PYNUOPT

マニュアル
V1.1

株式会社NTTデータ数理システム

2017年1月

MATHEMATICAL SYSTEMS INCARNATION

目次

1. はじめに

PYNUOPT とは Numerical Optimizer の Python インタフェースのことです．本インタフェースで実現している機能は「Numerical Optimizer/SIMPLE 外部接続マニュアル」(以後，外部接続マニュアル)の 3 章で説明をしている solveLP 関数，solveQP 関数になります．

以下では，

■Numerical Optimizer のモデリング言語 SIMPLE について

■外部接続マニュアルの 3 章について

■一般的な Python の使用方法

についてご理解いただいていることを想定しておりますが，1 項目目と 2 項目目については本マニュアルと外部接続マニュアルを逐次確認することで十分読み進めていくことができます．

1.1 サポートプラットフォーム

PYNUOPT がサポートするプラットフォームは Windows7，Windows8.1，Windows10 になります．いずれの Windows も 32bit 版，64bit 版で動作します．

また，Python のバージョンは 2.5，2.6，2.7 の 32bit 版，64bit 版になります．なお，テストは http://www.python.org/からダウンロードできる Python で行っておりますが，一般的な Python であれば，上記バージョンが一致すれば動作すると考えられます．

1.2 インストール

PYNUOPT のインストールは利用する Python にてコマンドを入力することで実行することができます．以下，インストールの手順になります．

PYNUOPT のデフォルトのインストール先は，利用する Python のインストール先の「lib¥site-packages」フォルダーになります．そのため，インストールするユーザは本フォルダーに対して書き込み権限が必要となります．インストール作業を実行する際は，本フォルダーに対して書き込み権限があるユーザでログオンしてください．

1. Numerical Optimizer のインストールメディアから PYNUOPT ファイルをコピーする

PYNUOPT ファイルは

%CD-ROM%¥pynuopt¥pynuopt-1.1.zip

です(%CD-ROM%はインストールメディアのトップフォルダーです)．

この pynuopt-1.1.zip をインストールする PC のローカルのハードディスクにコピーをしてください．たとえば，c¥tmp¥pynuopt-1.1.zip とします．

2．pynuopt-1.1.zip を展開する

1 の zip ファイルを解凍して展開してください．たとえば，c:¥tmp¥pynuopt-1.1 以下に展開したものとします.

3．Windows のコマンドプロンプトを起動する

スタートメニューからコマンドプロンプトを起動してください.

UAC(User Account Control)がオンになっている場合は，Python のインストール先によってはコマンドプロンプトを起動する際に右クリックメニューの「管理者として実行」から起動する必要があります.

4．pynuopt-1.1.zip を展開したフォルダーに移動する

3 で起動したコマンドプロンプト上で

```
> cd /d c:¥tmp¥pynuopt-1.1
```

と実行し，展開したフォルダーに移動してください.

5．PYNUOPT をインストール

コマンドプロンプト上で次のように入力して PYNUOPT をインストールしてください.

```
> python setup.py install
running install
running build
running build_py
copying nuopt¥utils.py -> build¥lib¥nuopt
…
running install_egg_info
Writing C:¥Python27¥Lib¥site-packages¥pynuopt-1.1-py2.7.egg-info
```

最後の行の「C:¥Python27¥Lib¥site-packages¥pynuopt-1.1-py2.7.egg-info」というファイル名を記録してください．アンインストールをする際に必要となります.

なお，この時，PYNUOPT を実行(利用)する python で実行する必要があります.

6．%NUOPT%¥bin にパスを通す

PYNUOPT を実行するには Numerical Optimizer の実行パスにパスを通す必要があります.

スタートメニューから

[MSI Solutions]-[NUOPT]-[NUOPT の環境設定]

を実行していただくことで，Numerical Optimizer の実行パスにパスを通すことができます.

なお，この時点で一度 PC を再起動してください.

7．サンプルを実行する

インストールメディアにはサンプルプログラムスクリプトがあります.

%CD-ROM%¥pynuopt¥sampleLP.py

%CD-ROM%¥pynuopt¥sampleQP.py

この 2 つのファイルをローカルな PC のカレントフォルダーにコピーしてから，次のように実行してください.

```
> python sampleLP.py
> python sampleQP.py
```

インストールに成功している場合は，下にある出力例と同じ出力が得られます．なお，2 つ目のスクリプトは Numerical Optimizer のモジュールがフルセットである必要があります.

