実際の数理最適化問題を瞬時に解くための実装技術

久保　幹雄

本稿では，実際問題であらわれる難しい最適化問題を短時間で解くための方法論について論じる．最適化プロジェクトを路頭に迷わせないためには，次々とクライアントから要求される付加条件や変更条件を柔軟に取り扱えること，修正が短時間で完了すること，可視化やデータ解析が容易であることが必要である．ここでは，単なる哲学ではなく，Python 言語による具体的な実装例を用いてコツを伝授する．

キーワード：Python 言語，数理最適化，メタヒューリスティクス，制約最適化

1. はじめに

OR のさらなる普及のためには，アカデミックサイドから実際問題の解決のための手助けをすることが不可欠である．特に，最適化に関しては実務家が手助けなしで解決できる問題は限られており，必然的に専門家に頼ることになる．筆者も OR 普及の一助として，しばしば実際問題の解決のお手伝いをしているが，その際にはある程度のところまでは，自ら実装をすることを心がけている．ここで実装というのは，何らかのプログラミング言語を用いて，問題解決のコアになるプログラムを記述することを意味する．

一方，大学に勤める（特に工学部の）研究者たちは，「大学教授という仕事」（水曜社，杉原厚吉著）に書かれているように結構多忙である．通常は，会議や講義や学生指導の合間にプログラムの実装を行うことになるが，これは容易なことではない．実際には，途中まで書いたプログラムを見直して過去の記憶を呼び覚ましている最中に，学生が判子をもらいに来たり，会議の時間が来たりする．

したがって，なるべく短い時間で実装を完了することが必須なスキルとなる．これにはコツがある．なるべく自分でプログラムを「書かない」ことである．これは，学生に実装の仕事を丸投げするという意味ではない．よほど優秀で仕事が早い学生でない限り，これはやってはいけない．筆者は，この方法を用いたために失敗したプロジェクトを山のように見てきている．

プログラムを自分で「書かない」というのは，できるだけ既存の安定したライブラリを用いることを意味する．以下では，超高水準言語 Python[1] とそれに付随するライブラリ[2] を用いて，問題を解決するための手順について解説する．

くぼ　みきお
東京海洋大学大学院海洋工学系流通情報工学
〒 135–8533 東京都江東区越中島 2–1–6

2. 全体の流れ

通常，最適化によって実際問題を解決するためのプロジェクトを遂行するためには，以下の手順を経る．

1. モデルの抽出
2. データ収集と確認
3. データの読み込みと解析
4. モデルの構築と実装
5. モデル実行（最適化）
6. 結果解析（可視化と評価）．そして 1. に戻る．

以下，順に解説していく．

3. モデルの抽出

実務家は解きたい問題があるからわざわざ聞きに来るので，最初は話をよく聞いて分析すれば良い．ただし，問題は最適化に馴染まないような形態で話される場合が多い．まずは，全部をまとめて一緒くたにして解きたい実務家を説得して，問題を意思決定レベル[3] で分類して整理し，解ける規模の問題に分解する．また，細かすぎるデータを出してくる実務家には，デー

[1] http://www.python.org/ から無償でダウンロードできる．Python 言語についての詳細は，Guttag による MIT の人気講義のテキストを翻訳（「Python 言語によるプログラム・イントロダクション」近代科学社，久保監訳，2014 年秋刊行予定）したのでそれを参照されたい．
[2] Python では，ライブラリはモジュールと呼ばれる．
[3] （サプライ・チェインの）意思決定レベルについては，拙著「サプライ・チェイン最適化ハンドブック」や「ロジスティクス工学」（ともに朝倉書店）を参照されたい．

タは多少集約しても大丈夫だと説得するのも大事である．可能ならば現場を見に行き，ベテランの人と飲む．それによって会議室では出しにくい裏の制約や実情を知ることができる．なお，最初に持ってきた問題は常に変化することを想定して，二転三転しても大丈夫なようなアプローチを選択することを心がける．

4. データ収集と確認

通常，最適化を依頼されたプロジェクトにおいてデータを収集するのは筆者の仕事ではない．実際には，問題を依頼してきた企業のほうが，自社データベースから抽出したり，紙ベースのデータを電子化して準備してくれる．餅は餅屋と割り切って，データが準備されたことを前提に話を進めても良いが，若干の注意を書いておく．古典的な格言が示すように「ゴミを入れればゴミが出てくる」からである．

