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

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

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

Yuichi Motoyama (ISSP)

2021-04-20
for 2DMAT ver 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

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**

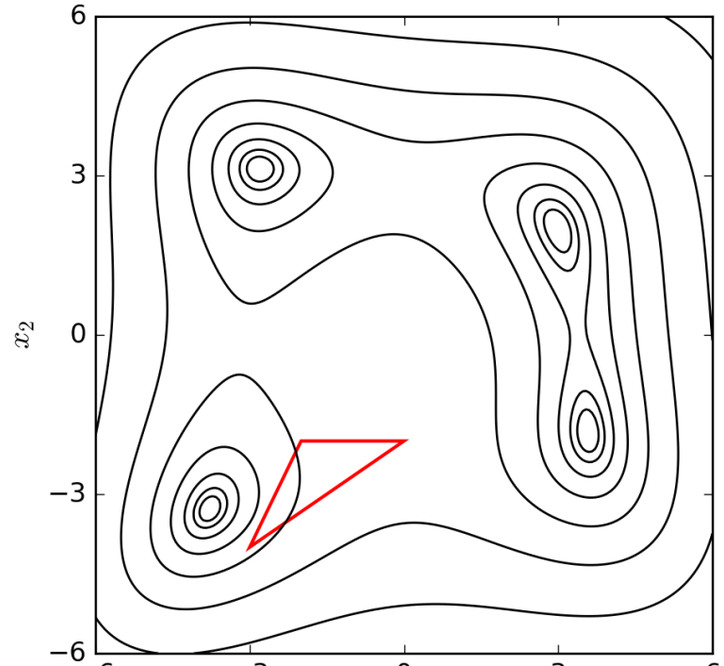
py2dmat の入力ファイル

- py2dmat コマンドはひとつの入力ファイルを引数に取る
- 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
  - Rosenblock 関数を ([solver])
  - Nelder-Mead 法で最適化する ([algorithm])

```
$ 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"


# 実行方法
$ py2dmat input.toml
```

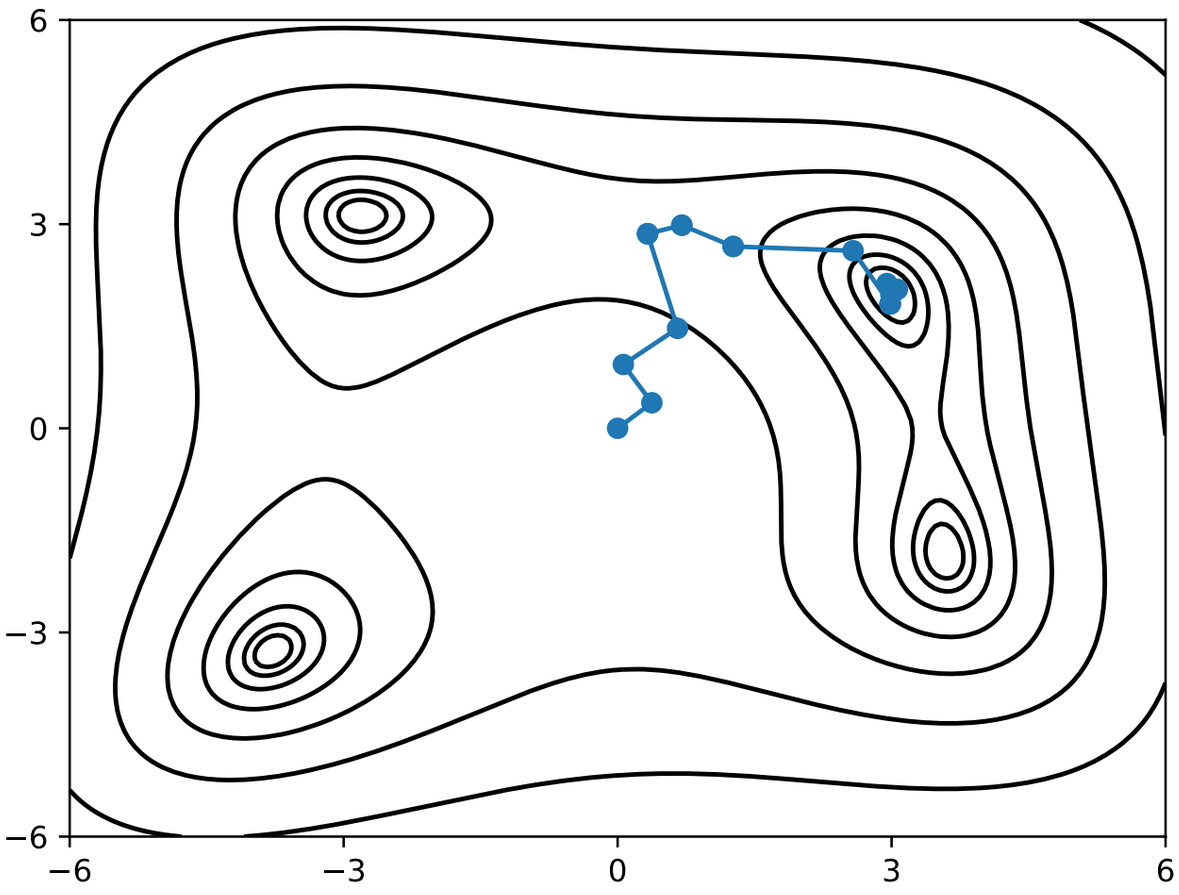
Nelder-Mead(a.k.a Simplex downhill)