実行に失敗した場合は，nuopt-support@msi.co.jp までご連絡ください.

sampleLP の出力例

```
=====================================solveLP============================

[About Numerical Optimizer]
MSI Numerical Optimizer xx.x.x (NLP/LP/IP/SDP module)
      <with META-HEURISTICS engine "wcsp"/"rcpsp">
      <with GLOBAL-OPTIMIZATION add-on "global">
      <with DERIVATIVE-FREE-OPTIMIZATION add-on "DFO">
     , Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Problem and Algorithm]
PROBLEM_NAME                                        anon.LP
NUMBER_OF_VARIABLES                                       2
NUMBER_OF_FUNCTIONS                                       4
PROBLEM_TYPE                                   MINIMIZATION
METHOD                                         HIGHER_ORDER

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=3.8e+001 .... 3.2e-004 . 7.9e-009
<iteration end>
```

```
[Result]
STATUS                                          OPTIMAL
VALUE_OF_OBJECTIVE                          765.7142858
ITERATION_COUNT                                       7
FUNC_EVAL_COUNT                                      10
FACTORIZATION_COUNT                                   8
RESIDUAL                                 7.898421803e-009
ELAPSED_TIME(sec.)                                 0.01
SOLUTION_FILE                                 solver.sol
[Objective]:
   180.0*x1 + 160.0*x2
[Constraint]:
   [  1]  6.0*x1 + 1.0*x2 >= 12
   [  2]  3.0*x1 + 1.0*x2 >= 8
   [  3]  4.0*x1 + 6.0*x2 >= 24
errorCode: 0
optValue: 765.714285831
x:
[1] val=1.71428571489 / dual=3.41506745151e-08
[2] val=2.8571428572 / dual=-2.73205437291e-08
c:
[1] val=13.1428571465 / dual=5.12259531374e-08
[2] val=8.00000000186 / dual=31.4285713025
[3] val=24.0000000027 / dual=21.4285714422
```

sampleQP の出力例

```
=====================================solveQP============================


[About Numerical Optimizer]
MSI Numerical Optimizer xx.x.x (NLP/LP/IP/SDP module)
      <with META-HEURISTICS engine "wcsp"/"rcpsp">
      <with GLOBAL-OPTIMIZATION add-on "global">
      <with DERIVATIVE-FREE-OPTIMIZATION add-on "DFO">
     , Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Problem and Algorithm]
```

```
PROBLEM_NAME                                          anon.QP
NUMBER_OF_VARIABLES                                         2
NUMBER_OF_FUNCTIONS                                         4
PROBLEM_TYPE                                       MINIMIZATION
METHOD                                          TRUST_REGION_IPM

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=7.1e+000 .... 1.3e+000 .... 5.2e-001 .... 5.3e-003 .... 4.4e-006 .
        5.3e-009
<iteration end>

[Result]
STATUS                                                  OPTIMAL
VALUE_OF_OBJECTIVE                                  2.318181825
ITERATION_COUNT                                            22
FUNC_EVAL_COUNT                                            26
FACTORIZATION_COUNT                                        37
RESIDUAL                                       5.297266072e-009
ELAPSED_TIME(sec.)                                         0.01
SOLUTION_FILE                                        solver.sol
[Objective]:
   -3.0*x1 + 1.0*x2 + 5.5*x1*x1 + 11.0*x2*x2
[Constraint]:
   [  1]  (-1.0*x1) + 0.1*x2 >= -1, <= 10
   [  2]  (-0.2*x1) + (-1.0*x2) >= -2, <= 10
   [  3]  2.0*x1 + 1.0*x2 >= 2, <= 10
errorCode: 0
optValue: 2.31818182484
x:
[1] val=0.939393937084 / dual=-1.09791154693e-07
[2] val=0.121212127647 / dual=5.48971367648e-08
c:
[1] val=-0.927272724319 / dual=9.14930620068e-08
[2] val=-0.309090915064 / dual=3.93485595971e-09
```

```
[3] val=2.00000000181 / dual=3.66666675192
```

1.3 アンインストール

アンインストーラはついておりませんので,物理的に削除していただくことでアンインストールが完了します．インストールの「5．PYNUOPT をインストール」で記録していたファイル及び同じフォルダー下に「NUOPT」というフォルダーがありますので,両方とも削除してください.