重要なことは正しいデータを集めることである．以前，既存の工場の統廃合の最適化を依頼されたときには，閉鎖されたくない工場が（意図的にかどうかは知らないが）誤った生産費用（製造原価）を出してきた．当然，安価に製造が可能な工場は生き残り，そうでない工場は閉鎖されるからだ．このときには，過去の製造量と会計情報を突き合わせることによって誤りを発見できたが，実際は意図的に改ざんされたことを見抜くのは難しい．信頼できるデータとの整合性を念入りにチェックする必要があるが，これには，以下で述べるデータ解析のツールやテクニックが役に立つ．

また生データには必ずといってよいほど欠損や誤りがあるので注意する必要がある．たとえば，ある企業の商品の販売量データでは，ある日の需要がない場合には空白を入れていた．在庫があるのに需要がない場合には，単に空白を 0 に置換すれば良いが，在庫がなくて販売できない場合には，前後の需要から適当な需要予測をして真の需要量を推定する必要がある．（某航空機会社では，この変換ができないがために，1 億円相当の海外製の収益管理ソフトウェアをゴミ箱行きにしていた．）このように，欠損データを補うためには多少のテクニックが必要である．それについては以下の節で述べる．

5. データの読み込みと解析

実務家から提供されるデータは，Microsoft 社の Excel 形式もしくは Excel から出力されたテキストファイルであることが多い．Excel の普及は我々研究者から見ると驚くほどであり，実務では Excel の限られた環境で名人芸のように意思決定を行っていることが多い．Excel でもある程度のデータ解析を行うことができるが，大規模データになると非常に煩雑になる．複数の Excel データからデータを抽出したり，欠損データを適切に処理し，最適化モデルに入力可能な形式に加工することは手間と時間がかかる．実務家はこれを Excel 上で半分手作業で行ったりするので，データ量が多い場合には時間もかかり，再現性がなくなり，正確性に欠けることになる．

さらに，提示されたデータが正しいという保証はない．たとえば，ユニークなはずの ID に重複があったり，欠損データがあったり，手作業を前提とした例外処理が含まれていたりするのは，ざらである．そのような場合は，筆者は，Python の pandas[4] と呼ばれるモジュール（Python のライブラリ）を用いてデータ解析ならびに加工を行う．これは，最近流行の言葉を使えば，データサイエンスとかビジネス解析に相当する．ビジネス解析用のさまざまな商用ソフトウェアがあるが，`pandas` を使えば高度な解析が高速に（しかも無償で）できる．

例として以下のような仮想の顧客データ `Custmer.csv`[5] を考える．想定しているのは，データ内の顧客に複数の営業マンが班を組んで訪問して商品の販売を行う問題である．データ項目は順に，顧客名 (name)，訪問する班の必要レベル (level)，班の最低人数 (number)，訪問時間 (time) である．

```
name,level,number,time
Kitty,A,3,5.03333
Panda,B,1,1.36667
Bear,C,2,2.2
Polar Bear,B,2,
Rabbit,B,2,7.03333
...
```

通常，データの読み込みは結構面倒であるが，pandas モジュールを用いて Python で記述するとたった 2 行（実質は 1 行）で済む．

```
1 import pandas as pd
2 df = pd.read_csv("Customer.csv")
```

最初の行では pandas モジュールを `pd` という名前で

[4] Python Data Analysis Library の省略らしいが，どこをどうやって省略したかは不明である．http://pandas.pydata.org/ から無償でダウンロードできる．
[5] カンマで区切られたテキストファイルであり，csv ファイルと呼ばれる．

読み込み，2 行目で csv ファイルを読み込んでいる．

pandas にはデータフレームと名づけられたオブジェクトがある．これは Excel のシートと同じものを Python 上で扱えるようにしたものだ．Excel のシートは行と列から構成されるが，データフレームでは行はインデックスと呼ばれる点が異なる．上のコードでは，csv ファイルから，データフレームオブジェクト `df` を生成した．

