Pythonによる数理最適化モデルの実装

小林　和博

本稿では，Python 言語を用いて，数理最適化の中でも凸最適化（特に線形最適化，2 次錐最適化，半正定値最適化），およびネットワーク最適化を実現する方法を述べる．数理最適化のためのモデリングツールを利用することで，人間が読みやすい形式で迅速にモデルを開発することが可能になる．

キーワード：Python 言語，CVXOPT，PICOS，NetworkX

1. はじめに

本稿では，Python 言語を用いて数理最適化を実現するための方法を述べる．具体的には，凸最適化問題を解くための方法と，ネットワーク最適化問題を解くための方法を取り上げる．

2. 凸最適化パッケージ CVXOPT

CVXOPT は，凸最適化 (convex optimization) のための Python パッケージである [1]．CVXOPT では，線形最適化，2 次錐最適化，半正定値最適化を実行するためのルーチンが提供されている．CVXOPT は，pip によりインストールすることができる．

```
$ pip install cvxopt
```

CVXOPT には，疎行列と密行列を扱う効率的なクラスが用意されている．たとえば，密行列 $\boldsymbol{A}$ を定義するには，CVXOPT の matrix を用いて，次のようにするとよい：

```
from cvxopt import matrix
A=matrix([1,2,3,4,5,6],(2,3))
```

これにより 2 行 3 列の行列

$$\boldsymbol{A}=\begin{bmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{bmatrix}$$

が定義される．場合によっては，行列の要素を列ごとに指定することが便利である．CVXOPT には列のリストを指定して行列を定めることもできる．この場合は行数と列数を指定する必要はない．

```
from cvxopt import matrix
A=matrix([[1,2],[3,4],[5,6]])
```

こばやし　かずひろ
東京理科大学理工学部
〒 278–8510 千葉県野田市山崎 2641
kkoba@rs.tus.ac.jp

疎行列は，spmatrix を用いて定める．次の行列

$$\boldsymbol{B}=\begin{bmatrix}2 & 0 & 3 & 0\\ 0 & 5 & 0 & 0\end{bmatrix}$$

を疎行列として扱うことにする．この行列 $\boldsymbol{B}$ の三つの非零成分を（行番号，列番号，値）の形式で書くと，$(0,0,2),(0,2,3),(1,1,5)$ となる．ただし，最初の行と列はそれぞれ 0 行目，0 列目とする．これらの行番号だけを取り出したリスト $(0,0,1)$，列番号だけを取り出したリスト $(0,2,1)$，値だけを取り出したリスト $(2,3,5)$ を用いて，CVXOPT の spmatrix として次のように定義する．

```
from cvxopt import spmatrix
B=spmatrix([2,3,5],[0,0,1],[0,2,1],(2,4))
```

`spmatrix()` の最初の引数は値のリスト，2 番目の引数は行番号のリスト，3 番目の引数は列番号のリスト，そして 4 番目の引数は，行数と列数を要素とするタプルである．

2.1 線形最適化

では，CVXOPT を用いて，次の線形最適化問題 (LP) を解いてみよう．

$$\begin{aligned}\min\ & -x_1-x_2\\ \text{s.t. }& x_1+2x_2\le 4,\\ & 4x_1+2x_2\le 12,\\ & -x_1+x_2\le 1,\\ & x_1\ge 0, x_2\ge 0.\end{aligned}$$

線形最適化問題の標準形を

$$\begin{aligned}\min\ & \boldsymbol{c}\cdot\boldsymbol{x}\\ \text{s.t. }& \boldsymbol{a}_i\cdot\boldsymbol{x}\le b_i\ (i=1,2,\ldots,m),\\ & \boldsymbol{x}\ge\boldsymbol{0},\end{aligned}$$

と書くことにする．ここで，m は制約式の数を表す．いま，b_i を成分とするベクトルを $\boldsymbol{b}=[b_1,b_2,\ldots,b_m]^\top$ と定めると，前の問題例を表すベクトル $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3,\boldsymbol{b},\boldsymbol{x}$

そして $\boldsymbol{c}$ は，それぞれ次のようになる：

$$\boldsymbol{a}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \boldsymbol{a}_2 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}, \boldsymbol{a}_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix},$$

$$\boldsymbol{b} = \begin{bmatrix} 4 \\ 12 \\ 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

