パレート最適とパレート最適解の求め方

鈴木　将之*

Introduction of Pareto optimality

Masayuki Suzuki*

多目的最適化は複数の目的関数の最適化を図る問題である。多目的最適化において、いくつかの目的関数がトレードオフの関係にある場合、ある目的関数の評価値を改善すると他の目的関数の評価値が悪化する状態であるパレート最適を求めることが一般的である。パレート最適を求めるアルゴリズムやライブラリが、近年では機械学習のハイパーパラメータのチューニングなどで注目されている。本稿ではパレート最適解を求めるアルゴリズムとして NSGA-II とその実装の一つ pymoo を紹介する。
Keywords: 多目的最適化、ブラックボックス最適化、パレート最適、NSGA-II、pymoo、制約付き最適化問題

1. はじめに

ビジネス上の問題解決や効率化を図るべく様々な最適化が行われる。最適化は目的関数を定義し、目的関数の出力である評価値の最小化を図る(最小化問題)ことが一般的である。なお、逆に評価値の最大化を図る場合は、評価値をマイナスとすることで最小化問題に落とし込むことができる。

目的関数が複数ある場合は多目的最適化と呼ばれる。近年では機械学習のハイパーパラメータのチューニングを目的として Optuna といった様々な多目的最適化ライブラリが開発されている[1]。これらの多目的最適化ライブラリは同時に多変量最適化も行うことができ、様々な目的関数の最適化を行うことができる。

多目的最適化において、目的関数はトレードオフの関係にある場合がある。ある目的関数を改善すると別の目的関数が悪化するような状態をパレート最適と呼ぶ。またパレート最適解の集合をパレートフロントと呼ぶ。多目的最適化アルゴリズムやライブラリはパレートフロントを求めることを目的とする。

本稿ではパレートフロントを求めるアルゴリズムとして 2 章に NSGA-II を紹介する。また、NSGA-II の実装として pymoo ライブラリを 3 章に紹介する。最後にまとめを 4 章に示す。

2. NSGA-II によるパレートフロントの求解

パレートフロントを求めるアルゴリズムとして NSGA-II [2]や TPE [3]などが広く使われる。本稿では NSGA-II を紹介する。

NSGA-II は進化的アルゴリズムの一種であり。実装の容易さと、前提条件なしに多変量多目的最適化問題へ容易に適用可能なことから、広く使われるアルゴリズムである。

NSGA-II を説明する前に、多目的最適化の各種アルゴリズムで使用される概念を定義する。図 1 に示す白丸のように、目的関数f_1とf_2のいずれかの評価値がよい黒丸の個体がある場合、白丸は黒丸に支配されると定義する。また図 2 に示すような、支配される個体がない個体の集合を非優越ランク 1 とし、非優越ランク$n-1$より小さい非優越ランクの個体以外に支配される個体がない個体の集合を非優越ランクnとする。

*アドバンスソフト株式会社　第 2 事業部
2nd Computational Science and Engineering Group, AdvanceSoft Corporation