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

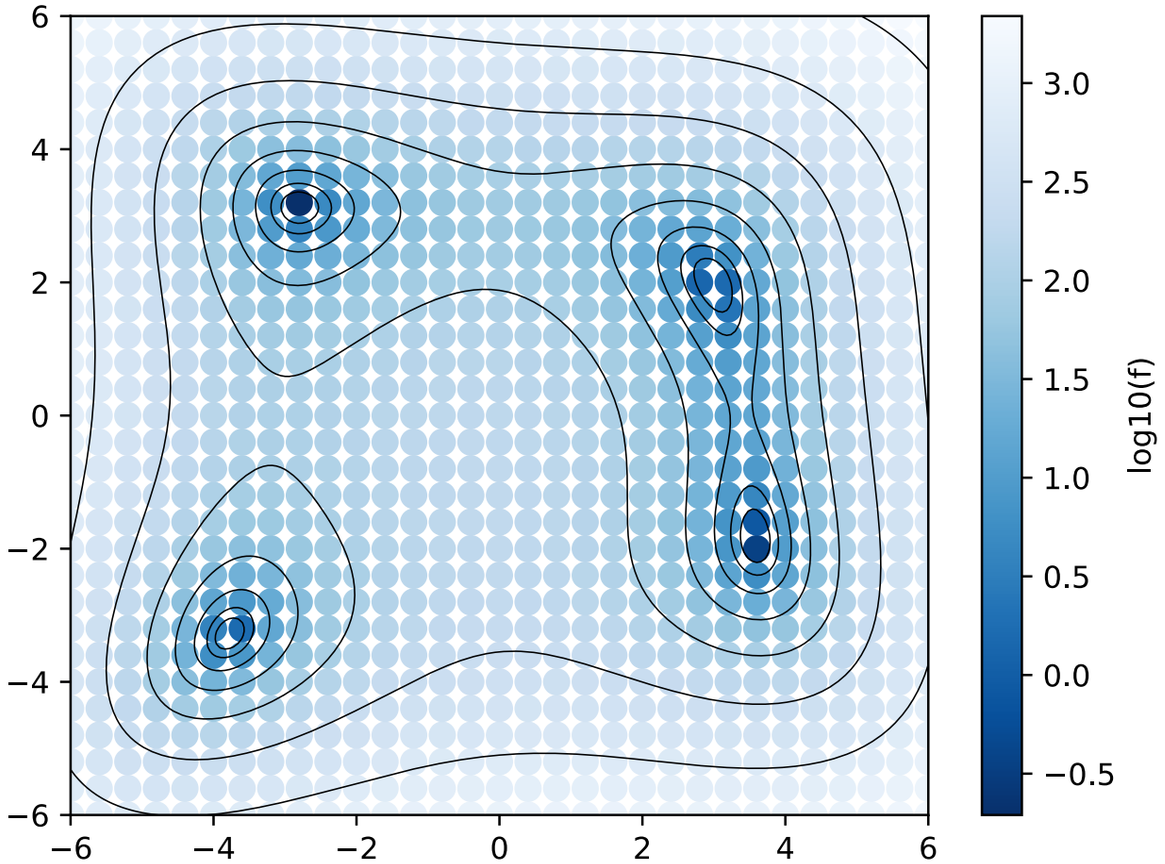
```
fx = 4.227837036199490 4e-08
x1 = 2.999669562950175
x2 = 1.999973389336225
```

