JAROS2019 研究発表大会〈チュートリアルセッション〉

Python パッケージを用いた数理最適化の実践

小林　和博
(青山学院大学理工学部)

1. はじめに

数理最適化問題を、コンピュータを用いて解くには，問題例の入力データを効率よく作成するモデリング言語と，その問題例の最適解を効率よく計算できるソルバをあわせて用いると便利である．Python 言語では，人間の読みやすい形式で問題例を表現できるモデリング言語と，最適解を効率よく計算できるソルバを，パッケージとして簡単に利用することができる．このチュートリアルでは，基本的な数理最適化問題を Python 言語のパッケージを用いて解く方法を述べる．

2. なぜ Python か？

Python は，最近よく用いられているコンピュータ言語である．キーワードが少ない，変数の宣言がいらない，タブなどの使い方が決められているため読みやすい，などの特徴がある．その中でも，様々な便利なパッケージが容易に利用可能である，という点が，利用者を増やしている最も大きな理由と考えられる．例えば，重力から解き放たれるには，

```
import antigravity
```

と打てばよい [1].

また，コンパイルの必要がなく（してもよい），多くのプラットフォームで動く．パッケージのインストールも楽に行うことができる．パッケージの管理には pip というパッケージ管理ツールを用いると便利である．例えば，PuLP というパッケージをインストールしたければ，コマンドラインで

```
$pip install pulp
```

を実行すればよい．こうすると，依存関係をチェックした上で必要なパッケージとあわせて PuLP をインストールすることができる．

[1] https://xkcd.com/353/

3. 数理最適化問題

数理最適化とは，解決したい問題を、数式を用いて表現し，それに対して様々な計算を行うことで最適な決定を見出す方法論のことを指す．一般の数理最適化問題は，次のように表される．

$$\text{最小化}\quad f(x)\quad \text{制約}\quad x \in S. \tag{1}$$

ここで，$f(x)$のことを目的関数，Sのことを実行可能領域とよぶ．

数理最適化問題は，用いられる数式のタイプによって様々なクラスに分類される．

4. 線形計画モデルの作成

最もよく用いられる数理最適化問題が，線形最適化問題（線形計画問題）である．これは，目的関数$f(x)$が線形関数で表され，かつ，実行可能領域Sが線形式で表されたもののことを指す．次に示すのは，線形最適化問題の例である．

$$\begin{aligned} &\text{最小化} && 2x_1 + 3x_2 \\ &\text{制約} && 3x_1 + 4x_2 \geq 1, \\ & && x_1 \geq 0, x_2 \geq 0. \end{aligned} \tag{2}$$

ここでは，$S = \{(x_1, x_2): 3x_1 + 4x_2 \geq 1, x_1 \geq 0, x_2 \geq 0\}$であり，$S$は線形不等式の共通部分で表されている．

4.1 線形最適化パッケージ PuLP

Python を用いて線形最適化問題を解くには PuLP というパッケージを用いることができる（参考文

献・資料 1)．PuLP をインストールした後，Python のプログラムで PuLP の機能を使うには，冒頭に

```
from pulp import *
```

と書けばよい．

4.2 問題生成

PuLP を用いて線形最適化問題を解くには，まず，問題を生成する．そのためには，

```
prob = LpProblem("ex1",LpMinimize)
```

を実行する．

4.3 変数生成

次に，問題に現れる決定変数x_1, x_2を生成する．そのためには，

```
x1 = LpVariable("x1", lowBound=0,cat=LpContinuous)
x2 = LpVariable("x2", lowBound=0,cat=LpContinuous)
```

を実行する．最初の引数で変数の名前を指定し，2 番目の引数では lowBound を用いて下限を 0 と指定している．これにより，$x_1 \geq 0$が課される．3 番目の引数では cat を用いてこの変数を連続変数とすることを指定している．

4.4 目的関数生成

これらの変数を用いて，目的関数を生成する．そのためには，問題 prob に線形関数を「追加」する．線形関数を「追加」するには演算子 += を用いる．

```
prob += 2*x1 + 3*x2
```