1.4 簡単なサンプル(LP)

次のような SIMPLE で記述されたモデル(2 変数，3 制約のモデル)を例にします.

```
Variable x;
Variable y;
// 目的関数
Objective cost(type=minimize);
cost = 180*x + 160*y;
// 制約
6*x + y >= 12;
3*x + y >= 8;
4*x + 6*y >= 24;
// 変数の上下限
0 <= x <= 5;
0 <= y <= 5;
solve();

x.val.print();
```

変数 x を変数番号 1，変数 y を変数番号 2 に対応すると考えてください．変数番号とは 1 始まりの整数の番号ですが, n 変数の問題の場合は各々の変数が番号 1 から n までのいずれかと 1 対 1 対応している必要があります.

制約は 3 制約あります．変数と同様に上から順に制約番号 1，2，3 に対応すると考えてください.

PYNUOPT の記述例

```
 1: import nuopt.utils
 2: if __name__ == '__main__':
 3:   solver = nuopt.utils.solveLP(3, 2)
 4:
 5:   solver.x[1].l = 0
 6:   solver.x[1].u = 5
 7:   solver.x[2].l = 0
 8:   solver.x[2].u = 5
 9:
10:   solver.c[1].l = 12
11:   solver.c[2].l = 8
12:   solver.c[3].l = 24
13:
14:   solver.A[1,1] = 6
15:   solver.A[1,2] = 1
16:   solver.A[2,1] = 3
17:   solver.A[2,2] = 1
18:   solver.A[3,1] = 4
19:   solver.A[3,2] = 6
20:
21:   solver.objL.value([180, 160])
22:
23:   res = solver.solve()
24:
25:   print res
```

1 行目：

PYNUOPT 用のモジュールを import します.

2 行目：

PYNUOPT を使用する際は必ず

```
if __name__ == '__main__':
```

を記述し，そのあとに Python スクリプトを作成してください.

3 行目：

2 変数 3 制約の線形な問題を表すインスタンスを作成します. 変数の数と制約の数の指定の順が逆になっている点についてご注意ください.

5 行目から 8 行目：

変数 x，y の上下限値 0，5 を設定します.

10 行目から 12 行目：

制約の下限値を設定します．各々制約には上限値がないため設定をしていません.

14 行目から 19 行目：

3 つの制約式の変数の係数を設定します.「solver.A[制約番号，変数番号] = 値」という形式で設定します.

21 行目：

目的関数の係数を設定します.

23 行目：

最適化計算を実行します．この例では，次のような出力が表示されます．通常の Numerical Optimizer の出力と全く同じです．なお，カレントフォルダーに solver.sol というファイルが出力されており，この解の詳細情報が出力されています.

```
[About Numerical Optimizer]
MSI Numerical Optimizer xx.x.x (NLP/LP/IP/SDP module)
          <with META-HEURISTICS engine "wcsp"/"rcpsp">
          <with GLOBAL-OPTIMIZATION add-on "global">
          <with DERIVATIVE-FREE-OPTIMIZATION add-on "DFO">
         , Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Problem and Algorithm]
PROBLEM_NAME                                    anon.LP
NUMBER_OF_VARIABLES                                   2
NUMBER_OF_FUNCTIONS                                   4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                      HIGHER_ORDER
```

```
[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=3.8e+001 .... 3.2e-004 . 7.9e-009
<iteration end>

[Result]
STATUS                                          OPTIMAL
VALUE_OF_OBJECTIVE                           765.7142858
ITERATION_COUNT                                        7
FUNC_EVAL_COUNT                                       10
FACTORIZATION_COUNT                                    8
RESIDUAL                                 7.898421803e-009
ELAPSED_TIME(sec.)                                  0.01
SOLUTION_FILE                                 solver.sol
```

25 行目：

計算結果を表示します. この例では次のような出力がなされます.

```
errorCode: 0
optValue: 765.714285805
x:
[1] val=1.71428571475 / dual=2.64337467775e-08
[2] val=2.85714285718 / dual=-2.11469998886e-08
c:
[1] val=13.1428571457 / dual=3.96505850368e-08
[2] val=8.00000000144 / dual=31.428571331
[3] val=24.0000000021 / dual=21.4285714391
```