まず，列 `time` は時間単位であったので，それを最適化で処理しやすいように分単位に変換した列 `minutes` を作成しておこう．データフレームの列は Python の辞書のようにアクセスできるので，1 行で記述できる．

```
df["minutes"] = df["time"] * 60
```

データを確認するためには，データの先頭もしくは末尾のみを表示させるためのメソッド[6] `head( )` もしくは `tail( )` を用いると良い．以下に対話型シェル[7]で `df.head( )` と記述したときの出力を示す．

```
          name  level  number      time   minutes
0       Kitty      B       3   5.03333  301.9998
1       Panda      B       1   1.36667   82.0002
2        Bear      C       2   2.20000  132.0000
3  Polar Bear      B       2       NaN       NaN
4      Rabbit      B       2   7.03333  421.9998
```

ちゃんと分を表す列 `minutes` が生成されていることがわかる．上のデータには `NaN` という記号が含まれている．これは “Not A Number” の略で pandas では欠損値を意味する記号である．ここでは欠損値を他の顧客への訪問時間の平均値で置き換えることにしよう．これも 1 行で書ける．

```
df["minutes"].fillna(value=
        df["minutes"].mean(),inplace=True)
```

上のコードでは，分単位の訪問時間の平均を `mean` メソッドで得て，データフレームの `fillna` メソッドで欠損値を置き換えている．

データのクロス分析（Excel のピボットテーブル）も簡単である．たとえば，レベルと人数のクロス分析は以下のように 1 行ででき，結果が表示される．

```
print pd.crosstab(df["level"], df["number"])
```

```
number   1   2    3
level
A       10  106   20
B      101  749  123
C       30   50    0
```

データを読み込んで解析したら，モデルに内在するデータの固まりをクラスとして実装しておくと後々便利だ．特に，後でプログラムを見直すときに，クラスができていると記憶を呼び起こすのが比較的楽だ．たとえば，顧客を表すクラスは，以下のように書ける．

```
1  class Customer():
2      def __init__(self,name,level,number,
           minutes):
3          self.name    = name
4          self.level   = level
5          self.number  = number
6          self.minutes = minutes
7      def __str__(self):
8          ret="Name=%s  Level= %s  Number=%s
               Time= %s"%(self.name,self.level,
               self.number,int(self.minutes))
9          return ret
```

ここで，`__init__` はクラスオブジェクトの初期化を行うためのメソッドであり，コンストラクタとも呼ばれる．上のコードでは，名前 (name)，レベル (levle)，必要レベル (level)，最低人数 (number)，訪問時間 (time) をコンストラクタの引数として与え，クラスの属性として保管している（ちなみに，Python では `self` はクラスオブジェクト自身を表す）．また，`__str__` は文字列を返すメソッドであり，これは内容を確認するときに役に立つ．たとえば，ある顧客オブジェクトを print 関数で出力すると，以下のように表示される．

```
Name=Kitty Level= B Number=3 Time= 301
```

行ごとにデータフレームに保管されている顧客情報から顧客クラスを生成するには，以下のように記述する．

```
1  Customers=[ ]
2  for i in df.index:
3      r = df.loc[i]
4      c = Customer(r["name"], r["level"],
5              r["number"], r["minutes"])
6      Customers.append( c )
```

ここで `index` メソッドはデータフレームの行（インデックス）のリストを返す．反復内では，各インデックスに対して，行情報にアクセスするために `loc` メソッドを用いて行データ `r` を抽出し，顧客クラスオブジェクト `c` を生成して，リスト `Customers` に追加している．

上のように基本となるオブジェクトの集合を保持する際のデータ構造は，リストか辞書[8]が良い．データに

[6] オブジェクトに付随した関数．

[7] Python はインタプリタとしても使えるので対話形式でデータ分析ができる．

[8] Python の組み込みの複合型．ハッシュ関数のようにキーと値の組を保管する写像型．

ランダムアクセスする必要がある場合には辞書，順番にアクセスするときにはリストと覚えておくと良い．面倒な場合には，何でも辞書で保管しても大丈夫である．Python による数理最適化のモデルについて記述した本 [2] ではすべて辞書で実装した例を示している．辞書だと順序が任意になるので気持ちが悪いという人には，Python 2.7 で導入された順序付き辞書（`collections` モジュールの `OrderedDict`）があるのでそれを使えば良い．

