Pythonで始めよう非線形最適化

小林　和博

プログラミング言語 Python では，非線形最適化を実行するためのさまざまなパッケージが存在する．本稿では非線形最適化問題のモデリングツールである Pyomo と凸最適化問題のモデリングツールである CVXPY を取り上げる．これらのツールを用いると，目的関数，制約式にさまざまな式を用いる非線形最適化問題を容易に表現し，解くことができる．

キーワード：Python，非線形最適化，凸最適化，プログラミング

1. 非線形最適化

非線形最適化問題は，次のように定められる．

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & c(x) = 0 \\ & d^L \leq d(x) \leq d^U \\ & x^L \leq x \leq x^U \end{aligned}$$

ここで，x は変数ベクトル，$f(x)$ は目的関数，$c(x)$ は等式制約を定める数式，$d(x)$ は不等式制約を定める数式である．また，d^L と d^U は不等式制約の定数項を表すベクトル，x^L と x^U はそれぞれ変数 x のとりうる最小値と最大値を表すベクトルとする．

非線形最適化問題を解くアルゴリズムは色々提案されているが，一般にはそれらを実行して得られるのは局所的最適解であり，大域的最適解ではないことに注意する．本稿でソルバーを用いて得られる解も一般には局所的最適解であり大域的最適解ではないが，記述をシンプルにするために局所的最適解のことも単に最適解と書くことにする．

2. Pyomo と Ipopt

非線形最適化問題を解くには，まず問題をプログラムとして表現する必要がある．このために，Pyomo を用いる [1]．Pyomo は非線形最適化のほかにもさまざまな問題のモデリングに用いることができる．Pyomo で使用可能な演算子の例を表 1 に挙げた．

表 1　Pyomo で使用可能な演算子の例

演算	演算子	使用例
積	*	expr = model.x * model.y
商	/	expr = model.x / model.y
冪乗	**	expr = (model.x+2.0)**model.y
正弦	sin	expr = pyo.sin(model.x)
余弦	cos	expr = pyo.cos(model.x)

Pyomo は，モデリングのためのツールであり，それ自体に問題を解く機能はない．モデリングした問題を解くためには，Pyomo から利用可能なソルバーを用いる必要がある．ここでは，非線形最適化問題を解くためのソルバーとして，Ipopt を用いる [2, 3]．

Pyomo と Ipopt を Anaconda 環境でインストールするには，次のコマンドを実行する [4]．

```
conda install -c conda-forge pyomo
conda install -c conda-forge ipopt
```

Pyomo を用いて最適化問題を解く手順は，次のとおりである．

1. 最適化モデルを生成し，必要な要素（パラメータ，変数，目的関数，制約条件）を宣言する．
2. モデルをインスタンス化する．
3. ソルバーを適用して問題を解く．
4. 解いた結果を確認する．

次に，これらの手順を具体的に述べる．

抽象モデルと具象モデル

Pyomo で数理最適化問題を表現するには抽象モデルを用いる方法と具象モデルを用いる方法がある．抽象モデルを用いる方法では，まずパラメータを用いて問題を定義し，その後，実際に問題を解く時点でパラメータのデータを与える．たとえば，次の線形最適化問題を，a_{ij}, b_i, c_j をパラメータのまま定めるには，抽象モデルを用いる．

こばやし　かずひろ
青山学院大学理工学部
〒 252–5258　神奈川県相模原市中央区淵野辺 5–10–1
kobayashi@ise.aoyama.ac.jp


```
1:  import pyomo.environ as pyo
2:  model = pyo.ConcreteModel()
3:  model.x = pyo.Var(initialize=1.5)
4:  model.y = pyo.Var(initialize=1.5)
5:
6:  def rosenbrock(model):
7:      return (1.0 - model.x)**2 \
8:          + 100.0*(model.y - model.x**2)**2
9:
10: model.obj = pyo.Objective(rule=rosenbrock, \
11:                    sense=pyo.minimize)
12: opt = pyo.SolverFactory('ipopt')
13: status = opt.solve(model)
14: pyo.assert_optimal_termination(status)
15: print("目的関数値:",pyo.value(model.obj))
16: print("x:",pyo.value(model.x), \
17:                   " y:",pyo.value(model.y))
```

図 1　The Rosenbrock Problem を解くプログラム

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 \qquad\qquad\quad \forall j \in \{1, \ldots, n\} \end{aligned}$$

