クレジット:
UTokyo Online Education 数理手法Ⅲ 2018 寒野善博

東京大学 THE UNIVERSITY OF TOKYO

CC BY NC ND

CVXPY による線形計画問題の解き方

この資料では，Python 上で動作する CVXPY [1] を用いて線形計画問題を解く方法を解説する．Python そのものの使い方については，たとえば文献 [3–5] などを参照されたい．CVXPY は凸計画問題のモデリングツールであり，最適化問題をほぼ数式の通りに記述すれば，それを最適化ソルバーの入力形式に変換した上でソルバーをよび出し，結果を返す，という動作をする．このようなツールは algebraic modeling language とよばれているが，同様のツールとして PICOS [7] や PuLP [6] などもよく知られている．また，CVXPY の Matlab 版に CVX [2] がある [*1]．これらのモデリングツールを利用すると，最適化問題を個々のソルバーの入力形式に合わせて変形する手間が不要になるので，便利である．ただし，解きたい問題によっては，モデリングツールによる問題の変換に大きな計算コストが費やされることもあるので，注意が必要である．

CVXPY のインストールの方法については，web ページ `http://www.cvxpy.org/install/` を参照されたい．

- この資料は，主に「最適化のプログラムを実際に動かしてみたい」という人向けの，補助資料です．講義の履修や単位の取得に，プログラミングは必要ではありません．
- 以下の実装例のファイルは，`https://www.or.mist.i.u-tokyo.ac.jp/kanno/lecture/` からダウンロードできます．
- プログラミングが自分でできなくても，実行してみると，実際の計算の感覚がつかめて有益です．

---

1. 生産計画問題

生産計画問題を取り上げて，CVXPY の使い方の例を示す．ある工場で，3 種類の原材料 M1, M2, M3 を用いて，2 種類の製品 P1, P2 を生産しているとする．このとき，各製品をそれぞれどれだけ生産すればより多くの利益が得られるかを考えるのが，生産計画問題である．

製品 P1, P2 の生産量をそれぞれ x_1 kg, x_2 kg とする．また，製品 1 kg を生産することで得られる利益は，P1, P2 のそれぞれに対して 70 万円, 30 万円 であるとする．つまり，総利益は $70x_1 + 30x_2$ で表される．さらに，各製品を 1 kg 生産するために必要な原材料 M1, M2, M3 の量を表 1 に示す．ただし，原材料の在庫はそれぞれ，M1 は 80 kg, M2 は 40 kg, M3 は 70 kg であるとする．実際に使用する原材料の量は在庫の量を超えないので，たとえば原材料 M1 の使用量について不等式

$$5x_1 + 2x_2 \leq 80$$

表 1: 製品 1 kg あたりの原材料の使用量

	M1	M2	M3
P1	5 kg	2 kg	3 kg
P2	2 kg	3 kg	7 kg

[*1] 先に開発されたのは CVX であるので，正確には，CVX の Python 版が CVXPY であるという位置づけになる．

が満たされなければならない．原材料 M2, M3 に関しても，同様の不等式制約を考慮する必要がある．また，各製品の生産量は非負であるから，$x_1 \geq 0$, $x_2 \geq 0$ が満たされなければならない．

以上より，利益を最大化する生産計画を求める問題は，線形計画問題

$$\begin{aligned}
&\text{Maximize} && 70x_1 + 30x_2 && \text{(1a)}\\
&\text{subject to} && 5x_1 + 2x_2 \leq 80, && \text{(1b)}\\
& && 2x_1 + 3x_2 \leq 40, && \text{(1c)}\\
& && 3x_1 + 7x_2 \leq 70, && \text{(1d)}\\
& && x_1 \geq 0, \quad x_2 \geq 0 && \text{(1e)}
\end{aligned}$$

として定式化できる．この問題を CVXPY で解くには，次のような Python のコードを記述すればよい．

```
1  # ファイル名：‘product_plan.py’
2  import cvxpy as cp                                  #まずはモジュールの読み込み
3  x1, x2 = cp.Variable(), cp.Variable()               #最適化の決定変数の定義
4  obj  = cp.Maximize( (70 * x1) + (30 * x2) )         #目的関数を記述
5  cons = [(5 * x1) + (2 * x2) <= 80,
6          (2 * x1) + (3 * x2) <= 40,
7          (3 * x1) + (7 * x2) <= 70,
8          x1 >= 0,
9          x2 >= 0]                                    #制約条件を記述
10 P = cp.Problem(obj, cons)                           #最適化問題を定義
11 P.solve(verbose=True)                               #求解
12 print("x1␣=␣", x1.value)                            #ここから最適解の出力
13 print("x2␣=␣", x2.value)
```