ファイルサイズが大きくデータの読み込みに時間がかかる場合には，ファイルから読んだら保管したデータ構造をバイトストリームに変換して保管しておくと良い．pandas のデータフレームも直接バイトストリームに保管できるが，ここでは作成したクラスオブジェクトを保管しておくことにしよう．

Python には，`pickle`[9] と呼ばれるモジュールが準備されている．たとえば，ファイル名 `cust` に，顧客オブジェクトのリスト `Customers` を保管するには，以下のようにする．

```
1 import cPickle
2 f = open('cust.dmp','w')
3 cPickle.dump( Customers, f )
4 f.close()
```

ちなみに上では `pickle` モジュールの高速版である `cPickle` を用いている．一度保管しておけば，次回からはテキストファイルから読み込まずに，直接顧客オブジェクトのリストをファイルから高速に読み込むことができる．

```
1 import cPickle
2 f = open('cust.dmp','r')
3 Customers = cPickle.load(f)
4 f.close()
```

6. モデルの構築と実装

モデルの構築とそのモデルを解くための解法の選択は同時に行うべきである．解けないモデルは無意味であるし，解法を選んでからモデルを構築するのは我田引水シンドローム [3] に陥っていると言える．解法としては，数理最適化ソルバーか，制約最適化ソルバーか，問題に特化したソルバーか，はたまた自作のメタヒューリスティクスとなる．どの解法を選択するのが良いかについては，以前書いた論文 [4] を参照されたい．モダンな数理最適化ソルバーを用いれば，列生成法，分枝切除法，数理最適化ソルバーベースのメタヒューリスティクスなどの問題に特化した解法も容易に実装できる．ここで注意だが，重要な実際問題に対して劣勾配法を用いた Lagrange 緩和は使うべきではない．（もちろん，問題を一回だけ解く場合や，論文を書くために研究で用いるのは問題ない．）Lagrange 緩和は列生成法の線形緩和と同じかそれより悪い限界値を算出するが，劣勾配法の動作は不安定でパラメータのチューニングが必要であり，実務で頻繁に用いる場合には，いつ破綻するかわからないからだ．

モデルの構築の際には，問題例に含まれる数値（データ）も重要である．簡単な簡易（封筒の裏の計算）モデルを構築することによって，どの変数や制約が重要かがわかり，簡易モデルを用いた分析によって，現状を改善するとどのくらいの効果があるのかが，本実験をする前にわかることもある．

通常，実務的な問題はほぼ 100% $\mathcal{NP}$-困難であるので，与えられた問題のすべての問題例（インスタンス）を解けるようにすることは難しい．もちろん将来どのような問題例が入力されるかはわからないが，想定される最大規模の問題例に対して，データ解析によって得られた標準的な問題例を解けるようにすることが重要なのである．

さてモデルの構築であるが，前節で考えた営業班による顧客の訪問の仮想の例を用いて説明する．いま，ヒアリングから以下の仮定が抽出できたものとする．

- 1 週間の営業班の編成を最適化したい．
- 毎日，班編成は変更しても良い．
- 営業マンのレベルは A，B，C の順で，上位のレベルを持つ営業マンは下位のレベルも有する．
- 班のレベルは，班に含まれる営業マンで最もスキルが高い人のレベルとする．
- 1 つの顧客は 1 つの班で処理する．
- 1 つの班が 1 日に訪問する顧客は，できるだけ地理的に近い必要がある．
- 班が 1 日に稼働できる時間の上限は決まっている．

上の仮定から直接最適化モデルを構築すると比較的複雑な問題になってしまうが，データ分析の結果を用いると問題を簡単にすることができる．前節で行ったクロス分析によって，顧客の必要とする班の人数は最大 3 人，レベルは 3 種類 (A，B，C) であることがわかっている．このような場合には，可能な班を列挙してしまうのが良い．可能な班を列挙したものをパターンと呼ぶと，定式化に必要な記号は以下のようになる．

[9] キュウリの漬け物の意味．漬け物のようにオブジェクトを保管しておくという意味．

t: 期（日）を表す添え字.
U_{pt}: 期 t での班 p の作業時間の上限.
J: 顧客の集合.
L: レベルの集合；添え字は l.
N_l: レベル l を持つ営業マンの人数.
C_p: パターン p の班で処理可能な顧客の集合.
n_p: パターン p の人数.
PAT: パターン集合.
PAT_l: 最上位レベル l の営業マンを含むパターンの集合.
M_i: 顧客 i の訪問時間.
d_{ij}: 顧客 i, j 間の移動距離.