4.5 制約式生成

最後に，制約式を生成する．これには，目的関数と同じく演算子 += を用いるが，今度は追加するのは線形関数ではなく線形不等式である．

```
prob += 3*x1 + 4*x2   >= 1
```

これで，(1)の線形最適化問題を、PuLP を用いて表すことができた．この線形最適化モデル prob は，print()文を用いて画面表示することができる．

```
print(prob)
```

表示結果は次のとおりである．

```
ex1:
MINIMIZE
2*x1 + 3*x2 + 0
SUBJECT TO
_C1: 3 x1 + 4 x2 >= 1

VARIABLES
x1 Continuous
x2 Continuous
```

この機能を使うと，意図どおりに問題が生成されているかを目視で確認できる．

4.6 線形最適化問題の一般形

線形最適化問題の一般形は, 次のように書かれる．

$$
\begin{aligned}
&\text{最小化} \quad \sum_{j=1}^{n} c_j x_j \\
&\text{制約} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (i = 1,2,\dots m), \\
&\qquad x_j \geq 0 \qquad\qquad\quad (j = 1,2,\dots,n).
\end{aligned}
$$

5. モデリング言語とソルバー

ここまで述べてきた PuLP は，モデリング言語と言われるものである．数理最適化のモデリング言語とは，数理最適化問題を人間の読みやすい・書きやすい形式で表現するための言語である．前節で述べた PuLP を用いた線形最適化問題の表現では，線形関数の表現が数式での表現とよく似ており，人間に読みやすい・書きやすい形式であることがわかる．線形最適化を扱えるモデリング言語には, PuLP の他に, ZMPL, GNU MathProg, AMPL, PICOS などがある．

モデリング言語はあくまで問題を表現するためのもので，それ自体には問題を解く機能はない．問題を解くには，別途，ソルバと呼ばれるソフトウェアを用意する必要がある．線形最適化問題を解くためのソルバに は，Cbc, Gurobi, Mosek などがある．Python パッケージとして PuLP をインストールすると, 同時に Cbc がインストールされる. したがって，PuLP を用いる際には, 別途ソルバをインストールする作業はなしで Cbc を用いることができる．

5.1 ソルバによる求解

さて，PuLP で表現した線形最適化モデル prob を

解くには，prob.solve()を実行すればよい．これにより，問題 prob の最適解が，指定されたソルバによって計算される．

```
status = prob.solve()
```

prob.solve()の返り値は，計算実行により至った終了状態を示す整数値である．この整数値は，LpStatus に引数として渡すことで，状態を示す文字列を得ることができる．

```
status=prob.solve()
print("終了状態:",LpStatus[status])
print("目的関数値:",value(prob.objective))
for var in prob.variables():
    print(var.name,":",var.varValue)
```

LpStatus[status]として Optimal か得られれば，最適解が得られたことがわかる．得られた最適値は，value(prob.objective)で得られる．モデル prob に含まれる変数には，prob.variable()でアクセス可能であるが，その要素 var の値は var.varValue で得られる．最適解が得られた状態であれば，そのときの最適解は，var.varValue で得られる．このプログラムを実行すると，下記の表示がされる．

```
終了状態: Optimal
目的関値: 0.66666666
x1 : 0.33333333
x2 : 0.0
```

これより，最適解が得られており，そのときの最適値は 2/3(=0.66666666), 最適解は$x_1 = \frac{1}{3}, x_2 = 0$であることがわかる．

6. 二次錐最適化問題

線形最適化問題の一般化として，二次錐最適化問題がある．この問題は，線形最適化問題よりもモデル化能力に優れ，かつ，最適解を得るための効率的なアルゴリズムが知られているので，様々な分野で用いられている．リスク管理の分野だと，線形最適化問題にロバスト性を取り入れたロバスト線形最適化問題は，パラメータの不確実性に楕円を仮定すると，二次錐最適化問題として定式可能であることが知られている（参考文献・資料 2)．

