Nuorium Optimizer

マニュアル
V24

株式会社NTTデータ数理システム

2022年5月

MATHEMATICAL SYSTEMS INCARNATION

目　次

第 1 章 マニュアル紹介

1.1 マニュアルラインアップ紹介

Nuorium Optimizer には用途に応じて以下のマニュアルが準備されております.

1. **Nuorium Optimizer マニュアル**
   Nuorium Optimizer の詳細機能が説明されています. 読者にはある程度 Nuorium Optimizer に馴染みがあることを想定しております.
2. **Nuorium Optimizer/PySIMPLE マニュアル**
   Nuorium Optimizer 付属の Python ベースのモデリング言語 PySIMPLE のマニュアルです.
3. **Nuorium Optimizer/SIMPLE マニュアル**
   Nuorium Optimizer 付属の C++ベースのモデリング言語 SIMPLE のマニュアルです.
4. **Nuorium Optimizer/RSIMPLE マニュアル**
   Nuorium Optimizer 付属の R ベースのモデリング言語 RSIMPLE のマニュアルです.
5. **Nuorium Optimizer/SIMPLE 例題集**
   典型的な数理最適化問題に対する Nuorium Optimizer/SIMPLE の記述方法が記されています. 具体的な問題を手本として学ぶのに最適なマニュアルです.
6. **Nuorium Optimizer/SIMPLE チュートリアル**
   初めて Nuorium Optimizer をご利用の方におすすめのチュートリアルです.
7. **Nuorium スタートガイド**
   初めて数理最適化専用 GUI Nuorium をご利用の方におすすめのスタートガイドです.
8. **Nuorium Optimizer/Excel アドインマニュアル**
   Nuorium Optimizer と Microsoft Excel との連係機能である Excel アドインの詳細機能が説明されています.
9. **Nuorium Optimizer/SIMPLE 外部接続マニュアル**
   外部のプログラムから Nuorium Optimizer を呼び出して利用する方法が説明されています.

一部マニュアルに関してはオンラインマニュアルを提供しております. オンラインマニュアルは https://www.msi.co.jp/nuopt/docs/index.html からご覧ください.

1.2 本マニュアルの構成

本マニュアルは以下のような内容から構成されています.

2〜6:求解ソルバ Nuorium Optimizer の解説

7:高度な利用法
付録:エラーメッセージ・アルゴリズムの解説，参考文献の紹介

第2章 標準出力

数理最適化問題が Nuorium Optimizer で解かれた場合，最終的な解情報の一部が標準出力に出力されます．以下はその一例です．

```
[Problem and Algorithm]
PROBLEM_NAME                                        a
NUMBER_OF_VARIABLES                                 5
NUMBER_OF_FUNCTIONS                                 2
PROBLEM_TYPE                                MAXIMIZATION
METHOD                                      HIGHER_ORDER

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=7.3e+001 .... 4.1e-003 .. 6.4e-010
<iteration end>

[Result]
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                  1049
ITERATION_COUNT                                        8
FUNC_EVAL_COUNT                                       11
FACTORIZATION_COUNT                                    9
RESIDUAL                              6.352800465e-010
ELAPSED_TIME(sec.)                                  0.01
SOLUTION_FILE                                      a.sol
```

2.1 アルゴリズム共通の出力

[Problem and Algorithm] で始まるセクションでは，以下のように問題の概要が出力されます．

```
PROBLEM_NAME                                        a
NUMBER_OF_VARIABLES                                 5
NUMBER_OF_FUNCTIONS                                 2
```

```
PROBLEM_TYPE                                    MAXIMIZATION
METHOD                                          HIGHER_ORDER
```

PROBLEM_NAME は「扱うモデルのファイル名」です．この例では a というモデルを解いています．

NUMBER_OF_VARIABLES は「変数 Variable の数」です．この例では変数が 5 個あります．

NUMBER_OF_FUNCTIONS は「関数の数」です．ここで言う関数とは，目的関数 Objective と制約式 Constraint を合わせたものになります．この例では，関数が 2 個（目的関数 1 個，制約式 1 個）あります．

PROBLEM_TYPE は問題が最小化問題（MINIMIZATION）なのか最大化問題（MAXIMIZATION）なのかを表示します．

METHOD は「最適化計算に用いたアルゴリズムの種類」です．この例では線形計画専用内点法（HIGHER_ORDER）を用いています．

[Result] で始まるセクションでは，以下のように最適化計算結果の要約が出力されます．

```
STATUS                                        OPTIMAL
VALUE_OF_OBJECTIVE                               1049
ITERATION_COUNT                                     8
FUNC_EVAL_COUNT                                    11
FACTORIZATION_COUNT                                 9
RESIDUAL                            6.352800465e-010
ELAPSED_TIME(sec.)                              0.01
SOLUTION_FILE                                  a.sol
```

最適化計算結果の要約の詳細については 2.10 をご参考ください．

2.2 内点法における出力

内点法を用いたアルゴリズムでは，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=7.3e+001 .... 4.1e-003 .. 6.4e-010
<iteration end>
```

<preprocess begin>と<preprocess end>の間は収束計算に入る前の処理の進行を，<iteration begin>と<iteration end>の間は収束計算の進行を示しています．計算の進行中に表示される数字（7.3e+001, 4.1e-003 など）は最適性条件の残差で，この表示はそれが計算の進行とともに減少していく様子を示しています．

2.3 単体法（simplex/dual_simplex），有効制約法，クロスオーバーにおける出力

単体法（hsimplex は除く），有効制約法，クロスオーバーを用いた場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
<preprocess begin>.........<preprocess end>
<iteration begin>
      ...1...................2
<iteration end>
```

<preprocess begin>と<preprocess end>の間は単体法の反復に入る前の処理の進行を示しています．

<iteration begin>と<iteration end>の間にあるドットは単体法の反復の進行を示しています（1 つのドットにつき，数回の反復を示しています）．また，文字 1 は実行可能解を探索するフェーズに遷移したことを示し，文字 2 は最適解を探索するフェーズへの遷移を示しています．

線形計画法，二次計画法に対して内点法からのクロスオーバー（options.crossover="on"）を指定した場合には

```
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=2.4e+001 .... 2.7e-005  1.4e-008
<iteration end>
<iteration begin>
        1222
<iteration end>
```

のように，内点法の経過表示の後に単体法の経過表示が現れます．

2.4 単体法（hsimplex）における出力

単体法（hsimplex）により線形計画問題を解く場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
[Progress]
dual-phase1 start
    Iter.       Objective    Primal Inf.      Dual Inf.  Time(s)
       1  -1.130341e+04   1.104730e+05   0.000000e+00       0.0
      38   0.000000e+00   0.000000e+00   0.000000e+00       0.0
dual-phase2 start
      39  -2.564421e+03   6.472308e+04   0.000000e+00       0.0
      89  -7.870890e+02   1.608960e+04   0.000000e+00       0.0
     151   1.368297e+03   2.325698e+03   0.000000e+00       0.0
```

```
      236   1.487996e+03   5.042927e+01   0.000000e+00      0.0
      295   2.691577e+03   0.000000e+00   0.000000e+00      0.0
cleanup perturbation
      296   2.690013e+03   0.000000e+00   0.000000e+00      0.0
```

また，二次計画問題を解く場合には，[Progress] で始まるセクションに以下のような実行経過が出力されます．

```
[Progress]
primal-phase1 start
    Iter.      Objective    Primal Inf.      Dual Inf.  Time(s)
       1   0.000000e+00   5.748232e+04   6.502000e+01      0.0
      51   1.628902e+04   9.463961e+03   3.889075e+01      0.0
     101   8.832791e+01   8.832791e+01   1.276675e+00      0.0
     105   0.000000e+00   0.000000e+00   0.000000e+00      0.0
cleanup perturbation
     106   0.000000e+00   0.000000e+00   0.000000e+00      0.0
primal-phase2 start
    Iter.      Objective    Primal Inf.      Dual Inf.  Time(s)  Freedom
     106   6.399420e+07   0.000000e+00   4.043570e+05      0.0        0
     156   2.687252e+07   0.000000e+00   1.284500e+01      0.0        1
     180   2.686595e+07   0.000000e+00   0.000000e+00      0.0        1
```

各項目の意味は次の通りです．

項目	意味
Iter.	反復回数
Objective	目的関数値
Primal Inf.	主変数に関する実行不可能性の値
Dual Inf.	双対変数に関する実行不可能性の値
Time(s)	経過時間（秒）
Freedom	基底解の自由度（二次計画問題のときのみ表示される）

2.5 制約充足問題ソルバ（wcsp）における出力

制約充足問題ソルバ（wcsp）を用いた場合には，[Progress] で始まるセクションに進捗が表示されます．

まず，<preprocess begin>から<preprocess end>では wcsp 実行前の準備がおこなわれます．非常に大規模な問題でない限り，ここに要する時間は僅かです．

```
<preprocess begin>.........<preprocess end>
preprocessing time: 0.000465(s)
```

次に，<iteration begin>から<iteration end>では wcsp 実行の進捗が表示されます．ここでは，最良解におけるハード制約のペナルティ，セミハード制約のペナルティ，ソフト制約のペナルティと最良解を見つけたときの時間及び反復回数が逐次表示されます．問題にセミハード制約が存在しない場合，セミハード制約のペナルティは表示されません．また，最良解を更新していない場合は表示が更新されませんが，計算は行われています．試行回数 (TryCount) が 2 以上の場合は試行回数分の進捗表示が行われます．また，wcsp で並列化オプションを有効にした場合はメインスレッドの進捗のみ表示されます．以下は進捗表示の一例です．

```
<iteration begin>
--- TryCount = 1 ---
# random seed = 1
(hard/semihard/soft) penalty= 45/12/4807, time= 0.00(s)
<greedyupdate begin>..........<greedyupdate end>
greedyupdate time= 0.00066(s)
(hard/semihard/soft) penalty= 0/8/3157, time= 0.00(s), iteration= 1
(hard/semihard/soft) penalty= 0/6/2905, time= 0.00(s), iteration= 2
(hard/semihard/soft) penalty= 0/6/2831, time= 0.00(s), iteration= 3
(hard/semihard/soft) penalty= 0/6/2783, time= 0.00(s), iteration= 4
(hard/semihard/soft) penalty= 0/6/2746, time= 0.00(s), iteration= 5
(hard/semihard/soft) penalty= 0/4/2775, time= 0.00(s), iteration= 8
(hard/semihard/soft) penalty= 0/2/3320, time= 0.00(s), iteration= 14
(hard/semihard/soft) penalty= 0/2/3283, time= 0.00(s), iteration= 15
(hard/semihard/soft) penalty= 0/0/4746, time= 0.00(s), iteration= 254
--- End Phase-I iteration ---
(hard/semihard/soft) penalty= 0/0/4094, time= 0.01(s), iteration= 368
# (hard/semihard/soft) penalty= 0/0/4094
# time = 0.01/0.02(s)
# iteration = 368/1000
<iteration end>
```

各項目の意味は次の通りです．

項目	意味
TryCount	現在の試行回数
random seed	使用した乱数の種
greedyupdate time	貪欲法によって初期解を更新するのにかかった時間

項目	意味
(hard/semihard/soft) penalty	ハード，セミハード，ソフト制約のペナルティ量
time	経過した時間
iteration	反復回数
End Phase-I iteration	ハード制約とセミハード制約が 0 になったことを意味する
time = X/Y(s)	最良解の発見に X 秒，wcsp の計算開始から終了までに Y 秒かかったことを意味する
iteration = X/Y	最良解の発見に X 回，wcsp の計算開始から終了までに Y 回反復したことを意味する

TryCount が 2 以上の場合は乱数の種を変えて複数回計算が行われます．全試行が終了した後にサマリが表示されます．以下はサマリの一例です．

```
<summary>
 trycount         hard     semihard         soft    iteration  time(s)
        1            0            0         2284         9796     0.49
        2            0            0         2160         5969     0.32
        3            0            0         2370         9760     0.50
        4            0            0         2298         4029     0.21
        5            0            0         2420         9326     0.46
        6            0            0         2422         3382     0.18
        7            0            0         2297         9574     0.47
*       8            0            0         2119         7698     0.39
```

サマリの内容は各試行における最良解のハード，セミハード，ソフト制約のペナルティ量と最良解を見つけるのにかかった時間及び反復回数です．"*"は全試行の中で最も良い解を表します．この例の場合，8 回目の試行で得られた解が最も良いことを意味します．

計算終了後，`[Result]` セクションに求解結果が出力されます．wcsp 特有の項目は次の通りです．

項目	意味
STATUS	求解ステータス．正常終了した場合は NORMAL，異常終了した場合は ERROR が出力されます．
TERMINATE_REASON	wcsp の計算が終了した理由
PENALTY	総ペナルティ量．ハード，セミハード，ソフト制約のペナルティ量の総和が出力されます．
RANDOM_SEED	最良解を見つけた乱数の種

2.6 分枝限定法における出力

混合整数計画問題を解く場合は，通常，自動的に分枝限定法が起動されます．その場合 [Progress] で始まるセクションでの実行経過の出力は次のようになり，求解の進行状況を確認できます．

```
#sol         upper         lower   gap(%)  time(s)  list  mem(MiB)
              +inf       84121.2     +inf      2.4     1       68  cut: 44
              +inf       85090.4     +inf      2.7     1       71  cut: 12
              +inf       85570.3     +inf      3.0     1       74  cut: 6
#1          139000       88862.7   22.003      5.1   190       98  sol: rens
#2          135125       88862.7   20.654      5.2   190       98  sol: rens
#3          134125         89294   20.066      5.4   194       99  sol: rens
#4          134025         89294   20.030      5.5   195       99  sol: rens
#5          133850         89294   19.967      5.7   196       99  sol: rins
#6          129350         89294   18.320      5.9   197       99  sol: relax
```

分枝限定法の進捗は，

- 新しい暫定解が得られた
- 所要メモリが 50MiB 以上変動した
- 15 秒経過した
- （並列化機能が無効のときに）切除平面を追加した

のいずれかの条件を満たせば出力されます．

各項目の意味は次の通りです．

表示	意味
#sol	発見した実行可能解の個数
upper	目的関数の上界値
lower	目的関数の下界値
gap(%)	上下界の相対ギャップ（パーセンテージ）
time(s)	経過時間（秒）
list	探索していない分枝木の葉の数
mem(MiB)	使用メモリ（メビバイト）
cut	追加した切除平面の数（並列化機能が無効であり，切除平面を追加したときに表示）
sol	解を発見した手法（解を発見したときに表示）
#worker	求解中の worker の数（並列化機能が有効のときに表示）

分枝限定法の場合は元問題とスケーリング後の問題について，係数値の大きさ（絶対値）の情報等が出力されます．また，スケーリング値の範囲も出力されます．

```
Coefficient Statistics (before scaling)
  Coefficient range         [min,max] : [1.00e-01,2.00e+01]
  RHS and bounds            [min,max] : [1.00e+00,6.00e+01]
  Objective                 [min,max] : [1.60e+01,4.00e+01]

Coefficient Statistics (after scaling)
  Coefficient range         [min,max] : [1.75e-01,2.42e+00]
  RHS and bounds            [min,max] : [1.00e+00,3.44e+01]
  Objective                 [min,max] : [8.60e+01,8.60e+01]
  Row scaling range         [min,max] : [3.41e-02,3.34e+00]
  Column scaling range      [min,max] : [3.56e-02,1.00e+00]
```

出力される情報は以下の 5 つです.

項目	意味
Coefficient range	係数行列の各要素の大きさ
RHS and bounds	制約式と変数の上下限値の大きさ
Objective	目的関数の係数値の大きさ
Row scaling range	係数行列の行方向のスケーリング値の大きさ
Column scaling range	係数行列の列方向のスケーリング値の大きさ

スケーリング値は，目的関数，制約式及び変数に乗じられる定数値です．例えば列方向のスケーリング値が 1.0e-08〜1.0e-6 の場合は変数が「1」だけ変動すると，この変動はソルバ内部では 1.0e-08〜1.0e-6 という微小な変動として解釈されます．このため，内部の変数の上下限制約違反の閾値が 1.0e-08 であるとすると上下限制約を「1」違反する解が許容されてしまう可能性があることになります．行方向スケーリング値も同様に，制約違反の許容具合に影響します.

スケーリング値（Row scaling range / Column scaling range）が小さすぎる場合は，スケーリングオプションを off にする，あるいは問題に与える変数の単位を見直すこと等が推奨されます.

初期解の修復機能を有効にした場合，分枝限定法の計算開始前に以下のような進捗が表示されます.

```
phase       total_slack         objective  time(s)  ite.  mem(MiB)
feas.           250.392                 0      5.5     0       277
feas.           250.392                 0      6.1     1       278
 opt.           250.392      -6.01642e+09     24.3     2       302
feas.                 2       7.59344e+08     25.2     3       301
feas.                 1       7.77003e+08     25.9     4       301
feas.                 1       7.77003e+08     26.5     5       293
```

これは初期解の修復における各反復（ite.）で，制約式の総違反量（total_slack）と目的関数（objective）がどのように遷移しているかを表示しています．行頭の feas. と opt. はその反復でどのような計算が

おこなわれているかを表しています．feas. は総違反量の最小化をおこない，opt. は目的関数の最小化（あるいは最大化）をおこなっています．実行可能解が見つかるか，ある程度の反復がおこなわれると初期解の修復を終了し，分枝限定法に移行します．

2.7 重み付き局所探索法（wls）における出力

重み付き局所探索法（wls）を用いた場合には，[Progress] で始まるセクションに探索の情報が以下の順に表示されます．

開始時

```
<problem statistics>
# 0-1 Vars    / Total     = 400 / 625
# 0-1 Constrs / Total     = 150 / 540
#   - Set Multi-Cover     = 50
#   - Set Multi-Packing   = 100
#   - Set Multi-Partition = 0
# Int Range Max           = 12

<iteration begin>
# Initial Sol         = given
# Obj                 = 240.00
# (Hard/Soft) Penalty = 54.00 / 430.00
```

<problem statistics>では，読み込まれた最適化問題について WLS の性能に大きく影響する情報が表示されます．具体的には，変数や制約式の中に 0-1 変数や 0-1 制約式が多く含まれるほど WLS は高い性能を発揮します．ここで 0-1 制約式とはすべての係数が 0 か 1 である線形な制約式です．0-1 制約式は，集合被覆型（例：$x_1 + x_2 \geq 2$），集合充填型（例：$x_1 + x_2 \leq 1$），集合分割型（例：$x_1 + x_2 = 2$）に分類されます．

<iteration begin>以降では初期解の情報が表示されます．上の例の場合，ユーザが指定した初期解は目的関数値が 240 であり，ハード制約違反量は 54，ソフト制約違反量は 430 であると読み取れます．

それぞれの項目の意味は次のとおりです．

表示	意味
# 0-1 Vars / Total	0-1 変数の個数 / 変数の個数
# 0-1 Constrs / Total	0-1 制約式の本数 / 制約式の本数
# - Set Multi-Cover	集合被覆型制約式の本数
# - Set Multi-Packing	集合充填型制約式の本数
# - Set Multi-Partition	集合分割型制約式の本数
# Int Range Max	整数変数の「上限 - 下限」の最大値

表示	意味
# Initial Sol	初期解の設定方法 "given": ユーザ指定, "random": ランダム, "zero": ゼロ初期化
# Obj	初期解の目的関数値
# (Hard/Soft) Penalty	初期解のハード制約違反量 / ソフト制約違反量

途中経過

```
-------------------------------------------------------------------------------
      Resources          |          Current Sol         |          Best Sol
 #Itrs  Time(s)  Mem(MiB) |    Obj      Hard       Soft  |    Obj      Hard       Soft
-------------------------------------------------------------------------------
     1     0.67    250.23 | 688.23     11.00     123.00  | 688.23     11.00     123.00
     5     0.75    403.34 | 347.43      3.00      23.00  |    400      0.00      43.00
...
-------------------------------------------------------------------------------
```

初期解の情報に続き，探索の途中経過が表形式で逐次的に表示されます．現在の実行時間や，現在どういった解を探索しているのか，現在までに見つけた解の中で最も良い解は何なのか，といった情報が読み取れます．

上の表は，反復が一定回数行われたり最良解が更新されたりする度に行が追加されていきます．解が更新されないと表示の更新も鈍くなりますが，計算は行われています．局所探索法による探索の一般的な性質として，計算の後半には解の更新は鈍くなります．

それぞれの項目の意味は次のとおりです．

表示	意味
#Itrs	反復回数
Time(s)	実行時間（秒）
Mem(MiB)	メモリ使用量（メビバイト）
Current Sol	現在探索中の解
Best Sol	最良解
Obj	目的関数値
Hard	ハード制約違反量
Soft	ソフト制約違反量

終了時

```
========================================
#  Obj                  = 230.00
#  (Hard/Soft) Penalty = 0.00 / 4.00
#  Elapsed Time (s)     = 43.54 / 100.00
#  Iterations           = 81708 / 191770
========================================
<iteration end>
```

アルゴリズムが終了すると実行時間や最良解などの情報が表示されます．上の例の場合，目的関数値 230，ハード制約違反量 0，ソフト制約違反量 4 である解が最良解として得られており，アルゴリズムの実行時間は 100 秒間であったことが読み取れます．

それぞれの項目の意味は次のとおりです．

表示	意味
# Obj	最良解の目的関数値
# (Hard/Soft) Penalty	最良解のハード制約違反量 / ソフト制約違反量
# Elapsed Time (s)	最良解発見時の実行時間 / 総実行時間
# Iterations	最良解発見時の反復回数 / 総反復回数

2.8 資源制約付きスケジューリング問題ソルバ（rcpsp）における出力

資源制約付きスケジューリング問題ソルバ（rcpsp）を用いた場合，[Progress] で始まるセクションでの実行経過の出力は以下のようになります，目的関数の設定によって 2 種類の出力があります．

2.8.1 完了時刻最小化

```
<preprocess begin>.............................<preprocess end>
<iteration begin>
(soft) penalty= 18, time= 0.00(s),iteration= 0
(soft) penalty= 17, time= 0.00(s),iteration= 2
(soft) penalty= 16, time= 0.00(s),iteration= 3
(soft) penalty= 15, time= 0.00(s),iteration= 4
(soft) penalty= 14, time= 0.03(s),iteration= 6
(soft) penalty= 13, time= 0.05(s),iteration= 8
 ...
<iteration end>
```

項目の意味は次の通りです．

表示	意味
(soft)	発見された解に関する
penalty=値 1	値 1：ソフト制約違反量
time=値 2，iteration=値 3	値 2：経過時間，値 3：反復回数

目的関数は内部では，ソフト制約として扱われていますので，目的関数の値もソフト制約違反量にふくまれています.

2.8.2 納期遅れ最小化

```
<preprocess begin>.............................<preprocess end>
<interation begin>
(objective value) value= 18, time= 0.00(s),iteration= 0
(objective value) value= 10, time= 0.03(s),iteration= 1
(objective value) value= 4, time= 0.05(s),iteration= 8
 ...
<iteration end>
```

項目の意味は次の通りです.

表示	意味
(objective value)	発見された解に関する
value=値 1	値 1：目的関数値（総納期遅れ）
time=値 2，iteration=値 3	値 2：経過時間，値 3：反復回数

上記 2 つの表示は，制約充足ソルバの時と同様，最良解が更新される度に現れます．その為，解が更新されないと表示が停止する点においても制約充足ソルバの時と同様です.

2.9 実行不可能性要因検出機能（iisDetect）の出力

デフォルトの指定では，実行不可能性を検出する iisDetect と呼ばれる仕組みが自動的に起動され，実行不可能性の原因の探索を行い，その結果を解ファイルの出力に反映させます（制約充足問題ソルバ/資源制約付きスケジューリング問題ソルバ使用時以外）．ここでは，iisDetect 機能が起動した場合の出力結果を説明します.

以下のモデル記述（モデルファイル名 lp.smp とします）に書かれた線形計画問題は，実行不可能（制約を満たす解なし）です.