以下の 0-1 変数を用いてモデルを定式化する.

y_{jpt}: 顧客 j をパターン p の班が期 t に訪問するとき 1，それ以外は 0.
X_{pt}: 期 t にパターン p の班を編成するとき 1，それ以外は 0.

問題は二次の目的関数を持つ整数最適化問題として，以下のように定式化できる.

$$
\begin{aligned}
\text{min.} \quad & \sum_{p,t}\sum_{i<j} d_{ij} y_{ipt} y_{jpt} \\
\text{s.t.} \quad & \sum_{p\in PAT} n_p X_{pt} \le \sum_{l} N_l && \forall t \\
& \sum_{p\in PAT_l} X_{pt} \le N_l && \forall l, t \\
& \sum_{j\in C_p} M_j y_{jpt} \le U_{pt} X_{pt} && \forall p, t \\
& \sum_{p,t} y_{jtp} = 1 && \forall j \\
& y_{jtp} \in \{0,1\} && \forall j, p, t \\
& X_{pt} \in \{0,1\} && \forall p, t
\end{aligned}
$$

最初の式は目的関数であり，班が期に訪問する顧客間の距離の合計を最小化することを表す．2，3 番目の式は，班編成のための制約条件である．4 番目の式は，作業時間上限を表す．5 番目の式は，各顧客がいずれかの班で処理されなければならないことを表す.

上の問題は非凸の二次の目的関数を持ち，かなり規模が大きい．よって，メタヒューリスティクスに基づく制約最適化ソルバー SCOP[10] を用いることにする.

SCOP では，変数を領域と呼ばれる集合から 1 つの値を選択することによって定義する．上の問題においては，$\{0,1\}$ の領域を持つ変数 y_{jtp} でなく，顧客 j を訪問可能なパターン p と期 t の組の集合を領域とした変数 Y_j を用いるほうが効率が良い．これによって変数の数が大幅に削減され，さらに 5 番目の制約が不要になる．実装は，Python を用いれば比較的容易にできる．実装法については拙著 [2] の付録 C を参照されたい.

[10] Solver for COnstraint Programming の略．Nonobe–Ibaraki [1] によって開発されたものに Python によるインターフェイスを付加したものである．詳細については http://www.logopt.com/scop.htm 参照.

7. モデル実行（最適化）

モデルを定式化し実装が済んだら，いよいよ最適化を行う．ここで注意すべきことは，実際問題においては常に最適解（はおろか実行可能解）が返されるとは限らないということだ.

制約最適化ソルバー SCOP では，すべての式を制約として扱う．制約には「重み」と呼ばれるパラメータを付加して，制約の逸脱量に重みを乗じた値の合計を最小化する．上の仮想の問題においては，目的関数に対応する制約に対する重みを 1 にし，その他の制約に対する重みを大きな値に設定する．破ることを許す制約がある場合には，重みを適当な値に設定する．たとえば，作業時間上限の逸脱を許す場合には，4 番目の制約の重みを 1 分あたりの残業代に設定すれば良い.

数理最適化ソルバーの場合には，自動的には制約からの逸脱をペナルティにしてくれないので，面倒くさがらずに，結果を解析するコードを入れておくことを推奨する．たとえば，代表的な数理最適化ソルバー Gurobi [2] を用いた場合には，以下のように最適化を行った後で，モデルの状態（Status 属性）を調べて適当な処理を追加する.

```
model.optimize()
status = model.Status
if status == 5 or status == 4:
    model.setObjective(0, GRB.MAXIMIZE)
    model.optimize()
    status = model.Status
if status == GRB.Status.OPTIMAL:
    print "Unbounded"
elif status == GRB.Status.INFEASIBLE:
    print "Infeasible"
else:
    print "status", status
```

上のコードでは，`Status` 属性が「非有界」を表す 5 (`GRB.Status.UNBOUNDED`) もしくは「実行不可能か非有界」を表す 4 (`GRB.Status.INF_OR_UNBD`) の場合には，目的関数を 0 ベクトルとして再び最適化を行い，それが最適解を持てば非有界，持たないならば実行不可能と判定している.