また，ベクトル $\boldsymbol{a}_i^\top$ を第 i 行目とする

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 4 & 2 \\ -1 & 1 \end{bmatrix}$$

を用いると，同じ問題を次のように表すことができる：

$$\begin{aligned} &\min \boldsymbol{c} \cdot \boldsymbol{x} \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \\ &\quad \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned}$$

そこで，これら $\boldsymbol{A}$, $\boldsymbol{b}$ と $\boldsymbol{c}$ を，CVXOPT の行列として次のように定義する．

```
from cvxopt import matrix,solvers
A=matrix([ [1.,4.,-1.],[2.,2.,1.] ])
b=matrix([4.,12.,1.])
c=matrix([-1.,-1.])
```

これで，問題を表す行列とベクトルが定まったので，線形最適化問題を解くことができる．そのためには，次の命令を実行する．

```
sol=solvers.lp(c,A,b)
```

これを実行すると，CVXOPT が内点法を実行し，その出力メッセージから最適解が得られたことがわかる．内点法は，適当な内点初期解から始めて反復的に最適解に近づいていく解法であるが，出力結果には，各反復における内点の状態が示される．

```
sol=solvers.lp(c,A,b)
pcost dcost gap pres dres k/t
0: -3.6667e+00 -3.6667e+00 1e+00 3e-01 0e
    +00 1e+00
1: -3.3103e+00 -3.3365e+00 8e-02 2e-02 7e
    -16 2e-02
2: -3.3331e+00 -3.3334e+00 9e-04 2e-04 2e
    -16 2e-04
3: -3.3333e+00 -3.3333e+00 9e-06 2e-06 0e
    +00 2e-06
4: -3.3333e+00 -3.3333e+00 9e-08 2e-08 3e
    -16 2e-08
Optimal solution found.
```

4: と記された最後の反復で gap が十分に小さくなっており，反復を終えている．内点法の実行結果は，solvers.lp() の返り値として得られる．たとえば，実行の結果として最適解が得られたかどうかは，sol['status'] の値によって確認できる．この問題例では，'optimal' が返ってくるので，最適解が得られていることがわかる．返り値 sol は辞書であり，状態のほかにもさまざまな値をもっている．たとえば，最適解の値も，キー x の値としてもっている．上の命令で得られた最適解を見るには，次のようにするとよい．

```
print(sol['x'])
```

そうすると，次の表示が得られる．

```
[ 2.67e+00]
[ 6.67e-01]
```

これより，最適解は，$x_1 = 2.67, x_2 = 0.667$ であることがわかる．

CVXOPT は，モデリングの機能をもっている．その機能を用いると，人間が見てわかりやすい数式に近い形で問題を記述することができる．上記の LP の問題例をモデリングの機能を用いて解くには，次のようにする．

```
from cvxopt.modeling import op,variable
x1=variable ()
x2=variable ()
c1=(x1+2*x2<=4)
c2=(4*x1+2*x2<=12)
c3=(-x1+x2<=1)
c4=(x1>=0)
c5=(x2>=0)
lp1=op(-x1-x2,[c1,c2,c3,c4,c5])
lp1.solve ()
```

variable() で変数 x1,x2 を生成し，これらの変数を用いて制約 c1,c2,c3 を定義する．ここでは，変数の非負条件 x1>=0,x2>=0 も制約 c4,c5 として定義することに注意する．変数と制約が定義できたら，目的関数と制約式のリストを引数として与えて op(-x1-x2,[c1,c2,c3,c4,c5]) を実行することで，線形最適化問題が定義できる．定義した線形最適化問題を解くには，lp1.solve() を実行する．実行結果の状態は lp1.status で確認できるが，これは optimal となっており，最適解が得られたことがわかる．得られた最適解の値を表示するには，

```
print(x1.value,x2.value)
```

を実行すればよい．先ほどの行列とベクトルを指定して解いた場合と同じ解が得られていることがわかる．

2.2 2 次錐最適化

$\boldsymbol{x} = (x_0, \boldsymbol{x}_1) \in R \times R^{n-1}$ としたとき，次の集合

$$\mathcal{K}(n) \equiv \{(x_0, \boldsymbol{x_1}) \,|x_0 \geq \|\boldsymbol{x_1}\|\}$$