これに対して，

$$\begin{aligned} \text{Minimize} \quad & 2x_1 + 3x_2 \\ \text{subject to} \quad & 3x_1 + 4x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

のように，パラメータに具体的な数値が与えられた問題を定めるには具象モデルを用いる．

3.　制約なし最適化問題

非線形最適化問題のベンチマークとしてよく用いられる Rosenbrock 関数の最小化問題を扱う．Rosenbrock 関数の最小化問題は，一般に n 変数で定められる制約なし最適化問題であるが，ここでは $n = 2$ の場合を扱う：

$$\text{Minimize}_{x,y} \quad f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

この 2 変数の Rosenbrock Problem を解くプログラムを図 1 に示す．各行の先頭に行番号を示したが，実際にプログラムとして実行する際にはこれらの番号は入力しない．

1 行目は，`pyo` として `Pyomo` の機能を用いるための `import` 命令である．2 行目では，`ConcreteModel()` によって具象モデルとして最適化問題を生成する．生成したモデルには `model` でアクセスできる．3 行目では，`Var()` によって変数 x に対応する `model` の変数 `model.x` を生成する．`Var()` のカッコ内に示した `initialize=1.5` は，変数の初期値を 1.5 に設定することを表す．4 行目も同様で `model.y` を生成する．初期値は，最適化アルゴリズムを実行する際に用いられ，この初期値を変えることで異なる局所的最適解が得られる可能性がある．6–8 行目で最小化する目的関数 $f(x, y)$ を，関数 `rosenbrock` として定義する．この関数は，`model.x` と `model.y` で定められるので，引数として `model` を与えている．10–11 行目では，問題 `model` の目的 `model.obj` を，`pyo.Objective()` により設定する．この際，引数として `rule=rosenbrock` と `sense=pyo.minimize` を与えている．`rule=rosenbrock` は，目的関数として 6–8 行目で定義した `rosenbrock` を用いることを指定する．そして，`sense=pyo.minimize` は目的を関数の最小化と指定する．このように，11 行目までで 2 変数の Rosenbrock Problem をプログラムとしてモデリングすることができた．

```
目的関数値: 7.013645951336496e-25
x: 1.0000000000008233  y: 1.0000000000016314
```

図 2　図 1 のプログラムの実行結果

続いて，こうしてモデル化した問題 `model` を解く．12 行目では，ソルバーとして `Ipopt` を用いて最適化するためのインスタンスを生成している．このインスタンスには `opt` によりアクセスできる．13 行目では，12 行目で生成した `opt` の `solve()` を用いて `model` で表される最適化問題の最適化を実行する．`opt.solve(model)` の実行結果は `status` によってアクセスすることができる．14 行目では，ソルバーを実行した結果として最適解が得られたかどうかを確認する．15 行目では `pyo.value(model.obj)` によって目的関数値の値を，16–17 行目では同様に `model.x` と `model.y` の値を表示する．

このプログラムの実行結果を，図 2 に示した．この結果を見ると，最適解として $(x, y) = (1, 1)$ が得られ，その点での最適値は 0 であることがわかる．

4.　制約あり最適化問題

続いて，一つの不等式制約と一つの等式制約をもつ次の制約あり最適化問題を扱う．

```
1:  import pyomo.environ as pyo
2:  model = pyo.ConcreteModel()
3:
4:  N=[1,2,3,4]
5:  model.x=pyo.Var(N,bounds=(1,5))
6:
7:  def obj_rule(model):
8:      return model.x[1]*model.x[4]* \
9:             (model.x[1]+model.x[2]+ \
10:             model.x[3]) + model.x[3]
11:
12:  model.obj=pyo.Objective(rule=obj_rule,\
13:             sense=pyo.minimize)
14:
15: def con_rule1(model):
16:     return model.x[1]*model.x[2]* \
17:            model.x[3]*model.x[4]>=25
18:
19: model.con1 = pyo.Constraint(rule=con_rule1)
20:
21: def con_rule2(model):
22:     return model.x[1]**2+model.x[2]**2 \
23:        +model.x[3]**2+model.x[4]**2 == 40
24:
25: model.con2 = pyo.Constraint(rule=con_rule2)
26:
27: opt=pyo.SolverFactory('ipopt')
28: status=opt.solve(model)
29: pyo.assert_optimal_termination(status)
30: print("目的関数値:",pyo.value(model.obj))
31: print("最適解:")
32: for i in N:
33:         print(i,"\t",pyo.value(model.x[i]))
```

