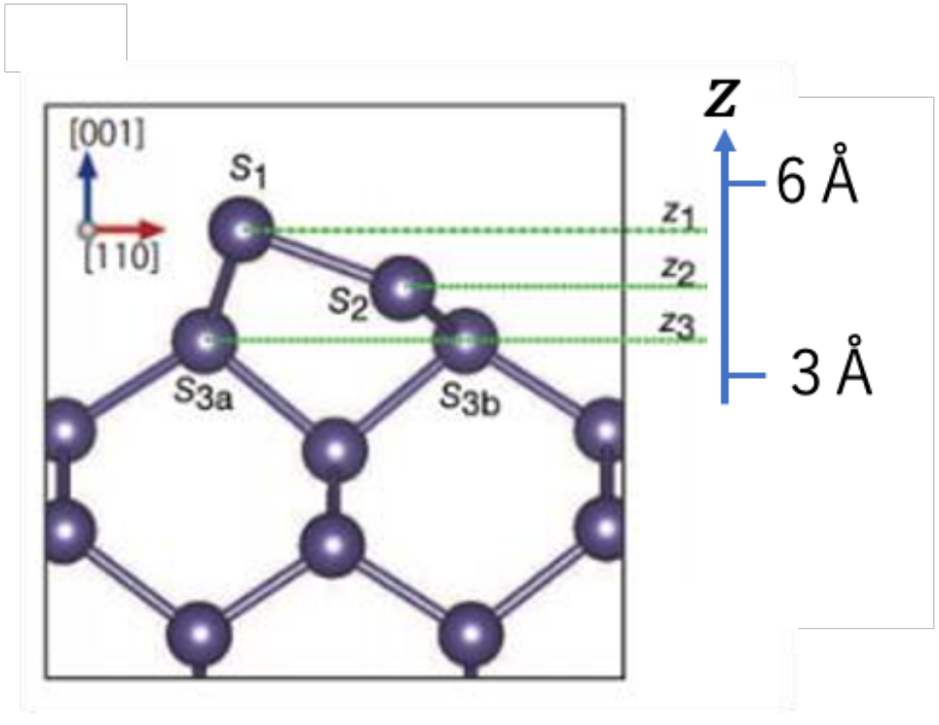




解析の実行例 (Nelder-Mead法)

- Ge(001)-c4x2 表面の2つの Ge 原子の z 座標 (z1,z2)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
- Nelder-Mead で最適化した座標における rocking 曲線





