

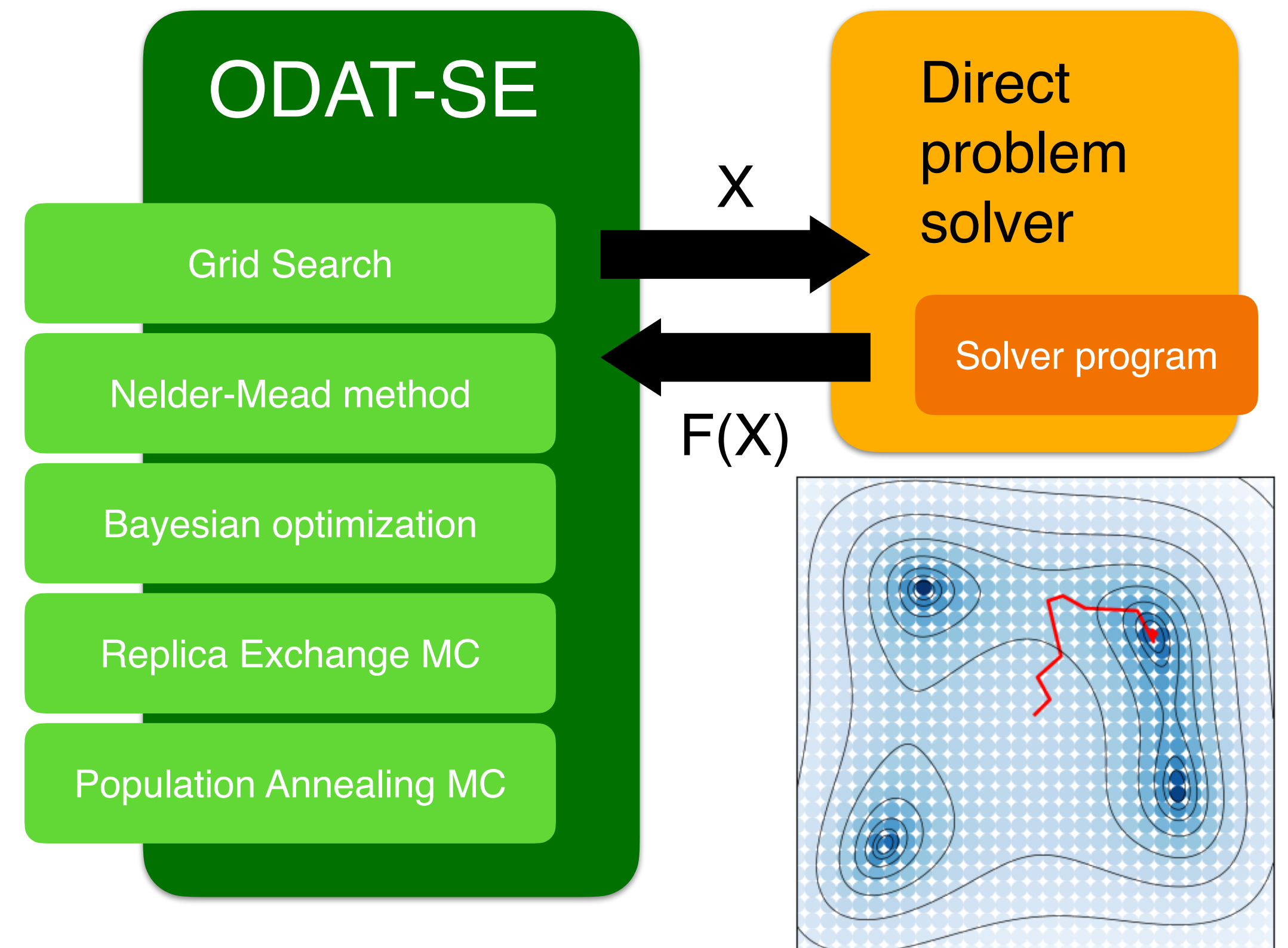
# ODAT-SEの 順問題ソルバーを作成する

データ解析フレームワークODAT-SEミーティング  
2025年5月8日 @NIFS

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

# ODAT-SEの構成

- 順問題ソルバー
  - 目的関数  $F(X)$  を評価するモジュール
  - 課題に合わせて作成する
- 逆問題解析アルゴリズム
  - $F(X)$  を最小化する  $X^*$  を探索する
  - 複数のアルゴリズムを実装
    - グリッド探索・Nelder-Mead法・
    - ベイズ最適化・レプリカ交換モンテカルロ法・
    - ポピュレーションアニーリング
- ポスト処理ツール
  - 事後確率分布の評価・可視化など