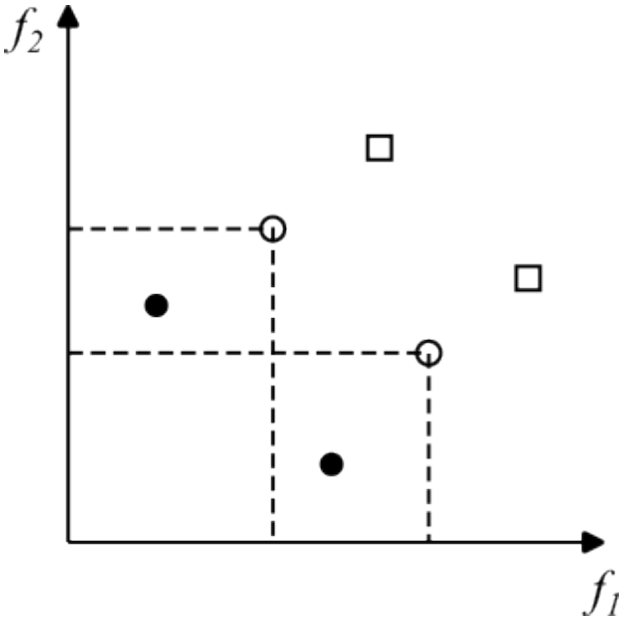

図 1 支配関係の例

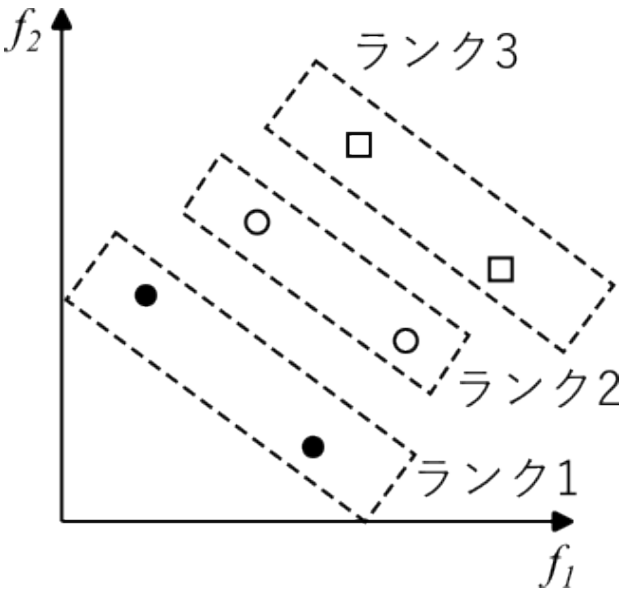

図 2 非優越ランクの例

NSGA-II の処理手順を以下に示す。

1. 個体を高速非優越ソートにより非優越ランク順に並び替える。
2. 非優越ランクの小さい個体からあらかじめ設定した個体数を親集団に登録する。設定した個体数より多い場合は、混雑距離の大きい個体を優先して親集団に登録する。
3. 親集団の個体から交差と突然変異を行い、子集団を生成する。
4. あらかじめ設定した世代数だけ 1 から 3 を繰り返す。

高速非優越ソートは非優越ランクを求める高速なアルゴリズムである。高速非優越ソートの処理手順を以下に示す。

1. 各個体に対して、支配している個体のリストと支配されている個体数を探索する。
2. 支配されている個体が 0 の個体を非優越ランク$n+1(n=0,1,2,\dots)$とする。
3. 非優越ランクnの個体に対し、その個体が支配している個体の、支配されている個体数を 1 減らす。
4. 2 と 3 を繰り返す。

混雑距離は他の個体との目的関数の差の和である。ある個体の混雑距離$I[i]_{distance}$を求める手順を以下に示す。

1. 全ての目的関数に対し、2 から 4 を行う。
2. m 番目の目的関数の評価値で個体をソートする。ソート済みの個体番号を i とする。
3. i(i=1,2,...l)に対し、$I[1]_{distance}=I[l]_{distance}=\infty$とする。
4. i(i=2,3,...,l-1)に対し、式 1 より混雑距離を更新する。式 1 において、ソートした i 番目の個体の評価値を$\mathcal{I}[i]^m$、m 番目の目的関数の最大値と最小値をそれぞれv_{max}^mとv_{min}^mとする。

$$I[i]_{distance}=I[i]_{distance}+\frac{\mathcal{I}[i+1]^m-\mathcal{I}[i-1]^m}{v_{max}^m-v_{min}^m} \quad (1)$$

すなわち、NSGA-II では高速非優越ソートによる高速な最良個体の選別と、混雑距離計算による大きく離れた個体同士を残すことにより局所解に陥ることを避けることができるアルゴリズムである。

3. Pymoo による最適化

NSGA-II の実装として、Python ライブラリの Optuna [4]や pymoo [5]、R のライブラリの mco [6]などがある。Optuna と pymoo の大きな違いとして、制約付き最適化問題に対し Optuna は制約条件を厳密に満たさない解も探索する(ソフト制約)のに対し、pymoo は制約条件を厳密に満たすよう解を探索する点(ハード制約)が挙げられる。本稿では制約付き問題の最適化の紹介も兼ねて、pymoo を紹介する。

まずは単純な単目的最適化の例としてヒンメルブラウの関数 [7]の最小値を求める例を示す。ヒンメルブラウの関数を式 2 に示す。

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

ヒンメルブラウの関数は $f(3.0, 2.0)$ 、$f(-2.805118, 3.131312)$ 、$f(-3.779310, -3.283186)$ と $f(3.584428, -1.848126)$のとき最小値 0 を取る。