図 3　Hock-Schittkowsky 71 番を解くプログラム

$$\begin{aligned} \text{Minimize} \quad & x_1x_4(x_1+x_2+x_3)+x_3 \\ \text{subject to} \quad & x_1x_2x_3x_4 \geq 25 \\ & x_1^2+x_2^2+x_3^2+x_4^2 = 40 \\ & 1 \leq x_1, x_2, x_3, x_4 \leq 5 \end{aligned}$$

これは，Hock-Schittkowsky Test Suite の 71 番の問題である．この問題を解くための Python プログラムを図 3 に示す．この問題で用いられる四つの変数は，x_1, x_2, x_3, x_4 と，x に添字をつけたものである．これらの変数を，プログラムでも `x[i]` と添字を用いて表すことにする．

4 行目で，`N` を変数の添字 1,2,3,4 を要素とする配列として定めている．5 行目では `Var()` によって `N` の要素を添字とする変数を生成している．こうして生成した `i` 番目の変数は，`model.x[i]` によりアクセスできる．`Var()` の引数として与えている `bounds=(1,5)` は，`x[i]` のとりうる最小値 1 と最大値 5 を指定している．7–10 行目では目的関数を定める Python の関数 `obj_rule()` を定義する．12–13 行目ではこの

```
目的関数値: 17.014017145174137
最適解:
1       0.9999999923254504
2       4.742999641809567
3       3.821149981789044
4       1.3794082897520368
```

図 4　図 3 のプログラムの実行結果

`obj_rule()` を `rule=`を用いて `model` の目的関数として設定している．この目的関数には `model.obj` でアクセスすることができる．15–17 行目では最初の制約である不等式制約を定めるための関数 `con_rule1()` を定める．目的関数を定める `obj_rule()` は式を返すが，`con_rule1()` は不等式を返すことに注意する．19 行目では，`pyo.Constraint()` を用いて関数 `con_rule1()` が返す不等式制約を `model` に設定する．この不等式制約は，`model.con1` によりアクセスすることができる．21–23 行目では 2 番目の制約である等式制約を定める関数 `con_rule2()` を定める．25 行目ではこの関数を用いて等式制約を `model` に設定する．27–29 行目は 25 行目まででモデル化した問題を解くための手順であり，ここは制約なし問題の場合と同様である．30–33 行目では，得られた最適解とその解における目的関数値を表示する．

図 3 に示したプログラムの実行結果を，図 4 に示した．これより，得られる最適解は $(x_1, x_2, x_3, x_4) = (1.000, 4.743, 3.821, 1.379)$ であり，そのときの目的関数値は 17.014 であることがわかる．

5.　凸最適化問題

非線形最適化問題には，凸な問題と非凸な問題がある．解きたい問題が凸であることがわかっていれば，その構造を生かして効率的に大域的最適解を求めることができる [5].

凸最適化問題は，次の形で表される最適化問題である．

$$\begin{aligned} \text{Minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq b_i \qquad \forall i \in \{1, \ldots, m\} \end{aligned}$$

ここで，$f_0, ..., f_m : \mathbb{R}^n \to \mathbb{R}$ は凸関数であるとする．すなわち，すべての $x, y \in \mathbb{R}^n$ と $\alpha+\beta = 1, \alpha \geq 0, \beta \geq 0$ を満たすすべての $\alpha, \beta \in \mathbb{R}$ に対して

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

を満たすとする．

線形関数とアフィン関数は凸関数であるが，それ以外にも凸関数の例として次のものが挙げられる．

- 指数関数 e^{ax}
- べき乗関数 x^a（$a \geq 1$ または $a \leq 0$），ただし $x > 0$
- 対数関数 $\log x$，ただし，$x > 0$
- 負エントロピー $x \log x$
- $\mathbb{R}^n$ 上のノルム
- 最大値関数 $f(x) = \max\{x_1, x_2, ..., x_n\}$

また，これらの凸関数に凸性を保つ演算を施した結果得られる関数も，凸関数である．凸性を保つ演算の例として，次のものが挙げられる．

- f が凸関数であるとすると，正の定数 $\alpha > 0$ に対して定数倍 αf は凸関数である．
- f_1 と f_2 がともに凸関数とすると，その和 $f_1 + f_2$ は凸関数である．

上記の二つの事実より，凸関数の非負重み付き和 $f = w_1 f_1 + w_2 f_2 + \cdots + w_m f_m$ は凸関数である．凸関数と，凸性を保つ演算を組み合わせると，さまざまな制約条件を凸関数として表すことができる．詳しくは凸最適化の教科書を参照するとよい [5].