二次錐は，n次元ベクトルの集合であり，次で定義される．

$$\mathcal{K}_q = \left\{ \boldsymbol{x} \mid x_1 \geq \sqrt{x_2^2 + x_3^2 + \cdots + x_n^2} \right\}.$$

この記号を用いて，二次錐最適化問題は，次のように表される．

$$\begin{aligned} \text{最小化} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{制約} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (i = 1,2,\ldots m), \\ & \boldsymbol{x} \in \mathcal{K}_q. \end{aligned}$$

(1)の形に対応させると，実行可能領域Sは

$$S = \left\{ \boldsymbol{x} \mid \sum_{j=1}^{n} a_{ij} x_j \geq b_i \ (i = 1,2,\ldots,m), \boldsymbol{x} \in \mathcal{K}_q \right\}$$

となる．

次に示すのは，二次錐最適化問題の例である．

$$\begin{aligned} \text{最小化} \quad & x_1 + 2x_2 + 3x_3 \\ \text{制約} \quad & x_1 + 2x_2 + 2x_3 \leq 10, \\ & 3x_1 + x_3 \leq 11, \\ & 4x_1 + 3x_2 + 2x_3 \leq 20, \\ & x_1 \geq \sqrt{x_2^2 + x_3^2}. \end{aligned}$$

6.1 錐線形最適化パッケージ PICOS

二次錐最適化問題を Python で解くには，PICOS というパッケージを使うとよい．

6.2 二次錐最適化問題を解くプログラム

PICOS で二次錐最適化問題を生成して解くプログラムは，次のとおりである．

```
import picos as pic
pb=pic.Problem()
x=pic.RealVariable("x",3)
pb.add_constraint( x[0]+2*x[1]+2*x[2] <= 10 )
pb.add_constraint( 3*x[0]+x[2] <= 11 )
pb.add_constraint( 4*x[0]+3*x[1]+2*x[2] <= 20 )
pb.add_constraint( x[0]>=abs(x[1:3]) )
pb.set_objective("min",x[0]+2*x[1]+3*x[2])
solution=pb.solve()
```

```
print("終了状態:",solution.claimedStatus)
print("目的関数値:",round(pb.value,2))
print("最適解:",solution.primals)
```

このプログラムを実行すると，次の結果を得る．

```
終了状態: optimal
目的関数値: -13.47
最適解: {<3x1 Real Variable: x>:
[5.247680246068714,-2.2453757938834125,
-4.743040743311351]}
```

これより，最適解を求まり，最適値は-13.47 であることがわかる．

6.3　ロバスト線形最適化問題

パラメータa_{ij}に不確実性がある線形最適化問題は，ロバスト線形最適化問題とよばれる．ロバスト線形最適化問題では，制約を定める係数a_{ij}が不確実性集合$\mathcal{U}_i$の中に値をとる，という状況を扱う．数式で表すと，$\boldsymbol{a}_i = (a_{i1}, a_{i2}, \dots, a_{in}) \in \mathcal{U}_i$となる．この$\mathcal{U}_i$を, 不確実性集合という. 不確実性集合としては様々なものが考えられるが，その一つに楕円がある．具体的には，次のように表される集合である．

$$\mathcal{U}_i = \{\bar{\boldsymbol{a}}_i + P_i\boldsymbol{u} \mid \|\boldsymbol{u}\|_2 \le 1\}. \qquad (3)$$

ロバスト線形最適化問題では，この不確実性集合の中のどの値$\boldsymbol{a}_i$に対しても，制約式$\sum_{j=1}^{n} a_{ij}x_j = \boldsymbol{a}_i \cdot \boldsymbol{x} \le b_i$が満たされなければならない，という制約が課されている．これは，数式で表現すると，次のとおりとなる．

$$\begin{aligned} \text{最小化}\quad & \sum_{j=1}^{n} c_j x_j \\ \text{制約}\quad & \max\{\boldsymbol{a}_i \cdot \boldsymbol{x} \mid \boldsymbol{a}_i \in \mathcal{U}_i\} \le b_i \quad (i = 1,2,\dots,m), \\ & \boldsymbol{x} \in \mathcal{K}_q. \end{aligned}$$