```
Variable x, y, z;
Objective f(type = minimize);
```

```
f = x + y + z;
x >= 2 * y;       // IIS
1 + 2 * z >= x;   // IIS
y >= 2 + z;       // IIS
x >= z;
y >= 0;
z >= 0;
```

よく見るとモデルで“IIS”のマークが付いた制約式群のどの一つを除去しても実行不可能性は解消しますが，すべてを満たす x, y, z は存在しません．また，マークされていない最後の制約式は実行不可能性とは無関係で，除去する，しないにかかわらず，問題は実行不可能であることもわかります．

iisDetect 機能はこのように，実行不可能性の原因となっている行の組（Irreducible Infeasible Set：IIS と呼ばれます）を特定して出力します．一般に実行不可能な問題について IIS は複数存在しますが，このアルゴリズムは可能な限り小さなもの（含まれている行が少ない）ものを求めるようなヒューリスティクスが導入されています．

この問題を解かせたとき，[Result] で始まるセクションに以下のような出力がなされます．

```
ERROR_TYPE                          (NUOPT 11) infeasible.
DETECTED_IIS_SIZE                                        3
(#IIS_RELATED_VAR)                                       3
INFEASIBILITY_OF_IIS                                   1.5
```

それぞれの出力の意味は，以下のようになります．

タイトル	解説	備考
DETECTED_IIS_SIZE	検出された IIS に含まれる行の数	成功時のみ
(#IIS_RELATED_VAR)	IIS に含まれている行に含まれる変数の数	成功時のみ
INFEASIBILITY_OF_IIS	IIS 全体での実行不可能性	成功時のみ
NO_IIS_FOUND_BY	IIS 検出失敗の原因	失敗時のみ
(#NONLINEAR_CONSTR.)	IIS 検出失敗の原因の可能性がある非線形制約の数	失敗時のみ

モデルに非線形の式が含まれていた場合，IIS の正確な検出はできません．その場合には，ヘッダー部には IIS の検出が非線形性のために失敗したというメッセージが現れ，非線形な制約がいくつあるかを示します．例えば，次のモデルに対する出力は以下のようになります．

```
Variable x, y, z;
Objective f(type = minimize);
f = x + y + z;


x * x >= 2 * y * y;
1 + z >= x;
```

```
y >= 2 + z;
x + y + z >= 0;
```

出力

```
ERROR_TYPE                              (NUOPT 11) infeasible.
NO_IIS_FOUND_BY                                NON_LINEARLITY
(#NONLINEAR_CONSTR.)                                        1
```

2.10 最適化計算結果標準出力内容

以下は，標準出力に出力される内容の一覧です.

タイトル	解説	備考
STATUS	最適化計算終了時の状態	NORMAL/OPTIMAL/NON_OPTIMAL/ERROR のいずれかを取る
(#IIS_RELATED_VAR)	IIS に含まれている行に含まれる変数の数	IIS 特定成功時のみ
(#INTEGER/DISCRETE)	整数変数の総数	
(#NONLINEAR_CONSTR.)	IIS 検出失敗の原因の可能性がある非線形制約の数	IIS 特定失敗時のみ
BOUND_INFEASIBILITY	変数の上下限制約違反量の最大値	値が小さい場合は出力が省略される
CONSTRAINT_INFEASIBILITY	制約式違反量の最大値	値が小さい場合は出力が省略される
DETECTED_IIS_SIZE	検出された IIS に含まれる行の数	IIS 特定成功時のみ
ELAPSED_TIME(sec.)	計算時間	SIMPLE の展開時間を含みません
ERROR_TYPE	エラー番号とエラーメッセージ	STATUS が NON_OPTIMAL や ERROR のときに出力される
FACTORIZATION_COUNT	行列の分解回数	内点法のみ
FUNC_EVAL_COUNT	関数の評価回数	内点法のみ
GAP	上界値と下界値の差	分枝限定法のみ
INFEASIBILITY_OF_IIS	IIS 全体での実行不可能性	IIS 特定成功時のみ
ITERATION_COUNT	アルゴリズム内の反復回数	内点法/wcsp/rcpsp/wls のみ
METHOD	適用した最適化手法	

タイトル	解説	備考
NO_IIS_FOUND_BY	IIS 検出失敗の原因	IIS 特定失敗時のみ
NUMBER_OF_ACTIVITIES	アクティビティの総数	rcpsp のみ
NUMBER_OF_FUNCTIONS	関数（目的関数を含む）の総数	
NUMBER_OF_GENERAL_CONSTRAINT	一般の考慮制約の総数	rcpsp のみ
NUMBER_OF_IMPRECEDENCE	直前先行制約の総数	rcpsp のみ
NUMBER_OF_MODES	モードの総数	rcpsp のみ
NUMBER_OF_PRECEDENCE	先行制約の総数	rcpsp のみ
NUMBER_OF_RESOURCES	資源の総数	rcpsp のみ
NUMBER_OF_VARIABLES	変数の総数	
PARTIAL_PROBLEM_COUNT	部分問題数	分枝限定法のみ
PENALTY	重みつき制約違反量	wcsp/rcpsp 適用時のみ
PROBLEM_NAME	問題名	SIMPLE 版:モデル名 MPS 版:TITLE の内容
PROBLEM_TYPE	MINIMIZATION（最小化） MAXIMIZATION（最大化）	
RANDOM_SEED	乱数の種	wcsp のみ
RESIDUAL	最適性条件の充足度合	分枝限定法/wcsp/rcpsp/wls 以外
SIMPLEX_PIVOT_COUNT	単体法の反復回数	単体法のみ
SOLUTION_FILE	解ファイルのパス名	出力される解ファイルの名前は，環境や設定により異なる
TERMINATE_REASON	計算終了理由	wcsp/rcpsp 適用時のみ
THREADS	並列化実行において使用したスレッド数	分枝限定法/制約充足問題ソルバ wcsp など並列化実行可能なアルゴリズムにおいて出力される
VALUE_OF_OBJECTIVE	目的関数値	実行不可能（infeasible）の場合，出力される目的関数の値は不定となる

STATUS は「最適化計算終了時の状態」です.

- OPTIMAL は局所最適解が得られたことを意味します.
- NON_OPTIMAL は何かしらの理由で局所最適解が得られなかったことを意味します．詳細は ERROR_TYPE を参照します.
- NORMAL は wcsp,rcpsp,wls が正常終了したことを意味します.

• ERROR は wcsp,rcpsp,wls が異常終了したことを意味します.

2.11 標準出力の抑制

以下の設定を行う事で，標準出力に出力される情報を出力しないように設定できます．設定方法は 2 つあります.

一つは，求解オプションファイル nuopt.prm に記述する方法です．nuopt.prm 内に以下のように記述します.

```
output:mode = silent
```

もう一つは，SIMPLE モデルファイル内に記述する方法です．モデルファイル内に以下のように記述します.

```
options.outputMode = "silent";
```

第3章 解ファイル

Nuorium Optimizer は最適化計算の詳細情報を解ファイルというファイルに出力します．コマンドラインで Nuorium Optimizer を起動した場合，モデル名.sol という解ファイルが作成されます．

具体例として，次の例題に対する解ファイルを見ていくことにします．

$$
\begin{aligned}
\text{最小化} \quad & -3x_1 - 2x_2 - 4x_3 \\
\text{条件} \quad & x_1 + x_2 + 2x_3 \leq 4 \\
& 2x_1 + 2x_3 \leq 5 \\
& 2x_1 + x_2 + 3x_3 \leq 7 \\
& x_1 \geq 0, x_2 \geq 0, x_3 \geq 0
\end{aligned}
$$

3.1 冒頭部分

解ファイルの冒頭部分には

```
%%
%%
%%
%% RESULT OF NUOPT #1
%%
%%
%%
%%
```

と表示されます．これは 1 回目の求解結果であることを示しています．`solve()` を複数回記述した場合，1 回目の求解結果の後に 2 回目以降の求解結果が順に表示されます．

各求解結果では，標準出力の [Problem and Algorithm] および [Result] で始まるセクションに出力される内容と，同等の情報が出力されます．

例えば，上記の例題を線形計画専用内点法 higher で解いた際の，解ファイルの冒頭部分は，次のようになります．

```
PROBLEM_NAME                                        sample
NUMBER_OF_VARIABLES                                      3
NUMBER_OF_FUNCTIONS                                      4
```

```
PROBLEM_TYPE                                MINIMIZATION
METHOD                                      HIGHER_ORDER
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                -10.5
ITERATION_COUNT                                       7
FUNC_EVAL_COUNT                                      10
FACTORIZATION_COUNT                                   8
RESIDUAL                             3.903545017e-009
ELAPSED_TIME(sec.)                                 0.02
```

上記例題を，単体法 simplex で解いた際，解ファイルの冒頭部分は次のようになります．ITERATION_COUNT，FUNC_EVAL_COUNT，FACTORIZATION_COUNT という行が表示されないかわりに SIMPLEX_PIVOT_COUNT という行に単体法の反復回数が表示されます．

```
PROBLEM_NAME                                   sample
NUMBER_OF_VARIABLES                                 3
NUMBER_OF_FUNCTIONS                                 4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                          SIMPLEX
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                -10.5
SIMPLEX_PIVOT_COUNT                                 3
RESIDUAL                             2.340507908e-016
ELAPSED_TIME(sec.)                                 0.02
```

上記例題において変数をすべて整数変数に直した問題を，分枝限定法＋単体法 simplex を用いて解いた場合，分枝限定法の部分問題の数 PARTIAL_PROBLEM_COUNT が出力されます．分枝限定法を行った場合に RESIDUAL が表示されないのは，整数解においては最適性条件（連続変数を仮定）が充足しているとはいえないので，RESIDUAL の値が目的関数値の正しさを示す尺度とはならないためです．

```
PROBLEM_NAME                                   sample
NUMBER_OF_VARIABLES                                 3
(#INTEGER/DISCRETE)                                 3
NUMBER_OF_FUNCTIONS                                 4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                          SIMPLEX
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                                  -10
SIMPLEX_PIVOT_COUNT                                 4
```

```
PARTIAL_PROBLEM_COUNT                                    1
ELAPSED_TIME(sec.)                                    0.01
```

3.2 解ファイルの変数値表示部

```
%%
%% VARIABLES
%%
```

で始まる部分には最適化アルゴリズム停止時における変数の値，及びその上下限が記述されています．

```
%%
%% VARIABLES
%%
     NAME     VALUE     STATUS      SLACK      [      BOUND TYPE   ]
V# 1 x1             2.5  FREE  2.50000000e+000 [ 0 <=    x[1]      ]
V# 2 x2             1.5  FREE  1.50000000e+000 [ 0 <=    x[2]      ]
V# 3 x3    3.51218e-010  LOWER 3.51217955e-010 [ 0 <=    x[3]      ]
```

VALUE は「変数の値」，STATUS は「変数値が上下限のいずれに付着しているかの状態」，SLACK が「上下限値からの余裕量」，BOUND TYPE は「変数の上下限」を示しています．

例えば，`V# 1` から始まる行は x1 という名前の変数の値（2.5）と，それが下限にも上限にも等しくなっていないこと（"FREE"），続いて x1 に課された制約の種類（[] 内）が示されています．

変数の状態を示す STATUS の意味を以下に示します．

状態を示す文字列	状態
LOWER	下限制約に付着（下限制約が active）
UPPER	上限制約に付着（上限制約が active）
FREE	上下限，いずれにも付着していない
REMVD	前処理によって削除された
INFS	上下限を違反している

INFS は得られた解が，変数の上下限あるいは制約式を違反している場合に出力されます．問題は正常に解けていません．

また，資源制約付きスケジューリング問題ソルバ（rcpsp）を用いた場合には，変数の部分は，以下に相当します．

```
%%
%% ACTIVITIES
%%
```

例えば,

```
%%
%% ACTIVITIES
%%
        NAME       MODE    STATUS SLACK [       BOUND TYPE       ]
V#  1 sourceActivity     "DummyMode"   ACT        [  0 <=  sourceActivity  <= 0  ]
V#  2 act[1]    "A_does"     ACT        [  6 <=  act[1]  <= 12 ]
V#  3 act[2]    "C_does"     ACT        [  0 <=  act[2]  <= 11 ]
V#  4 act[3]    "B_does"     ACT        [  0 <=  act[3]  <= 8  ]
```

のような出力があった場合には，MODE がアクティビティが処理されるモード，BOUND TYPE が（アクティビティの開始時刻）<=（アクティビティ）<=（アクティビティの終了時刻）を表します.

3.3 解ファイルの関数値表示部

```
%%
%% FUNCTIONS
%%
```

で始まる部分には最適化アルゴリズム停止時における関数[1]の値が記述されています.

```
%%
%% FUNCTIONS
%%
          NAME     VALUE STATUS      SLACK      [     BOUND TYPE       ]
F# 1 obj           -10.5  FREE                  [    OBJECTIVE (MINIMIZE)       ]
F# 2 g1                4  UPPER 1.75611081e-010 [             g1 <= 4 ]
F# 3 g2                5  UPPER 7.02441660e-010 [             g2 <= 5 ]
F# 4 g3              6.5  FREE  5.00000001e-001 [             g3 <= 7 ]
```

VALUE が「関数の値」，STATUS が「関数値が上下限のいずれに付着しているかの状態」，SLACK が「上下限値からの余裕量」，BOUND TYPE が「関数の上下限」を示しています.

F# 1 から始まる行には F という名前の関数（目的関数）の値（-10.5）と，それが最小化された（"MINIMIZE"）目的関数である旨が示されています．F# 2 から始まる行は g1 という名前の制約式の値が 4 であり，それが上限に等しくなっていること（"UPPER"），続いて g1 に課された制約の種類

[1] 目的関数と制約式を総称してこう呼んでいます.

（[] 内）が示されています.

関数（制約式）の状態を示す STATUS の意味を以下に示します.

状態を示す文字列	状態
LOWER	下限制約に付着（下限制約が active）
UPPER	上限制約に付着（上限制約が active）
FREE	上下限，いずれにも付着していない
INFS	上下限を違反している
TGIN	ソフト制約が制約を違反していない
TGOUT	ソフト制約が制約を違反し，ペナルティが発生している.

INFS は数値的な性質の悪い問題や最適化が途中で失敗した際に出力されます.

また，半正定値制約を課した対称行列の各要素の値に関しては以下のような出力がされます.

```
      NAME   VALUE  STATUS SLACK [   BOUND TYPE   ]
F# 1 X[1,1] 25.6995              [   MATRIXELEM   ]
F# 2 X[2,1]       1              [   MATRIXELEM   ]
F# 3 X[2,2] 25.6995              [   MATRIXELEM   ]
```

3.4 解ファイルの上下限，制約と対応する双対変数表示部

解ファイルの

```
%%
%% BOUNDS
%%
```

から続く部分には変数の上下限と双対変数，また

```
%%
%% CONSTRAINTS
%%
```

から続く部分には制約式の上下限と双対変数がそれぞれ表示されます.

```
%%
%% BOUNDS
%%
     [            BOUND TYPE            ]     DUAL VALUE
B# 1 [     0      <=  x1                ]   4.891837406e-010
B# 2 [     0      <=  x2                ]   8.151927281e-010
B# 3 [     0      <=  x3                ]         1.000000001
```

```
%%
%% CONSTRAINTS
%%
     [    CONSTRAINT/OBJECTIVE TYPE     ]     DUAL/WGT
C# 1 [       OBJECTIVE (MINIMIZE)       ]                   0
C# 2 [                 g1  <=     4      ]       -1.999999998
C# 3 [                 g2  <=     5      ]      -0.4999999986
C# 4 [                 g3  <=     7      ]  -2.443858094e-009
```

BOUND TYPE 及び CONSTRAINT/OBJECTIVE TYPE が「上下限の種類」，DUAL VALUE 及び DUAL/WGT が「双対変数の値」を示しています．

B# 1 の行では，x1 に対する制約が下限制約 0 <= x1 であること，またその双対変数がほぼ 0 である旨が示されています．

双対変数は制約式や変数の上下限を単位あたり変化させたときの目的関数の変動を示しています．双対変数は別名，シャドウプライス，または reduced cost とも呼ばれ，その正負や大きさから上下限のいずれが active かを次のように判断することができます．

解ファイルの双対変数値	状態
正	下限制約に付着（下限制約が active）
負	上限制約に付着（上限制約が active）
零/零に近い値	上下限，いずれにも付着していない

目的関数の双対変数値に対応する部分には零が出力されます．

3.5 解ファイルの実行不可能性要因出力部

解ファイルには，実行不可能性検出機能によって判定された「実行不可能な制約式の組」が出力されます．次のモデルに対しては，以下の出力が解ファイルになされます．

```
Variable x, y, z;
Objective f(type = minimize);
f = x + y + z;
x >= 2 * y;       // IIS
1 + 2 * z >= x;   // IIS
y >= 2 + z;       // IIS
x >= z;
y >= 0;
z >= 0;
```

解ファイル出力

```
%%
%% IIS
%%
--------------------------------------------------------------------------------
#2      sample.smp:4      :   -1*x+2*y
                                          <=          0   (-1.11e-016)
--------------------------------------------------------------------------------
#3      sample.smp:5      :   -1+1*x-2*z
                                          <=          0   (         0)
--------------------------------------------------------------------------------
#4      sample.smp:6 INFS :   2-1*y+1*z
                                          <=          0   (       1.5)
--------------------------------------------------------------------------------
```

上記のように，IIS に含まれる行の制約式の名前が出力されます．内部表現に従って移項されていますので，x, y, z の順番はモデル記述とは異なります．() 内は現在の変数値の設定（以降に出力されます）における制約式の値です．INFS とマークがある行は現在の変数値の設定において，上下限を破っている行です．これを解消しようとすると，IIS の定義から，ここに現れている行のほかのいずれかに波及します．IIS 検出に成功した場合の変数の設定は IIS に含まれる行の違反の合計が最も小さくなるように行われます．その際の IIS に含まれる行の違反の合計が，標準出力に現れる

INFEASIBILITY_OF_IIS

という値です．

実行不可能性が検出された場合は，実行不可能性の要因と関連する変数，関数のみが解ファイルに出力されます．

3.6 解ファイルのハード制約，セミハード制約およびソフト制約表示部

アルゴリズムとして制約充足問題ソルバ wcsp を用いている場合，解ファイルには，最終的に満たすことのできていないハード制約，ソフト制約が出力されます．次が出力サンプルです．

解ファイル出力