# もう少し詳しく

## ODAT-SEのコードの構造

- 探索アルゴリズムと順問題ソルバーの連携
  - アルゴリズムに従って候補点  $X$  を探索
  - ソルバーは候補点での関数値  $F(X)$  を評価
- ソルバーのインターフェース
  - evaluate メソッド

- ソルバークラスの構造
  - 初期化
    - パラメータの処理、参照データ読み込みなど
  - 目的関数  $F(X)$  の評価
    - 関数値を計算する/外部プログラム呼び出し/  
参照データとの比較

```
def evaluate(self, xs, args, nproc, nthr) -> float
```

- xs: 候補点の座標 (numpy.ndarray)
- args: 探索点のインデックスなど
- 戻り値: 関数値(float)

# もう少し詳しく Functionソルバーの構成

## • Functionソルバークラス

- ODAT-SEに組み込みの順問題ソルバー
- 多変数関数の最小化問題を扱う
- 対象の関数を `set_function` メソッドでセット

## • 解析関数のサンプル

- いくつかの解析関数が用意されている
  - Himmelblau • Rosenbrock • Ackley • Quadratics • Quartics
  - (最適化問題のベンチマーク)
- 関数を作成して追加する

## • 関数を定義する

- 関数の型

```
def func(xs: np.ndarray) -> float
```

- 例

```
def himmelblau(xs):  
    x, y = xs  
    return (x**2+y-11)**2+(x+y**2-7)**2
```

- 使い方

- `odatse` の入力パラメータに指定:

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

定義されている関数名

# 解析関数ソルバーを作成する

## 線形回帰問題ソルバーを追加する

### ・ 問題設定

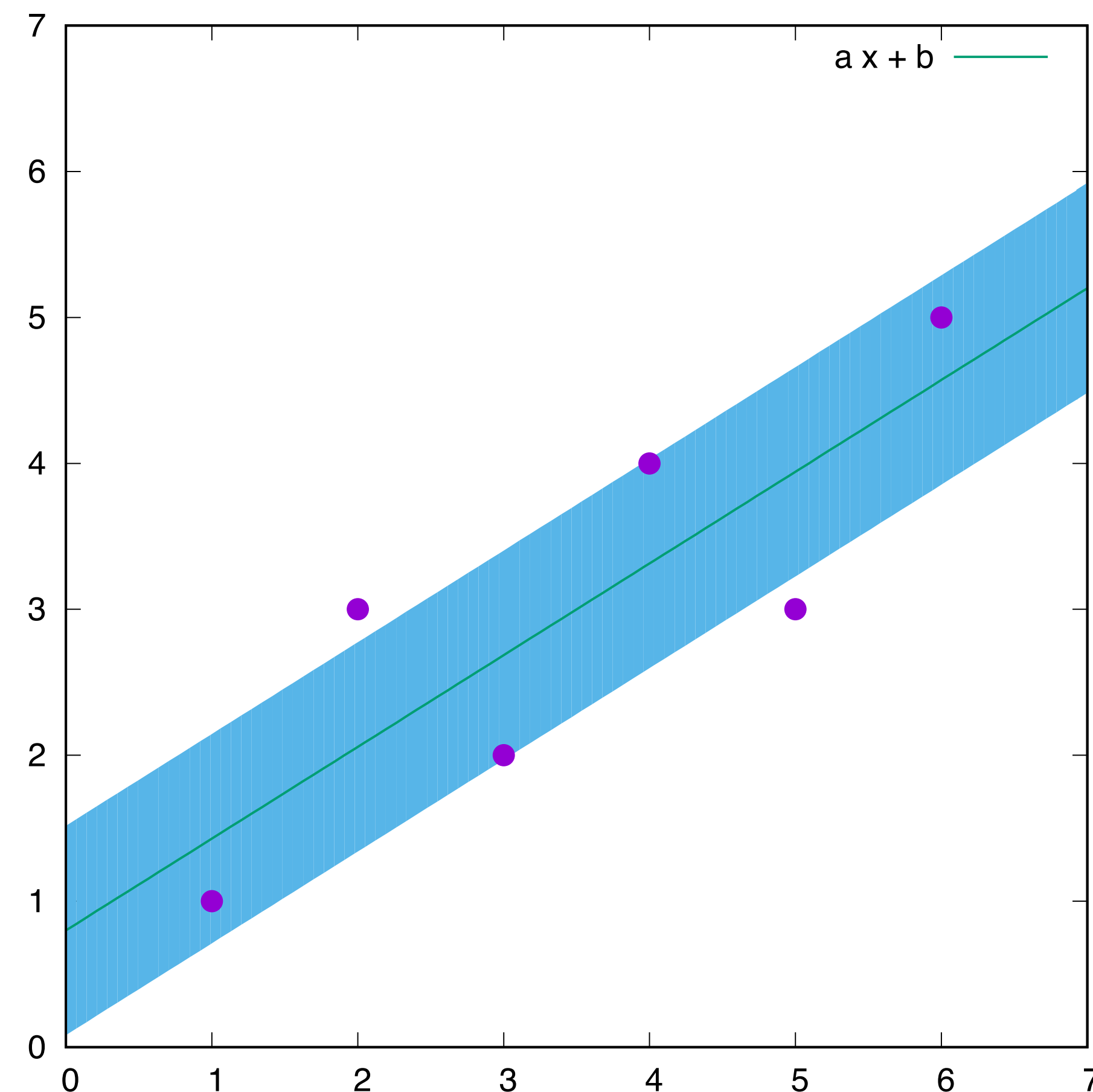
$(x_i, y_i)$ の組が与えられたとき、 $y_i$ が  $y = ax + b$  の周りに分散  $\sigma$  で正規分布しているとして、 $a, b, \sigma$  の値を推定する。対数尤度関数