凸最適化問題をモデル化することができる Python パッケージがいくつか存在するが，ここでは `CVXPY` を取り上げる [6]．Anaconda 環境で `CVXPY` をインストールするには次のコマンドを実行する．

```
conda install -c conda-forge cvxpy
```

たとえば, 次の最小二乗問題は凸な最適化問題である.

$$\text{Minimize} \quad \sum_{i=1}^{m} \left(a_i^\top x - b_i \right)^2$$

ここで，$a_i^\top$ は $m \times n$ の行列 A の第 i 行，b_i は m 次元ベクトル b の第 i 成分，x は n 次元の変数ベクトルとする．この最小二乗問題は，与えられた観測値 A，b に対して，Ax が b に最も近くなるような x を求める問題である．最小二乗問題の最適解を x^* とすると，Ax^* と b の差 $r = Ax^* - b$ は残差といわれる．$\sum_{i=1}^{m} \left(a_i^\top x - b_i \right)^2$ は行列とベクトルを用いて

$$(\|Ax - b\|_2)^2$$

と表される．ここで，$\|\cdot\|_p$ は ℓ_p ノルムとよばれ，$\|x\| = \left(\sum_{j=1}^{n} x_i^p \right)^{1/p}$ である．最小二乗問題を定める A と b を擬似乱数により生成し，その問題を解くプログラムを図 5 に示した．図 5 と，後出の図 7，9 に示したプログラムは，`CVXPY` のホームページの Examples に記載されたものをもとにしている [7].

```
1:  import cvxpy as cp
2:  import numpy as np
3:  m=20
4:  n=15
5:  np.random.seed(1)
6:  A=np.random.randn(m,n)
7:  b=np.random.randn(m)
8:
9:  x=cp.Variable(n)
10: cost=cp.sum_squares(A@x-b)
11: prob=cp.Problem(cp.Minimize(cost))
12: prob.solve()
13:
14: print("状態:",prob.status)
15: print("最適値:",prob.value)
16: print("最適解:")
17: print(x.value)
18: print("残差:",cp.norm(A@x-b,p=2).value)
```

図 5　最小二乗法を解くプログラム

1–2 行目は，それぞれ `CVXPY`，`NumPy` をインポートするものである [8]．3 行目では行列 A の行数を 20 に，4 行目では列数を 15 に指定している．列数の 15 は変数ベクトルの次元に対応する．5 行目は `numpy` の擬似乱数の種を 1 に指定するものである．6 行目では標準正規分布に従う擬似乱数を生成する `np.random.randn()` により，2 次元配列を表す `A` を生成している．7 行目では同様に `np.random.randn()` により，`m` 次元のベクトル `b` を生成する．9 行目では `n` 次元の変数 `x` を生成する．10 行目の `sum_squares()` は二乗和を計算する `CVXPY` の関数である．その引数として与えている `A@x-b` は，行列を表す `A` とベクトル `x` との積からベクトル `b` を引いたものを表す．なお，演算子`@`は行列と行列，または行列とベクトルの積を求めるものである．11 行目では，最小二乗問題を `CVXPY` の `Problem` として生成している．この際，`Problem()` の引数として `cp.Minimize(cost)` を与えることで，問題の目的を関数 `cost` の最小化と指定している．12 行目では，11 行目までで定めた最小二乗問題 `prob` を解く．14–18 行目では最適化の結果を表示する．

このプログラムの実行結果を図 6 に示した．この結果から，最適値，最適解，残差 r の $\|r\|_2$ を確認することができる．この解は大域的最適解である．

さまざまな問題が凸最適化問題として定式化されるが，それらの中からポートフォリオ最適化問題と線形不等式系の疎な解を求める問題を取り上げる．

ポートフォリオ最適化

二次関数は最も馴染み深い凸関数の一つである．制約条件が線形制約であり，二次関数を目的関数とする最

```
最適値: 7.005909828287485
最適解:
 [ 0.17492418 -0.38102551  0.34732251  0.0173098  -0.0845784  -0.08134019
   0.293119    0.27019762  0.17493179 -0.23953449  0.64097935 -0.41633637
   0.12799688  0.1063942  -0.32158411]
残差: 2.6468679280023557
```