実行不可能になった場合には，制約条件に無理がある場合が多い．どの制約に無理があるのかを調べるため

の 1 つの方法として，既約不整合部分系 (Irreducible Inconsistent Subsystem: IIS) を使う方法がある．既約不整合部分系とは，実行不可能になっている原因の制約と変数から構成され，Gurobi ではモデルオブジェクトの `computeIIS` メソッドで計算できる．

実行可能解からの逸脱ペナルティの和を最小化したい場合には，簡易メソッド `feasRelaxS` を用いるか，自分で制約の逸脱をペナルティを支払えば許すように定式化を書き換えて再び最適化を行う．これらの方法の詳細については，拙著 [2] の 1.10 節を参照されたい．

8. 結果解析（可視化と評価）

データ解析のところでも可視化は重要だが，結果の可視化はもっと重要である．特に，最適化をした結果とモデルの妥当性は，人間が判断せざるをえないが，それには人が見てわかるような図に出すことが望ましい．幸い，Python には `matplotlib`[11] と呼ばれる可視化モジュールがあり，`pylab` インターフェイスを通して商用の MATLAB と同じようにさまざまな図を「気軽に」作成することができる．

たとえば，上の例題で作成した顧客データを保管したデータフレーム `df` の訪問時間（分単位）の度数分布表を作成するには，以下のようにすれば良い．

```
import pylab
pylab.hist(df["minutes"])
```

Python にはネットワークを描画したり簡単な最適化解析を行うためのモジュールである `networkX`[12] もある．このモジュールは，大規模なグラフを描画機能だけでなく，グラフに関する基本的なアルゴリズムをすべて搭載しているので短時間での開発には便利である．

9. おわりに

以上，Python 言語のさまざまなモジュールを用いることによって，データ解析，最適化，可視化が容易にできることを示した．最後に，筆者がしばしば利用している（最適化以外の）便利なモジュール群をまとめておく．

1. numpy　http://www.numpy.org/
   数値計算を効率的に行うためのモジュール．以下の各モジュールはこのモジュールを基礎として作られているので，まずはこれをインストールする必要がある．
2. matplotlib　http://matplotlib.org/
   グラフ描画モジュール．これに含まれる pylab インターフェイスは，商用の MATLAB[13] と同じような機能を持つ．
3. pandas　http://pandas.pydata.org/
   データ解析ライブラリ．特定用途向けの言語である R（S 言語の無償版）も同様の機能を持つが，pandas と較べると遅い．
4. networkX　http://networkx.github.io/
   グラフ・ネットワーク関連のモジュール．筆者は大規模なネットワークの描画の際によく利用する．搭載されている基本グラフアルゴリズムは便利だが，最短路アルゴリズムなどの実装はいまいちであるので，大規模問題の場合には自分で実装するか，より高速な専用ライブラリ[14] を用いたほうが良い．
5. SciPy　http://www.scipy.org/
   さまざまな科学技術計算のための実装を含むモジュール．非線形最適化はいまいちだが，計算幾何学，統計などのモジュールは便利である．

参考文献

[1] K. Nonobe and T. Ibaraki, “A tabu search approach to the constraint satisfaction problem as a general problem solver,” *European J. Operational Research*, **106**, 599–623, 1998.

[2] 久保幹雄，ジョア・ペドロ・ペドロソ，村松正和，アブドル・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.

[3] 久保幹雄，“モデリングのための覚え書き，” オペレーションズ・リサーチ，**50**, 255–258, 2005.

[4] 久保幹雄，“数理最適化とメタヒューリスティクス，” オペレーションズ・リサーチ，**58**, 723–728, 2013.

[11] http://matplotlib.org/ からダウンロードできる．
[12] http://networkx.github.io/ からダウンロードできる．
[13] MATLAB は特定用途向けの言語である．pylab は Python 内で使えるので柔軟性があり，何かと便利である．
[14] 高速な最短路アルゴリズムの実装は，筆者らが以前やった最短路プロジェクトのページ http://www.logopt.com/mikiokubo/SP/ からダウンロードできる．