$$\ln P = -\frac{1}{2} \left( nt + \sum_i (y_i - (ax_i + b))^2 / e^t \right)$$

を最大化する  $a, b, \sigma$  を求める ( $-\ln P$ を最小化,  $t = \ln \sigma^2$ )

### ・ プログラムについて

- ・ 簡単のため、データ点はプログラムに埋め込み
- ・ 解析関数ソルバーに追加する形で実装する





# 解析関数ソルバーを作成する

## コード例

- 関数を定義する

```
def linear_regression_test(xs: np.ndarray) ↵  
                                -> float:  
  
    a, b, t = xs  
    xdata = np.array([1,2,3,4,5,6])  
    ydata = np.array([1,3,2,4,3,5])  
    n = len(ydata)  
  
    lnP = -0.5*(n*t  
            + np.sum((a*xdata+b-ydata)**2)/np.exp(t))  
    return -lnP
```

- Functionソルバーに登録する

- src/odatse/solver/analytical.py を編集

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

- 使い方

- odatse の入力パラメータに指定する

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

# 解析関数ソルバーを作成する

## 実行例

- Nelder-Mead法による解析 (右図)
  - output/res.txt に結果が出力される:

```
fx = 1.0050710911347074
x1 = 0.6285685294406518      a
x2 = 0.8000048609536232      b
x3 = -0.6649435572227187     t = log σ2
```

- 解析解は:

```
a = 22/35 = 0.6285714...
b = 4/5 = 0.8
t = log(18/35) = -0.664976...
```

```
$ odatse input.toml
name      : minsearch
seed      : 12345
param.min_list : [-2.0, -2.0, -2.0]
param.max_list : [2.0, 2.0, 2.0]
param.initial_list: [0.0, 0.0, 0.0]
minimize.initial_scale_list: [0.1, 0.1, 0.1]
eval: x=[0.1 0.1 0.1], fun=21.559155136754867
eval: x=[ 0.2 -0.1  0.2], fun=17.32666928538317
eval: x=[0.4 0.  0.1], fun=9.058826206588089
eval: x=[0.4 0.  0.1], fun=9.058826206588089
eval: x=[ 0.76666667 -0.3      0.36666667], fun=3.0755507898130716
eval: x=[ 0.82222222 -0.1      0.28888889], fun=2.379192076124161
eval: x=[ 0.82222222 -0.1      0.28888889], fun=2.379192076124161
eval: x=[ 0.82222222 -0.1      0.28888889], fun=2.379192076124161
.....
eval: x=[ 0.62856905  0.79999382 -0.66490654], fun=1.0050710978414183
eval: x=[ 0.62858829  0.79992621 -0.66498858], fun=1.0050710955564233
eval: x=[ 0.62855794  0.8000666  -0.66497555], fun=1.0050710945095704
eval: x=[ 0.62855779  0.80005489 -0.66497758], fun=1.0050710926859194
eval: x=[ 0.62856853  0.80000486 -0.66494356], fun=1.0050710911347074
eval: x=[ 0.62856853  0.80000486 -0.66494356], fun=1.0050710911347074
Optimization terminated successfully.
      Current function value: 1.005071
      Iterations: 73
      Function evaluations: 137
end of run
```