原点から始めたら右上に到達した

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
```

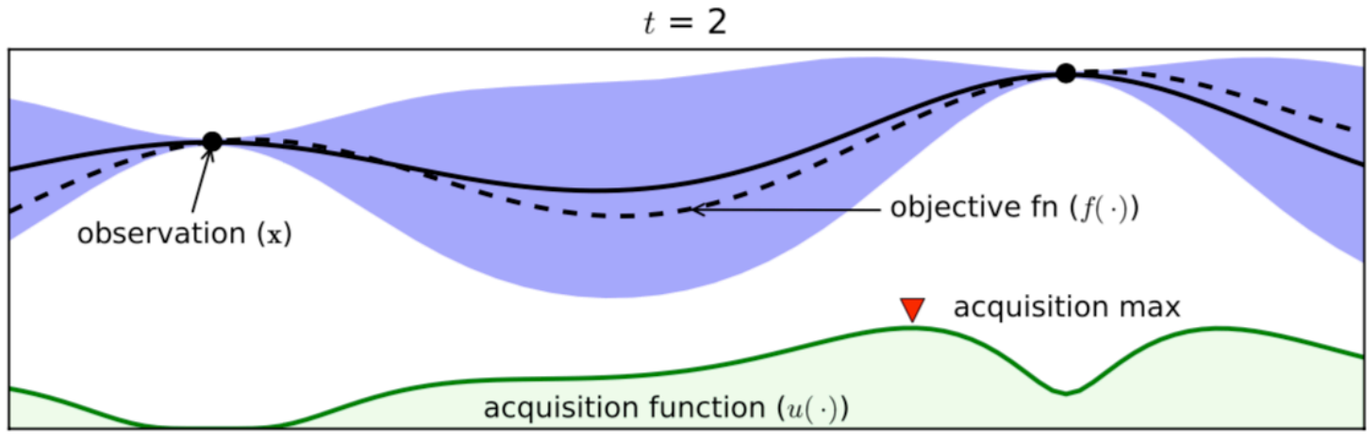
ベイズ最適化(Bayesian Optimization)

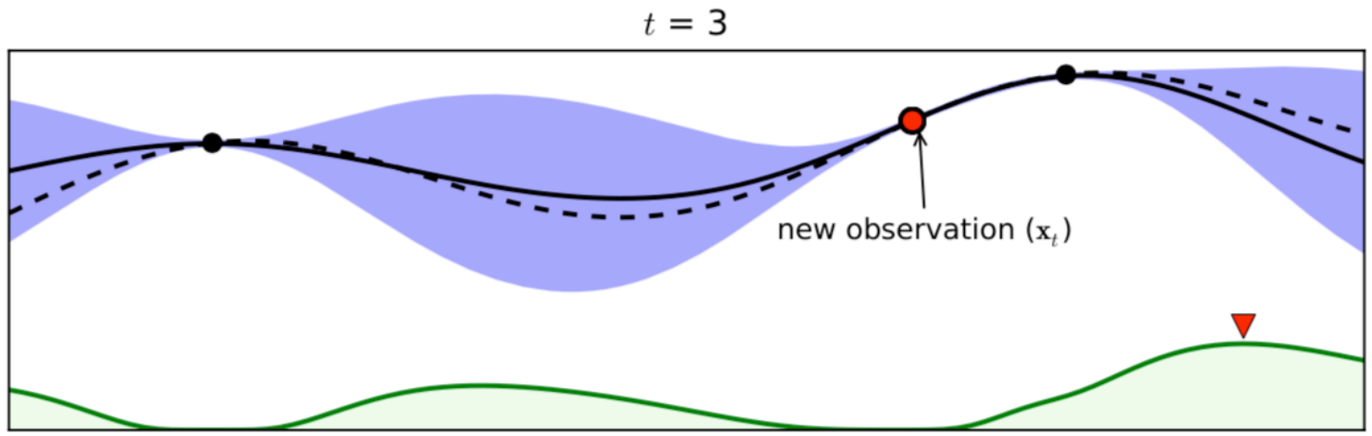
- f(x) の評価が難しい（コスト高）であるときに特に有用な最適化手法
- 目的関数 f(x) を、計算しやすいモデル関数 g(x) で近似する
  - x_i を数点適当にサンプルし、 y_i=f(x_i) を計算
  - (x_i, y_i) のセットを用いて g(x) を学習する（フィッティングする）