コード 1 にヒンメルブラウの関数に対し pymoo による最小値の一つを求める例を示す。コード 1 は以下を行う。

- HimmelblauProblem クラスにてヒンメルブラウの関数の最小値問題の定義を行う。Himmelblau Problem クラスのコンストラクタで引数と戻り値の数と引数の範囲を指定する。_evaluate メンバ関数にて himmelblau 関数の戻り値を評価値とする。
- 変数 algorithm に NSGA2 クラスのインスタンスを代入する。
- minimize 関数にて最小値問題を解く。世代数 10 とし、シード値を 1 の定数にする。
- print 関数で結果出力を行う。

```
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
from pymoo.core.problem import Problem



def himmelblau(x,    y):
    """
    ヒンメルブラウの関数
    """
    return (((x**2+y-11)**2) + (((x+y**2-7)**2)))



class HimmelblauProblem(Problem):
    """
    問題の定義
    """


    def __init__(self):
        super().__init__(n_var=2,
                         n_obj=1,
                         xl=-6.0,
                         xu=6.0)


    def _evaluate(self, x, out, *args, **kwargs):
        h = himmelblau(x[:, 0], x[:, 1])
        out["F"] = h



problem = HimmelblauProblem()
# アルゴリズムとして NSGA-II を選択
algorithm = NSGA2(pop_size=100)

# 最小化問題として解く
res = minimize(problem,
               algorithm,
               ('n_gen', 10),
               # シード値を固定
               seed=1,
               verbose=True)

# 結果出力
print(res.X)
print(res.F)
print(res.pop.get("X"))
print(res.pop.get("F"))
```

コード 1 pymoo によるヒンメルブラウの関数の最小値問題の求解

コード 1 を実行すると、print(res.X)行では最も最小値を得られた引数の組「[3.00184243 1.99302461]」を得ることができる。これはヒンメルブラウの関数の最小値を取る引数の一つに近い。またコード 2 に、コード 1 の 最終世代の全個体を出力する print(res.pop.get("X"))行の出力一部を示す。コード 2 より、いくつかの個体はヒンメルブラウの関数の、他の最小値を取る引数周辺も探索していることが分かる。

```
[[ 3.00184243  1.99302461]
 [ 3.00184243  1.98494982]
 [ 3.57443527 -1.83269187]
 [ 2.98525559  1.97930005]
```

```
 [ 3.57648103 -1.88285203]
 [ 2.98257168    1.97682432]
 [ 3.554806      -1.84376473]
 [-2.78540986    3.16038891]
(以下略)
```

コード 2 コード 1 の print(res.pop.get("X"))行出力一部

次に、コード 3 にコード 1 に引数が全て正である制約を追加したものを示す。コード 3 において、制約は Himmelblau Problem クラスの_evaluate メンバ関数の戻り値 out[“G”]として与えている。

```
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
from pymoo.core.problem import Problem



def himmelblau(x,    y):
    """
    ヒンメルブラウの関数
    """
    return (((x**2+y-11)**2) + (((x+y**2-7)**2)))



class HimmelblauProblem(Problem):
    """
    問題の定義
    """

    def __init__(self):
        super().__init__(n_var=2,
                         n_obj=1,
                         n_ieq_constr=2,
                         xl=-6.0,
                         xu=6.0)

    def _evaluate(self, x, out, *args, **kwargs):
        h = himmelblau(x[:, 0], x[:, 1])
        out["F"] = h
        # 引数は正である制約
        out["G"] = [-x[:, 0], -x[:, 1]]


problem = HimmelblauProblem()
# アルゴリズムとして NSGA-II を選択
algorithm = NSGA2(pop_size=100)

# 最小化問題として解く
res = minimize(problem,
               algorithm,
               ('n_gen', 10),
               # シード値を固定
               seed=1,
               verbose=True)

# 結果出力
print(res.X)
print(res.F)
print(res.pop.get("X"))
print(res.pop.get("F"))
```

コード 3 pymoo によるヒンメルブラウの関数の制約付き最小値問題の求解

コード 4にコード 3の print(res.pop.get("X"))行出力一部を示す。コード 4 においてコード 2 とは異なり、制約の通り全て正の値の引数が探索されていることが分かる。すなわち pymoo ではハード制約による制約付き最適化問題を解くことができることが分かる。

```
[[3.01321132 1.98879235]
 [3.01351832 2.00759939]
 [3.01077765 2.01479369]
 [3.01439904 1.97195716]
 [3.01612813 2.00759939]
 [3.01351832 2.01551744]
 [2.98904965 1.97655554]
(以下略)
```

コード 4 コード 3 の print(res.pop.get("X"))行出力一部

最後に、pymoo の公式サンプル ZDT1 を紹介する [8]。ZDT [9]は式 3 に示す 2 つの目的関数の最小化を図る問題である。

$$\min f_1(x)$$
$$\min f_2(x) = g(x)h(f_1(x), g(x)) \quad (3)$$

ZDT 問題の一つ ZDT1 では式 2 の$f_1(x)$、$g(x)$、$h(f_1, g)$を式 4 のように定義する。

$$f_1(x) = x_1$$
$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \quad (4)$$
$$h(f_1, g) = 1 - \sqrt{f_1/g}$$