```
%%
%% WCSP_PENALTY
%%
           NAME                 TYPE    VALUE   BOUND    AMOUNT WEIGHT PENALTY
F#   246   model.smp:83[6]       HARD    0    >=    1        1
F#   432   model.smp:91[13]      S.HARD  0    >=    1        1
F#   946   model.smp:119[17,E]   S.HARD  7    <=    6        1
F#  7080   model.smp:152[1](u)   SOFT    3    <=    2        1      100    100
```

```
F#  7586   model.smp:156[7,3]   SOFT   5   <=   2      3      10     30
F#  7675   model.smp:156[21,6]  SOFT   3   <=   2      1      10     10
F#  7756   model.smp:156[2,10]  SOFT   4   <=   2      2      10     20
F#  7780   model.smp:156[1,11]  SOFT   3   <=   2      1      10     10
F#  8293   model.smp:162[19,25] SOFT   0   ==   5      5       1      5
                                . . .
```

上記のように，ハード制約，ソフト制約で満たされていないもののみがハード制約，セミハード制約，ソフト制約の順に表示されます．各行は，制約式の名前，タイプ（ハード：HARD，セミハード：S.HARD，ソフト：SOFT），現在の値と制約，違反量，ウエイト（ソフト制約のみ），ペナルティ量（ソフト制約のみ）を示しています．

この出力例ではハード制約である 246 番目の制約（model.smp:83[6]）が 0 ですが，本来 1 以上とならねばならないので，ハード制約違反の違反量が 1 であることなどがわかります．

上記で上から 4 つめの制約（model.smp:152[1]）の名前の後に (u) とあるのは，この制約が上下限制約であり，違反しているのは制約の上限であることを示しています（2 という上限に対して 3 という値を取っています）．上下限制約の下限に違反しているのであれば制約の名前の後に (l) と表示されます．ソフト制約については，違反量に設定されたウエイトを掛けた値である「ペナルティ」があわせて表示されます．制約充足問題ソルバ wcsp は，ハード制約，セミハード制約の違反量，ソフトペナルティのペナルティの合計値を最小化します．

第4章 Nuorium Optimizerの適用範囲とアルゴリズム

本章では，Nuorium Optimizer が扱う事のできる数理最適化問題の範囲及び備えているアルゴリズムを列挙し，それらの対応関係を与えます．また，アルゴリズムの設定方法も示します．

4.1 数理最適化問題一覧

Nuorium Optimizer では，以下のような数理最適化問題を取り扱うことができます．

- LP (Linear Programming：線形計画問題)
  目的関数と制約式がすべて線形である問題で，整数変数を含まないものです．
- MILP (Mixed Integer Linear Programming：混合整数計画問題)
  目的関数と制約式がすべて線形で，整数変数を含むものです．MIP（Mixed Integer Programming）と呼ばれることも多いです．
- MIQP (Mixed Integer Quadratic Programming：混合整数二次計画問題)
  制約式がすべて線形，目的関数が二次関数で，整数変数を含むものです．
- MINLP (Mixed Integer Nonlinear Programming：混合整数非線形計画問題)
  制約式および目的関数が非線形で整数変数を含むものです．
- CQP (Convex Quadratic Programming：凸二次計画問題)
  目的関数が凸な二次関数，制約式がすべて線形であるもの（ただし，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できるもの）です．
- CP (Convex Programming：凸計画問題)
  目的関数，制約式に非線形なものが含まれていますが，実行可能領域が凸で，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できる問題です．ここでは整数変数は含まないものを言います．
  半正定値計画問題も凸計画問題の一部ですが，ここには含めません．
- NLP (Nonlinear Programming：非線形計画問題)
  上記以外で，整数変数を含まない一般の非線形計画問題です．
- SDP (SemiDefinite Programing：半正定値計画問題)
  行列の半正定値制約を含む線形計画問題です．
- NLSDP (NonLinear SemiDefinite Programing：非線形半正定値計画問題)
  行列の半正定値制約を含み，なおかつ目的関数・制約式に非線形項が含まれる問題です．
- WCSP (Weighted Constraint Satisfaction Problem：重み付き制約充足問題)
  各々重みの付いた制約条件をなるべく満足するためには値をどのように割り当てると良いかを決定する問題です．制約充足問題ソルバ wcsp により高速に解を得ることができます．

- RCPSP (Resource Constrained Project Scheduling Problem：資源制約付きスケジューリング問題）
  一定の資源制約の下で，決められた作業の開始・終了時刻を決定する問題です．一般の整数計画問題（MILP）として記述することも可能ですが，特殊な記法を行うと資源制約付きスケジューリング問題ソルバ rcpsp により高速に実行可能解を得ることができます．

4.2 アルゴリズム一覧

Nuorium Optimizer は以下のようなアルゴリズムを備えています．見出しの解法名は，アルゴリズムを指定する際に用います．カッコ内の解法名は，標準出力に METHOD として出力されるものです．

- simplex：単体法（SIMPLEX）
  線形計画法の解法として古くから知られている方法です．大規模問題では内点法に速度的に劣りますが，可能基底解が求まり原理的に内点法/外点法よりも高精度です．
  整数変数を含む問題に対して指定すると，単体法を分枝限定法（Branch and bound method）という枠組のなかで繰り返し行って，最適性の保証のある整数解を求めます．大規模問題において基底解が必要な場合には，"cross:on"と指定して内点法からのクロスオーバーを用いるのが有利です．
- dual_simplex：双対単体法（DUAL_SIMPLEX）
  （主）単体法が主実行可能な基底解をたどりながら最適解にたどり着くのに対し，双対単体法は双対実行可能な基底解をたどりながら最適解にたどり着きます．
  大規模な線形計画問題に対して，（主）単体法と比較して有利であることがあります．
- hsimplex：単体法 (HSIMPLEX)
  スパース性をより活用した，線形計画問題および凸二次計画問題に対する単体法です．整数変数問題を含んだ問題に対しては整数性を緩和した問題を解くのでご注意ください．
- asqp：有効制約法（ACTIVE_SET_QP）
  単体法と同様，古典的な凸二次計画問題の厳密解法です．1 万変数以上の大規模問題では，一般に内点法（直線探索法（Line Search Method））に劣りますが，
  - 変数に比べて制約式の数が非常に少ない（1/10 以下）場合
  - 目的関数のヘッセ行列が密行列である場合

  には内点法よりも高速かつ高精度です．また，整数計画法に対応しているので，整数変数が含まれている凸二次計画問題を解くことができます．`"cross:on"`と指定することで内点法からのクロスオーバーを用いることができるので，大規模問題に対して高精度な解を求めることができます．
- higher：線形計画問題専用内点法（HIGHER_ORDER）
  線形計画法に特化した内点法で，大規模な線形計画問題の解法としては最も高速です．単体法と違い，可能基底解は求まりません．
- lipm：直線探索法（LINE_SEARCH_IPM）
  一般の凸計画問題に適用可能な内点法です．問題が凸であることがわかっている場合には信頼領域法よりも高速です．幅広い範囲の問題に対して有効です．
- bfgs：準ニュートン法（BFGS_LINE_SEARCH）
  準ニュートン法を用いた直線探索に基づく内点法です．ヘッセ行列の近似行列を密行列として保

持します．小規模（50〜500 変数以下）かつ非線形性の強い問題に対して tipm よりも有効な場合があります．

- tipm：信頼領域内点法（TRUST_REGION_IPM）
  大規模なものを含む一般の非線形計画問題に適用可能な内点法です．幅広い範囲の問題に対して有効です．
- lsqp：直線探索法に基づく逐次二次計画法（LINE_SEARCH_SQP）
  準ニュートン法によって二階微係数を求める逐次二次計画法です．小規模（50〜100 変数以下）な非線形計画問題に適しています．
  問題によっては直線探索内点法（lipm）よりも安定的により精度の良い解を導くことができます．
- tsqp：信頼領域法に基づく逐次二次計画法（TRUST_REGION_SQP）
  二階微係数をそのまま用いる逐次二次計画法です．大規模なものを含む一般の非線形計画問題に適用可能な方法です．一般に内点法よりも低速ですが，問題によっては内点法よりも安定的に，より精度の良い解を導くことができます．変数の数よりも制約式数が多い場合には内点法（tipm）よりも高速な場合があります．
  変数の数よりも制約式数が多い場合には内点法（tipm）よりも高速な場合があります．
- lsdp：線形半正定値計画問題に対する主双対内点法
  線形の半正定値計画問題に対する主双対内点法です．目的関数・制約式に出現する項は線形である必要があります．内部でメリット関数の計算を行いません．
- trsdp：信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
  目的関数・制約式に非線形項が出現する半正定値計画問題に対する主双対内点法です．メリット関数の降下を保証するために，信頼領域法を利用しています．
- wcsp：制約充足問題ソルバ（WCSP）
  京都大学「問題解決エンジン」グループの開発による制約充足問題に対するアルゴリズムです．必ずしも厳密解が求まるわけではありませんが，大規模な整数計画問題に対し，非常に高速に実行可能解（近似解）を求めることができます．
  整数変数のみを含み，かつすべての変数に上限と下限がある問題に対してのみ有効です．目的関数，制約式に重みを設定することができます．制約の重みには，ハード制約，セミハード制約，ソフト制約の三種類があります．
- wls：重み付き局所探索法 (WLS)
  PySIMPLE から呼び出して使える近似解法です．0-1 係数のみを含む制約式に対して特別な処理をおこなっています．そのため，集合被覆や集合分割といった特定の問題を得意としています．線形及び二次の制約式・目的関数を扱うことができます．変数は整数変数のみを扱うことができます．
- rcpsp：資源制約付きスケジューリング問題ソルバ（RCPSP）
  京都大学「問題解決エンジン」グループの開発による資源制約付きスケジューリング問題に対するアルゴリズムです．資源制約の下，決められた作業の開始・終了時刻を決定する問題の実行可能解を高速に求めることができます．rcpsp の記述にあたっては問題を SIMPLE の特殊なクラスを用いて記述する必要があります．完了時刻の最小化問題と，納期遅れ最小化問題を扱うことができます．前者を扱う際にはソフト制約，後者を扱う際にはハード制約のみが使用できます．

Nuorium Optimizer のアルゴリズムは，SIMPLE で記述される目的関数，制約で四則演算および数学

関数を用いて記述されたものをサポートします.

4.3 数理最適化問題とアルゴリズムの対応

以下は，Nuorium Optimizer で取り扱い可能な数理最適化問題と，Nuorium Optimizer が備えているアルゴリズムの対応一覧です.

表の内容は

◎　最も適している
○　適している
R　整数性を緩和した問題を解く

を示します.

	LP	MILP	MIQP	WCSP※5	RCPSP	MINLP	CQP	CP	NLP	SDP	NLSDP
simplex	◎	◎									
dual_simplex	◎										
hsimplex	◎	R					◎				
asqp	○	○	◎				◎				
higher	◎	R									
lipm	○	R	R			R	◎	◎	○		
bfgs	○	R	R			R	○	○	◎		
tipm	○	R	R			R	○	○	◎		
lsqp	○	R	R			R	○	◎	○		
tsqp	○	R	R			R	○	○	◎		
lsdp	○	R※4	R※4							◎	○
trsdp	○	R※4	R※4							○	◎
wcsp		◎※1	◎※1	◎		○※1					
wls		◎※2	◎※2	◎※3							
rcpsp					◎						

- ※ 1 について，0-1 整数変数と離散変数（DiscreteVariable）のみを含む問題のみ扱うことができます.
- ※ 2 について，整数変数を含む問題のみ扱うことができます．PySIMPLE から呼び出すことができます.
- ※ 3 について，wls が扱えるのは制約式・目的関数がそれぞれ二次までの問題です．また，ハード制約とソフト制約のみ扱うことができます.
- ※ 4 について，SymmetricMatrix を用いて定式化した場合に扱うことができます.
- ※ 5 について，本表では WCSP は「ソフト制約・セミハード制約・ハード制約が定義された連続変数を含まない問題」を指します.

4.4 アルゴリズムの設定方法

アルゴリズムを設定する方法には,

- モデルファイル内で指定する方法
- 求解オプションファイル nuopt.prm 内で指定する方法

の二通りがあります.

詳細については 5.2.1 アルゴリズムの選択を参照してください.

4.5 アルゴリズムの自動選択

アルゴリズムの指定を明示的に行わない場合には，入力された問題の内容から自動的にアルゴリズムを選択します.（**アルゴリズムの自動選択**）

適用アルゴリズム	判定基準
simplex	関数がすべて線形で整数変数を含む
asqp	目的関数が非線形で整数変数を含む
higher	関数がすべて線形で整数変数を含まない
wcsp	DiscreteVariable, selection を含む
rcpsp	Activity, ResourceRequire, ResourceCapacity を含む
lsdp	関数が全て線形で半正定値制約を含む
trsdp	上記以外の半正定値制約を含む
tipm	上記以外

次のような場合，個別に設定するとよりよい結果が得られる可能性があります.

4.5.1 整数変数が含まれている非線形計画問題

混合整数二次計画問題（MIQP）であれば asqp が利用可能です．混合整数非線形計画問題（MINLP）の場合，アルゴリズムを明示的に指定していない場合はエラーを返します．そのような場合は以下をお試しください.

- 非線形項を線形化して混合整数線形計画問題（MIP）として解く.
- 変数がすべて 0-1 整数変数であれば wcsp で解く.
- 整数性を無視した問題を tipm や tsqp で解く.

4.5.2 整数変数が含まれない非線形計画問題

整数変数が含まれない非線形計画問題の場合，デフォルトでは内点法による信頼領域法（tipm）となりますが，問題が二次計画問題（目的関数のみ二次関数）の場合，特に変数に比べて一般の制約式の数が少ない問題には有効制約法 asqp が有利な場合もあります.

逐次二次計画法 tsqp は若干時間を所要するケースもございますが，最も精度良く非線形最適化を行う方法です．

bfgs は小規模（50〜500 変数以下）かつ非線形性の強い問題に対して tipm よりも有効な場合があります．

4.5.3　凸計画問題

凸計画問題に対しては直線探索法を用いた方が一般に高速ですので，非線形ながら問題が凸であるとわかっている場合には（SIMPLE は凸であるかを自動判定できません）としていずれかの直線探索法を指定してください．通常お勧めできるのが lipm です．

4.6　実行不可能性要因検出機能 iisDetect

実行不可能性要因の検出を行うアルゴリズム iisDetect は，実行不可能性の原因となっている制約式の組の内，できるだけ式の少ないものを特定します．与えられた問題が実行不可能と判定されたら自動的に起動します．

大規模問題で実行不可能と判定された場合，iisDetect で多大な計算時間を要する場合があります．その時には求解オプションでオフにすることが可能です．

4.7　クロスオーバー

線形計画問題（LP）あるいは二次計画問題（QP）に対しては，内点法（higher/lipm/bfgs/tipm）によって得られた解の情報をもとにして単体法を起動する，クロスオーバーと呼ばれる手法を用いることができます．大規模問題に関して精度の良い解を得るにはこの方法が最も有効です．

混合整数計画問題（MIP）に対しても，起動可能です．

クロスオーバーを行うためには内点法の手法を設定した後に，以下のように指定します．

モデルファイルに記述する方法

```
options.crossover = "on";
```

求解オプションファイル nuopt.prm に記述する方法

```
cross:on
```

第5章 求解オプション設定

求解オプションは，Nuorium Optimizer の求解動作をより細かく制御するためのものです．求解オプションを利用することにより，アルゴリズムの選択や，標準出力の制御，終了条件の調整などを行う事ができます．求解オプションを設定する方法には次の二通りの方法があります．

1. モデルファイル中に記述する
2. 求解オプションファイル nuopt.prm に記述する

求解オプションの中には，2 の方法でしか利用できない種類のものも存在します．

求解オプション指定が競合した場合，求解オプションファイル nuopt.prm から指定する方法の方が優先されます．

5.1 求解オプションファイル nuopt.prm

求解オプションファイルは nuopt.prm という名前でなければなりません．

求解オプションファイルの冒頭の行は begin，最後の行は end である必要があります．end の後は必ず改行しなければなりません．次の例は，特に何も指定がなされていない，最も簡単な求解オプションファイルです．

```
begin
end
```

begin と end の間に各種求解オプションの設定を行うことができます．次の例ではアルゴリズムとして単体法 simplex を指定しています．

```
begin
method:simplex
end
```

求解オプションは:ではなく，=で指定する場合もあります．次の例では，停止条件を 10^{-12} に設定しています．

```
begin
crit:eps=1.0e-12
end
```

求解オプションファイル中には半角スペースを適宜入れる事ができます．次の例は，上記の例と同じ意味です．

```
begin
crit : eps = 1.0e-12
end
```

求解オプションファイルは，複数の求解オプションを指定する事もできます．以下の例では，停止条件を 10^{-4} に設定し，アルゴリズムに線形計画専用内点法 higher を設定しています．

```
begin
crit:eps = 1.0e-4
method:higher
end
```

行頭の半角アスタリスク*は，その行がコメント文であることを意味します．例えば次の例では 3 行目の method:higher が読み込まれません．

```
begin
crit:eps = 1.0e-4
*method:higher
end
```

求解オプションファイルで設定する値には整数や小数のほか，

```
9.836d-5   1.347D-4   3.4e-3   4.562384E-2
```

のような浮動小数点表記が許されています．

求解オプションファイルが読み込まれた場合，標準出力には求解オプションファイルを読み込んだことを示すメッセージと内容が表示されます．求解オプションファイル中の空白，コメントは無視されます．以下は，求解オプションファイルが読み込まれた場合の出力例です．

求解オプションファイル

```
begin
maximize
method:tipm
scaling:on
crit:eps = 1.0e-8
end
```

標準出力

```
<reading solver option file: nuopt.prm>
nuopt.prm:1:     begin
nuopt.prm:2:     maximize
nuopt.prm:3:     method:tipm
nuopt.prm:4:     scaling:on
```

```
nuopt.prm:5:      crit:eps = 1.0e-8
nuopt.prm:6:      end


          ...
```

5.2 共通求解オプション

本節では，求解アルゴリズムに依存しない求解オプションについて解説します.

5.2.1 アルゴリズムの選択

求解オプションを用いてアルゴリズムを設定する事ができます.

アルゴリズム名	日本語名
simplex	単体法（+分枝限定法）
dual_simplex	双対単体法
hsimplex	単体法（PySIMPLE からのみ使用可能）
asqp	有効制約法
higher	線形計画専用内点法
lipm	直線探索内点法
bfgs	準ニュートン法
tipm	信頼領域内点法
lsqp	直線探索を利用した逐次二次計画法
tsqp	信頼領域を利用した逐次二次計画法
lsdp	線形半正定値計画問題に対する主双対内点法
trsdp	信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
wcsp	制約充足問題ソルバ
wls	重み付き局所探索法（PySIMPLE からのみ使用可能）
rcpsp	資源制約付きスケジューリング問題ソルバ

モデルファイル内で指定する方法

```
options.method = "アルゴリズム名";
```

求解オプションファイル nuopt.prm 内で指定する方法

```
method:アルゴリズム名
```

以下の例では，アルゴリズム simplex（単体法）をそれぞれモデルファイルから，あるいは求解オプションファイルから設定しています.

モデルファイル内で指定する方法

```
options.method = "simplex";
```

求解オプションファイル nuopt.prm 内で指定する方法

```
method:simplex
```

5.2.2　標準出力制御

デフォルト設定の Nuorium Optimizer は，標準出力に求解情報を表示し，解ファイル（モデル名.sol）を作成します．求解オプションを用いる事で，これらを抑制できます．

標準出力による求解情報の表示を抑制するには，次のように記述します．

モデルファイルに記述する方法

```
options.outputMode = "silent";
```

求解オプションファイル nuopt.prm に記述する方法

```
output:mode = silent
```

5.2.3　解ファイル出力制御

求解オプションを用いて解ファイルの出力を抑制するには，次のように記述します．

モデルファイルに記述する方法

```
options.outfilename = "_NULL_";
```

求解オプションファイル nuopt.prm に記述する方法

```
output:name = _NULL_
```

求解オプションを用いて，出力される解ファイルのファイル名を変更することもできます．

モデルファイルに記述する方法

```
options.outfilename = "ファイル名";
```

求解オプションファイル nuopt.prm に記述する方法

```
output:name = ファイル名
```

以上の指定により，ファイル名.sol という名前の解ファイルが作成されます．

モデル内で解ファイルを作成するタイミングを制御することもできます．このためにはまず以下の記述を行います．

モデルファイルに記述する方法

```
options.noDefaultSolout = 1;
```

この記述により，求解関数 solve を実行した段階では解ファイルの生成を行わなくなります．さらに，解ファイルを生成したい箇所に次の関数を記述することにより，タイミングを制御できます．

```
solout(); // この関数の実行時に解ファイルを生成する
```

5.2.4 Nuorium/Excel アドインへの出力制御

求解オプションを用いて Nuorium/Excel アドインへの出力を制御することができます．
以下のように記述をすると出力を全て抑制することができます．
モデルファイルに記述する方法