sxrdcalとの接続

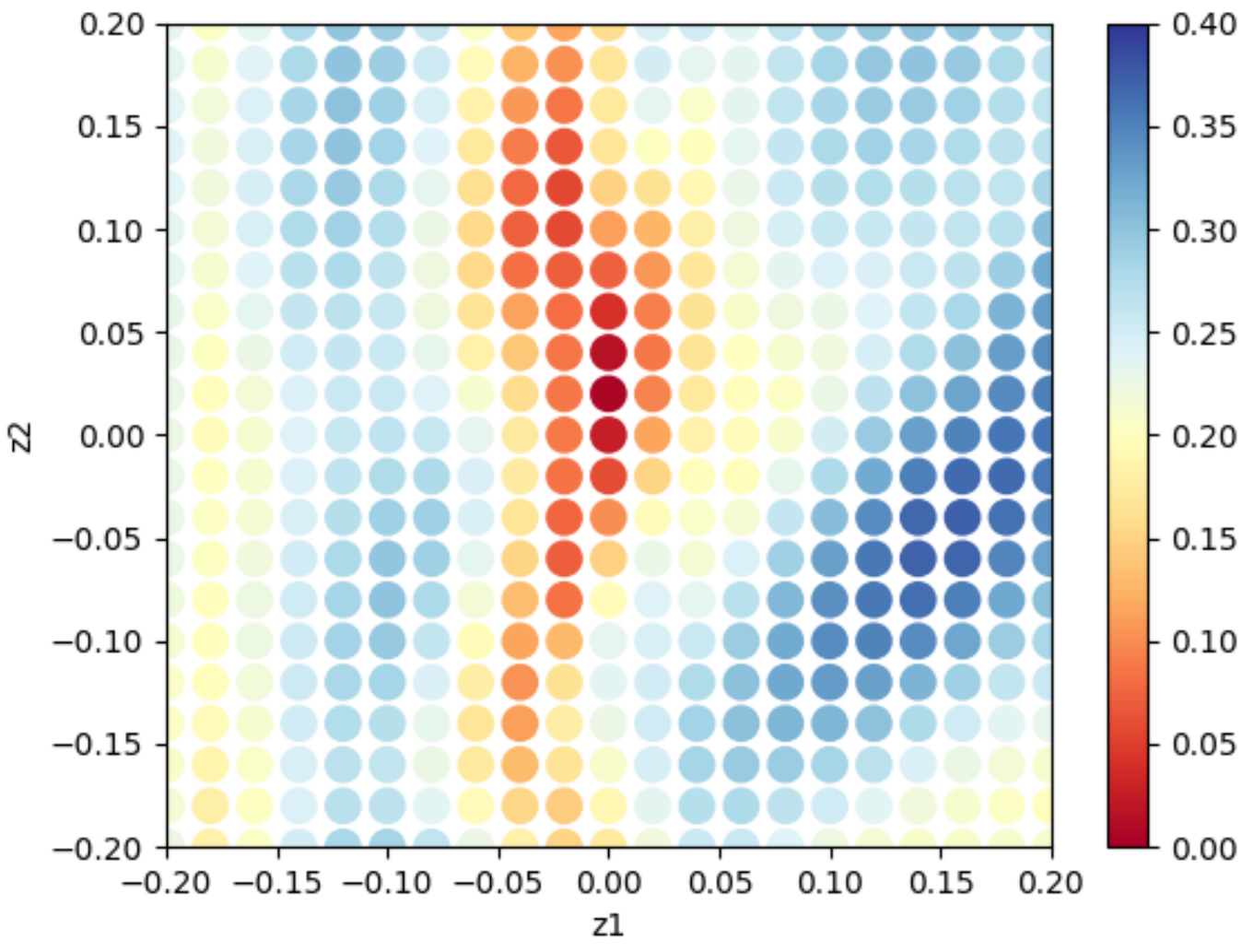
- sxrdcalc は 表面X線散乱(SXRD) において、原子座標から構造因子を計算したり、散乱データにあうように原子座標を最適化したりするソフトウェア
  - https://github.com/sxrdcalc/sxrdcalc
  - by Wolfgang Voegeli 氏@学芸大
- odatse-SXRD は sxrdcalc を用いた Solver を提供している
  - 入力xは原子座標
  - 目的関数$f(x)$は、（実験などで）別に用意した構造因子 (D_{exp}) と 理論計算によるもの (D_{sim}) との距離
- ベイズ最適化や Nelder-Mead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