図 6　図 5 のプログラムの実行結果

適化問題を二次最適化問題とよぶ．二次最適化問題として定式化することができる代表的な応用問題としてポートフォリオ最適化問題が挙げられる．ポートフォリオ最適化問題は，n 個の金融商品それぞれに投資する全資産の割合を決定する問題である．資産 $i \in \{1, 2, \ldots, n\}$ への投資割合を w_i と表す．w_i を i 番目の要素とする n 次元ベクトルを w と表し，ポートフォリオとよぶ．このベクトルは $\sum_{i=1}^{n} w_i = 1$ と $w_i \geq 0$ $(i = 1, \ldots, n)$ を満たす．

資産 i の収益率を r_i とすると，ポートフォリオの収益率は $R = \sum_{i=1}^{n} r_i w_i$ と定められる．収益率 $r = (r_1, r_2, \ldots, r_n)^\top$ を確率変数としてその期待値を μ，共分散を Σ とする．このとき，R は期待値が $\sum_{i=1}^{n} \mu_i w_i$，分散が $w^\top \Sigma w$ の確率変数となる．この期待値 $\sum_{i=1}^{n} \mu_i w_i$ をポートフォリオの期待収益率という．

ポートフォリオ最適化問題では，期待収益率の最大化とリスクの最小化という二つの相反する目的を扱う．

ポートフォリオ最適化問題における平均・分散モデルは次の最適化問題として定式化される．

$$\begin{aligned} \text{Maximize} \quad & \mu^\top w - \gamma w^\top \Sigma w \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1 \\ & w_i \geq 0 \ (i = 1, \ldots, n) \end{aligned}$$

ここで，$\gamma > 0$ はリスク指向度を表すパラメータである．この問題は凸最適化問題であるので，大域的最適解を求めることができる．

平均・分散モデルの問題例を生成し，数理最適化問題として解くプログラムを図 7 に示した．

4 行目では変数ベクトルの次元 `n` を指定している．これは，資産数に対応している．5 行目では，擬似乱数によって μ の値 `mu` を設定する．7 行目では Σ の値 `Sigma` を設定するが，そのために 6 行目で擬似乱数を用いて n 行 n 列の `Sigma` を設定し，それを用いて `Sigma.T.dot(Sigma)` の値をあらためて `Sigma` と設定している．10 行目では変数ベクトル w を生成する．11 行目ではリスク指向度を表す `gamma` を 100 に

```
 1:  import numpy as np
 2:
 3:  np.random.seed(1)
 4:  n=10
 5:  mu = np.abs(np.random.randn(n,1))
 6:  Sigma = np.random.randn(n,n)
 7:  Sigma = Sigma.T.dot(Sigma)
 8:
 9: import cvxpy as cp
10: w=cp.Variable(n)
11: gamma=100
12: ret=mu.T @ w
13: risk = cp.quad_form(w,Sigma)
14: prob = cp.Problem( \
15:        cp.Maximize(ret-gamma*risk), \
16:        [cp.sum(w)==1,w>=0])
17: prob.solve()
18:
19: print("状態:",prob.status)
20: print("最適値:",prob.value)
21: print("最適解:")
22: print(w.value)
```

図 7　ポートフォリオ最適化問題を解くプログラム

設定している．この値が大きくほどリスク最小化を指向し，小さいほど期待収益率の最大化を指向することを表す．12 行目では期待収益率を表す `ret` を，`mu` と `w` の内積として定めている．13 行目では，リスクを表す `risk` を，ベクトル `w` と行列 `Sigma` の二次形式として定める．一般に，行列 A の二次形式は，ベクトル x によって $x^\top Ax$ と定められるが，`CVXPY` では `quad_form()` によって二次形式を表すことができる．14–16 行目では，`ret-gamma*risk` の最大化を目的関数，`cp.sum(w)==1` と `w>=0` の二つの式を制約条件として最適化問題 `prob` を定めている．17 行目では最適化問題を解く．19–22 行目では最適化の結果を表示する．

このプログラムの実行結果を図 8 に示した．状態が `optimal` であるので最適解が得られていることがわかる．この問題は 2 次最適化問題であり，得られた解は大域的最適解である．

線形不等式系の疎な解を求める問題

$m \times n$ の行列 A $(m > n)$ と m 次元ベクトル b について，次の線形不等式系

```
状態: optimal
最適値: -12.011609005555467
最適解:
[1.61997763e-01 5.06325337e-23 2.69644881e-01 9.84425227e-02
 1.78560550e-23 9.70423962e-02 3.81900752e-02 3.13913398e-02
 1.36371276e-01 1.66919746e-01]
```

