使い方

本パッケージは、SCIOLICという名前で、分離可能凸整数整数係数線型制約付き整数最適化問題(Separatable Convex Inter Optimization with Linear Integer Condition)を解くことができます。具体的には、目的函数fが

$$f(x) = \sum_i f(x_i)$$

と書け、各fは一変数凸函数であるようなものであって、各変数x_iは整数値をとり、$l_i, u_i, a_i, b_i, c_i d_i$は全て整数であるとして、制約条件が

$$\forall i, l_i \leq x_i \leq u_i$$

$$\sum_i a_{ij} x_j \leq b_i$$

$$\sum_j c_{ij} x_j = d_i$$

と書けるようなものです。例えば、各$f(x_i)$が線型函数であれば、これは凸函数になります。

では、使い方を以下に説明していきます。まず、githubからoptimizerとgraverという二つの名前のファイルを`git clone`で適当なディレクトリにダウンロードしてください。さらに、本パッケージは内部でnumpyに依存しているので、あらかじめ`pip install numpy`としてnumpyをダウンロードしておきます。

では、問題を解いていきましょう。

まず、`Solver = SCIOLIC()`でSCIOLICクラスをインスタンス化し、

`Solver.N_search_feasible`で実行可能解の探索回数を指定、

`Solver.alpha`で近似度、即ち$x*$を出力される解として、

$$\alpha = 1 - \frac{f(x^*) - \min f}{\max f - \min f} = \frac{\max f - f(x^*)}{\max f - \min f}$$

を$0 < \alpha < 1$の範囲で指定できます。定義より$0 \leq \alpha \leq 1$となり、1に近ければ近いほど最適解に近いと言えますが。αが1に近いほど解の改善アルゴリズムの繰り返し回数が増えます。

`Solver.verbose` でログ出力を指定できます。即ち、`True`ならログ出力し、`False`ならしません。

問題を解くためには。

`Solver.add_variable("x", 3, 5)`のようにすれば、$3 \leq x \leq 5$の値をとる変数xを追加できます。

制約条件に不等式

$$2x - y - 2z > 2$$

を追加したいときは、

`Solver.add_condition(">", 2)`のようにして、条件式の定数と不等号を定義し、

`Solver.define_coefficient(0, "x", 2)`のようにすれば、0番目の条件式（条件式の番号は、追加された順番です。）の変数xの係数を2に指定できます。

最後に、`Solver.add_convexfunc("x", 3)`によって、目的函数の変数xに依存する部分の係数を指定できます。係数の代わりに一変数凸函数fを定義して代入、即ち

`def f(x):`

`return 3*x`

`Solver.add_convexfunc("x", f)`

のようにすることもできます。fが凸であるかどうかはあらかじめ人間が判断しておく必要があります。

具体例を見ると、例えば、以下のような目的函数を最小化するためには、以下のように記述します。

、、

$$\text{objective} : 3x + 4y + 2z$$

$$2x - y - 2z > 2$$

$$3 \leq x \leq 5$$

$$1 \leq y \leq 4$$

$$-2 \leq z \leq 3$$

`from optimizer import *`

`# SCIOLICクラスをインスタンス化する`

`Solver = SCIOLIC()`

`# 実行可能解を最大何回探索するか`

`Solver.N_search_feasible = 10`

`# 近似度`

```
Solver.alpha = 0.9
# ログ出力するかどうか
Solver.verbose = True
Solver.add_variable("x", 3, 5)
Solver.add_variable("y", 1, 4)
Solver.add_variable("z", -2, 3)
Solver.add_condition(">", 2)
Solver.define_coefficient(0, "x", 2)
Solver.define_coefficient(0, "y", -1)
Solver.define_coefficient(0, "z", -2)
Solver.add_convexfunc("x", 3)
Solver.add_convexfunc("y", 4)
Solver.add_convexfunc("z", 2)
```

以上で問題が定義できたので、

```
print(Solver.search())
```

によって解の探索を実行します。`search`メソッドは、0番目に解、1番目に最適解かそうでないかを返します。例えば、上の問題の場合は、実行可能解の探索に成功している限り

```
({'x': 3, 'y': 1, 'z': -2}, True)
```

という答えが返ってくるはずです。