  - g(x) を最小化・最大化するような点 x' を次の観測点として、 y'=f(x') を計算し、 g(x) を再学習する
  - 適当な回数繰り返す
- 一般的にはg(x) として、ガウス過程を事前分布とした事後分布を用いる
  - g(x) という関数の分布をベイズ推定するという意味で「ベイズ」最適化

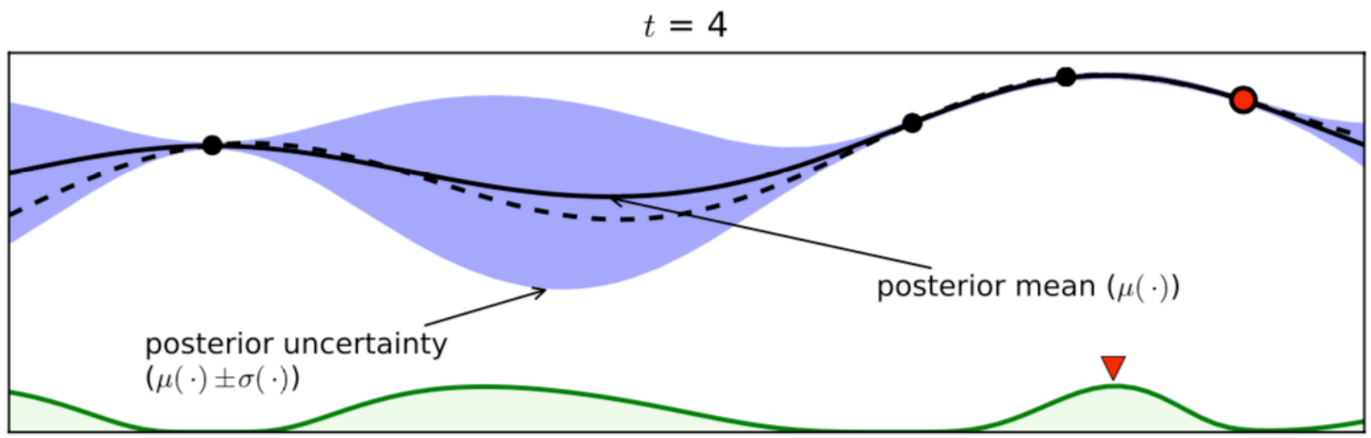
ベイズ最適化の流れ

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」





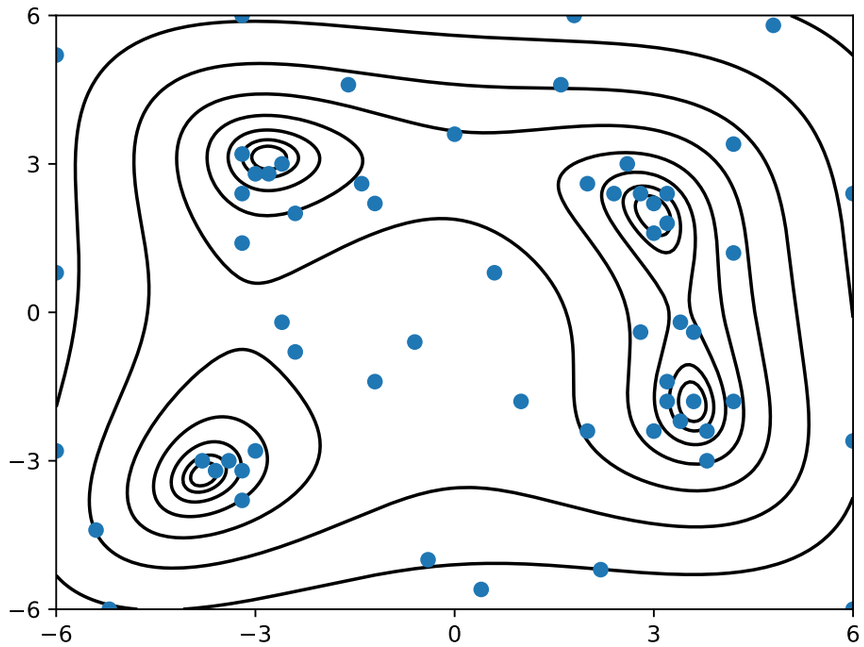