ここで，最初の制約は，次のように表すことができる．

$$\max\{\boldsymbol{a}_i \cdot \boldsymbol{x} \mid \boldsymbol{a}_i \in \mathcal{U}_i\} = \bar{\boldsymbol{a}}_i \cdot \boldsymbol{x} + \|P_i^{\mathsf{T}}\boldsymbol{x}\|_2 \le b_i. \qquad (4)$$

この式の，右 2 つの項は，$\|P_i^{\mathsf{T}}\boldsymbol{x}\|_2 \le b_i - \bar{\boldsymbol{a}}_i \cdot \boldsymbol{x}$と等しく，これは，二次錐制約として表される．

次に，ロバスト線形最適化問題の問題例をとりあげる．まず，線形最適化問題の問題例を示す．

$$\begin{aligned} \text{最小化}\quad & -4x_1 + 9x_2 + 3x_3 \\ \text{制約}\quad & -x_1 + 3x_2 \le 7, \\ & 3x_1 + 3x_2 \le 3, \\ & 3x_2 + 2x_3 \le 6, \\ & -3x_1 - 5x_3 \le -5, \\ & x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{aligned} \qquad (5)$$

さて，この線形最適化問題に対するロバスト最適化問題を扱う．不確実性を定める行列P_1, P_2, P_3, P_4は，それぞれ次で与えられるとする．

$$P_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_2 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}, P_4 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

(3)におけるベクトル$\bar{\boldsymbol{a}}_i$は，$\bar{\boldsymbol{a}}_1 = [-1,3,0], \bar{\boldsymbol{a}}_2 = [3,3,0], \bar{\boldsymbol{a}}_3 = [0,3,2], \bar{\boldsymbol{a}}_4 = [-3,0,-5]$である．こうして，次のように PICOS による二次錐最適化モデルが定められる．

```
import cvxopt as cvx
A=pic.Constant("A",cvx.matrix([-1,3,0,-3,3,3,3,0,0,0,2,-5
],(4,3)))
b=pic.Constant("b",cvx.matrix([7,3,6,-5],(4,1)))
c=pic.Constant("c",cvx.matrix([-4,9,3],(3,1)))
P=[cvx.matrix([1,0,0,0,3,0,0,0,1],(3,3)),
cvx.matrix([2,0,0,0,1,0,0,0,2],(3,3)),
cvx.matrix([2,0,0,0,0.5,0,0,0,1],(3,3)),
cvx.matrix([1,0,0,0,2,0,0,0,1],(3,3))]
prob = pic.Problem()
x = pic.RealVariable ("x",3)
prob.add_list_of_constraints([abs(P[i]*x)<=b[i]-A[i,:]*x
for i in range(4)])
prob.add_list_of_constraints([x[i]>=0 for i in range(3)])
objective=c|x
prob.set_objective(“min”,objective)
solution = prob.solve()
print("status:", solution.claimedStatus)
print("objective value:", prob.value)
print(solution.primals)
```

このプログラムを実行すると, 次の結果が得られる．

```
status:  optimal
```

```
objective value: 2.060032365677577
{<3x1 Real Variable: x>: [0.27462203493052445, -3.
54048787281071e-11, 1.052840168572773]}
```

PICOS では，ベクトル$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$のノルム$\|x\|$を，abs(x)で表すことができる．したがって，$\|P_i^{\mathrm{T}}\boldsymbol{x}\|_2 \leq b_i - \bar{\boldsymbol{a}}_i \cdot \boldsymbol{x}$は，次の行で表される．

```
abs(P[i]*x)<=b[i]-A[i,:]*x
```

実行結果より，最適値として 2.06 が得られることがわかる．この最適値は，線形最適化問題(5)の最適値よりも悪い（大きい）が，P_iで定められる範囲で$\boldsymbol{a}_i$どのような値を実現しても，制約式(4)は破られることはない．