```
options.noDefaultSolout = 1;
```

関数 solout を用いると，Nuorium/Excel アドインへの出力を関数呼び出し時に行うことができます．

```
solout(); // この関数の実行時に Nuorium/Excel アドインへの出力を行う
```

Parameter/Expression は出力の制御が可能です．
Parameter の出力を抑制するには，以下のように記述します．
モデルファイルに記述する方法

```
options.outputParameter = 0;
```

Expression の出力を抑制するには，以下のように記述します．
モデルファイルに記述する方法

```
options.outputExpression = 0;
```

Parameter/Expression の出力に時間を要する場合などに有効です．

5.2.5 実行不可能性の検出

実行不可能性検出アルゴリズム iisDetect の設定解除には，以下の設定方法が用意されています．
モデルファイル内で指定する方法

```
options.iisDetect = "off";
```

求解オプションファイル nuopt.prm 内で指定する方法

```
param:iis = off
```

5.3 アルゴリズム固有の求解オプション

本節では特定のアルゴリズムを選んだ際にのみ有効な求解オプションと，その設定について解説し

ます.

5.3.1 線形計画問題専用内点法（higher）/信頼領域内点法（tipm）/直線探索法（lipm）/逐次二次計画法（lsqp/tsqp）/半正定値計画専用内点法（lsdp/trsdp）に有効な求解オプション

以下，線形計画問題専用内点法のみ有効な求解オプションと，その設定について解説します.

- スケーリング

  アルゴリズムを効率よく安定に動作させるため，目的関数，制約式，変数に定数値を乗じる処置がスケーリングです．この求解オプションはスケーリングを行うか否かと，スケーリングの種類を指定するものです．指定できる値は“off”，“minmax”，“cr”，“on”の 4 つです．“off”ではスケーリング処理を行いません．“minmax”では，係数行列の各行と列について，非零要素の絶対値の最大値と最小値との幾何平均が 1 になるようにスケーリングを施します．“cr”では係数行列全体について，非零要素の絶対値の対数の 2 乗和を最小化します．“on”は Nuorium Optimizer V15 以前との互換性のために用意されており，“minmax”と等価です.

  モデルファイルに記述する方法

  ```
  options.scaling = "cr";
  ```

  求解オプションファイル nuopt.prm に記述する方法

  ```
  scaling:cr
  ```

- 単体法へのクロスオーバー（線形計画専用内点法 higher のみ）

  この指定を行うと，内点法によって得られた解の情報をもとにして単体法を起動することができます．大規模問題に関して可能基底解を得るにはこの方法が最も有効です.

  モデルファイルに記述する方法

  ```
  options.crossover = "on";
  ```

  求解オプションファイル nuopt.prm に記述する方法

  ```
  cross:on
  ```

- 停止条件

  停止条件として用いる最適性条件の残差です．最適性条件の残差がこの値以下になったときに，計算が収束したとみなして反復計算を終了します．ただし，higher では最適性条件の残差と双対ギャップの内大きい方の値が設定した値以下になったときに反復計算を終了します.

  モデルファイルに記述する方法

  ```
  options.eps = 1.0e-8;
  ```

  求解オプションファイル nuopt.prm に記述する方法

  ```
  crit:eps = 1.0e-8
  ```

- 反復回数上限

  停止条件として用いる反復計算の回数の上限です．反復回数のデフォルト値は 150 回となっています．反復回数がこの回数を越えた場合には解が得られなかったとみなしてエラー（(NUOPT 10) IPM iteration limit exceeded.）を出力します．逐次二次計画法（lsqp/tsqp）を用いた場合にはエラー（(NUOPT 40) SQP iteration limit exceeded.）を出力します．

  モデルファイルに記述する方法

```
options.maxitn = 150;
```

  求解オプションファイル nuopt.prm に記述する方法

```
crit:maxitn = 150
```

- 内点法内に現れる連立一次方程式を反復法で解く設定（higher のみ）

  この方法（mtxfree）を指定すると，探索方向を求めるために解く連立一次方程式を反復法（クリロフ部分空間法）を用いて解きます．デフォルトの求解法よりもメモリの使用量を減らすことができます．mtxfree には下記の制限が伴います．

  - 単体法へのクロスオーバーと併用することができない
  - 上界も下界も設定されていない変数がある場合は使用できない
  - 実行不可能性要因検知機能（iisDetect）と併用することができない

  モデルファイルに記述する方法

```
options.mtxfree = "on";
```

  求解オプションファイル nuopt.prm に記述する方法

```
linear:mtxfree = on
```

5.3.2 単体法（simplex/dual_simplex/hsimplex）/有効制約法（asqp）に有効な求解オプション

- スケーリング

  アルゴリズムを効率よく安定に動作させるため，目的関数，制約式，変数に定数値を乗じる処置がスケーリングです．この求解オプションはスケーリングを行うか否かと，スケーリングの種類を指定するものです．指定できる値は“off”，“minmax”，“cr”，“on”の 4 つです．“off”ではスケーリング処理を行いません．“minmax”では，係数行列の各行と列について，非零要素の絶対値の最大値と最小値との幾何平均が 1 になるようにスケーリングを施します．“cr”では係数行列全体について，非零要素の絶対値の対数の 2 乗和を最小化します．“on”は Nuorium Optimizer V15 以前との互換性のために用意されており，“minmax”と等価です．

  hsimplex に対して，スケーリング“cr”は指定できません．

  モデルファイルに記述する方法

```
options.scaling = "cr";
```

求解オプションファイル nuopt.prm に記述する方法

```
scaling:cr
```

- 主変数の実行可能性判定閾値

主変数の上下限違反判定に関する閾値です．この値以下の変数値に関する上下限違反は許容されます．初期値は 1.0e-8 です．

モデルファイルに記述する方法

```
options.tolx = 1.0e-8;
```

求解オプションファイル nuopt.prm に記述する方法

```
simplex:tolx = 1.0e-8
```

この判定値はスケーリングの適用後の問題に対して適用されます．したがって，スケーリング適用前の問題においては，判定値に対して上下限違反を起こす可能性があります．

- 双対変数の双対実行可能性判定閾値

双対変数（シャドウプライス）の双対実行可能性の判定値です．この値以下の解の改善可能性を無視します．初期値は 1.0e-8 です．

モデルファイルに記述する方法

```
options.told = 1.0e-6;
```

求解オプションファイル nuopt.prm に記述する方法

```
simplex:told = 1.0e-6
```

主変数の実行可能性判定閾値と同様，スケーリング適用後の問題に対して適用されます．

- アルゴリズムの選択（hsimplex のみ）

hsimplex で使用するアルゴリズムを選択します．線形計画問題に対しては「自動設定」と「主単体法」「双対単体法」が選択できます．二次計画問題に対しては「自動設定」と「主単体法」が選択できます．現在は，二次計画問題に対して双対単体法を選択しても，主単体法が使用されます．

求解オプションファイル nuopt.prm に記述する方法

```
hsimplex:method = -1    *  -1, 0, 1 を指定できます
```

指定された値に応じ，次の表のようにアルゴリズムが選択されます．現在，デフォルト値の自動設定では双対単体法が選択されます．

hsimplex:method	**アルゴリズム**
-1	自動設定（デフォルト値）
0	主単体法
1	双対単体法

5.3.3 制約充足アルゴリズム（wcsp/rcpsp）に有効な求解オプション

タブー・サーチによる制約充足アルゴリズム（wcsp/rcpsp）は制約をできるだけ充足する解をいずれか一つ求めるという手法で，厳密な最適解を求めるものではありません．

- 反復回数上限

  停止条件の一つである，反復回数上限を設定します．初期設定値-1 は，無制限を意味します．

  モデルファイルに記述する方法

```
options.maxitn = -1;
```

  求解オプションファイル nuopt.prm に記述する方法

```
crit:maxitn = -1
```

- 計算時間上限

  停止条件の一つである，計算時間上限を設定します．初期設定値-1 は，無制限を意味します．

  モデルファイルに記述する方法

```
options.maxtim = -1;
```

  求解オプションファイル nuopt.prm に記述する方法

```
crit:maxtim = -1
```

- 制約式の重み設定

  求解オプション defaultConstraintWeight を用いると，モデルファイルで出現する制約式全てに一律に重みを設定できます．次の例では，一律に重み 12 のソフト制約を指定しています．defaultConstraintWeight の値は，モデルファイル内でのみ設定できます．

  モデルファイル内

```
options.defaultConstraintWeight = 12;
```

  defaultConstraintWeight に 0 を設定すると，ハード制約として扱われます．defaultConstraintWeight の初期設定値は 0 です．

  defaultConstraintWeight と Constraint 関数（hardConstraint 関数，semiHardConstraint 関数，softConstraint 関数）が競合した場合，後者が優先されます．

- 目的関数の重み設定

  目的関数の重みは求解オプション defaultObjectiveWeight で指定します．defaultObjectiveWeight の値は，モデルファイル内でのみ設定できます．次の例では，目的関数の重みに 5 を指定しています．

  モデルファイル内

```
options.defaultObjectiveWeight = 5;
```

  求解オプション defaultObjectiveWeight の初期設定値は 1 です．

- 目的関数の目標値設定（**wcsp のみ**）

目標値 target の値は，求解オプション defaultObjectiveTarget で指定することができます．

```
options.defaultObjectiveTarget = 5; // wcsp のみ有効
```

Objective の引数での target 値と，求解オプション defaultObjectiveTarget の値が競合した場合は，Objective の引数の値の方が優先されます．目標値 target の初期設定値は 0 です．この求解オプションは rcpsp には無効です．

- 初期解生成時に使用する乱数発生の種

  初期解生成時に使用する乱数を発生するための値（種）を指定することができます．メタヒューリスティクス解法では初期解が最終的に得られる解の精度に影響することが知られており，初期解を変更することによって，より良い解が得られる場合があります．乱数を発生するための値は wcspRandomSeed で指定することができます．

  ```
  options.wcspRandomSeed = 3;
  ```

  初期解生成時に使用する乱数発生の種の初期設定値は 1 です．

- 計算回数

  上記 wcspRandomSeed で述べた通り，初期解を変更することによって最終的に得られる解が変化することがあります．初期解生成時に使用する乱数を変更し何回か解くことによって解の精度向上が期待されます．計算回数は wcspTryCount で指定することができます．

  ```
  options.wcspTryCount = 5;
  ```

  計算回数を複数回に設定した場合には，初期解生成時に用いる乱数を変更し，指定した回数求解を行い，その中で最もよい結果を自動的に残します．計算回数の初期設定は 1 です．

- スレッド数の上限（**wcsp のみ**）

  WCSP はマルチコア環境において並列処理を行うことにより，異なる初期値からの解の探索を効率的に行うことができます．スレッド数の上限は wcspthreads で指定することができます．

  ```
  options.wcspthreads = 8;
  ```

  本求解オプションは「計算回数」と合わせることにより有効です．例えば，計算回数を 8 回，スレッド数上限を 2 に設定する場合は以下のように記述します．

  ```
  options.wcspTryCount=8;
  options.wcspthreads = 2;
  ```

  上記のように設定をすることにより，各スレッドで計算回数を 4 (8/2) 回とする実行が行われます．実行 PC に指定スレッド数以上の CPU（コア）が搭載されている必要があります．本求解オプションのデフォルト値は 1 ですので，並列化を有効にする場合は 2 以上の値を設定してください．

- 制約充足フェーズにおける計算時間上限

  制約充足フェーズ（ハードペナルティ及びセミハードペナルティが残っているフェーズ）における計算時間上限は wcspPhaseOneMaxtime で指定することができます．

```
options.wcspPhaseOneMaxtime = 60;
```

-1 を設定した場合，無制限と解釈されます．wcspPhaseOneMaxtime の初期設定は-1 です．

wcspPhaseOneMaxtime に値を設定した場合，ハード・セミハードペナルティが 0 になるか設定した時間が経過した時点で経過時間をリセットし，そのあとで maxtim で設定した時間解の探索を行います．経過時間をリセットした後の探索は，ハード・セミハードペナルティのそれ以上の改善よりも，ソフトペナルティの改善を目指して行われます．

ある一定の時間が経過した後にハード・セミハードペナルティが改善することはないとわかっている問題に対して wcspPhaseOneMaxtime を使用することで，よりソフトペナルティの小さい解が得られる可能性があります．

- 解更新間隔計算時間上限

  解が更新されてから指定した時間が経過すると計算を終了します．本機能はハード制約とセミハード制約のペナルティが 0 になった後に有効になります．大規模問題では計算時間の設定は非常に難しいのですが，この機能を用いると簡単に有用な求解を行うことができます．解更新間隔計算時間上限は wcspPhaseTwoMaxInterval で指定することができます．

```
options.wcspPhaseTwoMaxInterval = 5;
```

-1 を設定した場合，無制限と解釈されます．解更新間隔計算時間上限の初期設定は-1 です．

- 指定した初期値からの探索（**wcsp のみ**）

  ユーザ指定による初期値から wcsp の探索をスタートするか否かを指定ことができます．

```
options.wcspInitialValueActivation = "on";
```

本求解オプションのデフォルト値は"off"ですので，ユーザ指定による初期値から探索をスタートする場合は"on"を設定してください．なお，tryCount で計算回数を 2 以上に設定した場合，全ての回においてユーザ指定による初期値から探索を出発します．

5.3.4 重み付き局所探索法（wls）に有効な求解オプション

重み付き局所探索法（wls/rcpsp）は内部で重みを自動調整しながら解を求める手法で，厳密な最適解を求めるものではありません．

- 反復回数上限

  停止条件の一つである，反復回数上限を設定します．初期設定値-1 は，無制限を意味します．反復回数上限と計算時間上限が無制限に設定されている場合，自動的に反復回数上限が設定されます．

  求解オプションファイル nuopt.prm に記述する方法

```
crit:maxitn = -1
```

- 計算時間上限

  停止条件の一つである，計算時間上限を設定します．初期設定値-1 は，無制限を意味します．反復回数上限と計算時間上限が無制限に設定されている場合，自動的に反復回数上限が設定されます．

求解オプションファイル nuopt.prm に記述する方法

```
crit:maxtim = -1
```

- メモリ利用量上限

  Nuorium Optimizer プロセスが使用することのできる最大メモリ量を制限するためのパラメータで，MiB 単位で指定します．例えば 1024 とすると 1GiB を上限とすることを意味し，1GiB を超えた場合に実行を停止します．初期設定値-1 は，無制限を意味します．

  求解オプションファイル nuopt.prm に記述する方法

```
wls:maxmem = 1000
```

- 目的関数の目標値設定

  目的関数の目標値を設定します．設定した場合，探索中に実行可能解でかつ目的関数値が目標値より良い（最小化問題であれば目標値以下）解が得られたら探索を終了します．

  求解オプションファイル nuopt.prm に記述する方法

```
wls:objectiveTarget = 10
```

- 計算回数

  wls では初期解を変更することによって最終的に得られる解が変化することがあります．初期解生成時に使用する乱数を変更し何回か解くことによって解の精度を向上することが期待されます．初期値は 1 です．

  求解オプションファイル nuopt.prm に記述する方法

```
wls:tryCount = 5
```

5.3.5　整数計画法（simplex/asqp）に有効な求解オプション

以下の求解オプションは単体法（simplex)/有効制約法（asqp）を，整数変数を含む問題について適用した場合にのみ有効です．

これらのアルゴリズムは，整数変数の整数条件を一部緩和した問題を繰り返し解きながら，分枝限定法というスキームによって，整数条件を満たす実行可能解の発見と，実行可能解の最適性の保証を行うものです．本質的に難しいクラスに属する問題であるため，連続変数のみからなる同程度の規模の問題に比べて，ときとして数倍程度の時間がかかることはよくあります．

また，この分枝限定法というスキームの好ましからざる特徴として，最適解を発見して最適性を証明するまでの時間が問題の規模のみならず，問題の性質に大きく左右されるということがあります．具体的には，数万変数の問題が数秒で解けてしまったり，一方では数百変数の問題を解くのに一時間以上を所要したりということがございます．

Nuorium Optimizer は現在得られる最新の技法を組み込み，できるだけ多様な問題が安定にとけるようにチューニングしておりますが，あらゆる性質の問題に対して常に良い設定を見出すことはできておりません．そのため，以下にご紹介する求解オプションを適宜設定することによって，デフォルト

の状態ではかなり時間を所要した問題をより効率よく解くことができることも十分にあり得ます．以下では効果的と思われる順にチューニングのための求解オプションをご紹介します．

以下説明の便宜のため，解いている問題が最小化問題だと仮定します．最大化問題は目的関数の符号を逆にして最小化を行っていることと等価なので，意味は同じですが以下に出てくる下界値と上界値という言葉を逆転させて解釈する必要があります．

- 切除平面法の求解オプション（simplex のみ有効）

  分枝限定法は整数性あるいは非凸性の条件を緩和した部分問題を解き，その解を，現在求められている実行可能解の下界値（これ以上小さくなることはあり得ないと理論的に保証された値）を参照しながら探索を行います．現在求められている実行可能解と下界値の差が零に近いとみなされたときに，現在得られている実行可能解の最適性が証明されたと判断し，アルゴリズムは停止します．

  切除平面法は，整数解が満たすであろう線形な制約式（妥当不等式と呼びます）を生成し，下界値を押し上げて，分枝限定法の収束を早める手法です．切除平面を加えるほど下界値は上がり，分枝限定法は早期に停止することが期待されますが，加えすぎると扱う問題の規模が増大するため，緩和解の計算コストがかさみ，結局実行時間の増大につながる可能性もあります．この求解オプションは切除平面を加える個数を調整するものです．値としては 0, 1, 2 のいずれかを取ることができ，デフォルト値は 1（標準的な量）です．2 を設定すると，切除平面を多めに加えるようになり，0 を設定すると全く加えなくなります．

  実行可能解は発見されるが，最適性がなかなか証明されずに停止しない状況ではこの求解オプションを 2 に設定するのが効果的な可能性がございます．一方で最適解が問題なく得られている場合には，切除平面の追加が計算のオーバーヘッドになっている可能性がありますので，0 に設定するのが有効な場合もあります．Nuorium Optimizer は V12 から追加される切除平面の種別を増やしたため，切除平面を比較的多めに加えるようになっています．結果として難しい（性質の悪い）問題のパフォーマンスは向上しましたが，以前の状態で高速にとけていた問題のパフォーマンスは低下している可能性はあります．切除平面の追加数が多すぎると感じる場合は，clevel を 0 に設定して試してみることをお勧めします．

  モデルファイルに記述する方法

  ```
  options.clevel = 1;
  ```

  求解オプションファイル nuopt.prm に記述する方法

  ```
  branch:clevel = 1
  ```

- ヒューリスティックサーチを調整する求解オプション（simplex/asqp のみ有効）

  良質な実行可能解を早期に得ることは分枝限定法の収束を加速するうえで大きく貢献します．緩和問題の変数を一つずつ整数値に固定していった結果として実行可能解を得るのが通常の分枝限定法ですが，問題の対称性が強い場合などには実行可能解の発見が非常に遅くなることはあり得ます．ヒューリスティックサーチとは緩和解を様々な方法で変形して，良質な実行可能解を早期に得るためのテクニックです．この手法がうまく機能すれば，分枝限定法の収束が加速する，あるいはより良質な実行可能解が早く得られる可能性があります．

各求解オプションは組み込まれているヒューリスティックサーチの手法（rounding, feasibility Pump, neighbour search, rins, rens）にそれぞれ対応しています．0 は行わない，1 は行うという意味となっています．また，rounding に関しては 2, 3 を設定することができ，値が大きいほど行う頻度が高くなります．feasibility Pump と neighbour search は大規模問題で実行可能解が得られにくい場合に効果的です．数千〜数万変数規模の実行可能解がなかなか現れない問題を解いている場合には，適用を試みるとよい結果が得られる可能性があります．
一方で規模の小さな問題（数百変数），あるいは実行可能解が既に多数出力される問題に対しては，ヒューリスティックサーチは効果がないばかりか時間を余計に所要する可能性があります．その際にはこれらの求解オプションをすべて 0 として，ヒューリスティクスサーチをやめてみてください．デフォルトでは，rins と rens は実行する（1），その他のヒューリスティックサーチは Nuorium Optimizer が実行有無を適当に判断する（負の値）です．
モデルファイルに記述する方法

```
options.rounding = -1; // -1，0，1，2，3 を選択できます
options.feasPump = -1; // -1，0，1 を選択できます
options.neighbourSearch = -1; // -1，0，1 を選択できます
options.rins = 1 // 0，1 を選択できます
options.rens = 1 // 0，1 を選択できます
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:round = -1     *  -1，0，1，2，3 を選択できます
branch:feas = -1      *  -1，0，1 を選択できます
branch:neigh = -1     *  -1，0，1 を選択できます
branch:rins = 1       *  0，1 を選択できます
branch:rens = 1       *  0，1 を選択できます
```

- 探索深さ

探索の深さを設定します．1 とすると深さ優先探索を行います．大きい値であるほど，発見的探索に近くなります．
p を大きめに設定すると実行可能解が見つかりにくく，所要メモリが大きくなりがちです．そのような場合には p=1（深さ優先探索）という設定が有効な場合があります．
モデルファイルに記述する方法