を，2 次錐という．2 次錐最適化問題は，変数ベクトル $\boldsymbol{x} \in R^n$ が 2 次錐 $\mathcal{K}(n)$ に入っており，かつ制約式 $\boldsymbol{a}_i \cdot \boldsymbol{x} = b_i$ を満たすという条件のもとで，目的関数 $\boldsymbol{c} \cdot \boldsymbol{x}$ を最小化する問題である：

$$\begin{aligned} &\min\ \boldsymbol{c} \cdot \boldsymbol{x} \\ &\text{s.t. } \boldsymbol{a}_i \cdot \boldsymbol{x} = b_i\ (i = 1, 2, \ldots, m), \\ &\qquad \boldsymbol{x} \in \mathcal{K}(n). \end{aligned} \tag{1}$$

m 本の制約式を行列を用いてまとめて書くためには，$\boldsymbol{a}_i^\top$ を第 i 行とする行列 $\boldsymbol{A}$ を用いて次のように書くとよい：

$$\begin{aligned} &\min\ \boldsymbol{c} \cdot \boldsymbol{x} \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ &\qquad \boldsymbol{x} \in \mathcal{K}(n). \end{aligned} \tag{2}$$

CVXOPT を用いると，2 次錐最適化問題を記述し，解くことができる．2 次錐最適化問題に対しては，モデルや定式化の意味を読み取るためには，問題 (1) よりもその双対問題の形で表すほうがよい場合がある．その双対問題は，次の形で表される：

$$\begin{aligned} &\max\ \boldsymbol{b} \cdot \boldsymbol{y} \\ &\text{s.t. } \boldsymbol{z} = \boldsymbol{c} - \sum_{i=1}^{m} \boldsymbol{a}_i y_i \in \mathcal{K}(n). \end{aligned} \tag{3}$$

双対問題も，主問題の表現で用いた行列 $\boldsymbol{A}$ の転置行列 $\boldsymbol{A}^\top$ を用いて，次のように表すことができる：

$$\begin{aligned} &\max\ \boldsymbol{b} \cdot \boldsymbol{y} \\ &\text{s.t. } \boldsymbol{z} = \boldsymbol{c} - \boldsymbol{A}^\top \boldsymbol{y} \in \mathcal{K}(n). \end{aligned} \tag{4}$$

2 次錐最適化の問題例として次のものを挙げる．

$$\begin{aligned} &\max\ -2y_1 - y_2 - 3y_3 \\ &\text{s.t. } \begin{bmatrix} 10 \\ 7 \\ 5 \end{bmatrix} - y_1 \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix} - y_2 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - y_3 \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \in \mathcal{K}(3). \end{aligned}$$

この 2 次錐最適化問題を CVXOPT で解くには，次のようにベクトル $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{a}_i$ を指定して，`solvers.socp()` に渡せばよい：

```
from cvxopt import matrix,solvers
b=matrix([-2.,-1.,-3.])
c=[ matrix([10.,7.,5.]) ]
At=[ matrix([
     [2.,3.,0.],[1.,1.,1.],[1.,1.,2.] ]) ]
sol=solvers.socp(-b,Gq=At,hq=c)
```

ここで，`solvers.socp()` の引数として `b,c,At` を渡すときに，`solvers.socp(-b,Gq=At,hq=c)` として渡した．これは，問題 (3) とは異なる標準形を採っている CVXOPT の入力形式に合わせるためである．CVXOPT の 2 次錐最適化の標準形は，公式ホームページにあるドキュメントで確認してもらいたい [1].