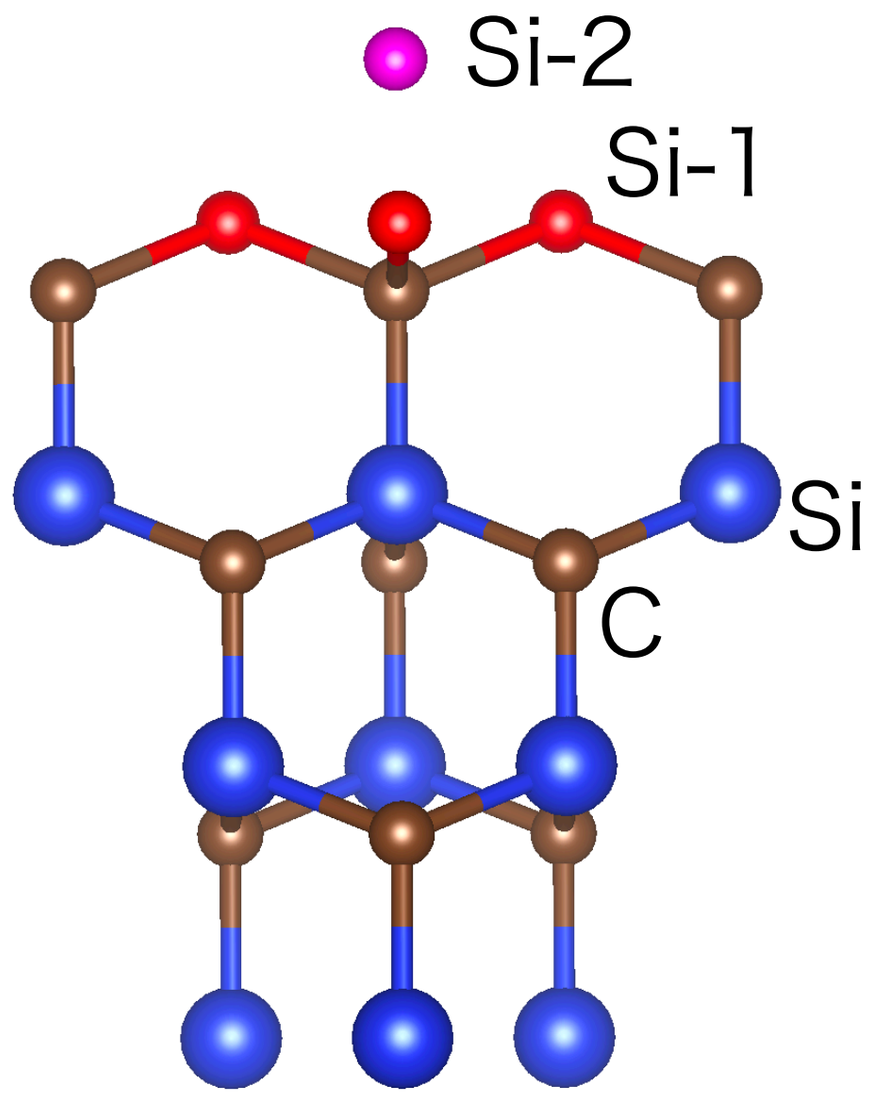
sxrdcalとの接続

- sxrdcalcでは表面X線回折の構造因子やR-factor を計算できる
- odatse-SXRD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 odatse-SXRD 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 odatse-SXRD のマニュアルや、本スライドの付録を参照

解析の実行例 (グリッド探索)