```
options.p = 10;
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:p = 10
```

- wcsp ヒューリスティクスに関する求解オプション（simplex/asqp のみ有効）

制約が線形である問題に限り，分枝限定法内で wcsp タブーサーチ法を用いる「wcsp ヒューリスティクス」を使用することができます．問題の変数が全て 0-1 変数の場合，デフォルトで有効に

なっています.

wcsp ヒューリスティクスは通常の wcsp タブーサーチ法と以下が異なります.

1. 0-1 変数以外の変数は適当な刻み幅の DiscreteVariable として扱われる
2. wcsp の target 値は，変数の上下限を考慮した目的関数の下限あるいは上限が使用される
3. 内部で収束判定が行われ，自動的に終了する

wcsp ヒューリスティクスの計算時間消費が多すぎる場合，探索の反復回数上限や求解時間を制限することで求解時間の増加を抑えられる可能性があります.

求解オプションの設定方法は下記の通りです.

- モデルファイルに記述する方法

```
options.useWcsp = 1;              // 0(使用しない), 1(使用する)
options.branchWcspMaxitn = -1;  // -1 または正の整数
options.branchWcspMaxtim = 180; // 正の整数
```

- 求解オプションファイル nuopt.prm に記述する方法

```
branch:useWcsp = 1         * 0(使用しない), 1(使用する)
branch:wcspMaxitn = -1     * -1 または正の整数
branch:wcspMaxtim = 180    * 正の整数
```

- 足切り点

分枝限定法による探索時の足切り点です．設定した場合，足切り点（cutoff）より悪い解しか与えないことが解った問題は探索の対象から外します．従って，適切に設定すると計算の無駄を省くことができます．この値を最小化問題の場合は小さく（最大化問題の場合は大きく）設定するほど，足切り条件は厳しく，探索の対象は狭くなり，計算の手間は減ります．厳しく設定しすぎると実行可能解がない（(NUOPT16) Infeasible MIP.）というエラーになりますので，注意が必要です．初期設定では何も設定されていません.

モデルファイルに記述する方法

```
options.cutoff = 1.0e-2;
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:cutoff = 1.0e-2
```

- 分枝変数のスコアの算出方法

Nuorium Optimizer の分枝限定法では分枝時に改善される単位あたりの目的関数値の予測値（擬コスト）を算出しています．この擬コストから変数のスコアを算出し，分枝する変数を決定しています．スコアの算出方法は，分枝時に生成される二つの子問題における目的関数値の改善量を擬コストから算出し，それらを組み合わせます．この組み合わせ方として auto（デフォルト）, sum, product を用意しています.

スコアの算出方法は次のように設定します.

モデルファイルに記述する方法

```
options.branchVariableSelectScore = "product";
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:variableSelectScore = product
```

- 計算時間上限

計算時間の上限です．計算開始から秒単位の計算時間が計算時間上限 maxtim を越えると，下記のエラーメッセージとともに現在までの最適解を出力して実行を終了します．（0 以下の値は設定していないのと同じ意味になります．）計算時間には，前処理や最初の緩和解を求める時間が含まれます．初期設定では計算時間上限なしを意味する-1 が設定されています．

```
(NUOPT 21) B&B itr. timeout (with feasible.sol).
```

```
(NUOPT 22) B&B itr. timeout (no feasible.sol).
```

モデルファイルに記述する方法

```
options.maxtim = -1;
```

求解オプションファイル nuopt.prm に記述する方法

```
crit:maxtim = -1
```

- 整数解個数上限

実行可能解の個数の上限です．0 以下は設定していない（無制限）と同じと見なされます．1 とすれば，実行可能解を 1 つだけ求めて終了する，ということが可能になります．計算開始から見つかった実行可能解の数が maxintsol を越えると，以下のエラーメッセージとともに，現在までの実行可能解を出力して実行を終了します．

```
(NUOPT 37) B&B terminated with given # of feasible.sol.
```

モデルファイルに記述する方法

```
options.maxintsol = -1;
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:maxintsol = -1
```

- 最大メモリ量制限

Nuorium Optimizer プロセスが使用することのできる最大メモリ量を制限するための求解オプションで，MiB 単位で指定します．例えば 1024 とすると 1GiB を上限とすることを意味し，1GiB を超えた場合に実行を停止します．なお，メモリ使用量は物理メモリだけでなく，仮想メモリを含んだ量です．

負の値を設定すると，システムで利用可能なメモリ量が残り -maxmem 以下になったときに実行を停止します．メモリ上限によって実行が停止した場合には NUOPT43 エラーが，実行可能解が見

つかっていない場合には NUOPT44 エラーが出力されます．この場合，現在までの最適解を出力して実行を終了します．

```
(NUOPT 43) B&B memory error (with feasible.sol.).
```

```
(NUOPT 44) B&B memory error (no feasible.sol.).
```

モデルファイルに記述する方法

```
options.maxmem = -10;
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:maxmem = -10
```

- 上下界値のギャップによる停止

上下界値のギャップが，指定した値を下回る場合に解の探索を停止します．停止した際は，以下のエラーが出力されます．

```
(NUOPT 45) B&B gap reaches under the limit.
```

実行可能解が求まってはじめて上下界値のギャップは意味を持ちますので，このエラーで停止した場合には必ず実行可能解の出力が成されます．初期状態では，設定されていません．

ギャップ閾値の設定方法として，絶対値で指定する方法と相対値で指定する方法の二つがあります．

- 絶対値で設定する

  絶対値で設定する場合は，(上界値 - 下界値) に対する閾値を設定します．

  モデルファイルに記述する方法

  ```
  options.gaptol = 10;
  ```

  求解オプションファイル nuopt.prm に記述する方法

  ```
  branch:gaptol = 10
  ```

- 相対値で設定する

  相対値で設定する場合は，上界値を Z_p 下界値を Z_d とした時の以下の式で定義される相対ギャップに対する閾値を設定します．

$$relgap := \begin{cases} 0, & Z_p = Z_d = 0 \\ \frac{|Z_p - Z_d|}{\max(|Z_p|, |Z_d|)}, & Z_p \cdot Z_d \geq 0 \\ 1, & Z_p \cdot Z_d < 0 \end{cases}$$

  モデルファイルに記述する方法

  ```
  options.relgaptol = 1.0e-2;
  ```

  求解オプションファイル nuopt.prm に記述する方法

```
branch:relgaptol = 1.0e-2
```

- 目的関数値による停止

  最小化（最大化）問題において目的関数値が設定値以下（以上）になった場合に解の探索を停止します．停止した際には以下のメッセージが出力されます．

```
(NUOPT 23) B&B objective reaches under the limit.
```

  停止する値は次のように設定します．

  モデルファイルに記述する方法

```
options.branchObjTarget = 100;
```

  求解オプションファイル nuopt.prm に記述する方法

```
branch:objTarget = 100
```

- スレッド数の上限

  マルチコア環境において並列分枝限定法を用いることができます．ユーザは並列分枝限定法が使用するスレッド数の上限を指定することができます．

  -1 を指定すると，Nuorium Optimizer が内部で適切なスレッド数を設定します（設定の結果シングルスレッドの分枝限定法と並列分枝限定法のいずれになるかは環境により異なります）．

  0 あるいは 1 と設定した場合は，シングルスレッドの分枝限定法が動作します．

  モデルファイルに記述する方法

```
options.bbthreads = 1;
```

  求解オプションファイル nuopt.prm に記述する方法

```
branch:threads = 1
```

- 並列分枝限定法の手法

  Nuorium Optimizer V22 から並列分枝限定法の手法を racing（デフォルト）, deterministicRacing, subtree の 3 つから選択できるようになりました．それぞれの手法の特徴については付録の B2.4 並列分枝限定法をご覧ください．

  並列分枝限定法の手法は次のように設定します．

  モデルファイルに記述する方法

```
options.branchParallelMethod = "deterministicRacing";
```

  求解オプションファイル nuopt.prm に記述する方法

```
branch:parallelMethod = deterministicRacing
```

- 初期解の修復

  Nuorium Optimizer の分枝限定法にはユーザが与えた解を実行可能解として用いる機能があります．使い方は求解前に変数に初期値を設定します．設定された値を初期解として認識し，初期解が全

ての制約を満たしていれば上界値を算出してから分枝限定法を開始します．この機能によって計算開始直後でも限定操作が働き，計算時間の短縮に繋がります．

以下は初期値設定方法の具体例です．

```
IntegerVariable x;
Variable y;
0 <= x <= 6;
1.5 <= y <= 6;
x + y >= 4;
x = 3;    // 変数 x に初期値を設定
y = 1.5;  // 変数 y に初期値を設定
solve();
```

一方，与えた初期解が制約を満たさない場合は実行可能解として認識されません．連続変数の値を正確に設定するのが面倒な場合や，数値的な理由で制約違反と見なされる場合に不都合です．このようなケースにおいて初期解の修復機能を有効にすると，与えた初期解から実行可能解が得られる可能性があります．具体的には整数変数のみに値を設定して連続変数の値は Nuorium Optimizer に任せたいケース，多段階の求解をおこなう際に前回の結果を用いたいケースにおいて非常に有効です．

初期解の修復機能の実行に必要な条件は次の二つです．

- 求解オプション branchRepairSolution が有効になっている
- 整数変数に与えた初期値が上下限制約を満たしている

branchRepairSolution の有効化は次の通りです．

モデルファイルに記述する方法

```
options.branchRepairSolution = "on";
```

求解オプションファイル nuopt.prm に記述する方法

```
branch:repairSolution=on
```

なお，分枝限定法のスレッド数のオプション（bbthreads）は初期解の修復機能にも有効であり，スレッド数を増やすと初期解の修復機能によって実行可能解が得られる可能性も高まります．

5.4 MPS ファイルに関する設定

この節では MPS ファイルから問題を入力する際の付加情報の指定方法を解説します．MPS ファイルに対する付加情報に関しては，求解オプションファイル nuopt.prm を用いて指定する方法のみが提供されています．

- 最小化，最大化の指定

  MPS ファイルから読み込んだ問題を目的関数の最小化問題/最大化問題のいずれとして解くかを指定します．初期設定は最小化です．

```
maximize
```

- 各種ラベル名

  これらは MPS ファイル中に複数の RHS/BOUNDS/RANGE/目的関数行があるとき，実際の計算で用いるものを指定します.

  デフォルトでは最初に現れたものとなります.

```
mpsfile:rhs = 文字列　(RHS ラベル名)
mpsfile:bou = 文字列　(BOUNDS ラベル名)
mpsfile:ran = 文字列　(RANGE ラベル名)
mpsfile:obj = 文字列　(目的関数行ラベル名)
```

  上記に関して該当するラベルを持つものが存在しない場合には，以下のようなエラーが出力されます.

```
(MPS FILE 13) Specified rhs: RHS データラベル not found
(MPS FILE 11) Specified bound: BOUND データラベル　not found
(MPS FILE 15) Specified range data: RANGE データラベル not found.
(MPS FILE 12) Specified objective: 目的関数行名 not found
```

5.5 求解オプション一覧

Nuorium Optimizer で設定可能な求解オプションの一覧です.

名称	選択	デフォルト値	意味
bbthreads [branch:threads = 1]	int	1	並列分枝限定法のスレッド数の上限
branchObjTarget [branch:objTarget = 100]	double	未定義	目的関数値による停止条件
branchParallelMethod [branch:parallelMethod=racing]	"racing" "deterministicRacing" "subtree"	"racing"	並列分枝限定法の手法
branchRepairSolution [branch:repairSolution=off]	"on" "off"	"off"	初期解の修復機能

名称	選択	デフォルト値	意味
branchVariableSelectScore [branch:variableSelectScore=auto]	"auto" "sum" "product"	"auto"	分枝変数のスコアの算出方法
branchWcspMaxitn [branch:wcspMaxitn = -1]	int	-1	分枝限定法内の wcsp ヒューリスティクスでの最大反復回数
branchWcspMaxtim [branch:wcspMaxtim = -1]	int	-1	分枝限定法内の wcsp ヒューリスティクスでの最大求解時間
clevel [branch:clevel = 1]	0/1/2	1	導入される切除平面の数の目安（0 は導入しない）（分枝限定法専用，asqp には無効）
crossover [cross:off]	"off" "on"	"off"	単体法へのクロスオーバー（higher 専用）

名称	選択	デフォルト値	意味
cutoff [branch:cutoff = 1.8]	double	未定義	足切り点（分枝限定法専用）
defaultConstraintWeight	double	-1	指定のない制約式の重み（デフォルトはハード制約）
defaultObjectiveTarget	double	0	目的関数の目標値（デフォルトは 0）（wcsp のみ）
defaultObjectiveWeight	double	1	目的関数を変形した制約式の重み（デフォルトは重み 1 のソフト制約）
eps [crit:eps = 1.0e-8]	double	自動設定	停止条件

名称	選択	デフォルト値	意味
feasPump [branch:feas = 0]	-1/0/1	-1	Feasibility Pump によるヒューリスティックサーチの頻度（-1 はシステムが適当に設定する）（分枝限定法専用）
gaptol	double	-1（指定なし）	上下界値のギャップの閾値（絶対値で設定）
iisDetect [param:iis = on]	"off" "on"	"on"（行う）	実行不可能な行集合（IIS）探索を行う/行わない
maxintsol [branch:maxintsol = 3]	int	-1（無制限）	整数解取得個数上限（個）
maxitn [crit:maxitn = 150]	int	内点法： 150 wcsp/wls/rcpsp： -1（無制限）	反復回数の最大（内点法と wcsp/wls/rcpsp）

名称	選択	デフォルト値	意味
maxmem [branch:maxmem = 500]	int	-10（残り 10MiB）	分枝限定法のメモリ利用量上限（MiB），残り利用可能メモリによる制限（負値の場合，MiB）
maxnod [branch:maxnod = 100000]	int	-1（無制限）	探索問題数上限（分枝限定法専用）
maxtim [crit:maxtim = 3600]	int	-1（無制限）	計算時間上限(秒)（分枝限定法と wcsp/wls/rcpsp）

名称	選択	デフォルト値	意味
method	"auto"	"auto"	求解アル
[method:auto]	"lipm"		ゴリズム
	"higher"		
	"tipm"		
	"bfgs"		
	"simplex"		
	"dual_simplex"		
	"hsimplex"		
	"asqp"		
	"lsqp"		
	"tsqp"		
	"lsdp"		
	"trsdp"		
	"wcsp"		
	"wls"		
	"rcpsp"		
mtxfree	"on"	"off"	クリロフ
[linear:mtxfree = on]	"off"		部分空間
			法を使用
			して連立
			一次方程
			式を解く
			か否か

名称	選択	デフォルト値	意味
multDataPolicy	int	0（許さない）	同一のデータについてデータを重複して与えることを許すかどうか（1 に設定すると警告扱いとなる）
neighbourSearch [branch:neigh = 1]	-1/0/1	-1	Neighbour search によるヒューリスティックサーチの頻度（-1 はシステムが適当に設定する）（分枝限定法専用）
noDefaultSolout	int	0	solout() を陽に呼ばないと解の出力を行わない

名称	選択	デフォルト値	意味
noDefaultSolve	int	0	solve()を陽に呼ばないと求解を行わない
outfilename [output:name = myout]	char*	0 （未定義の場合「モデル名.sol」）	Nuorium Optimizerの解ファイル名（"_NULL_"とすると出力を行わない）
outputExpression	int	1	Expressionの CSV ファイル出力を行うかどうか
outputMode [output:mode = normal]	"silent" "normal"	"normal"	標準出力モード
outputParameter	int	1	Parameterの CSV ファイル出力を行うかどうか
p [branch:p = 10]	int	10	探索深さ（分枝限定法専用）

名称	選択	デフォルト値	意味
relgaptol	double	-1（指定なし）	上下界値のギャップの閾値（相対値で設定）
rens [branch:rens = 1]	0/1	1	rens によるヒューリスティックサーチの頻度（分枝限定法専用）
rins [branch:rins = 1]	0/1	1	rins によるヒューリスティックサーチの頻度（分枝限定法専用）
rounding [branch:round = 1]	-1/0/1/2/3	-1	rouding によるヒューリスティックサーチの頻度（-1 はシステムが適当に設定する）（分枝限定法専用）

名称	選択	デフォルト値	意味
scaling [scaling:cr]	"off" "minmax" "cr" "on"	"cr" (hsimplex のみ "minmax")	スケーリングの種類
told [simplex:told = 1.0e-6]	double	1.0e-6	双対変数の双対実行可能性判定閾値(単体法のみ)
tolx [simplex:tolx = 1.0e-8]	double	1.0e-8	主変数の実行可能性判定閾値(単体法のみ)
useWcsp [branch:useWcsp = 1]	0/1	1	分枝限定法で wcsp ヒューリスティクスを使用する
wcspInitialValueActivation	"off" "on"	"off"(行わない)	指定した初期値からの探索を行う/行わない(wcsp のみ)
wcspPhaseOneMaxtime	int	-1(無制限)	制約充足フェーズにおける計算時間上限(wcsp のみ)

名称	選択	デフォルト値	意味
wcspPhaseTwoMaxInterval	int	-1（無制限）	解更新間隔計算時間上限（wcspのみ）
wcspRandomSeed	int	1	乱数発生の種（wcspのみ）
wcspthreads	int	1	wcspの並列化時のスレッド数上限（wcspのみ）
wcspTryCount	int	1	乱数を変更しての計算回数（wcspのみ）
[hsimplex:method = -1]	-1/0/1	-1	hsimplexで使用するアルゴリズム（hsimplexのみ）
[wls:maxmem = 500]	int	-1（無制限）	wlsのメモリ利用量上限（MiB）（wlsのみ）
[wls:objectiveTarget = 10]	double	未定義	目的関数の目標値（wlsのみ）

名称	選択	デフォルト値	意味
[wls:tryCount = 5]	int	1	乱数を変更しての計算回数（wls のみ）

第6章 MPSファイル・LPファイル

Nuorium Optimizer は MPS ファイル形式で記述された数理最適化問題を解くことができます．コマンドラインでの使用法は，以下の通りです．

```
prompt% nuopt ファイル名
```

上記の場合，拡張子が.lp であるファイルを LP ファイル形式として，それ以外を Free-MPS ファイル形式として読み込みます．

コマンドラインで使用する場合、ファイル形式をオプションとして指定することが可能です．この場合ファイル拡張子は無視されます．

Fix-MPS ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -fix-mps ファイル名
```

Free-MPS ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -free-mps ファイル名
```

LP ファイル形式で読み込む場合には以下のようにします．

```
prompt% nuopt -lp ファイル名
```

6.1 MPS ファイルに対する標準出力

MPS ファイルに対する求解コマンド nuopt を実行すると標準出力に計算の進行が表示されます．

```
prompt% nuopt ex1.mps
[About Nuorium Optimizer]
Nuorium Optimizer x.x.x  (NLP/LP/IP/SDP module)
         <with META-HEURISTICS engine "wcsp"/"rcpsp">
         <with Netlib BLAS>
        , Copyright (C) 1991 NTT DATA Mathematical Systems Inc.

[Reading MPS file: ex1.mps]
MPS_FILE_NAME                                          ex1.mps
PROBLEM_NAME(TITLE)                                   example1
```

```
ROWS                                                    4
COLUMNS                                                 3
NONZEROS                                               11
OBJECTIVE                                               f
RHS                                                     b

[Problem and Algorithm]
NUMBER_OF_VARIABLES                                     3
NUMBER_OF_FUNCTIONS                                     4
PROBLEM_TYPE                                   MINIMIZATION
METHOD                                         HIGHER_ORDER

[Progress]
<preprocess begin>.........<preprocess end>
<iteration begin>
    res=2.6e+001 .... 1.6e-004 . 3.9e-009
<iteration end>

[Result]
STATUS                                               OPTIMAL
VALUE_OF_OBJECTIVE                                     -10.5
ITERATION_COUNT                                            7
FUNC_EVAL_COUNT                                           10
FACTORIZATION_COUNT                                        8
RESIDUAL                                    3.903545017e-009
ELAPSED_TIME(sec.)                                      0.00
SOLUTION_FILE                                        ex1.sol
```

この出力中以下の部分：

```
[Reading MPS file: ex1.mps]
PROBLEM_NAME(TITLE)                                  example1
ROWS                                                       4
COLUMNS                                                    3
NONZEROS                                                  11
OBJECTIVE                                                  f
RHS                                                        b
```

は MPS ファイルを読み込むインタフェース部分からのメッセージで，MPS ファイルの NAME セクションにあるタイトル example1，ROWS セクションで指定された行の数（4），COLUMNS セクションで指

定された変数の数（3）と総非零要素数（11），目的関数の行の名前（F）と右辺ラベル（B）を示しています．

以降の標準出力は SIMPLE モデルで最適化計算を行った場合と同じです．詳しくは 2 標準出力をご覧ください．

6.2 MPS ファイル及び LP ファイルに対する解ファイル

nuopt は計算終了と共に，解ファイル出力します．これらには最適化アルゴリズム停止時における，変数や関数（目的関数及び制約式），双対変数（シャドウプライス）の値が記されています．解ファイルは入力ファイルの拡張子を.sol に変えたものが作成されますが，特殊な場合は次のルールに則って解ファイルの名前が決定されます．

入力ファイル名	解ファイル名	備考
ex1	ex1.sol	
ex1.mps, ex1.lp	ex1.sol	. 以降が.sol に
ex1.4.mps, ex1.3.lp	ex1.4.sol	最後の. 以降のみが.sol に
/nuopt/samples/ex1.mps	ex1.sol	パス名は反映されない

6.3 MPS ファイルに対する求解オプション設定

MPS ファイルに対しても，求解オプションを用いて各種設定をすることができます．MPS ファイルに対する情報を求解オプションで指定するには，求解オプションファイル nuopt.prm を用いる方法のみが提供されています．MPS ファイルに特有の求解オプションとして，以下が提供されています．

- 最小化，最大化の指定

  MPS ファイルから読み込んだ問題を目的関数の最小化問題/最大化問題のいずれとして解くかを指定します．MPS ファイルの初期設定は最小化ですので，最大化問題として解くには，明示的に指定する必要があります．

```
maximize
```

- 各種ラベル名

  これらは MPS ファイル中に複数の RHS/BOUNDS/RANGE/目的関数行があるとき，実際の計算で用いるものを指定します．

  デフォルトでは最初に現れたものとなります．

```
mpsfile:rhs = 文字列   (RHS ラベル名)
mpsfile:bou = 文字列   (BOUNDS ラベル名)
mpsfile:ran = 文字列   (RANGE ラベル名)
mpsfile:obj = 文字列   (目的関数行ラベル名)
```

  上記に関して該当するラベルを持つものが存在しない場合には，以下のようなエラーが出力され

ます.