ベイズ最適化ライブラリPHYSBO

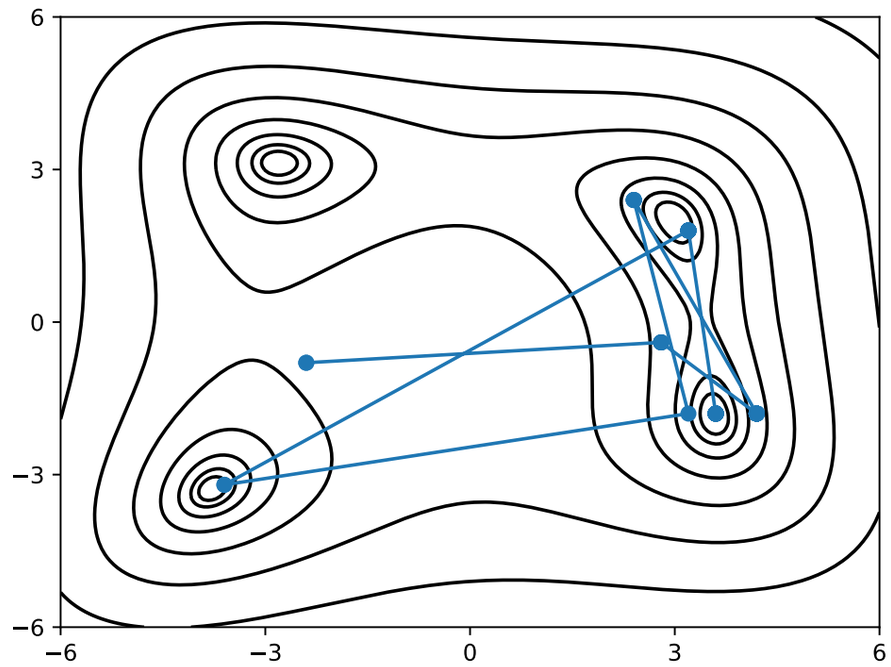
- py2dmat はベイズ最適化に PHYSBO ライブラリを利用している
  - https://www.pasums.issp.u-tokyo.ac.jp/physbo/
- 東大津田研で開発されているCOMBO というライブラリの派生
- py2dmat と同じく物性研ソフトウェア高度化プロジェクトで支援された
  - 今年中にPHYSBO の講習会もやる予定です
- 探索空間は離散化されている（最初に離散化した点の集合を与える）
  - 解像度や探索コストをコントロール可能
    - 気になる場合は最適化結果を初期値にして Nelder-Mead をする