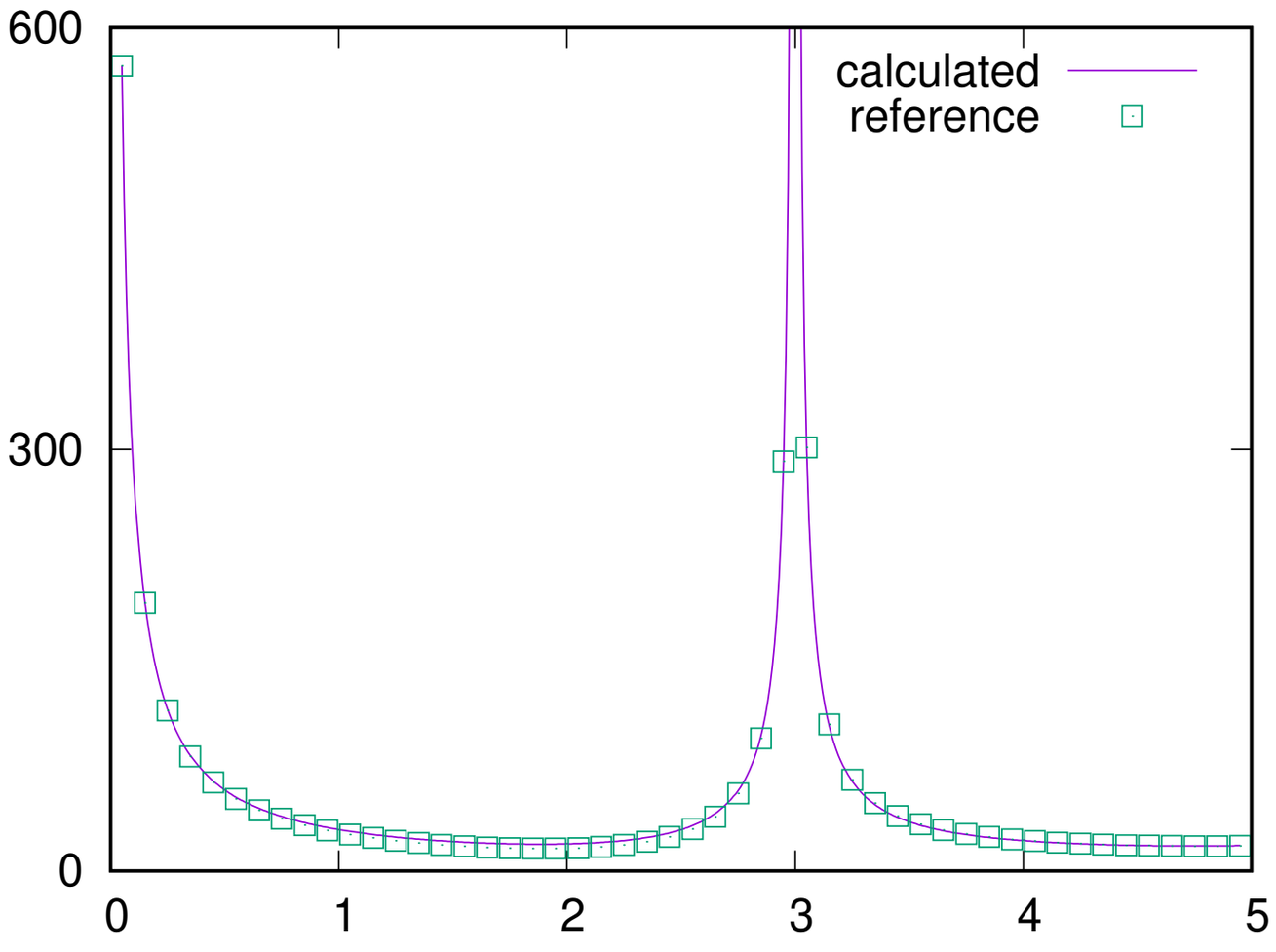
- SiC(111)-r3xr3 表面の2種類の Si 原子 (Si-1, Si-2)のz 座標 (z1,z2)に対するグリッドサーチ