```
(MPS FILE 13) Specified rhs: RHS データラベル not found
(MPS FILE 11) Specified bound: BOUND データラベル  not found
(MPS FILE 15) Specified range data: RANGE データラベル not found.
(MPS FILE 12) Specified objective: 目的関数行名 not found
```

6.4 MPS ファイルの具体例

MPS ファイルは次のような一般形の線形/二次計画問題を記述するためのものです.

$$
\begin{array}{ll}
\text{最小／最大化} & f(x) \\
\text{条　件} & c_{L_i} \le g_i(x) \le c_{U_i}, \quad i = 1, \cdots, m \\
 & b_{L_j} \le x_j \le b_{U_j}, \quad j = 1, \cdots, n \\
\text{初期値} & x_j = x_j^{\,0} \quad j = 1, \cdots, n
\end{array}
$$

ここで $f(x)$, $g_i(x)$ は二次関数で

$$f(x) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n + \frac{1}{2} x^t H_0 x$$

$$g_i(x) = a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n + \frac{1}{2} x^t H_i x$$

と表されます.

MPS ファイルは MPS フォーマットと呼ばれる形式で，次の情報を記述したものです.

目的関数，制約式の線形部分の係数	c_j, a_{ij}
制約式の上下限	c_{L_i}, c_{U_i}
目的関数，制約式の Hessian の要素	H_0, H_i
変数の上下限	b_{L_j}, b_{U_j}
変数の初期値	$x_j^{\,0}$

本マニュアルでは MPS ファイルのフォーマットに対する定義は述べず，具体的な問題に対する MPS ファイルの対応を記述するに留めます．MPS ファイルフォーマットの詳細をご所望の方は nuopt-support@ml.msi.co.jp までご連絡ください.

$$
\begin{aligned}
&\text{最小化} && 4x_1 - x_3 + {x_2}^2 && \\
&\text{条件} && x_1 + x_4 && = 4 \\
& && x_1 + 2x_2 - x_3 + x_1x_2 && \leq 10 \\
& && x_2 + x_3 && \geq 2 \\
& && x_1 \geq 3 && \\
& && 4 \geq x_2 \geq 1 && \\
& && x_3, x_4 \geq 0 && \\
&\text{初期値} && {x_1}^0 = 4, {x_2}^0 = 2 && \\
& && {x_3}^0, {x_4}^0 = 0 &&
\end{aligned}
$$

次は上記の二次計画問題を記述した MPS ファイルの例です.

```
NAME          SAMPLE
ROWS
 E   R1
 L   R2
 G   R3
 N   C
COLUMNS
    X1        R1                  1.    R2                  1.
    X1        C                   4.
    X2        R2                  2.    R3                  1.
    X3        R2                 -1.    R3                  1.
    X3        C                  -1.
    X4        R1                  1.
RHS
    B         R1                  4.    R2                 10.
    B         R3                  2.
BOUNDS
 LO BND1      X1                  3.
 LO BND1      X2                  1.
 UP BND1      X2                  4.
HESSIAN
    X2        X2                  2.
    R2
    X1        X2                  1.
INITIAL
    X1                            4.
```

```
    X2                                2.
ENDATA
```

6.5 LP ファイルの具体例とファイルフォーマット

LP ファイルとは，下記のようなフォーマットで数理最適化問題を記述したものです.

```
MIN
 2x1 + 3x2
SUBJECTTO
  x1 + x4           =  4
 -x1 + x2 - 0.5x3 <= 10
  x2 + 0.25x3      >=  2
BOUND
 x1 < -2
GEN
 x2
END
```

ここでは Nuorium Optimizer が対応している LP ファイルフォーマットについて簡単に説明します.

6.5.1 命名規則

変数，目的関数，制約式の名前に用いることができる文字は以下通りです.

- アルファベット：a-z A-Z
- 数字：0-9
- 記号：!"#$%&/,.;?@_‘´{}()|~’

```
4x2
```

のような記述は x2 の部分を名前とみて 4 × x2 と解釈します.

6.5.2 コメントと空行

\から行末までをコメントとします．また，空行は読み飛ばします.

6.5.3 半角スペースおよび式中の改行

"数 変数"および"変数 変数"という記述は，積と解釈します．式中の改行は，半角スペースと同様に扱われます.

6.5.4 lp ファイルの節

数理最適化問題を構成する節の記述順は以下の通りです.

1. 問題名（省略可）
2. 目的関数
3. 制約式
4. 境界条件（省略可）
5. 変数型（省略可）
6. 初期値（省略可）

各節の内容は，対応する指示語を行頭から記述した次の行から記述します．指示語は大文字・小文字を問いません．問題記述が終わった後に

```
END
```

と記述する必要があります．次節以降の lp ファイルの節についての説明では，下記の通りの書式を用います.

- `[ ]`：[] 内は省略可能
- `(a,b,..)`：`a b ..` のいずれか
- `{ }**`：{}内の繰り返し
- `number`：数値
- `name`：名前
- `term`：(number, [number] name [([*] name, ^2)])
- `expression`：[(+,-)] term { [(+,-) term] }**
  目的関数における定数項は 1 つまで可

6.5.5 問題名節

指示語：`prob`, `problem`

6.5.6 目的関数節

指示語：`minimize`, `maximize`, `min`, `max`, `mininum`, `maximum`

書式：

```
[name:] expression
```

name 前後の半角スペースは読み飛ばします．name が省略された場合`"Objective"`が目的関数名となります．二次式を記述する場合には

```
0.5 x1^2 + 6 x1 x2 + 2x2^2
```

もしくは

```
+ [ x1^2 + 3x1 * x2 + 2x2 ^2 ] / 2
```

と記述します.

6.5.7 制約式節

指示語：subject to, subject to:, such that, st, s.t., st., subjectto, suchthat, such
書式：

```
[name:] expression (<,<=,=<,>,>=,=>,=) number
```

name 前後の半角スペースは読み飛ばします．Name が省略された場合"co(開始行数)"が制約式名となります．不等号・等号と右辺値の間に改行を挟んではいけません.

6.5.8 境界条件節

指示語：bounds, bound
書式：

```
number (<,<=,=<,>,>=.=>,=) name
name (<,<=,=<,>,>=.=>,=) number
number (<,<=,=<,>,>=.=>,=) name (<,<=,=<,>,>=.=>,=) number
name free
(-inf, -infinite) (<,<=,=<) name
name (>,>=.=>) (-inf, -infinite)
(inf, infinite) (>,>=.=>) name
name (<,<=,=<) (inf, infinite)
```

<, >はそれぞれ<=, =>と解釈します．境界条件において，重複した定義はエラーとなります．上界値は，未設定の場合には+inf と設定されます．下界値が未設定の場合は

1. 上界値が 0 未満であれば-inf
2. そうでない場合 0

と設定されます.

6.5.9 変数型節

書式：

```
{name}**
```

指示語：generals, general, gens, gen
指示語に続く名前の変数を一般整数変数とします．境界条件を与えていない変数に対しては，境界

条件を [0, +inf) とします.

指示語：integers, integer, ints, int
指示語に続く名前の変数を整数変数とします．境界条件が与えられていない場合は，境界条件を [0, 1] とします.

指示語：binaries, binary, bins, bin
指示語に続く名前の変数を 0-1 整数変数とします.

6.5.10 初期値節

指示語：init, initial
書式：

```
name = number
```

6.6 変数の境界条件について

MPS ファイルにおいて 'INTORG' マーカーで指定した整数変数に対して，境界条件が与えられない場合は 0-1 整数変数と推定し，計算を実行します．また，MPS ファイルおよび LP ファイルのいずれについても，下記のルールが適用されます.

- 下界値のみが指定された場合，上界値+inf とする.
- 上界値のみが指定された場合，これが 0 未満であれば下界値を-inf とする.
- 上界値のみが指定された場合，これが 0 以上であれば下界値を 0 とする.

6.7 MPS ファイルおよび LP ファイルへの変換

6.7.1 変換方法

モデリング言語 SIMPLE を用いてモデルファイルを記述した後，mpsout, mpsout_e または lpout という関数を呼び出すことによって，システムの内容を MPS ファイル形式にて出力することができます．mpsout の場合は Fix-MPS ファイルとして，mpsout_e の場合は Free-MPS ファイルとして，lpout の場合は LP ファイルとして出力します．モデルファイルは，線形（整数）計画問題あるいは二次（整数）計画問題である必要があります.

```
mpsout();   // Fix-MPS ファイルの出力
mpsout_e(); // Free-MPS ファイルの出力
lpout();    // LP ファイルの出力
```

作成される MPS ファイルの名称はモデルファイル名（.smp を除いたもの）.mps となります．出力

される MPS ファイル名を指定したい場合には次のように.mps を除いた部分を引数として与えます.

```
mpsout("filename"); // filename.mps という MPS ファイルを出力
```

6.7.2 MPS ファイルへの変換機能使用時の注意

前節で述べた MPS ファイルへの変換機能を用いる場合には，以下の点に注意をする必要があります.

- 変数名，関数名

  MPS ファイルの変数名は文字数の制限がありますので，変数名は一律 X1,X2,...，関数名は F1,F2,... という名前に変更されます．これらの名前と SIMPLE 内部で付けた名前との対応は出力される MPS ファイルの先頭部分に，次のように MPS ファイル書式のコメントの形式で記述されます.

```
VARIABLE NAME (MPSFILE - original)

X1          -   var[1]
X2          -   var[2]
                    ...

FUNCTION NAME TABLE (MPSFILE - original)

F1          -   obj
F2          -   NONAME
       ...
```

- 最大化/最小化

  最大化問題は**最小化問題に変換**されます（目的関数の符号が反対になります）.

- 問題規模の制限

  変数，関数のいずれかが 9999999 個以上の問題は出力できません.

第 7 章　高度な利用法

本章では，Nuorium Optimizer の高度な利用法を扱います.

7.1　変数の初期値

変数の初期値の設定は，内点法，分枝限定法，wcsp，wls 及び rcpsp において有効です.

内点法では設定された初期値を探索の出発点として採用します．特に非線形計画問題においては，以下のようなケースで有効である可能性があります.

- 局所解が大域最適解とは保証されないケース
- 実行可能領域が連結でないケース
- 探索点によっては関数評価で浮動小数点エラーが発生する可能性があるケース

分枝限定法では設定された初期値が制約を満たす場合は，実行可能解として与えられます．与えられた実行可能解は枝刈りや実行可能化の更新等に用いられ，実行可能解が見つかりづらいが別のロジックでは見つけやすい場合や多段階求解の場合などに有効です．一方初期値が制約を満たさない場合は，初期値の設定は無視されます．この場合，微小な違反により初期値の設定が有効にならないケースがあります．そのような場合は求解オプション `branchRepairSolution` による「初期解の修復機能」を用いると初期値から実行可能解を構成できる可能性があります.

求解アルゴリズム wcsp では設定された初期値を探索の出発点として採用します．以下は利用にあたっての注意点です.

- 求解オプション `wcspInitialValueActivation` のデフォルト値が"off"のため，ユーザ指定による初期値から探索をスタートする場合は本オプションを"on"に設定する必要があります.
- 求解オプション `tryCount` で計算回数を 2 以上に設定した場合，全ての試行回においてユーザ指定による初期値から探索を出発します.
- 離散変数 `DiscreteVariable` の定義域に文字列を用いる場合は，該当の変数に初期値を設定することはできません.
- 以下のケースなどでは初期値の設定は無視されます.
    - 初期値が整数性を満たさない
    - 初期値が変数の固定条件を違反している
    - 初期値が `selection` 制約を違反している

求解アルゴリズム wls では設定された初期値を探索の出発点として採用します．ただし試行回数（`tryCount`）が 2 以上で設定された場合は 2 回目以降の試行では設定された初期値ではなくランダムに構成された初期値が採用されます.

付録 A Nuorium Optimizerのエラー/警告メッセージ

A.1 Nuorium Optimizer のエラー/警告メッセージ

Nuorium Optimizer 本体の出力するエラー/警告メッセージは,

- Nuorium Optimizer 本体部の計算で検出されたエラー/警告

```
(NUOPT 番号)　　エラー/警告メッセージ
```

- 求解オプションに関するエラー/警告

```
(SOLVER OPTION 番号)　エラーメッセージ
```

- MPS ファイル解釈時に検出されたエラー/警告

```
(MPS FILE 番号)　　エラーメッセージ
```

- LP ファイル解釈時に検出されたエラー/警告

```
(LP FILE 番号)　　エラーメッセージ
```

の 4 種類です．エラーメッセージは，標準出力に出力されます．SIMPLE ロードモジュールの場合には