以下に，‘product_plan.py’ の実行結果の様子を示す．

```
>>>
ECOS 1.1.1 - (c) A. Domahidi, ETH Zurich & embotech 2012-15. Support: ecos@embotech.com

It     pcost       dcost      gap   pres   dres    k/t    mu     step    IR
 0  -8.130e+02  -2.121e+03  +6e+02  4e-04  8e-01  1e+00  1e+02   N/A    1 1 -
 1  -1.123e+03  -1.369e+03  +1e+02  8e-05  2e-01  1e+01  2e+01  0.8758  0 0 0
 2  -1.126e+03  -1.136e+03  +6e+00  3e-06  7e-03  4e-01  1e+00  0.9575  0 0 0
 3  -1.127e+03  -1.128e+03  +2e-01  1e-07  3e-04  2e-02  4e-02  0.9736  1 0 0
 4  -1.127e+03  -1.127e+03  +2e-03  1e-09  3e-06  2e-04  5e-04  0.9890  0 0 0
 5  -1.127e+03  -1.127e+03  +3e-05  2e-11  3e-08  2e-06  5e-06  0.9890  1 0 0
 6  -1.127e+03  -1.127e+03  +3e-07  2e-13  4e-10  2e-08  6e-08  0.9890  1 0 0

OPTIMAL (within feastol=3.7e-10, reltol=2.7e-10, abstol=3.1e-07).
Runtime: 0.016330 seconds.

('x1 = ', 14.545454549604761)
('x2 = ', 3.6363636253616609)
>>>
```

つまり，製品 P1 を 14.55 kg, P2 を 3.64 kg だけ生産すればよいことが分かる．

問題の規模が大きくなってくると（つまり，決定変数や制約式の数が多くなってくると），問題 (1) を行列やベクトルを用いて次のように表現した方が便利である：

$$\text{Maximize} \quad \begin{bmatrix} 70 \\ 30 \end{bmatrix}^{\top} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2a}$$

$$\text{subject to} \quad \begin{bmatrix} 5 & 2 \\ 2 & 3 \\ 3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 80 \\ 40 \\ 70 \end{bmatrix}, \tag{2b}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq \mathbf{0}. \tag{2c}$$

この形式に対応した Python のコードは，以下のようになる．

```
# ファイル名：'product_plan_matrix.py'
import cvxpy as cp                          #まずはモジュールの読み込み
import numpy as np
x = cp.Variable(2,1)                        #最適化の決定変数の定義
A = np.matrix([                             #ここから問題のデータを記述
    [5.0, 2.0],
    [2.0, 3.0],
    [3.0, 7.0]])
b = [80.0, 40.0, 70.0]
c = np.matrix([ [70.0, 30.0] ]).T
obj  = cp.Maximize( c.T * x )               #目的関数を記述
cons = [A * x <= b,
        x >= 0]                             #制約条件を記述
P = cp.Problem(obj, cons)                   #最適化問題を定義
P.solve(verbose=True)                       #求解
print(x.value)                              #最適解の出力
```

2. 輸送問題

ある会社では m 個の倉庫 $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_m$ に商品を収容している．倉庫から運び出された商品は，n 個の消費地 $\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_n$ に輸送される．このときに，輸送費用を最小にするような輸送計画を考える．

倉庫 S_i から出荷できる商品の量（供給量）を p_i とおく．また，消費地 D_j に納品するべき商品の量（需要量）を q_j とおく．ただし，供給の総量と需要の総量は等しいと仮定する．つまり，$\sum_{i=1}^{m} p_i = \sum_{j=1}^{n} q_j$ が成り立つものとする．さらに，倉庫 S_i から消費地 D_j に輸送するときに，商品の単位量あたりにかかる輸送費用を r_{ij} とおく．

倉庫 S_i から消費地 D_j への輸送量を変数 x_{ij} で表す．需要と供給の制約条件を満たしながら，輸送

費用を最小化するような輸送量を求める問題は，次のように定式化される：

$$\text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}x_{ij} \tag{3a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij} = p_i, \quad i = 1,\ldots,m, \tag{3b}$$

$$\sum_{i=1}^{m} x_{ij} = q_j, \quad j = 1,\ldots,n, \tag{3c}$$

$$x_{ij} \geq 0, \qquad i = 1,\ldots,m;\ j = 1,\ldots,n. \tag{3d}$$

この問題は，制約条件や目的関数がすべて x_{ij} の 1 次関数で表されているから，線形計画問題である．

具体例として，$m = 2$, $n = 3$ の場合を考える．問題のデータが

$$\boldsymbol{p} = (p_i) = \begin{bmatrix} 20 \\ 15 \end{bmatrix}, \quad \boldsymbol{q} = (q_j) = \begin{bmatrix} 8.5 \\ 12.5 \\ 14 \end{bmatrix}, \quad R = (r_{ij}) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 8 & 7 \end{bmatrix}$$