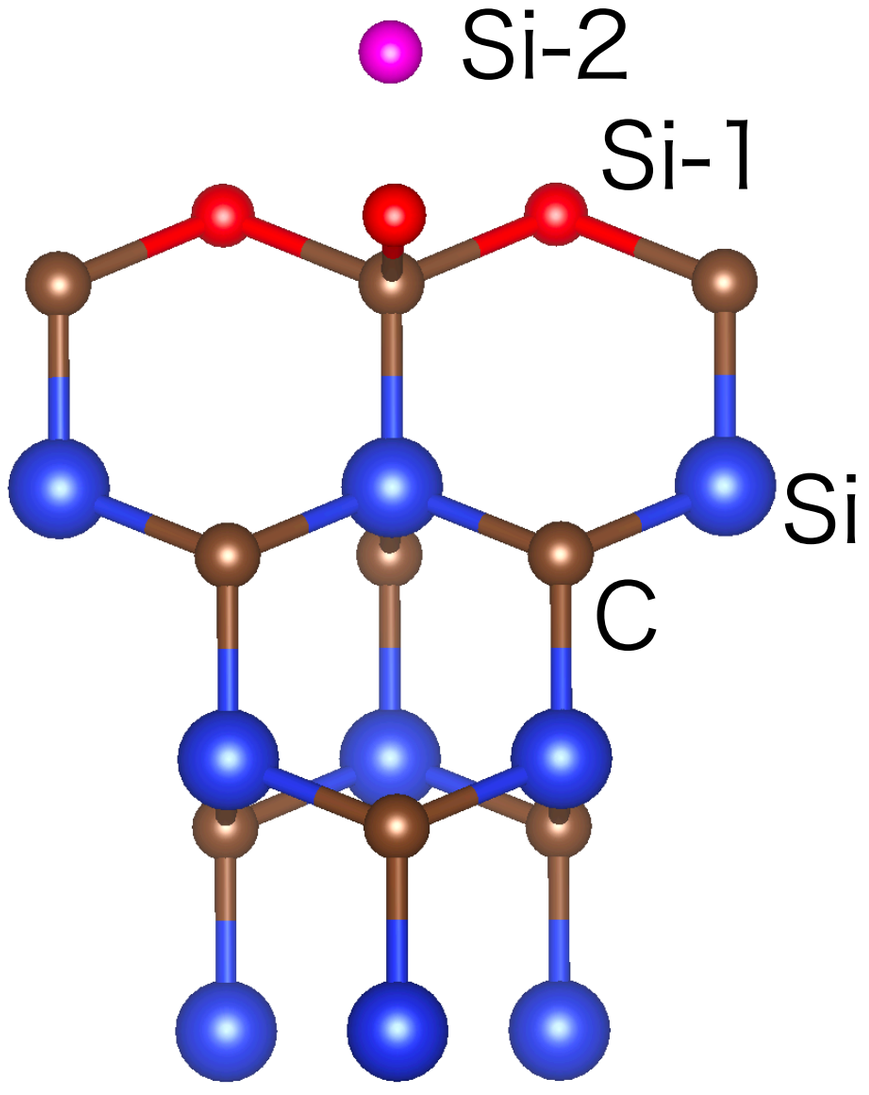




解析の実行例 (Nelder-Mead法)

- 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
- odatse-LEED 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 I-V曲線 (D_{exp}) と 理論計算によるもの (D_{sim}) 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、odatse-LEED 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

まとめと展望

- 2DMAT → ODAT-SE
  - 一般的な最適化・サンプリングのためのフレームワークです
  - 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
  - 論文: Comp. Phys. Commun. 280, 108465 (2022)
    - 2DMAT + 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

sim-trhepd-rheed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
  - odatse-STR 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
  - 表面の解析にあたって不変な情報を指定
- 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）