1.5 簡単なサンプル(QP)

次のような SIMPLE で記述されたモデル(2 変数, 3 制約, 目的関数に 2 次の項があるモデル)を例にします.

```
Variable x;
Variable y;
// 目的関数
Objective cost(type=minimize);
cost = -3.0*x + 1.0*y + 5.5*x*x + 11*y*y;
// 制約
-1 <= -0.1*x + 0.1*y <= 10;
-2 <= -0.2*x - 0.1*y >= 10;
2 <= 2*x + 1.0*y >= 10;
// 変数の上下限
0 <= x <= 1;
0 <= y <= 2;
solve();

x.val.print();
y.val.print();
```

PYNUOPT の記述例

```
 1: import nuopt.utils
 2: if __name__ == '__main__':
 3:   solver = nuopt.utils.solveQP(3, 2)
 4:
 5:   solver.x[1].bound(0,1)
 6:   solver.x[2].bound(0,2)
 7:   solver.x[1].type = nuopt.utils.VTYPE.REAL
 8:   solver.x[2].type = nuopt.utils.VTYPE.REAL
 9:
10:   solver.c[1].bound(-1, 10)
11:   solver.c[2].bound(-2, 10)
12:   solver.c[3].bound(2, 10)
13:
14:   solver.A[1,1] = -1.0
15:   solver.A[1,2] = 0.1
16:   solver.A[2,1] = -0.2
17:   solver.A[2,2] = -1.0
18:   solver.A[3,1] = 2.0
19:   solver.A[3,2] = 1.0
20:
21:   solver.objL.value([-3.0, 1.0])
22:
23:   solver.objQ[1,1] = 11
24:   solver.objQ[2,2] = 22
25:
26:   res = solver.solve()
27:
28:   print res
```

基本的には 1.4 節と同じですが，16 行目，17 行目に 2 次の項に相当する記述があります.

2．PYNUOPT 詳細

本章では PYNUOPT の各機能の詳細についてまとめています.

2.1 solveLP，solveQP

solveLP，solveQP の宣言は次の形式になります.

```
nuopt.utils.solveLP(制約式の数，変数の数，options=None)
nuopt.utils.solveQP(制約式の数，変数の数，options=None)
```

options は Numerical Optimizer に渡すオプションのことであり，デフォルト値は None(デフォルトオプションと解釈される)となります．オプションについては，2.8 節を参照してください.

上記のように宣言すると, solveLP, solveQP インスタンスを得ることができます. solveLP インスタンスは次のメンバー変数にアクセスすることができ, これらに値を代入することで数理計画問題を定義することができます．たとえば，solveLP インスタンスを ins とすると，

ins.x ： 変数の上下限，変数の型情報
ins.c ： 制約式の上下限
ins.A ： 制約式の変数に対する係数
ins.objL ： 目的関数の係数
ins.x0 ： 変数の初期値

でアクセスすることができます.

solveQP インスタンスでは, solveLP インスタンスのメンバーに加えて次の 2 つのメンバー変数にアクセスすることができます.

ins.QC ： 制約式の 2 次の項に対する係数
ins.objQ ： 目的関数の 2 次の項の係数

ins.x は 2.2 節，ins.c および ins.A は 2.3 節，ins.objL は 2.4 節，ins.x0 は 2.5 節，ins.QC は 2.6 節，ins.objQ は 2.7 節で詳細に説明しています.

また，数理計画問題を定義した後は，メンバー関数 solve()を実行することで最適化計算を開始します．solve()関数については，2.9 節をご覧ください.

2.2 変数

メンバー変数 x にて，変数の上下限，変数の型を指定することができます.

ins.x[変数番号].l ： 指定した変数番号の変数の下限値
ins.x[変数番号].u ： 指定した変数番号の変数の上限値
ins.x[変数番号].type ： 指定した変数番号の変数の型

変数番号とは，n 変数の問題の場合 1 から n までの整数値であり，各変数に 1 対 1 に対応しています.

下限値，上限値ともデフォルト値は指定なしであり，明示的に指定なしにする場合は None を代入してください．また，

ins.x[変数番号].bound(下限値，上限値)

のように bound()メソッドで上下限値を同時に指定することもできます.