と与えられたとする．このとき，CVXPY で輸送問題を解くには，次のような Python のコードを記述すればよい．

```
# ファイル名：transportation_problem.py
import cvxpy as cp                          #まずはモジュールの読み込み
import numpy as np
m, n = 2, 3                                 #ここから問題のデータを記述
p = [20.0, 15.0]
q = [8.5, 12.5, 14.0]
R = np.matrix([
    [1.0, 2.0, 3.0],
    [4.0, 8.0, 7.0] ])
r = np.reshape(R.T, (m*n,1));
A = np.kron(np.ones((1,n)), np.eye(m))
B = np.kron(np.eye(n), np.ones((1,m)));
x = cp.Variable(m*n, 1)                     #最適化の決定変数の定義
obj  = cp.Minimize( r.T * x )               #目的関数を記述
cons = [A * x == p,
        B * x == q,
        x >= 0]                             #制約条件を記述
P = cp.Problem(obj, cons)                   #最適化問題を定義
P.solve(verbose=True)                       #求解
print(x.value)                              #ここから最適解の出力
```

3. CVX を用いた Matlab コードの例

CVX [2] は，CVXPY と同様の使い方ができる Matlab のツールボックスである．たとえば，CVX 用いて問題 (1) を解くには，次のような Matlab コードを用意すればよい．

```
% ファイル名：product_plan.m
cvx_begin                                   %CVXの起動
    variable x1                             %ここから最適化の決定変数の定義
    variable x2
    maximize( (70 * x1) + (30 * x2) )       %目的関数の記述
    subject to                              %ここから制約条件を記述
        (5 * x1) + (2 * x2) <= 80;
        (2 * x1) + (3 * x2) <= 40;
        (3 * x1) + (7 * x2) <= 70;
        x1 >= 0;
        x2 >= 0;
cvx_end                                     %求解
fprintf('␣x1␣=␣%7.4f␣\n', x1);              %ここから最適解の出力
fprintf('␣x2␣=␣%7.4f␣\n', x2);
```

問題 (2) の形式では，Matlab コードは次のように書けばよい．

```
% ファイル名：product_plan_matrix.m
A = [5, 2;                                  %ここから問題のデータを記述
     2,  3;
     3,  7];
b = [80; 40; 70];
c = [70; 30];
cvx_begin                                   %CVXの起動
    variable x(2,1)                         %最適化の決定変数の定義
    maximize( c' * x )                      %目的関数の記述
    subject to                              %ここから制約条件を記述
        A * x <= b;
        x >= 0;
cvx_end                                     %求解
for j=1:2                                   %ここから最適解の出力
    fprintf('␣x%g␣=␣%7.4f␣\n', j, x(j));
end
```

問題 (3) を解く Matlab コードは，次のように書ける．

```
% ファイル名：product_plan_matrix.m
p = [20; 15];                           %ここから問題のデータを記述
q = [8.5; 12.5; 14];
R = [1, 2,  3;
     4,  8,  7];
m = size(R, 1);
n = size(R, 2);
r = reshape(R, m*n, 1);
A = kron(ones(1,n), eye(m));
B = kron(eye(n), ones(1,m));
cvx_begin                               %CVXの起動
    variable x(m*n, 1)                  %最適化の決定変数の定義
    minimize( r' * x )                  %目的関数の記述
```

```
    subject to                          %ここから制約条件を記述
        A * x == p;
        B * x == q;
        x >= 0;
cvx_end                                 %求解
for j=1:n                               %ここから最適解の出力
    for i=1:m
        fprintf('␣x(%g,%g)␣=␣%7.4f\n', i, j, x(m*(j-1)+i));
    end
end
```

参考文献

[1] S. Diamond, E. Chu, A. Agrawal, S. Boyd: *Welcome to CVXPY 1.0—CVXPY 1.0.8 Documentation*. http://www.cvxpy.org/ (Accessed September 2018).

[2] M. Grant, S. Boyd: *CVX: Matlab Software for Disciplined Convex Programming*. http://cvxr.com/cvx (Accessed September 2018).

[3] J. V. Guttag: *Introduction to Computation and Programming Using Python: With Application to Understanding Data (2nd ed.)*, MIT Press, Cambridge (2016). J. V. Guttag（著），久保 幹雄（監訳），麻生 敏正，木村 泰紀，小林 和博，関口 良行，並木 誠，藤原 洋志（訳）:『Python 言語によるプログラミングイントロダクション 第 2 版—データサイエンスとアプリケーション』，近代科学社 (2017).

[4] 久保 幹雄，J. P. ペドロソ，村松 正和，A. レイス:『あたらしい数理最適化—Python 言語と Gurobi で解く』，近代科学社 (2012).

[5] 並木 誠 :『Python による数理最適化入門』，朝倉書店 (2018).

[6] PuLP Documentation Team: *Optimization with PuLP*. https://pythonhosted.org/PuLP/ (Accessed September 2018).

[7] G. Sagnol: *The PICOS Documentation*. https://picos-api.gitlab.io/picos/ (Accessed September 2018).

（以上）