```
ex2.smp:10:error: (SIMPLE 193) アルゴリズム実行時の問題
ex2.smp:10:error: (NUOPT 10) IPM iteration limit exceeded.
```

などのように，SIMPLE のエラーメッセージの一部として表示されますが，これはモデリング言語 SIMPLE が Nuorium Optimizer を起動している形を取っているためです.

A.1.1 Nuorium Optimizer のエラー/警告メッセージ

エラーに関する解説の最後に [解出力なし] とあるエラーの場合には，解ファイルへの変数値や関数値に関する出力は行われません．[解出力あり] とあるエラーの場合には最適性の保証がない解を一応は出力します.

エラー番号	エラーメッセージ 説明
1	(NUOPT 1) memory error in preprocessing. 前処理部で所要メモリが，使用可能な量をオーバーしました．[解出力なし]

エラー番号	エラーメッセージ 説明
2	(NUOPT 2) infeasible (linear constraints and variable bounds). 線形制約や変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました．[解出力なし]
3	(NUOPT 3) neither valid objective nor constraint in this model. モデル（問題）に目的関数および制約式がありません．[解出力なし]
5	(NUOPT 5) infeasible variable bounds. (NUOPT 5) infeasible integer variable bounds. 連続変数もしくは整数変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました．（整数変数が整数性に違反する様な上下限を課されている場合等に発生します．）[解出力なし]
6	(NUOPT 6) unbounded (linear constraints and variable bounds). 線形制約と変数の上下限から問題の最適解は有界とはならないことが前処理部で判定されました．[解出力なし]
7	(NUOPT 7) internal error. [内部ルーチン名] 内部エラーが発生しました． （nuopt-support@ml.msi.co.jp にご連絡ください）．[解出力なし]
8	(NUOPT 8) memory error in optimization phase. 計算部において所要メモリが使用可能な量をオーバーしました．[解出力なし]
9	(NUOPT 9) step reduction limit exceeded. 直線探索アルゴリズムにおいて step reduction の失敗が起き，最適化実行が止まってしまいました（凸でない問題に直線探索アルゴリズムを適用した場合や，問題が infeasible である場合に起きます）．[解出力あり]
10	(NUOPT 10) IPM iteration limit exceeded. 内点法の反復回数が上限を越えました（上限は特に指定しない場合には 150 回です．求解オプションファイルからは criteria:maxitn = 300 として設定することができます）．[解出力あり] 内点法の収束状況が悪化しており，反復回数が上限を越えた場合に本エラーメッセージが出力されます． このような現象は，問題のスケールの悪さなど，数値的な問題に起因することが多いことが経験上知られています． この問題に対して万能な解決策を挙げることは難しいのですが，以下の方策により回避できることがあります． 初期点を変更する 自動スケーリング機能を off にする options.scaling = "off"; 反復回数上限を上げる（例えば，デフォルトで 150 であれば 300 にする） options.maxitn = 300;
11	(NUOPT 11) infeasible. 問題が実行不可能であると判定されました．[解出力あり]

エラー番号	エラーメッセージ 説明
12	(NUOPT 12) heap memory for NUOPT process exhausted.
13	(NUOPT 13) unbounded.
	問題の最適解が有界でないことが判定されました．[解出力あり]
14	(NUOPT 14) integrality is violated.
	整数変数を含む問題に単体法以外を適用したため，整数変数が整数となっていない解が出力されています．[解出力あり]
15	(NUOPT 15) 手法名 misapplied to 問題種類.
	非線形計画問題に単体法が適用されようとしています．[解出力なし]
16	(NUOPT 16) Infeasible MIP.
	混合整数計画問題に整数解が存在しないことがわかりました．[解出力あり]
17	(NUOPT 17) B&B node limit reached (with feas.sol.).
	分枝限定法において生成する部分問題数が上限を越えました．最適解である保証はありませんが，整数解は得られています．（branch:maxnod を設定した場合）．[解出力あり]
19	(NUOPT 19) B&B node limit reached (no feas.sol.).
	17 番と同じですが，整数解が得られていません．[解出力あり]
20	(NUOPT 20) Some subproblems remain unsolved in B&B (no feas.sol.).
	分枝限定法で数値的理由によりいくつかの部分問題が解かれずに残りました．実行可能解の出力はありません．スケーリングオプションを変更してもう一度お試しください．[解出力あり]
21	(NUOPT 21) B&B itr. timeout (with feas.sol.).
	整数計画法を解いている場合の Nuorium Optimizer の起動時間が上限を越えました．最適である保証はありませんが，整数解は得られています（crit:maxtim を設定した場合）．[解出力あり]
22	(NUOPT 22) B&B itr. timeout (no feas.sol.).
	21 番と同じですが，整数解が得られていません．[解出力あり]
23	(NUOPT 23) B&B objective reaches under the limit.
	分枝限定法において目的関数値が設定値に達しました．[解出力あり]
25	(NUOPT 25) Can’t open file in current directory [no ファイル種類 made]
	（警告）解ファイル（.sol ファイル等）のオープンに失敗したので解ファイルが出力されていません（計算は行われます）．[解出力なし]
27	(NUOPT 27) SIMPLEX iteration limit exceeded.
	単体法の反復回数の上限をオーバーしました．[解出力あり]
28	(NUOPT 28) higher-order method is only for LP.
	非線形計画問題に線形計画法専用の内点法が適用されようとしています．[解出力なし]

エラー番号	エラーメッセージ 説明
29	(NUOPT 29) iteration diverged.
	内点法が発散しました．ペナルティパラメータが増大しており，以下の可能性が考えられます. 実行不可能 実行不可能に近い状態 実行可能領域になかなか近づけない 制約想定が満たされていない [解出力あり]
30	(NUOPT 30) terminated by user.
	ユーザの指示により計算の中断を行いました．[解出力あり]
31	(NUOPT 31) B&B terminated by user (with feas.sol.).
	分枝限定法の演算中ユーザの指示により計算の中断を行いました．最適解である保証はありませんが，整数解は得られています．[解出力あり]
32	(NUOPT 32) B&B terminated by user (no feas.sol.).
	31 番と同じですが，整数解が得られていません．[解出力あり]
33	(NUOPT 33) Bound violated.
	変数の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
34	(NUOPT 34) Bound and Constraint violated.
	変数および制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
35,36	(NUOPT 35) Constraint violated. (NUOPT 36) Equality constraint violated.
	制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
37	(NUOPT 37) B&B terminated with given # of feas.sol.
	options.maxintsol で指定した数だけの整数解が発見されたので分枝限定法を停止しました．[解出力あり]

エラー番号	エラーメッセージ 説明
38	(NUOPT 38) dual infeasible. リスタート時に起動される双対単体法のプロセスで実行不可能性が検出されました．[解出力あり]
39	(NUOPT 39) IPM iteration timeout. options.maxtim で設定した時間に対して，内点法の反復がタイムアウトしました．[解出力あり]
40	(NUOPT 40) SQP iteration limit exceeded. SQP（lsqp,tsqp）の反復が上限を超えました．他のアルゴリズムをお試しください．[解出力あり]
41	(NUOPT 41) SQP internal error. SQP（lsqp,tsqp）の実行中に内部的なエラーが発生しました．他のアルゴリズムをお試しください．[解出力なし]
42	(NUOPT 42) You cannot apply crossover for MIP
43	(NUOPT 43) B&B memory error (with feas.sol.). 分枝限定法の実行中に，options.maxmem で設定したメモリオーバーが起こり，実行を停止しましたが，実行可能解は得られています．[解出力あり]
44	(NUOPT 44) B&B memory error (no feas.sol.). 分枝限定法の実行中に，options.maxmem で設定したメモリオーバーが起こり，実行を停止しました．実行可能解は得られていません [解出なし]
45	(NUOPT 45) B&B gap reaches under the limit. 上下界の差が options.gaptol または options.relgaptol で与えられたよりも小さくなりましたので，分枝限定法を停止します．[解出力あり]
47	(NUOPT 47) IntegerVariable 変数名 should be declared as "binary". 0-1 整数変数以外の整数変数が表れています．wcsp は一般の整数変数を扱うことができません．[解出力なし]
48	(NUOPT 48) Variable 変数名 appear in two selection (). 二つ以上の selection にまたがって現れている 0-1 整数変数が表れました．[解出力なし]
49	(NUOPT 49) Variable 変数名 is fixed to infeasible value. 0-1 整数変数が 0，1 以外の値に固定されました．[解出力なし]
50	(NUOPT 50) Both of two variables 変数名 1 and 変数名 2 cannot be 1. 変数名 1 と変数名 2 の両方は（単一の `selection` に現れているので）両方 1 になることができません．[解出力なし]
51	(NUOPT 51) 手法名 is currently not available without SIMPLE. wcsp は SIMPLE によって定義された問題に対してのみ有効です．[解出力なし]

エラー番号	エラーメッセージ 説明
52	(NUOPT 52) All variables in selection statement#数字 fixed to 0. 対応する変数がすべて 0 に固定されているような selection() 文が「数字」番目に現れました．[解出力なし]
53	(NUOPT 53) Constraint 制約式名's weight is 値 should be -1 or non-negative value. 重みとしては-1 もしくは非負の数を与えねばなりません．[解出力なし]
54	(NUOPT 54) Constraint 制約式名's weight is 値 should be -1 or non-negative value. 重みとしては-1 もしくは非負の数を与えねばなりません．[解出力なし]
57	(NUOPT 57) You cannot use any method but "rcpsp" for model with Activity. Activity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
58	(NUOPT 58) You cannot use any method but "rcpsp" for model with ResourceRequire. ResourceRequire の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
59	(NUOPT 59) You cannot use any method but "rcpsp" for model with ResourceCapacity. ResourceCapacity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
60	(NUOPT 60) You cannot use any method but "rcpsp" for model with tardiness and completionTime. 目的関数が tardiness/completionTime に設定されているのにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
63	(NUOPT 63) You cannot use Variable for "rcpsp" . rcpsp は Variable は用いる事が出来ません．[解出力なし]
64	(NUOPT 64) You cannot use IntegerVariable for "rcpsp" . rcpsp は IntegerVariable は用いる事が出来ません．[解出力なし]
65	(NUOPT 65) failed to generate any initial solution rcpsp において初期解生成に失敗しました．[解出力なし]
66	(NUOPT 66) can't find feasible solution. rcpsp において実行可能解を得る事が出来ませんでした.[解出力なし]
67	(NUOPT 67) DiscreteVariable （変数名）'s bound: [a,b] and domain: {..} conflicts. (Regard Bound as a definition). DiscreteVariable の定義された範囲では満たすことのできない上下限があたえられました．[解出力なし]
70	(NUOPT 70) initial order is invalid between XX and YY . activities related imprecedene have to continue. rcpsp において初期解として不正な順番が与えられました. 直前先行制約に関係する Activity の順番は連続していなければなりません．[解出力なし]

エラー番号	エラーメッセージ 説明
71	(NUOPT 71) initial order is invalid. XX's order have to be previous to YY by way of precedence.
	rcpsp において初期解として不正な順番が与えられました．先行関係がある `Activity` はその順番が守られなければなりません．[解出力なし]
72	(NUOPT 72) infeasible MIP (preprocess).
	前処理において実行可能解が無いと判定しました.
73	(NUOPT 73) Continuous Variable 変数名 cannot be included in model for wcsp.
	連続変数を含む問題を wcsp で解こうとしました．wcsp は連続変数を扱うことができません．[解出力なし]
81	(NUOPT 81) You cannot use any method but "wcsp" for model with DiscreteVariable.
	`DiscreteVariable` を含む問題に wcsp 以外の方法は適用できません．[解出力なし]
82	(NUOPT 82) Trust region too small
	関数の二次近似に失敗しつづけて信頼領域が小さくなりすぎましたので実行を停止します．[解出力あり]
83	(NUOPT 83) Some subproblems remain unsolved in B&B (with feas.sol.).
	分枝限定法で数値的理由によりいくつかの部分問題が解かれずに残りました．実行可能解が出力されます．[解出力あり]
100	(NUOPT 100) Cannot open NUOPT License file: "XX".
	UNIX/Linux 版のライセンスファイルを開くことができません.
101	(NUOPT 101) Invalid License file: "XX".
	UNIX/Linux 版のライセンスファイルの内容に問題があります.
102	(NUOPT 102) Machine key(YY) not consistent to License file: "XX".
	UNIX/Linux 版のライセンスファイルがご利用のマシンと整合していません.
103	(NUOPT 103) License expired on XX days ago
	試用版の有効期間が終了しています.
104	(NUOPT 104) Invalid License limit
105	(NUOPT 105) Cannot get license information check machine configuration.
110	(NUOPT 110) Internal error (dpotrf failed).
	正定値で無い行列が与えられ，コレスキー分解に失敗しました．[解出力なし]
111	(NUOPT 111) Internal error (dpotrf failed).
	メリット関数計算時において，正定値でない行列が与えられ，コレスキー分解に失敗しました．[解出力なし]

エラー番号	エラーメッセージ 説明
112	(NUOPT 112) Internal error (dpotrs failed). 逆行列を求める計算に失敗しました．[解出力なし]
113	(NUOPT 113) Internal error (dsygst failed). 一般化固有値問題を標準固有値問題に変換する事ができませんでした．[解出力なし]
114	(NUOPT 114) Internal error (dsytrd failed). 三重対角化に失敗しました．[解出力なし]
115	(NUOPT 115) Internal error (dstebz failed). 最小固有値の導出に失敗しました．[解出力なし]
120	(NUOPT 120) Primal-dual gap is too large. 主双対ギャップが十分小さくならない内に，エラーが発生し計算を終了致しました．主問題・双対問題の実行可能解は満たされています．[解出力あり]
121	(NUOPT 121) Primal-dual gap is too large. 主双対ギャップが十分小さくならない内に，エラーが発生し計算を終了致しました．主問題の実行可能解は満たされています．[解出力あり]
122	(NUOPT 122) minus stepsize detected 正であるべきステップサイズに負の値が検出されました．解法を tipm に変更してお試しください．[解出力なし]
123	(NUOPT 123) The SDP constraint cannot be treated by specified algorithm. 半正定値制約が存在しますが，半正定値制約を解釈できるアルゴリズムが選択されていません．[解出力なし]
124	(NUOPT 124) No SDP constraint is detected. 半正定値計画法用のアルゴリズムが起動されましたが，半正定値制約が存在しません．[解出力なし]
132	(NUOPT 132) overflow wcsp の target や hard penalty の概算値，soft penalty の概算値が INT_MAX を超えてしまったことを表します．また変数の上下限が INT_MAX を超えていた際もこのエラーが出力されます.
133	(NUOPT 133) no feasible solution found wcsp が求解を行ったが制約を満たす解が得られなかったことを表します.
134	(NUOPT 134) hard penalty overflow wcsp 求解後の hard penalty が overflow していることを表します．正しく求解が行われている保証はありません.
135	(NUOPT 135) soft penalty overflow wcsp 求解後の soft penalty が overflow していることを表します．正しく求解が行われている保証はありません.

エラー番号	エラーメッセージ 説明
141	(NUOPT 141) mtxfree parameter error
	mtxfree のための求解オプションの設定に誤りがあります．nuopt.prm をご確認ください.
142	(NUOPT 142) mtxfree option is valid only for higher-order method
	mtxfree は higher order でのみ使うことができます.
143	(NUOPT 143) mtxfree failed to solve linear equations
	反復法で連立一次方程式が解けませんでした．メモリが十分にある場合は mtxfree を使用せずに求解してください．いくつかの求解オプションを調整することで解けるようになるかもしれません．詳細は nuopt-support@ml.msi.co.jp にお問い合わせください.
144	(NUOPT 144) mtxfree cannot deal with free variables
	`mtxfree` では上限も下限も存在しない変数に対応していません．モデルを調整するか, `mtxfree` を使用せずに求解してください.
150	(NUOPT 150) asqp misapplied to nonconvex QP
	非凸二次計画問題に有効制約法が適用されようとしています．目的関数の二次項に対応する対称行列が半正定値ではありません．モデルと入力データを確認してください. 例えば二次の目的関数 `x*x + y*y` は凸関数で，対応する対称行列が半正定値です．一方で `x*x - 2*y*y` は半正定値ではありません．非凸二次計画問題を解く場合は信頼領域内点法 `tipm` といった別の解法を指定してください．整数変数を含む非凸二次計画問題を解く場合は問題を線形化して単体法ベースの分枝限定法 `simplex` を指定してください．整数変数のみの場合は解法 `wcsp/wls` も対応可能です.
151-165	（例）(NUOPT 156) irowA[0] = 9 should be in [1,3]
	`solveLP` や `solveQP` 実行時に与えた引数に問題があります．詳しくは「Nuorium Optimizer/SIMPLE 外部接続マニュアル」をご覧ください.
166	(NUOPT 166) Access error in calling "XX(YY)". Argument should be in [0,ZZ] (index for variable)
167	(NUOPT 167) Access error in calling "XX(YY)". Argument should be in [0,ZZ] (index for constraint)
168	(NUOPT 168) Invalid [int/double/string] value 求解オプション値 for nuoptPrm options. 求解オプション名 .
	options. 求解オプション名 に誤った値が設定されています.
171	(NUOPT 171) upper or lower bound of integer variable is too large.
	整数変数の上限値もしくは下限値の絶対値が INT_MAX（int 型の最大値）を超えています.
172	(NUOPT 172) 手法名 is currently not available with SIMPLE.
	[手法名] は SIMPLE では利用することができません．[解出力なし]

エラー番号	エラーメッセージ / 説明
175	(NUOPT 175) Continuous/unbounded variable cannot be included in model for wls.
	解法 WLS は連続変数，あるいは非有界の変数を含めることはできません．[解出力なし]
176	(NUOPT 176) SemiHard constraint cannot be included in model for wls.
	解法 WLS はセミハード制約を含めることができません．[解出力なし]
177	(NUOPT 177) Nonlinear constraints cannot be included in model for wls.
	解法 WLS は非線形制約を含めることができません．[解出力なし]
180	(NUOPT 180) Continuous Variable 変数名 cannot be included in model for wls.
	wls による求解時に連続変数が含まれている時に出力されます．[解出力なし]
182	(NUOPT 182) no feasible solution found
	wls による求解時に制約を満たす解が得られなかったことを表します．[解出力なし]
190	(NUOPT 190) SIMPLEX time limit exceeded.
	単体法が計算時間上限を超過しました．[解出力あり]
191	(NUOPT 191) dual unbounded(infeasible or unbounded).
	実行不可能あるいは非有界となりました．[解出力あり]
193	(NUOPT 193) hsimplex misapplied to QCQP.
	解法 hsimplex は QCQP に適用することができません．[解出力なし]

A.1.2 求解オプションのエラー/警告メッセージ

nuopt.prm または options を用いた Nuorium Optimizer の求解オプション設定のエラーです.

エラー番号	エラーメッセージ / 説明
1	(SOLVER OPTION 1) Syntax error in solver option file.
	求解オプションファイルの記述にエラーが検出されました.
2	(SOLVER OPTION 2) Solver option file is empty.
	求解オプションファイルが全く空になっています.
4	(SOLVER OPTION 4) Internal error [内部ルーチン名]
	求解オプションの解釈を行うルーチンにて内部エラーが発生しました（nuopt-support@ml.msi.co.jp にご連絡ください）.

求解オプションファイルの記述にエラーが発生した場合（エラー番号 1）には求解オプションファイル読み込み部から標準出力に補助的なメッセージが表示されますので，併せて参考にしてください.

エラーのある求解オプションファイル：