sim-trhepd-rheed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)
- odatse-STR 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
  - 入力パラメータの特定の要素 (ある原子のz 座標など) を value_01 などの文字列に置き換える
  - odatse-STR 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)
```

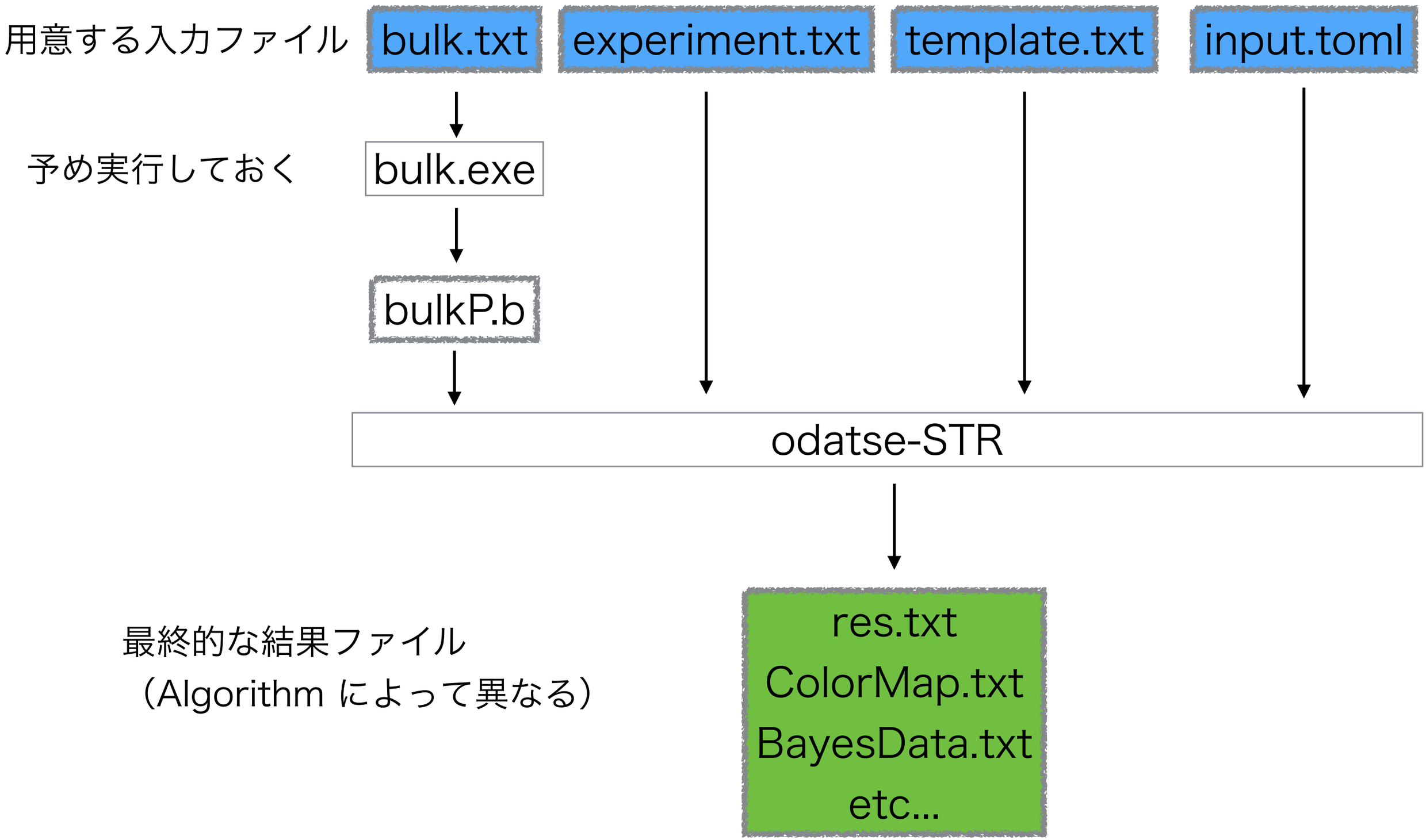
- surf.exe が出力する surf-bulkP.s から rocking curve を読み取る

sim-trhepd-rheedとの接続

- 入力ファイル例

```
[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（省略時: template.txt）
calculated_first_line = 5              # surf-bulkP.s 中、D(x) の最初の行番号
cal_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

[solver.reference]
path = "experiment.txt"     # 実験データ Dexp が書いてあるファイル
reference_first_line = 1    # Dexp として使う部分の最初の行番号（省略時: 先頭行）
reference_last_line = 70    # Dexp の最後の行番号（省略時: 最終行）
exp_number = 1              # Dexp の値が入っている列番号
```

odatse-STRの実行フロー



最終的な結果ファイル
（Algorithm によって異なる）

付録 B

sxrdcalc

sxrdcalcとの接続

- sxrdcalc で表面原子座標を最適化する際には、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r は