  - 獲得関数の計算で並列計算可能
    - 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
  - 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 をどうやって設定するかが問題（次ページ）

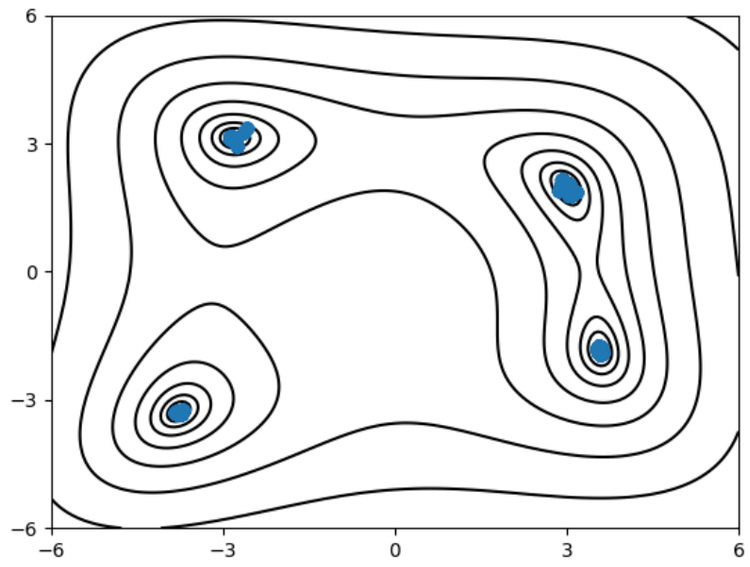
レプリカ交換モンテカルロ法

- 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

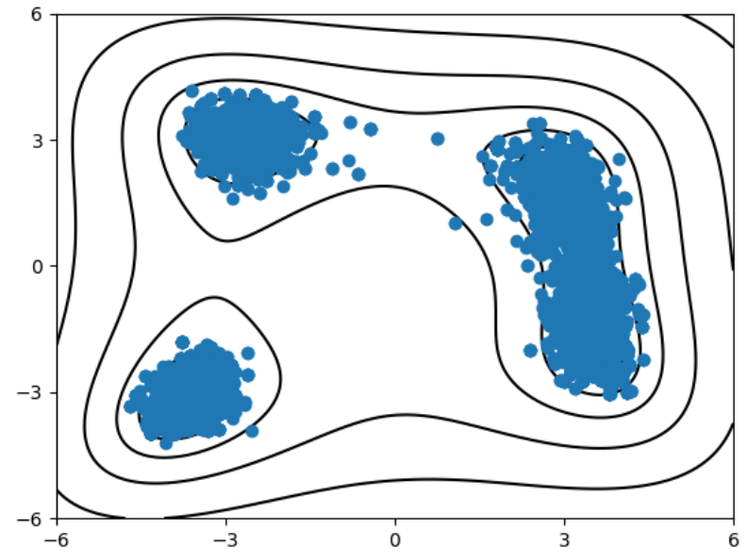
交換MC のデモ