変数の型は実数型，整数型，0-1 変数型の 3 種類から選ぶことができます.

実数型 ： nuopt.utils.VTYPE.REAL
整数型 ： nuopt.utils.VTYPE.INTEGER
0-1 変数型 ： nuopt.utils.VTYPE.BINARY

のいずれかを指定してください．なお，デフォルト値は実数型です.

変数情報についての確認は

ins.x.show()

とすることで行えます．上の 1.4 節の場合の show()関数の結果は次のようになります.

```
===Variable Bound Show===
[1]: type=REAL: >= 0, <= 5
[2]: type=REAL: >= 0, <= 5
```

2.3 制約式

メンバー変数 c，A にて制約式の上下限，制約式の係数を指定することができます.

ins.c[制約番号].l ： 指定した制約番号の制約式の下限値
ins.c[制約番号].u ： 指定した制約番号の制約式の上限値
ins.A[制約番号,変数番号] ： 指定した制約番号の制約式の指定した変数の係数

制約番号とは，m 制約の問題の場合 1 から m までの整数値であり，各制約に 1 対 1 に対応しています.

下限値，上限値ともデフォルト値は指定なしであり，明示的に指定なしにする場合は None を代入してください．また，

ins.c[制約番号].bound(下限値，上限値)

のように bound()メソッドで上下限値を同時に指定することもできます.

係数のデフォルト値は 0 になります.

制約情報についての確認は

```
ins.c.show()
ins.A.show()
```

とすることで行えます．上の 1.4 節の場合の show()関数の結果は次のようになります．

```
===Constraint Bound Show===
[1]: >= 12
[2]: >= 8
[3]: >= 24
===A Show===
[1,1]: 6.0
[1,2]: 1.0
[2,1]: 3.0
[2,2]: 1.0
[3,1]: 4.0
[3,2]: 6.0
```

2.4 目的関数

メンバー変数 objL にて目的関数の係数を指定することができます．

```
ins.objL[変数番号] : 指定した変数番号の目的関数の係数
```

係数のデフォルト値は 0 です．また，

```
ins.objL.value([係数リスト])
```

のように value()メソッドで全ての係数を同時に指定することもできます．

目的関数についての確認は

```
ins.x0.show()
```

とすることで行えます．上の 1.4 節の場合の show()関数の結果は次のようになります．

```
====objL Show===
[1]: 180.0
[2]: 160.0
```

2.5 変数の初期値

メンバー変数 x0 にて変数の初期値を設定することができます．

```
ins.x0[変数番号] : 指定した変数番号の変数の初期値
```

初期値のデフォルト値は 0 です．また，

```
ins.x0.value([初期値リスト])
```

のように value()メソッドで全ての係数を同時に指定することもできます．

初期値についての確認は

```
ins.objC.show()
```

とすることで行えます．上の 1.4 節の場合すべての初期値がデフォルト値であるため出力はされませんが，「2.4 目的関数」と同じ書式で出力されます．

2.6 制約式 (2 次の項)

メンバー変数 QC にて 2 次の制約式の係数を指定することができます．

```
ins.QC[制約番号][変数番号 1，変数番号 2] = 値
```

ここの制約番号は外部接続マニュアルの「3.2.5 制約式の 2 次の部分にかかわるもの」の ifunQC に相当します．同様に変数番号 1 は irowQC，変数番号 2 は jcolQC に対応しています．

また，外部接続マニュアルにある通り，たとえば

```
ins.QC[1][1,2] = 1
ins.QC[1][2,1] = 2
```

のように上三角部分と下三角部分の対称の要素の両方ともに値をいれた場合は同一の要素の非零要素が複数与えられたと解釈しますので，ご注意ください．詳細は外部接続マニュアルをご覧ください．

2.7 目的関数 (2 次の項)

メンバー変数 objQ にて 2 次の目的関数の係数を指定することができます．

```
ins.objQ[変数番号 1，変数番号 2] = 値
```

ここの変数番号 1 は外部接続マニュアルの「3.2.4 目的関数 2 次の部分にかかわるもの」の irowQ に相当します．同様に変数番号 2 は jrowQ に対応しています．