```
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
```

$$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \alpha_1 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

    - α1 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）
- odatse-SXRD 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
```

odatse-SXRDの実行フロー

用意する入力ファイル

バルク

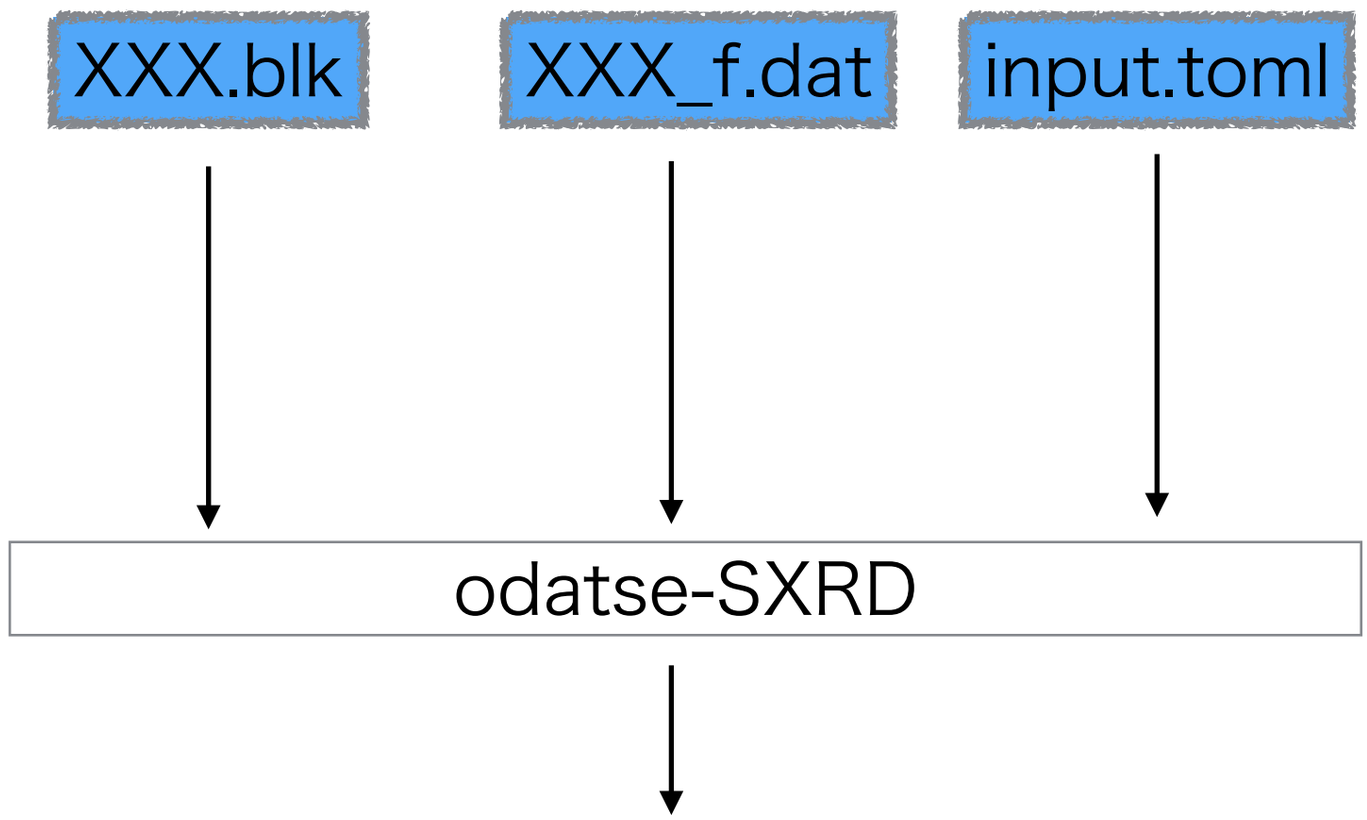
XXX.blk

参照データ
(構造因子)

XXX_f.dat

odatse-SXRD の入力

input.toml



最終的な結果ファイル
（Algorithm によって異なる）

res.txt
ColorMap.txt
BayesData.txt
etc...

49