図 8　図 7 のプログラムの実行結果

```
1:  import cvxpy as cp
2:  import numpy as np
3:
4:  np.random.seed(1)
5:  delta = 1e-8
6:
7:  m=100
8:  n=50
9:
10: A=np.random.randn(m,n)
11: x0=np.random.randn(n)
12: b=A.dot(x0)+np.random.random(m)
13:
14: x_l1 = cp.Variable(shape=n)
15: constraints = [A@x_l1 <= b]
16: obj=cp.Minimize(cp.norm(x_l1,1))
17: prob = cp.Problem(obj,constraints)
18: prob.solve()
19:
20: print("状態:",prob.status)
21:
22: nnz_l1 = (np.absolute(x_l1.value)> \
23: delta).sum()
24: print("変数ベクトルの次元:{}，求まった解の非ゼロ
要素の数:{}".format(n,nnz_l1))
25: print("最適解:", obj.value)
```

図 9　疎な解を求めるプログラム

$$Ax \leq b$$

が実行可能である，すなわちこの方程式系を満たす x が存在するとする．ここで，この不等式系を満たす x の中から，できるだけ疎な，すなわちできるだけたくさんの 0 を要素としてもつ x を求めたいとする．この問題の大域的最適解を求めることは難しいが，できるだけ良い解を求めるためのヒューリスティックとして，ℓ_1 ノルムを最小にする方法がある．この問題は，次の最小化問題として定式化される [7].

$$\begin{aligned} \text{Minimize} \quad & \|x\|_1 \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$

図 9 に，この ℓ_1 ノルム最小化問題を解くプログラムを示した．1–2 行目は必要なパッケージをインポートするものである．5 行目は，値が 0 か否かを判定するために用いる閾値である `delta` を定める．7–12 行目では，この方程式系を定める `A`，`b` を定める．ここで

```
状態: optimal
変数ベクトルの次元:50，求まった解の非ゼロ要素の
数:40
最適解: 28.582394103180178
```

図 10　図 9 のプログラムの実行結果

は，まず `A` を擬似乱数により生成し，その次に `x0` を生成している．もし続いて，ベクトル `b` も擬似乱数を用いて生成してしまうと $Ax \leq b$ が実行可能になるとは限らない．そこで，いったん実行可能なベクトルとして `x0` を生成し，`b` は `A` と `x0` の積 `A.dot(x0)` に正の擬似乱数ベクトル `np.random.random(m)` を足すことで定めている．こうすると，$Ax \leq b$ が常に実行可能となる．このように，いったん実行可能解を生成しておき，それを用いて実行可能な問題例を生成することは，さまざまな数理最適化問題で用いることができる便利な tip であるので覚えておくとよい．

14 行目では，n 次元の変数ベクトルを生成している．15 行目では制約条件を表す `constraints` を，16 行目では目的関数を表す `obj` を設定する．この問題では ℓ_1 ノルムを用いるので，`norm()` の 2 番目の引数に 1 を指定している．このプログラムの実行結果を，図 10 に示した．これより，50 個の要素のうち非ゼロ要素の数が 40 個の実行可能解が求まったことがわかる．

6. まとめ

モデリングツールを用いると，非線形最適化問題をわかりやすい形で表現することができる．そして，ソルバーを用いることで最適解を求めることができる．モデリングツールとソルバーは多くのものがあり，選択に迷うケースがあるかもしれない．その場合は専門家に相談することも一案である．

参考文献

[1] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson and D. L. Woodruff, *Pyomo: Optimization Modeling in Python, 3rd Edition*, Springer, 2021.

[2] A. Wächter and L. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, **106**, pp. 25–57, 2006.
[3] COIN-OR, Ipopt project, https://github.com/coin-or/Ipopt（2023 年 9 月 9 日閲覧）
[4] Anaconda, Inc., Anaconda Software Distribution, Computer software. Vers. 2-2.4.0. Anaconda, https://anaconda.com（2023 年 9 月 9 日閲覧）
[5] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
[6] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for Convex optimization,” *Journal of Machine Learning Research*, **17**(83), pp. 1–5, 2016.
[7] CVXPY, https://www.cvxpy.org（2023 年 9 月 9 日閲覧）
[8] S. R. Harris, K. J. Millman, S. J. van der Walt, et al., “Array programming with NumPy,” *Nature*, **585**, pp. 357–362, 2020.