2.3 半正定値最適化

半正定値最適化問題 (SemiDefinite Optimization Problem, SDP) は，変数行列 $\boldsymbol{X} \in \mathcal{S}^n$ が半正定値であり，かつ制約式 $\boldsymbol{A}_i \bullet \boldsymbol{X} = b_i\ (i = 1, 2, \ldots, m)$ を満たすという条件のもとで，目的関数 $\boldsymbol{C} \bullet \boldsymbol{X}$ を最小化する問題である：

$$\begin{aligned} &\min\ \boldsymbol{C} \bullet \boldsymbol{X} \\ &\text{s.t. } \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i\ (i = 1, 2, \ldots, m), \\ &\qquad \boldsymbol{X} \succeq \boldsymbol{O}. \end{aligned} \tag{5}$$

ここで，$\mathcal{S}^n$ は $n \times n$ 対称行列の空間を，$\boldsymbol{X} \succeq \boldsymbol{O}$ は行列 $\boldsymbol{X}$ が半正定値であることを，$\boldsymbol{X} \bullet \boldsymbol{Y}$ は行列の内積を表す．ただし，二つの行列 $\boldsymbol{A}, \boldsymbol{B} \in R^{m \times n}$ の内積は

$$\boldsymbol{A} \bullet \boldsymbol{B} = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} B_{ij}$$

で定義される．この半正定値最適化問題の双対問題は，次で表される：

$$\begin{aligned} &\max\ \boldsymbol{b} \cdot \boldsymbol{y} \\ &\text{s.t. } \boldsymbol{Z} = \boldsymbol{C} - \sum_{i=1}^{m} \boldsymbol{A}_i y_i \succeq \boldsymbol{O}. \end{aligned} \tag{6}$$

CVXOPT を用いると，半正定値最適化問題を解くことができる．ここでは，問題例として次のものを用いる：

$$\begin{aligned} &\max\ -y_1 + y_2 - y_3 \\ &\text{s.t. } \begin{bmatrix} 33 & -9 \\ -9 & 26 \end{bmatrix} - y_1 \begin{bmatrix} -7 & -11 \\ -11 & 3 \end{bmatrix} \\ &\qquad - y_2 \begin{bmatrix} 7 & -18 \\ -18 & 8 \end{bmatrix} - y_3 \begin{bmatrix} -2 & -8 \\ -8 & 1 \end{bmatrix} \succeq \boldsymbol{O}. \end{aligned}$$

この問題例を CVXOPT で解くには，次のようにする：

```
from cvxopt import matrix,solvers
b=matrix([-1.,1.,-1.])
A=[ matrix([ [-7.,-11.,-11.,3.],[
     7.,-18.,-18.,8.],[-2.,-8.,-8.,1.] ]) ]
C=[ matrix([ [33.,-9.],[-9.,26.] ]) ]
sol=solvers.sdp(-b,Gs=A,hs=C)
```

`solvers.sdp()` が SDP を解くための命令である．この命令の第 1 引数にベクトル $\boldsymbol{b}$ を，第 2 引数に行列 $\boldsymbol{A}$

を，第3引数に行列 $\boldsymbol{C}$ を指定している．CVXOPT では SDP の標準形として次のものを採用している：

$$\begin{aligned} \min\ & -\boldsymbol{b}\cdot\boldsymbol{y} \\ \text{s.t. } & \boldsymbol{G_s y}+\mathbf{vec}(\boldsymbol{Z})=\mathbf{vec}(\boldsymbol{h_s}), \\ & \boldsymbol{Z}\succeq\boldsymbol{O}. \end{aligned} \tag{7}$$

ここで，$\mathbf{vec}(\boldsymbol{A})$ は，行列 $\boldsymbol{A}\in R^{m\times n}$ の n 本の m 次元列ベクトルを縦に並べた mn 次元の列ベクトル

$$\mathbf{vec}(\boldsymbol{A})\equiv(A_{11},\ldots,A_{m1},\ldots,A_{1n},\ldots,A_{mn})^{\top}$$

を表す．また，$\boldsymbol{G_s}$ は，問題 (6) における制約行列 $\boldsymbol{A}_i$ に関して $\mathbf{vec}(\boldsymbol{A}_i)$ を第 i 列目とする行列である．前の問題例では，

$$\boldsymbol{G_s}=\begin{bmatrix} -7 & 7 & -2 \\ -11 & -18 & -8 \\ -11 & -18 & -8 \\ 3 & 8 & 1 \end{bmatrix} \tag{8}$$

となる．そこで，`solvers.sdp()` の第1引数として`-b` を，そして行列 $\boldsymbol{G_s}$ と $\boldsymbol{h_s}$ を与える引数として `Gs=A` と `hs=C` を与えている．