コード 5 に pymoo による ZDT1 の求解の例を示す。コード 5 では公式サンプルの problem インスタンスを使用し、またそのパレートフロントを NSGA-II により求めたパレートフロントと一緒にプロットする。図 3 にコード 5 のプロット結果を示す。図 3 より、実線で示されている理論上のパレートフロントと、NSGA-II によるパレートフロントが一致していることが分かる。

```
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.optimize import minimize
from pymoo.problems import get_problem
from pymoo.visualization.scatter import Scatter

problem = get_problem("zdt1")
algorithm = NSGA2(pop_size=100)

res = minimize(problem,
               algorithm,
               ('n_gen', 200),
               verbose=True)

plot = Scatter()
plot.add(problem.pareto_front(),  plot_type="line",
color="black", alpha=0.7)
plot.add(res.F, facecolor="none", edgecolor="red")
plot.show()
```

コード 5 ZDT1 の求解の例

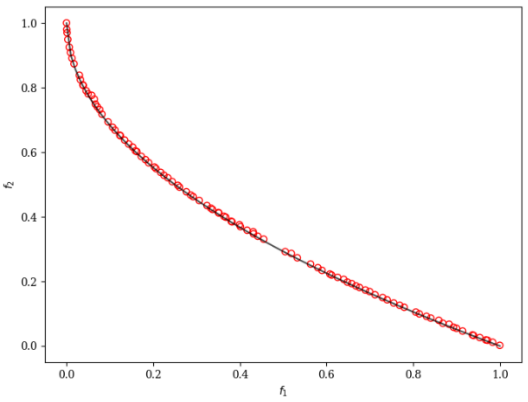

図 3 ZDT1 のパレートフロント(実線)とコード 5 により求められたパレートフロント

4. まとめ

本稿では多目的最適化のためのアルゴリズムとして NSGA-II と、その実装の一つ pymoo を紹介した。NSGA-II はパレートフロントを求めるアルゴリズムであり、NSGA-II を適用することで容易に多変量多目的最適化を行うことができる。

NSGA-II を用いる一番のメリットはブラックボックス最適化といわれる、目的関数の形状が未知であったり、外部プログラムの引数であったりするパラメータの最適化も行うことができる点が挙げられる。本稿で示したサンプルプログラムの通り、評価値を適切に返す関数とクラスを定義し、十分な世代数を設定すれば、各種ライブラリで何らかのパレートフロントを得ることができる。

アドバンスソフトでは Optuna や pymoo を用いた最適化に加え、様々なデータ解析の実績がある。お手元の大量のデータ解析に関してお悩み事があれば、お気軽に弊社に相談いただきたい。

参考文献

[1] 佐野, et al. Optuna によるブラックボックス最適化. 2023. ISBN 9784274230103.

[2] DEB, Kalyanmoy, et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings 6. Springer Berlin Heidelberg, 2000. p. 849-858.

[3] BERGSTRA, James, et al. Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 2011, 24.

[4] Optuna - A hyperparameter optimization framework. Optuna [online]. [no date] [viewed 30 April 2024]. Available from: https://optuna.org/

[5] pymoo: Multi-objective Optimization in Python. pymoo: Multi-objective Optimization in Python [online]. [no date] [viewed 30 April 2024]. Available from: https://pymoo.org/

[6] mco package - RDocumentation. Home - RDocumentation [online]. [no date] [viewed 30 April 2024]. Available from: https://www.rdocumentation.org/packages/mco/versions/1.16

[7] HIMMELBLAU, David M., et al. Applied nonlinear programming. McGraw-Hill, 2018.

[8] pymoo - ZDT. pymoo: Multi-objective Optimization in Python [online]. [no date] [viewed 30 April 2024]. Available from: https://pymoo.org/problems/multi/zdt.html

[9] ZITZLER, Eckart; DEB, Kalyanmoy; THIELE, Lothar. Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, 2000, 8.2: 173-195.

※ 技術情報誌アドバンスシミュレーションは、アドバンスソフト株式会社 ホームページのシミュレーション図書館から、PDF ファイル（カラー版）がダウンロードできます。（ダウンロードしていただくには、アドバンス/シミュレーションフォーラム会員登録が必要です。）