また，2.6 と同様に上三角部分と下三角部分の対称の要素の両方ともに値をいれた場合は同一の要素の非零要素が複数与えられたと解釈します．

2.8 オプション

Numerical Optimizer へのオプションは

nuopt.utils.Options

クラスで指定することができます．指定できるオプションは次になります．

- method
- scaling
- maxitn
- eps
- clevel
- rounding
- feasPump
- neighbourSearch
- rins
- addToCutoff
- res_addToCutoff
- cutoff
- p
- maxnode
- maxtim
- maxmem
- bbthreads
- gaptol
- tolx
- told
- maxintsol
- iisDetect
- noDefaultSolout
- mtxfree

各オプションの意味，指定値，デフォルト値は「Numerical Optimizer/SIMPLE マニュアル」の「15.5 パラメーター一覧」をご覧ください．ただ，method オプションで指定できる解法は wcsp，rcpsp 以外になります．

例えば，次のように指定します．

```
options = nuopt.utils.Options()
options.method = “simplex”
options.maxtim = 60
```

```
nuopt.utils.solveLP(3, 2, options=options)
```

2.9 求解

メソッド solve()にて最適化計算を実行できます.

solve()メソッドのオプションには次のようなものがあります.

■silent_flag

■minimize

■out

silent_flag オプションには True または False を指定することができます. True を指定すると solveLP, solveQP を実行した際の標準出力が表示されなくなります. デフォルトは False です.

minimize オプションは, 最適化問題において目的関数最小化または最大化を指定するオプションです. True を指定すると最小化となり, False を指定すると最大化となります. デフォルトは True(最小化)になります.

out オプションは silent_flag オプションが False の場合に意味があり, 2 種類の設定方法があります. 1 つ目は, Python のリストオブジェクトを設定する方法です. この設定では指定したリストオブジェクトに実行時の標準出力が格納されて返ってきます. たとえば, 次のように使います.

```
buf = []
solver.solve(out=buf)
for line in buf:
   print line
# print ‘¥n’.join(buf)と 1 行で記述することも可能です
```

2 つ目は, ファイルオブジェクトを設定する方法です. あらかじめ書き込みモードでオープンしたファイルオブジェクトを指定することで指定されたファイル等に実行時の標準出力をリダイレクトすることができます. たとえば次のように使います.

```
fout = open(‘log.txt’, ‘w’)
solver.solve(out=fout)
fout.close()
```

この場合では, ファイル log.txt に実行時の標準出力が出力されています.

本オプションのデフォルトは標準出力にそのまま出力するとなっています.

2.10 その他メソッド

showObjective()メソッド

目的関数を式の形に整形して出力します.

```
[Objective]:
  180.0*x1 + 160.0*x2
```

showConstraint()メソッド

制約式を式の形に整形して出力します.

```
[Constraint]:
  [  1]  6.0*x1 + 1.0*x2 >= 12
  [  2]  3.0*x1 + 1.0*x2 >= 8
  [  3]  4.0*x1 + 6.0*x2 >= 24
```

また，制約番号を入力することで，指定した制約のみを出力することもできます.

showSystem()メソッド

showObject()と showConstraint()メソッドの両方とも実行します.

addVariable()メソッド

変数を追加します．変数番号は addVariable()を実行する前の変数の数+1 となり，本メソッドの戻り値です.

また，本メソッドには次のオプションがあり，変数の追加と同時に変数の型，上下限値を設定できます.

type: 変数の型

lower: 下限値

upper: 上限値

いずれのオプションもデフォルト値は「指定しない」です.

addConstraint()メソッド

制約式を追加します.制約番号は addConstraint()を実行する前の制約式の数+1 となり，本メソッドの戻り値です.

また，本メソッドには次のオプションがあり，制約式の追加と同時に上下限値を設定できます.

lower: 下限値

upper: 上限値

いずれのオプションもデフォルト値は「指定しない」です.

deleteVariable()メソッド

指定した変数番号の変数を削除します.

ただ，完全に削除するわけではなく下で説明をする restoreVariable()メソッドで復元することができます.

deleteConstraint()メソッド

指定した制約番号の制約式を削除します.

ただ，完全に削除するわけではなく下で説明をする restoreConstraint()メソッドで復元することができます.

restoreVariable()メソッド

指定した削除された変数番号の変数を復元します.

restoreConstraint()メソッド

指定した削除された制約番号の制約式を復元します.