3. 最適化インターフェイス PICOS

PICOS は，錐最適化・整数最適化ソルバのための Python インターフェイスである [2]．高次元の行列変数を自然な形で扱うことができ，半正定値最適化問題や 2 次錐最適化問題を容易に記述することができる．加えて，ネットワーク構造を容易に記述する機能ももっている．PICOS は複数の異なるソルバを用いて最適化をすることができる．使用可能なソルバは，`picos.tools.available_solver()` により確認することができる．錐最適化問題を解くには錐最適化問題を解く機能をもつソルバが，混合整数最適化問題を解くには混合整数最適化問題を解く機能をもつソルバが利用可能でなければならない．解きたい問題を解く機能をもつソルバが，`picos.tools.available_solver()` の出力結果に含まれているかをあらかじめ確認しておく必要がある．

3.1 線形最適化

PICOS で線形最適化の問題例を定義するには，CVXOPT で用いた行列 `A` とベクトル `b`, `c` を用いればよい．PICOS によって， 2.1 節で用いた線形最適化の問題例を定義して解くには，次のようにすればよい：

```
from cvxopt import matrix
import picos as pic
A=matrix([ [1.,4.,-1.],[2.,2.,1.] ])
b=matrix([4.,12.,1.])
c=matrix([-1.,-1.])

P = pic.Problem ()
A = pic.new_param('A', A)
b = pic.new_param('b',b)
x = P.add_variable('x',2)
P.add_constraint(A*x<b)
objective = sum([c[i]*x[i] for i in range
    (2)])
P.set_objective('min',objective)
sol=P.solve ()
```

これを実行すると，PICOS は CVXOPT を呼び出して問題を解き，その反復の結果が表示される．

まず，`Problem()` によって問題を生成する．続いて，問題例を定めるための行列 `A` と `b` を `new_param()` によって定める．それに続く `P.add_variable('x',2)` は，問題例 `P` の変数を生成する命令である．最初の引数`'x'` は，変数名を指定する．2番目の引数は変数の次元を指定する．ここでは 2 を指定しているので，`x` は 2 次元ベクトルとなる．`P.add_constraint(A*x<b)` は問題例 `P` に制約式 `A*x<b` を追加する命令である．

ここで特徴的なのは，まず CVXOPT の行列として定義した `A` を，`A=pic.new_param('A',A)` という命令によって PICOS のパラメータにしているところである．`matrix()` コマンドによって `A` を定義した直後に，`type(A)` を実行して `A` の型を確認すると，`A` は CVXOPT の base.matrix であることがわかる．

```
A=matrix([ [1.,4.,-1.],[2.,2.,1.] ])
type(A)
```

`type(A)` の実行の結果，次の表示が得られる．

```
<class 'cvxopt.base.matrix'>
```

一方，PICOS の `new_param()` によって PICOS のパラメータとした後に `type(A)` を実行して `A` の型を確認すると，今度は PICOS でアフィン表現を表す `expression.AffinExp` となっていることがわかる：

```
A=matrix([ [1.,4.,-1.],[2.,2.,1.] ])
A=pic.new_param('A',A)
```

`type(A)` の実行の結果，次の表示が得られる．

```
<class 'picos.expression.AffinExp'>
```

`A` をただの行列ではなくアフィン表現とすることで，演算のオーバーロードを用いてさまざまなモデル化を容易に行うことができる．

3.2 2 次錐最適化

アフィン表現 $\boldsymbol{x}$ が PICOS で `x` と表されているとき，

$\boldsymbol{x}$ のユークリッドノルム $\sqrt{\boldsymbol{x}\cdot\boldsymbol{x}}$ を `abs(x)` で求めることができる．

2.2 節で解いた 2 次錐最適化の問題例を，PICOS で解くためのコードは次のとおりである．ただし，定式化としては主問題 (1) を用いるために，いったん行列 $\boldsymbol{A}$ の転置行列 $\boldsymbol{A}^\top$ として定義したデータ `At` に，転置を施して `At.T` とし，それを行列 $\boldsymbol{A}$ を表すデータ `A` として定めた．

```
import picos as pic
from cvxopt import matrix
P = pic.Problem ()
x = P.add_variable('x',3)
c=matrix([10.,7.,5.])
b=matrix([-2.,-1.,-3.])
At=matrix([
     [2.,3.,0.],[1.,1.,1.],[1.,1.,2.] ])
At=pic.new_param('At',At)
A=At.T
A=pic.new_param('A',A)
objective=sum([c[i]*x[i] for i in range
     (3)])
P.add_constraint(A*x<b)
P.add_constraint(abs(x[1:])<x[0])
P.set_objective('min',objective)
P.solve ()
```