7.　ネットワーク最適化問題

他によく用いられる最適化問題として，ネットワーク上の最適化問題がある．ネットワークは，グラフともよばれる．それは，点集合Vと辺集合Eのペアとして，$G = (V, E)$と定義される．図 1 に示したのは，

$V = \{0,1,2,3,4,5\}$,
$E = \{\{0,1\},\{0,2\},\{1,3\},\{1,4\},$
$\{2,3\},\{2,4\},\{3,5\},\{4,5\}\}$

で定義されるネットワークである．各ネットワークには，コストが関連づけられている．そのコストは，各辺の脇に書かれている．ネットワーク上の最適化問題として最も身近なものは，最短路問題である．最適化問題は，ある点から他の点への最も費用の安い経路を求める問題である．その応用としては，地図アプリを用いた経路探索が挙げられる．

Python でネットワークを扱うパッケージとしては，NetworkX が便利である．NetworkX を用いると，図 1 のネットワークは次のよう表すことができる．

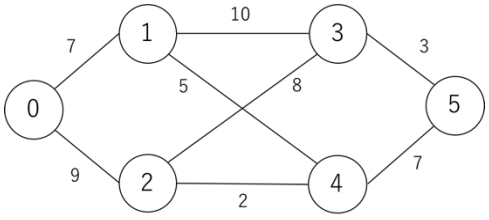

図 1：ネットワークの例

```
import networkx as nx
G=nx.Graph()
```

```
G.add_weighted_edges_from([(0,1,7),(0,2,9),(1,3,10),(1,4
,5),(2,3,8),(2,4,2),(3,5,3),(4,5,7)])
```

このグラフ上での最短路問題は，NetworkX の機能によって簡単に求めることができる．ある点 source から，それ以外の全ての点への最短路を求めるには，次の命令を実行すればよい．

```
p = nx.shortest_path(G, source=0)
```

これにより，source として指定した点 0 から，それ以外の点への最短路が求まる．このグラフ上での最短路問題は，NetworkX の機能によって簡単に求めることができる．求められた最短路は，print(p)によって表示されるが，その結果は次のとおりである．

```
{0: [0], 1:[0,1], 2:[0,2], 3:[0,1,3], 4:[0,1,4], 5:[0,1,3,5]}
```

ここで注意が必要なのは，shortest_path()の引数に特に指定をしなければ，辺の数が最小の路を最短路として求めることである．例えば，0 から 4 への最短路は，点 0,1,4 を順に辿るものであるが，このコストは，7+5=12 である．しかし，重みで比較すると，点 0,2,4 を順に辿る路のコスト 11 の方が小さい．ネットワーク G に設定した重みの総和を最小にする最短路を求めたければ，引数に明示する必要がある．

```
p = nx.shortest_path(G, source=0,weight="weight")
```

こうして求めた最短路は，次のとおりである．

```
{0: [0], 1:[0,1], 2:[0,2], 3:[0,1,3], 4:[0,2,4], 5:[0,2,4,5]}
```

この結果を見ると，点 4 と 5 への最短路として異なるものが得られている．

8.　まとめ

本稿では，Python を用いて，数理最適化問題を解く方法を述べた．Python を用いるメリットは，豊富なパッケージが使えることである．本稿で扱ったパッケージ以外にも，様々な機能を実現するパッケージが存在する．開発した数理モデルの動作をテストすることなどが簡単にできるので，活用をお勧めする．

参考文献・資料

1. Pulp documentation team (2009) , Optimizaiton with PuLP, https://coin-or.github.io/pulp/, 参照日: 2020 年 4 月 29 日.
2. Lobo, M.S., Vandenberghe, L., Boyd, S. and Lebret, H. (1998), Applications of second-order cone programming. Linear Algebra and its Applications, 284, 193—228.
3. Sagnol,G., Stahlberg, M. (2012), The PICOS Documentation, http://picos.zib.de, 参照日 : 2020 年 4 月 29 日.
4. NetworkX developers (2014), NetworkX – Network Analysis in Python, https://networkx.github.io/, 参照日:2020 年 4 月 29 日.