# ソルバークラスを作成する

## 線形回帰問題ソルバーをクラスに書き換え

- 目標

- データ点  $(x_i, y_i)$  をファイルから読み込む
- データファイルをパラメータで指定する

- ソルバークラス作成の手順

- SolverBase 基底クラスから派生させる
- コンストラクタでパラメータを受け取る
- evaluate メソッドを実装する

- コード例 (右図)

```
class LinearRegression(odatse.solver.SolverBase):
    def __init__(self, info):
        super().__init__(info)

        data_file = info.solver["reference"]["data_file"]
        data = np.loadtxt(data_file, unpack=True)
        self.xdata = data[0]
        self.ydata = data[1]
        self.n = len(self.ydata)

    def evaluate(self, xs, args, nprocs, nthreads):
        a, b, t = xs
        lnP = -0.5*(self.n*t + np.sum((a*self.xdata+b
                                     -self.ydata)**2)/np.exp(t))
        return -lnP
```



# ソルバークラスを作成する

## コード例

### • コンストラクタ

- odatse.Info型の入力パラメータを受け取る
- データ点をファイルから読み込み、メンバ変数 `xdata`, `ydata` にストアする
  - テキスト形式で各行に  $x_i$   $y_i$  の値を記述

### • 目的関数の評価

- `evaluate` メソッド
  - 内容はFunctionソルバーと同等

### • パラメータファイル

- TOMLファイルの `solver.reference.data_file` にファイル名を指定

```
class LinearRegression(odatse.solver.SolverBase):
```

```
    def __init__(self, info):  
        super().__init__(info)
```

```
        data_file = info.solver["reference"]["data_file"]  
        data = np.loadtxt(data_file, unpack=True)  
        self.xdata = data[0]  
        self.ydata = data[1]  
        self.n = len(self.ydata)
```

```
    def evaluate(self, xs, args, nprocs, nthreads):  
        a, b, t = xs  
        lnP = -0.5*(self.n*t + np.sum((a*self.xdata+b  
                                     -self.ydata)**2)/np.exp(t))  
        return -lnP
```

# ソルバークラスを作成する

## main関数を作成する

- **main 関数を作成する** (右図)

- ①初期化、入力パラメータを取得
- ②ソルバー・オブジェクト作成
- ③Runner 作成
- ④アルゴリズム選択とオブジェクト作成
- ⑤アルゴリズム実行

```
import odatse
from odatse.algorithm import choose_algorithm

def main():
    info, run_mode = odatse.initialize() ← ①
    solver = LinearRegression(info) ← ②
    runner = odatse.Runner(solver, info) ← ③
    alg_module = choose_algorithm(
        info.algorithm["name"]) ← ④
    alg = alg_module.Algorithm(info, runner,
        run_mode=run_mode)
    result = alg.main() ← ⑤

if __name__ == "__main__":
    main()
```