2 次錐制約は `P.add_constraint(abs(x[1:])<x[0])` によって追加される．`x[1:]` は Python のスライスと呼ばれる表現で，変数 `x` の 2 番目から最後の要素までを表す．数式で表すと，ベクトル $\boldsymbol{x} = (x_0, \boldsymbol{x_1}) \in R \times R^{n-1}$ に対して $\boldsymbol{x_1}$ を表す．したがって，`abs(x[1:])` は $\sqrt{\boldsymbol{x_1}\cdot\boldsymbol{x_1}}$ を表す．これらより，`P.add_constraint(abs(x[1:])<x[0])` は 2 次錐制約を表すことがわかる．

3.3 半正定値最適化

PICOS はモデル化のためのさまざまな機能をもっており，最大カット問題の半正定値最適化緩和も数式に近い形で記述することができる．最大カット問題は，重みつき無向グラフ $G(V,E)$ のノード集合 V の部分集合 $X \subset V$ で，各枝 $e_{ij} \in E$ $(i,j \in V, i \neq j)$ に与えられた非負の重み w_{ij} に関して，

$$\sum_{i\in X, j\in V\setminus X} w_{ij}$$

を最大にするものを求める問題である．

最大カット問題に対して，半正定値最適化による緩和が提案されている．各ノードに対して変数を $\boldsymbol{x} \in \{1,-1\}^{|V|}$ と定義する．ノード i が集合 X に入っているか否かによって，変数 x_i が 1 をとるか -1 をとるかが決まる．こうすると，カットの値は，$\frac{1}{4}\boldsymbol{x}\cdot(\boldsymbol{L}\boldsymbol{x})$ で表される [3]．ここで，行列 $\boldsymbol{L}$ はグラフ G のラプラシアンである．なお，グラフ G のラプラシアンとは，行列 $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ で，$\boldsymbol{A}$ が隣接行列，$\boldsymbol{D}$ はノードの重みつき次数を成分とする対角行列である．

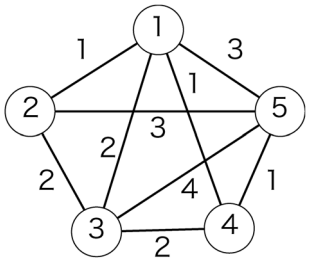

図 1　最大カット問題の問題例

ベクトル $\boldsymbol{x}$ に対して行列 $\boldsymbol{X}$ を $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^\top$ と定義すると，任意の行列 $\boldsymbol{A}$ に対して，$\boldsymbol{x}\cdot(\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{A}\bullet\boldsymbol{X}$ が成り立つこと，そして $x_i^2 = 1$ $(i \in V)$ であることに注意すると，最大カット問題は次の最適化問題として表される：

$$\begin{aligned} \max_{\boldsymbol{X}}\ & \frac{1}{4}\boldsymbol{L}\bullet\boldsymbol{X} \\ \text{s.t. } & \operatorname{diag}(\boldsymbol{X}) = \boldsymbol{1}, \\ & \boldsymbol{X} \succeq \boldsymbol{O}, \\ & \boldsymbol{X} \text{ は階数 } 1. \end{aligned}$$

ここで，$\operatorname{diag}(\boldsymbol{X})$ は行列 $\boldsymbol{X}$ の対角成分からなるベクトル，$\boldsymbol{1}$ はすべての成分が 1 のベクトルとする．この最適化問題から「$\boldsymbol{X}$ は階数 1」という制約を緩和した (=取り除いた) 最適化問題は，半正定値最適化問題となる．これが最大カット問題の半正定値最適化緩和である．