```
$ 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）

```
[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]
T_min =  0.1        # 温度の下限
T_max = 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"
```

f が小さいところを重点的にサンプリングできている

ポピュレーションアニーリング

- 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
    - （実際にはもう少しうまくやる）
  - 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
- 現在の py2dmat (v1.0) には未実装
  - 2021 年度中には実装されます

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視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 (Dexp) と 理論曲線 (D(x)) との距離 f(x) = √Σ(Dexp - 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い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

用意する入力ファイル

bulk.txt

experiment.txt

template.txt

input.toml

予め実行しておく

bulk.exe

bulkP.b

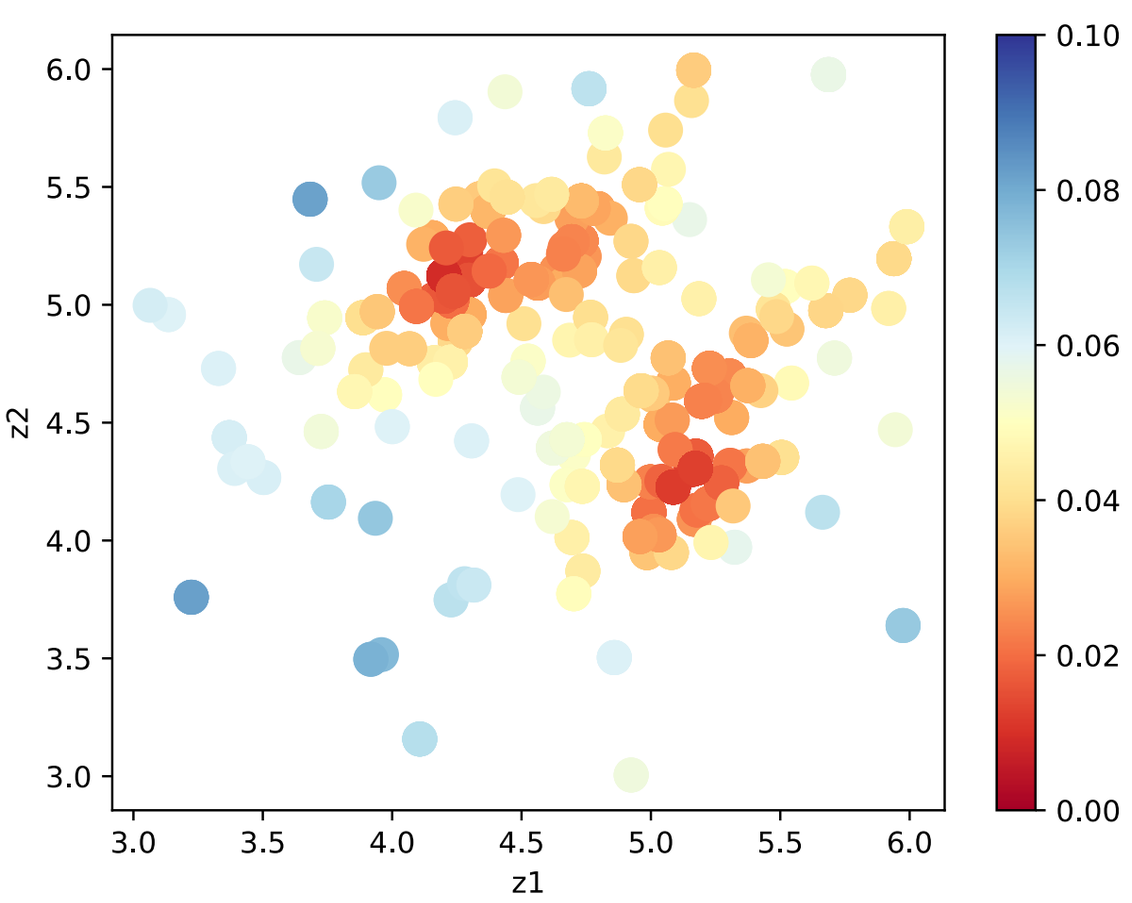
py2dmat

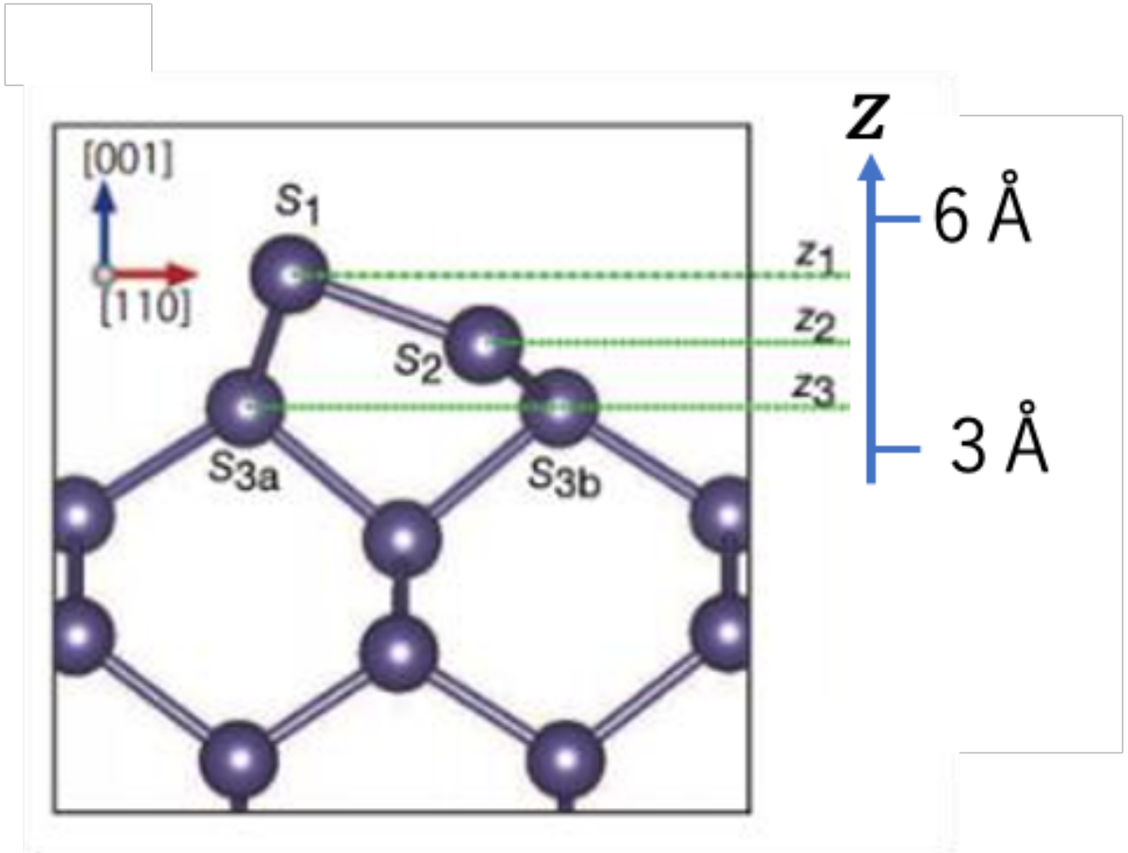
最終的な結果ファイル
（Algorithm によって異なる）

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

py2dmat(exchange) + STRの例

- 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
- この例では z1, z2 は交換に対して対称





自作のAlgorithm, Solver を使う

- 自分で独自のAlgorith, Solver を定義することでpy2dmat を拡張可能
- python でクラスを定義し、呼び出す
- 詳細はマニュアル参照
  - もちろん困ったときにはお問い合わせください

バージョンアップなど

- py2dmat のバージョンアップでは Algorithm や Solver が追加されます
  - Algorithm
    - population annealing Monte Carlo
  - Solver
    - surface x-ray diffraction (SXRD)
    - low-energy electron diffraction (LEED)
- その他、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、 警告メッセージを表示します
- 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