# ソルバークラスを作成する

## パッケージ化する

- シンプルな `pyproject.toml` を作成する (右図)
  - `name`, `version` を記述 (必須)
  - `packages` にインストール先を指定
  - `scripts` にはコマンドとしてインストールする関数を指定する
- **ファイル構成**
  - `src/Solver/` にソースコードを配置
    - `linear_regression.py`, `_main.py`, `__init__.py`
  - `pyproject.toml`, `README.md` を作成
- **インストール**

```
$ pip install .
```

```
[tool.poetry]
name = "ODAT-SE-linear-regression"
version = "1.0.0"
packages = [
    {include="Solver", from="src", to="odatse_solver"}
]

[tool.poetry.dependencies]
numpy = "^1.26"
ODAT-SE = ">=3"

[tool.poetry.scripts]
odatse-solver = "odatse_solver.Solver._main:main"

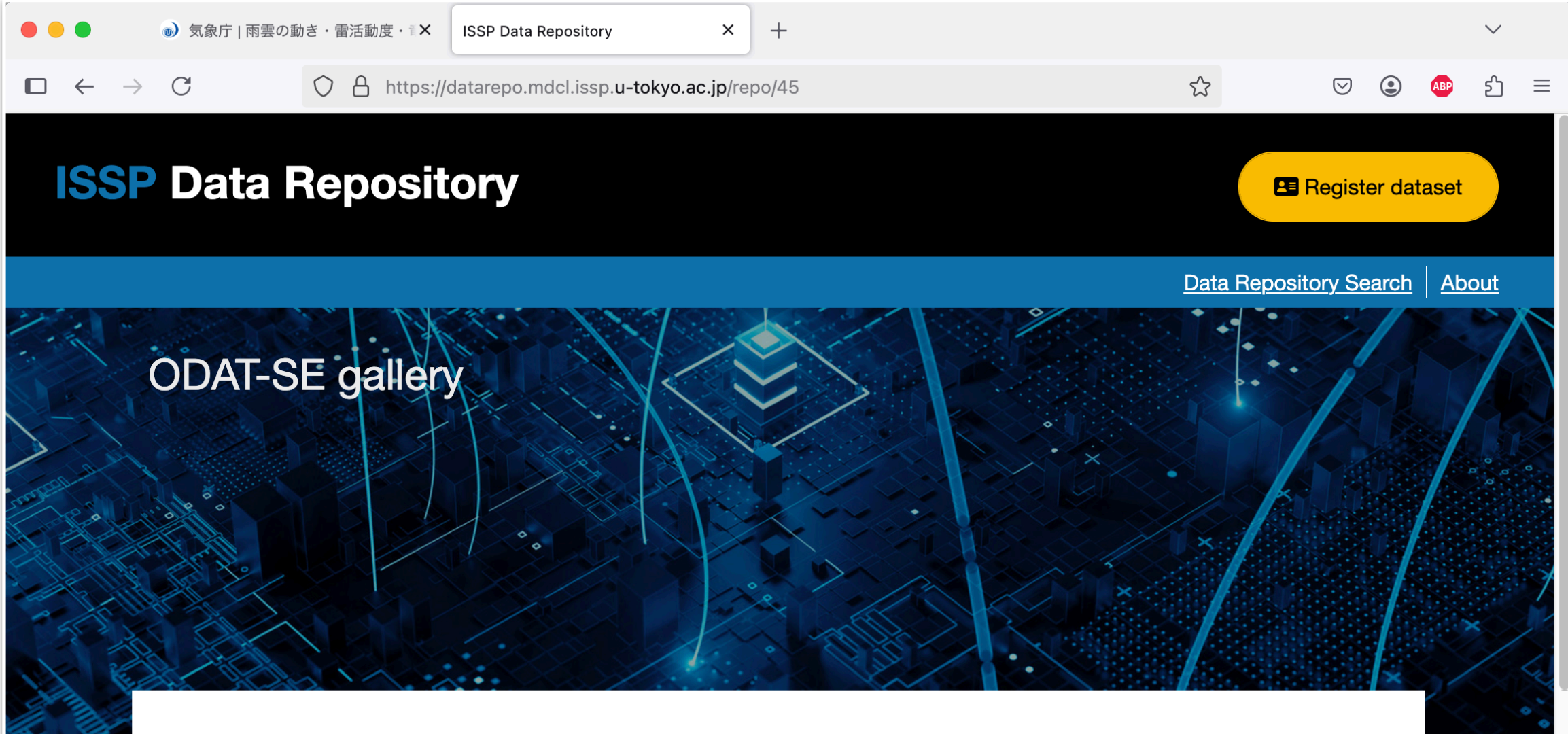
[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```



# 順問題ソルバーのサンプル・テンプレート

## ODAT-SE Galleryの紹介

- 物性研データリポジトリに ODAT-SE Gallery を用意しています。
  - ソルバーのサンプル・テンプレート集
  - (主に2次元物質構造解析の)サンプルデータ
- 今回のサンプル・テンプレートは `data/tutorial/solver-template/solver_module` にあります。
- 外部プログラムとの連携についてはサンプルを `data/tutorial/solver-template/external_solver_module` に置いています。



The screenshot shows a web browser window with the URL `https://datarepo.mdcl.issp.u-tokyo.ac.jp/repo/45`. The page title is "ISSP Data Repository" and it features a "Register dataset" button. The main content area is titled "ODAT-SE gallery" and displays a table with the following information:

About	Open Data Analysis Tool for Science and Engineering (ODAT-SE) is a framework for applying a search algorithm to a direct problem solver to find the optimal solution. It has been developed by the name 2DMAT, and since version 3.0, it is organized as an open platform for data analysis by modularizing direct problem solvers and search algorithms. This repository is a collection of examples for ODAT-SE.
URL	<a href="https://isspns-gitlab.issp.u-tokyo.ac.jp/takeohoshi/odat-se-gallery">https://isspns-gitlab.issp.u-tokyo.ac.jp/takeohoshi/odat-se-gallery</a>
DOI	
Authors	<a href="#">Takeo Hoshi</a> , <a href="#">Kazuyoshi Yoshimi</a>

<https://datarepo.mdcl.issp.u-tokyo.ac.jp/repo/45>

<https://isspns-gitlab.issp.u-tokyo.ac.jp/takeohoshi/odat-se-gallery>