図 1 に示したグラフ上での最大カット問題に対する半正定値最適化緩和を PICOS で定式化して解くためのコードを示す：

```
import cvxopt as cvx
import picos as pic
import cvxopt.lapack
import numpy as np
import networkx as nx
G=nx.Graph ()
G.add_edge('1','2',weight=1)
G.add_edge('1','3',weight=2)
G.add_edge('1','4',weight=1)
G.add_edge('1','5',weight=3)
G.add_edge('2','3',weight=2)
G.add_edge('2','5',weight=3)
G.add_edge('3','4',weight=2)
G.add_edge('3','5',weight=4)
G.add_edge('4','5',weight=1)
N = G.number_of_nodes ()
maxcut = pic.Problem ()
X=maxcut.add_variable('X',(N,N),'symmetric
     ')
nlist=['1','2','3','4','5']
gL=nx.laplacian_matrix(G,weight='weight',
     nodelist=nlist)
gL=gL.toarray ().astype(np.double)
L=pic.new_param('L',1/4.*gL)
```

```
maxcut.add_constraint(pic.tools.diag_vect(
    X)==1)
maxcut.add_constraint(X>>0)
maxcut.set_objective('max',L|X)
maxcut.solve ()
```

ここでは，ネットワーク解析のためのパッケージである NetworkX を用いている．まず，NetworkX によって無向グラフ G を生成する．そして，そのグラフに重みつき枝を add_edge() で追加する．add_edge() には第 1 引数と第 2 引数に両端点を，第 3 引数に枝の重みを指定する．重みは weight で指定する．グラフのノード数を N とし，$N \times N$ の行列変数を問題 maxcut に追加する．目的関数のためにグラフのラプラシアン $\boldsymbol{L}$ を計算する必要があるが，これには NetworkX の提供する laplacian_matrix() を用いる．この関数には第 1 引数としてグラフを指定する．重みつきグラフとして計算するにはさらに引数として weight=を指定する．nodelist=はオプションであるが，これを指定することによってラプラシアンの各行（列）が表すノードを指定することができる．ここでは，nlist=['1','2','3','4','5'] を指定しているので，計算されたラプラシアン gL では第 1 行（第 1 列）がノード '1' に，第 2 行（第 2 列）がノード '2' に対応する．nodelist=を指定しない場合は，ラプラシアンの各行各列の値は，リスト G.nodes() の順序に従って計算される．たとえば，上のコードを実行すると G.nodes() は ['1','4','5','3','2'] となるが，この場合，nodelist=を指定せずに laplacian_matrix() を実行すると，得られるラプラシアンの第 1 行（第 1 列）は，リスト G.nodes() の最初の要素 '1' に対応した値に，第 2 行（第 2 列）は 2 番目の要素 '4' に対応した値になる．出力結果を目で確認したい場合はこれは少しわかりにくいので，適宜 nodelist=を指定するのがよい．

laplacian_matrix() によって計算されたラプラシアン gL を PICOS のパラメータとして用いるためには，いくらかの処理が必要である．まず，gL.toarray() によって，numpy の array 型に変換する．変換された値の方は int64 型であるが，この型のデータは pic.new_param() の引数として指定することができない．そこで，この型を astype(np.double) によって numpy の double 型に変換する．これらの変換を施すことで，データ gL を pic.new_param() の引数として指定することができるようになる．さらに，行列 $\boldsymbol{L}$ と $\boldsymbol{X}$ の内積は L|X で表すことができる．

PICOS では，行列変数の対角要素が 1 に等しいという制約は，pic.tools.diag_vect(X)==1 によって指定することができる．また，行列変数が半正定値であるという制約は，X>>0 により指定することができる．

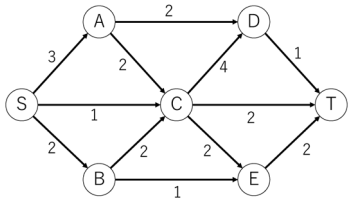

図 2　最大フローを求める有向グラフ

3.4　ネットワーク最適化

PICOS は，ネットワークのフロー制約を表現する機能を持っている．次に示すのは，ネットワーク解析で便利に用いることができるパッケージである NetworkX で有向グラフを作成し，各枝に流れるフローを表す変数 f[e] とフローの合計値を表す変数 F を定義するコードである．ただし，ネットワークとしては図 2 に示したものを用いた．枝に付記したのは枝の容量である．

```
import networkx as nx
G=nx.DiGraph ()
G.add_edge('S','A', capacity=3);
G.add_edge('S','B', capacity=2);
G.add_edge('S','C', capacity=1);
G.add_edge('A','C', capacity=2);
G.add_edge('A','D', capacity=2);
G.add_edge('B','C', capacity=2);
G.add_edge('B','E', capacity=1);
G.add_edge('C','D', capacity=4);
G.add_edge('C','E', capacity=2);
G.add_edge('C','T', capacity=2);
G.add_edge('D','T', capacity=1);
G.add_edge('E','T', capacity=2);
pb = pic.Problem ()
f={}
for e in G.edges () :
 f[e]=pb.add_variable('f[{0}]'.format(e
     ),1)
F = pb.add_variable('F',1)
```

ここでは，add_variable() で変数を定義する際に，最初の引数として'f[{0}]'.format(e) を与えている．最初の引数は変数の名前を与えるもので，文字列を指定する．format 関数は，Python で変数を文字列中に埋め込むことを可能にする．この命令は，文字列'f[{0}]'の{0}の部分に，e の値を文字列として埋め込むものである．

これらの変数間には，ネットワーク上のノードに関するフロー保存則，すなわち，各ノードに接続された枝から入るフローの合計と出るフローの合計との整合性が取れるための制約を課す必要があるが，これらの制約を記述する際には，変数の添え字などに誤りがないように十分に注意する必要がある．たとえば，始点と終点以外でのフロー保存則は，次のように書かれる：


```
pb.add_list_of_constraints( [ pic.sum([f[
    p,i] for p in G.predecessors(i)],'p','
    pred(i)') == pic.sum([f[i,j] for j in
    G.successors(i)],'j','succ(i)') for i
    in G.nodes () if i not in ('S','T')],
    'i','nodes-(s,t)')
```

このほかにも，各枝での容量制約と非負制約，始点と終点でのフロー制約も，誤りなく記述する必要があるが，それなりに煩雑になる．ところが，PICOS の機能を用いると，次の 1 行でそのフロー制約を記述することができる．

```
flowCons = pic.flow_Constraint(G, f,
    source='S', sink='T', capacity='
    capacity', flow_value= F, graphName='G
    ')
```

こうして生成したフロー制約 `flowCons` を問題の制約として追加すれば，変数間に適切にフロー制約が課される．あとは，目的関数を設定して解けばよい．ここでは目的関数を，始点から出て終点に到達するフローの合計値の最大化とする．

```
pb.addConstraint(flowCons)
pb.set_objective('max',F)
sol = pb.solve ()
flow = pic.tools.eval_dict(f)
```

フロー値は，`pic.tools.eval_dict(f)` によって得られる．フロー値 value(f) は，始点 s から出て終点 t に到達するフローの合計値であり，枝集合を E，枝 $(i,j) \in E$ のフローを $f(i,j)$ とすると，次の式で定められる：

$$\begin{aligned}\mathrm{value}(f) &= \sum_{(s,j)\in E} f(s,j) - \sum_{(i,s)\in E} f(i,s) \\ &= \sum_{(i,t)\in E} f(i,t) - \sum_{(t,j)\in E} f(t,j).\end{aligned}$$

4. まとめ

本稿では，CVXOPT と PICOS を用いて錐最適化を実行する方法を述べた．また，ネットワーク制約や半正定値最適化モデルを簡単に記述するための PICOS の機能も述べた．近頃は，ネットワーク上で非線形な制約や目的を取り扱う最適化モデルも増えてきている．本稿では，CVXOPT，PICOS，そして NetworkX の機能を取り混ぜて用いる方法を述べたが，今後はこのように最適化に関する複数のパッケージを組合わせて用いる機会も増えてくると考えられる．

参考文献

[1] M. Andersen, J. Dahl and L. Vandenberghe, “CVXOPT: Python software for convex optimization, version 1.1.8,” http://cvxopt.org/（2018 年 10 月 8 日閲覧）

[2] G. Sagnol, “PICOS: A Python interface for conic optimization solvers, version 1.1.2,” https://picos-api.gitlab.io/picos/（2018 年 10 月 8 日閲覧）

[3] 小島政和，土谷隆，水野眞治，矢部博，『内点法』，朝倉書店，2001.