```
begin
maximize
```

```
method:tipm
criteria:eps = 1.0e-8
```

標準出力：

```
<reading solver option file: nuopt.prm >
nuopt.prm:1:     begin
nuopt.prm:2:     maximize
nuopt.prm:3:     method:tipm
nuopt.prm:4:error: Unknown category     criteria:eps = 1.0e-8（行名が違う）
nuopt.prm:5:error: end command is needed.                    （end で終わっていない）
(SOLVER OPTION 1) Syntax error in solver option file.
```

A.1.3 MPS ファイルのエラー/警告メッセージ

エラー番号	エラーメッセージ 説明
1	(MPS FILE 1) Failed to open mps file: ファイル名. MPS ファイルのオープンに失敗しました.
2	(MPS FILE 2) Undefined row name: 行名. COLUMNS/RHS/RANGES セクションに未定義の行が現れました.
3	(MPS FILE 3) Internal error. 内部エラーが発生しました. （nuopt-support@ml.msi.co.jp にご連絡ください.）
4	(MPS FILE 4) Syntax error in セクション名 section. 「セクション名」の示すセクションで文法エラーが発生しました.
5	(MPS FILE 5) Too many 'INTORG' markers. 'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません.（'INTORG' マーカー行が多すぎます.）
6	(MPS FILE 6) Too many 'INTEND' markers. 'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません.（'INTEND' マーカー行が多すぎます.）
7	(MPS FILE 7) Unknown marker: フィールド 3 の内容 'INTORG','INTEND' 以外のマーカー行が現れました. （Nuorium Optimizer の MPS ファイル解釈部は'INTORG','INTEND' 以外のマーカー行を解釈しません.）

エラー番号	エラーメッセージ 説明
8	(MPS FILE 8) Undefined row: 行名 in HESSIAN section. HESSIAN セクションの二次の項を付加する行名として，定義されていないものが現れました.
9	(MPS FILE 9) Undefined column: 変数名 in HESSIAN section. HESSIAN セクションの非零要素の場所を示す際の変数名として，定義されていないものが現れました.
10	(MPS FILE 10) row: 行名 appeared more than once. ROWS セクションに同一の行名が二度以上現れました.
11	(MPS FILE 11) Specified bound: BOUND データラベル not found. 求解オプションファイルで指定された（mpsfile: bound = BOUND データラベル）ラベルを持つ BOUNDS データが MPS ファイルには存在しません.
12	(MPS FILE 12) Specified objective: 目的関数行名 not found. 求解オプションファイルで指定された（mpsfile: objective = 目的関数行名）名前を持つ目的関数行が MPS ファイルには存在しません.
13	(MPS FILE 13) Specified rhs: RHS データラベル not found. 求解オプションファイルで指定された（mpsfile: rhs = RHS データラベル）ラベルを持つ RHS データが MPS ファイルには存在しません.
14	(MPS FILE 14) Range data: RANGE データラベル contains unsuitable row. 「RANGE データラベル」を持つ RANGE データが目的関数行に対しての指定を行っています.
15	(MPS FILE 15) Specified range data: RANGE データラベル not found. 求解オプションファイルで指定された（mpsfile: range = RANGE データラベル）ラベルを持つ RANGE データが MPS ファイルには存在しません.
17	(MPS FILE 17) Undefined column name: 変数名 in INITIAL section. INITIAL セクションに定義されていない変数名が現れました.
18	(MPS FILE 18) Bound spec. on column : 変数名 should appear earlier. （警告）「変数名」に関する上下限設定の現れるのはより前でなければなりません.（最初に発見されたものに対してのみこのメッセージが出力されます.）
19	(MPS FILE 19) 数字 column(s) appeared disorderly in BOUNDS section （警告）BOUNDS section において「数字」個の変数の現れる順番が違法です.（エラー 18 と組になって出力されます.）
20	(MPS FILE 20) Memory allocation error. ファイルの読み込み時にメモリエラーが発生しました.
21	(MPS FILE 21) Undefined column name: 変数名 in BOUNDS section. BOUNDS section において定義されていない変数名が現れました.

エラー番号	エラーメッセージ 説明
22	(MPS FILE 22) Hessian is implicitly bound for hlinexistent objective. HESSIAN セクションでは暗黙のうちに（行名を指定せず）目的関数に対して二次の項が設定されていますが，MPS ファイルには全く目的関数が存在しません.
23	(MPS FILE 23) Same bound spec. on column: 変数名 appeared more than once. BOUNDS section で「変数名」に関して同一の制約指定が二度以上行われました.
24	(MPS FILE 24) Column : 変数名 has bound specification FX and other. BOUNDS section で「変数名」に関して FX 制約と他の制約指定が同時に行われました.
25	(MPS FILE 25) Column : 変数名 has bound specification FR and other. BOUNDS section で「変数名」に関して FR 制約と他の制約指定が同時に行われました.
26	(MPS FILE 26) Column : 変数名 has bound specification LO and MI. BOUNDS section で「変数名」に関して LO 制約と MI 制約指定が同時に行われました.
27	(MPS FILE 27) Column : 変数名 has bound specification UP and PL. BOUNDS section で「変数名」に関して UP 制約と PL 制約指定が同時に行われました.
28	(MPS FILE 28) Unknown bound specification フィールド 1 の内容 BOUNDS セクションに LO, LI, UP, UI, BV, PL, MI, FR, FX 以外のラベルが現れました.
29	(MPS FILE 29) row : 行名 appreaed more than once in セクション名 section. RHS セクションおよび RANGE セクションにおいて同一の行名が二度以上現れました.
30	(MPS FILE 30) Unsupported section. フィールド 1 の内容 未対応のセクション名が現れました.
31	(MPS FILE 31) Bound of column 列名 infeasible. 列名に対する境界条件が実行不可能です.
32	(MPS FILE 32) Invalid mps file. その他の理由で読み込むことができないファイルです.

A.1.4 LP ファイルのエラー/警告メッセージ

エラー番号	エラーメッセージ 説明
1	(LP FILE 1) Failted to open lp file : ファイル名. ファイルのオープンに失敗しました.
2	(LP FILE 2) Memory allocation error. ファイルの読み込み時にメモリエラーが発生しました.
3	(LP FILE 3) Internal error. 内部エラーが発生しました.（nuopt-support@ml.msi.co.jp にご連絡ください.）
4	(LP FILE 4) Syntax error. 構文エラーが見つかりました.

エラー番号	エラーメッセージ 説明
5	(LP FILE 5) Non-ascii char appeared. 非 ASCII 文字が現れました.
6	(LP FILE 6) The order of sections is wrong. セクションの記述順が間違っています.
7	(LP FILE 7) Variable 変数名 appeared more than once in 式名. 同一の変数が一つの式に二度以上現れました.
8	(LP FILE 8) Term 変数名 * 変数名 appeared more than once in 式名. 同一の項が一つの式に二度以上現れました.
9	(LP FILE 9) Undefined variable name : 変数名. 未定義の変数名が現れました.
10	(LP FILE 10) Lower/Upper bound of variable 変数名 appeared more than once. 同一の変数に対する境界条件が二度以上現れました.
11	(LP FILE 11) Bound of variable 変数名 is infeasible. 変数について矛盾した境界条件が与えられました.
12	(LP FILE 12) Length of name 名前... is too longer. 変数や式の名前が長すぎます.（255 文字までです.）
13	(LP FILE 13) セクション名 section unsupported. 未対応のセクション名が現れました.
14	(LP FILE 14) general/integer/binary section appeared more than once. 同一のセクションが二度以上現れました.
15	(LP FILE 15) Invalid lp-format. 対応できないファイルです.

付録 B Nuorium Optimizer アルゴリズム概説

B.1 内点法

B.1.1 問題

次の最適化問題

$$\begin{aligned} &\text{最小化} \quad f(x), && x \in R^n, \\ &\text{条　件} \quad g(x) = 0, && x \geq 0 \end{aligned}$$

を考えます[2]．ここで，変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで，関数 f は目的関数です．また，$g : R^n \to R^m$ は m 次元のベクトル値関数です．この問題のラグランジュ関数を

$$L(x, y, z) = f(x) - y^t g(x) - z^t x$$

としたとき，Karush-Kuhn-Tucker（KKT）条件（最適性の一次の必要条件）は次式で与えられます：

$$\begin{aligned} \nabla_x L(x, y, z) &= 0, \\ g(x) &= 0, \\ XZe &= 0, \quad x \geq 0, z \geq 0. \end{aligned}$$

ただし，$X = diag(x_1, \cdots, x_n) \in R^n, Z = diag(z_1, \cdots, z_n) \in R^n, e = (1, \cdots, 1)^t \in R^n$ です．ここで，相補性条件 $XZe = 0$ を $XZe = \mu e$（$\mu > 0$）で置き換えたものを修正 KKT 条件と呼びます．

Nuorium Optimizer ではバリヤペナルティ関数：

$$F(x, z) = f(x) - \mu \sum_{i=1}^{n} \log(x_i) + \rho \sum_{i=1}^{m} |g_i(x)| + \upsilon \left(x^T z - \mu \sum_{i=1}^{n} \log(x_i z_i) \right)$$

をメリット関数として採用します．ここで，$\mu > 0$ はバリヤパラメータ，$\rho > 0$ はペナルティパラメータ，$\upsilon > 0$ は主双対項の考慮度合いを表すパラメータです．

非線形最適化問題に対する内点法では，「修正 KKT 条件を満たす点を求めて，修正 KKT 条件を更新する」という作業を逐次行います．この際に，メリット関数 $F(x, z)$ の一次近似 $F_l(x, z)$ あるいは二次近似 $F_q(x, z)$ の変化量が重要な手がかりとなります[3]．次項以降で示すように，修正 KKT 条件を求める方法は複数存在します（直線探索を利用する方法・信頼領域を利用する方法）．

この作業を通して，最終的に元来の KKT 条件を満たす点を求めます．

---

[2] ここでは，説明の便宜上このような形式を考えますが，Nuorium Optimizer では制約関数に上下限が存在する場合，等式条件，変数に上下限が存在する場合などの一般形を扱うことができます．また，制約条件が存在しない問題も扱うことができます．

[3] 詳細は論文 [13][14][15] を参照

B.1.2 直線探索を利用する方法

修正 KKT 条件に対するニュートン法は次のようになります.

$$\begin{bmatrix} G + X^{-1}Z & -A^t \\ -A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_N \\ \Delta y_N \end{bmatrix} = \begin{bmatrix} -r_L - X^{-1}r_C \\ r_E \end{bmatrix},$$

$$\Delta z_N = -X^{-1}Z\Delta x_N - X^{-1}r_C.$$

ここで, $(\Delta x_N, \Delta y_N, \Delta z_N)$ は各変数に対する探索方向ベクトルで, G はラグランジュ関数のヘッセ行列あるいはその近似行列です. このとき,「ρ が十分大きく, G が半正定値行列である場合 $\Delta F_l(x, z, \Delta x_N, \Delta z_N) < 0$ である」という性質が成り立ちます.

この性質を利用して, 直線探索を利用して大域的に収束するアルゴリズムを構築することができます. 主双対変数に対するステップ幅 α は以下のように計算されます. まず, 次の三式から α の上限値 $\alpha_{\max}$ を求めます.

$$\alpha^x{}_{\max} = \min_i \left\{ \frac{-x_i}{(\Delta x_N)_i} \mid (\Delta x_N)_i < 0 \right\},$$

$$\alpha^z{}_{\max} = \min_i \left\{ \frac{-z_i}{(\Delta z_N)_i} \mid (\Delta z_N)_i < 0 \right\},$$

$$\alpha_{\max} = \min\{\alpha^x{}_{\max}, \alpha^z{}_{\max}\}$$

次に,

$$\alpha = \bar{\alpha}\beta^l, \quad \bar{\alpha} = \min\{\gamma\alpha_{\max}, 1\}$$

と定義します. ここで, $\gamma \in (0,1), \beta \in (0,1)$ です. そして, l は

$$F(x + \bar{\alpha}\beta^l\Delta x_N, z + \bar{\alpha}\beta^l\Delta z_N) - F(x,z) \leq \varepsilon_0\bar{\alpha}\beta^l\Delta F_l(x, z, \bar{\alpha}\beta^l\Delta x_N, \bar{\alpha}\beta^l\Delta z_N)$$

をみたす最小の正整数として定義されます[4]. $\varepsilon_0 \in (0,1)$ です.

このように, 各種パラメータを適切に設定すれば, メリット関数 $F(x,z)$ が確実に減少するような探索方向ベクトル $(\Delta x_N, \Delta y_N, \Delta z_N)$ 及びステップ幅 α を定める事ができます. ここからアルゴリズムの大域的収束性が保証されます.

ラグランジュ関数のヘッセ行列が非負定値となるのは

- 線形計画問題（ヘッセ行列は 0）
- 一般の凸計画問題

です. また, 一般の非線形計画問題ではヘッセ行列を準ニュートン法で近似することによって行列 G を常に正定値に保つことができます. 従って, このような場合に直線探索を利用した手法を使用することができます.

B.1.3 信頼領域を利用する方法

行列 G が非負定値でないとき直線探索法を利用することは困難です. そこで, G が不定であるとき

[4] この一連のステップ幅の設定ルールを Armijo's Rule と呼びます.

も利用できる方法として主双対変数に対する信頼領域法を採用します．このとき，探索方向ベクトル w，サイズ $\delta > 0$，ステップ幅 α は次のような関係を満たします．

$$\|w\| \leq \delta,$$
$$\alpha \leq \min\left\{\frac{\delta}{\|w\|}, \gamma\alpha_{\max}\right\}$$

$\alpha_{\max}$ の導出方法は「直線探索を利用する方法」同様です．信頼領域のサイズ調整は通常の方法で行います．

大域的収束性を得るために基準となる最急降下方向ベクトル $(\Delta x_{SD}, \Delta y_{SD}, \Delta z_{SD})$ を

$$\begin{bmatrix} D + X^{-1}Z & -A^t \\ -A & 0 \end{bmatrix}\begin{bmatrix} \Delta x_{SD} \\ \Delta y_{SD} \end{bmatrix} = \begin{bmatrix} -r_L - X^{-1}r_C \\ r_E \end{bmatrix},$$
$$\Delta z_{SD} = -X^{-1}Z\Delta x_{SD} - X^{-1}r_C,$$

によって定義します．ここで $D \succ 0$ は対角行列です．α^* を方向 Δ_{SD} に沿って信頼領域で与えられる区間内で，メリット関数変化量の二次近似 $\Delta F_q(x, z, \Delta x, \Delta z)$ を最小化するステップ幅として定義します．

$$\alpha^* = \arg\min\{\Delta F_q(x, z, \alpha\Delta x_{SD}, \alpha\Delta z_{SD}) \mid \|\alpha(\Delta x_{SD} + \Delta z_{SD})\| \leq \delta, \alpha \in [0, \bar{\alpha}]\}$$
$$\bar{\alpha} = \min\{1, \gamma\alpha_{\max}\}, \gamma \in (0, 1),$$

信頼領域のステップ幅 α は以下の条件をみたすように設定します．

$$\Delta F_q(x, z, \alpha\Delta x, \alpha\Delta z) \leq \frac{1}{2}\Delta F_q(x, z, \alpha^*\Delta x, \alpha^*\Delta z) < 0,$$
$$\|\Delta x_{Nk}\| \leq M\|\Delta x_{SDk}\|,$$
$$\|\Delta z_{Nk}\| \leq M\|\Delta z_{SDk}\|$$

「信頼領域を利用する方法」では探索方向ベクトル $(\Delta x, \Delta z)$ は

$$\begin{pmatrix} \Delta x \\ \Delta z \end{pmatrix} = \nu\begin{pmatrix} \Delta x_{SD} \\ \Delta z_{SD} \end{pmatrix} + (1 - \nu)\begin{pmatrix} \Delta x_N \\ \Delta z_N \end{pmatrix},$$

として計算されます．ここで，パラメータ $\nu \in [0, 1]$ は $\nu = 0, 0.1, 0.2, \cdots, 0.9, 1.0$ の中で条件：

$$\Delta F_q(x, z, \alpha\Delta x, \alpha\Delta z) \leq \frac{1}{2}\Delta F_q(x, z, \alpha^*\Delta x, \alpha^*\Delta z) < 0$$

をみたす最小の数です．このように，「信頼領域を利用する方法」では探索方向ベクトルの設定に異なる二方向 $(\Delta x_{SD}, \Delta z_{SD})$，$(\Delta x_N, \Delta z_N)$ を利用します．この結果，大域的収束性が保証されます．

B.1.4 線形計画問題専用内点法

問題が線形の場合，KKT 条件

$$\nabla_x L(x, y, z) = 0,$$

$$g(x) = 0,$$
$$XZe = 0 \tag{1}$$

の第 1 式，第 2 式は線形であり，非線形なのは第 3 式のみとなります．この方程式系に関するステップ幅 1 のニュートン法を適用すると線形な式の残差は原理的に零になり，非線形な第 3 式のみに

$$\Delta X_N \cdot \Delta Z_N e$$

なる形の残差が現れます（ΔX_N，ΔZ_N はニュートン法のステップ方向を対角に並べた行列）．この式にこの残差が発生することを見越してニュートン法のステップ方向を修正する（高次方向（Higher Order）の修正を加える）ことによって，より良いステップ方向を得ることができます．ニュートン法のステップ方向修正分を計算するためにはニュートン法のステップ方向の計算時に得られる副産物を有効に利用できるため，少ない計算コストでニュートン方向を改善することができ，計算を効率化することができます．

Nuorium Optimizer にはこのことを利用し，さらに問題が線形であることに特化したチューニングを加えた手法が組み込まれています．

B.1.5 半正定値計画問題専用内点法

次の最適化問題

$$\begin{array}{lll} \text{最小化} & f(x) & x \in R^n, X \in S^p \\ \text{条件} & g(x) = 0, X_0(x) = X, & x \geq 0, X \geq 0 \end{array}$$

を考えます．ここで, 変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで, 関数 f は目的関数です．また, $g : R^n \to R^m$ は m 次元のベクトル値関数です．変数 X は p 次元対称正方行列で，$X \geq 0$ は行列 X の半正定値制約を意味するものとします．$X_0 : R^n \to S^p$ は n 次元ベクトルを対称正方行列空間に写す写像です．問題が線形の場合，X_0 は $X_0(x) = \sum_{i=1}^{n} A_i x_i - B$ と表現しても構いません．

この問題のラグランジュ関数を $w = (y, Y, Z)$ として

$$L(w) = f(x) - y^T g(x) - \langle X_0(x) - X, Y \rangle - \langle X, Z \rangle$$

としたとき，Karush-Kuhn-Tucker(KKT) 条件（最適性の一次の必要条件）は次式で与えられます：

$$\nabla_x L(w) = \nabla f(x) - \nabla g(x)^T \Delta y - A^*(x)Z = 0$$
$$\nabla_X L(w) = Y - Z = 0$$
$$g(x) = 0$$
$$X_0(x) - X = 0$$
$$X \circ Z = 0, X \geq 0, Z \geq 0$$

ここでは，$A_i(x) = \dfrac{\partial X_0(x)}{\partial x_i}$，$A^*(x)Z = \begin{pmatrix} \langle A_1, Z \rangle \\ \cdots \\ \langle A_n, Z \rangle \end{pmatrix}$，$X \circ Z = \dfrac{1}{2}(XZ + ZX)$ であるものとします．

Nuorium Optimizer では，この KKT 条件を満たす点を，Newton 法を利用して反復解法で求めます．

線形な半正定値計画問題の場合，大域的収束性は保証されますが，そうでない問題を扱う場合，大域的収束を保証するには，メリット関数を定義する必要があります．

非線形半正定値計画問題を扱う場合，バリヤペナルティ関数：

$$F(x,X,Z) = F_{BP}(x,X) + \nu F_{PD}(x,X,Z)$$

$$F_{BP}(x,X) = f(x) - \mu \log(\det X) + \rho\|g(x)\|_1 + \rho'\|X_0(x) - X\|_1$$

$$F_{PD}(X,Z) = \langle X,Z\rangle - \mu \log(\det X \det Z)$$

をメリット関数として採用します．ここで，$\mu > 0$ はバリヤパラメータ，$\rho, \rho' > 0$ はペナルティパラメータ，$\nu > 0$ は主双対項の度合いを表すパラメータです．

探索方向を求める Newton 方程式は以下のように記述されます．

$$\begin{pmatrix} G+H & -\nabla g(x) \\ -\nabla g(x)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} \nabla f(x) - \nabla g(x)^T y - \mu A^*(x) X^{-1} + A^*(x)(ZX_0X^{-1} - Z) \\ g(x) \end{pmatrix}$$

$$\Delta X = X_0(x + \Delta x) - X$$

$$\Delta Z = \mu X^{-1} - Z - \frac{1}{2}(Z\Delta X X^{-1} + X^{-1}\Delta X Z)$$

ここでは，$H_{ij} = \langle A_i, X^{-1}A_j Z\rangle$ であるものとします．A_i, A_j の構造に応じて H_{ij} を計算する方法は異なります．

修正 KKT 条件を満たす点を逐次求め反復するという枠組み自体は，一般の非線形用内点法と同等です．Newton 方程式を解く際には，探索方向を対称化するため，KSH 方向へのスケーリングを行っています．

問題が非凸な場合，大域的収束性を確保するための工夫が必要になります．Nuorium Optimizer では信頼領域法に基づく方法を利用する場合，大域的収束を保証する方向と，局所的収束に有利な方向を混ぜ合わせて探索方向を決定します．

B.2 単体法・有効制約法

単体法は線形最適化問題

$$\begin{aligned} &\text{最小化} \quad c^t x && x \in R^n \\ &\text{条　件} \quad b_U \geq Ax \geq b_L \end{aligned}$$

に対して，有効制約法は二次計画問題

$$\begin{aligned} &\text{最小化} \quad \frac{1}{2}x^t Q x + c^t x && x \in R^n \\ &\text{条　件} \quad b_U \geq Ax \geq b_L \end{aligned}$$

に対して，それぞれ有効な方法です．一度可能基底解が得られれば，問題に対して小さな変更を行った際の解を比較的高速に求めることができるなど，内点法にはない特徴を備えています．

B.2.1 改訂単体法

Nuorium Optimizer に実装されている単体法は大規模問題用の改訂単体法と呼ばれる手法です．単体

法自身に関する解説は例えば [3] を参照してください.

B.2.2 有効制約法

Nuorium Optimizer に実装されている有効制約法は改訂単体法に基づいています．有効制約法に関する解説は例えば [12] を参照してください．この手法は 5 千変数以上の大規模問題では，一般に内点法（直線探索法（Line Search Method））に劣りますが,

- 変数に比べて制約式の数が非常に少ない（1/10 以下）場合
- 目的関数のヘッセ行列が密行列である場合

には内点法よりも高速かつ高精度です.

B.2.3 分枝限定法

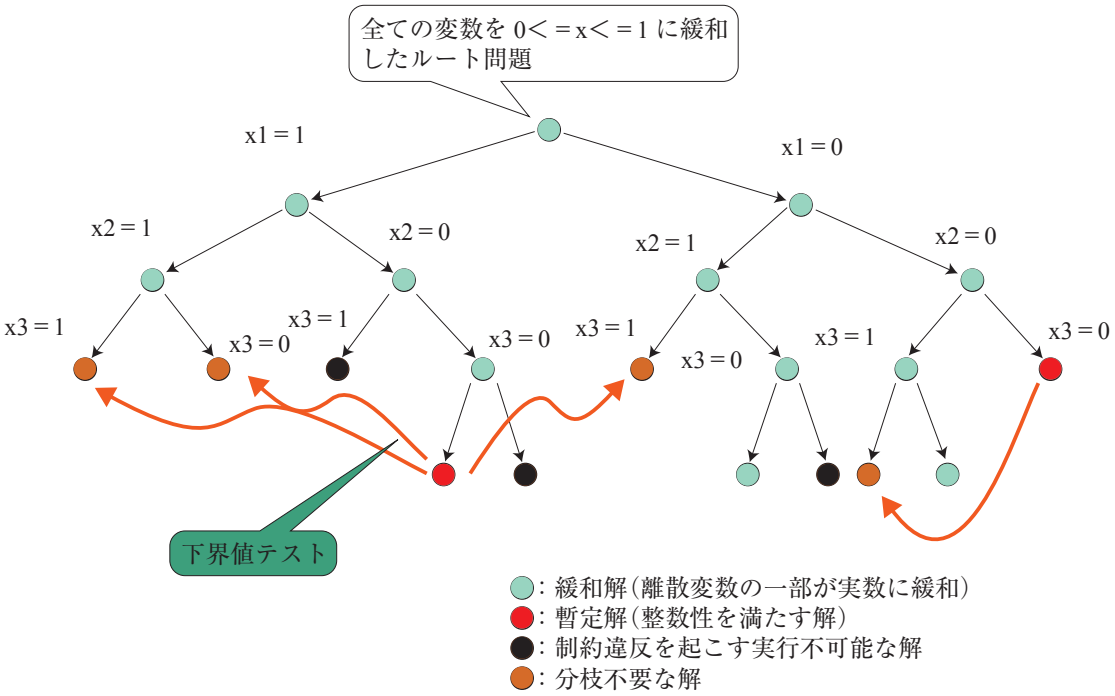
Nuorium Optimizer は，整数変数を含む線形/二次計画問題に対して以下のアルゴリズムが指定された場合，分枝限定法と呼ばれるアルゴリズムを用います.

- 単体法 （options.method = "simplex"）
- 有効制約法（options.method = "asqp"）

分枝限定法とは一般に元の問題の制約を一部緩和した問題（緩和問題）を繰り返し解くことにより厳密解を求めるアルゴリズムです．特に整数計画問題に適用される場合，緩和問題は整数変数を連続変数に緩和した問題（連続緩和問題）を考えます.

ここでは 0-1 整数変数を含む線形計画問題（目的関数を最小化する）の例を説明します．0-1 整数変数は連続緩和すると上限値が 1，下限値が 0 となる連続変数になることにご注意ください.

分枝限定法では緩和問題の解から整数になっていない変数を一つ選択し，0 または 1 に固定する，という操作（分枝操作）を行います．この過程は次の図のような木（分枝木）の形に表すことができ，整数制約を満たす解はこの木の「葉」の部分（図の下側の先端部分）に現れます.



しかしながら，分枝操作だけでは「葉」が整数変数の数に対して指数的に増える可能性があり，現実的な規模の問題に対して効率の良い解法となりません．このため分枝限定法は，分枝木の「分枝するごとに緩和問題の目的関数値が同じあるいは増大する」という性質を用いて枝の削減を行います（限定操作）．具体的には，分枝時に緩和解がその時点で知られている暫定解（整数性を満たす解）よりも目的関数値が大きければ，それ以上分枝する必要がありません．例えば緩和解の目的関数値が 11 で，暫定解として 10 が得られていればそれ以上分枝して探索する必要がありません（上の図の橙色の解に相当）．分枝限定法はこのような分枝操作と限定操作を繰り返し最適解を得るアルゴリズムです．

Nuorium Optimizer は更なる工夫により分枝限定法を高速化しています．

ヒューリスティクス

通常の分枝限定法では緩和解が整数性を満たす時のみ暫定解が得られます．Nuorium Optimizer ではそれに加え，実行可能解探索を別のロジックで行うことにより効率よく暫定解を求めます．例えばヒューリスティクス RINS では，緩和解と暫定解を比較し，共通して整数となっている変数は固定し，解が非共通の変数を非固定にした状態で分枝限定法を新たに解くことで，実行可能解を得ようとします．

切除平面

整数計画問題では，整数性を考慮することにより「切除平面」と呼ばれる制約式を生成できます．この制約式を元の制約式に付与すると緩和解が改善するため，分枝木を小さくできます．ただし緩和問題のサイズは大きくなるため，トレードオフの調整が必要です．

前処理

整数計画問題は，最初の緩和問題を解く前に「前処理」により問題変換あるいは問題削減を行います．例えば「検針（ probing ）」と呼ばれる操作は 0-1 変数を 0 あるいは 1 に固定しそこから実行不可能性が導かれるのであればどちらかに固定します．

これらの手法の具体的な求解オプションの設定方法については「整数計画法（simplex/asqp）に有効

な求解オプション」の項に解説があります.

B.2.4 並列分枝限定法

並列分枝限定法では「Supervisor」および「Worker」と呼ばれる役割が協調して処理を行います [17]. Supervisor は並列化動作を統制し，Worker は与えられたタスクを処理します.

Nuorium Optimizer は次の 3 つの並列分枝限定法を提供しています.

1. Racing（デフォルト）
2. Deterministic Racing
3. Subtree

Racing と Subtree は [18] を参考にしています.

Racing

Racing は複数の Worker が同一の整数計画問題を並行して解く手法です．各 Worker では異なる求解オプションが設定されています．同一の問題を解くため，冗長な計算を多く含みますが，デフォルトの求解オプションでは発見することが容易でない解を得ることができます．短時間で優良な解を得ることが重視される場合に有効な手法です．非決定性並列処理のため，同じ計算環境かつ同じ入力データであっても，異なる計算プロセスを経る可能性があります.

Deterministic Racing

Deterministic Racing は Racing に決定性を持たせた手法です．ここで言う決定性とは，同じ計算環境かつ同じ入力データであれば同じ結果が得られる（同じ計算プロセスを経る）ことを意味します.

デバッグやテストがしやすく，システムの計算エンジンとして利用するなど，より安定的に利用したい場合に有効です．注意点として決定性を持たせている分計算性能が犠牲になっているため，問題によっては通常の分枝限定法よりも遅くなるケースがあります．また，計算性能はスレッド数を多くするほど悪化する可能性があるため，適切なスレッド数を事前に調べておくことが重要です.

Subtree

Subtree は並列に分枝木を探索する手法です．Racing と比べると探索の効率が良く，小から中規模の問題で最適解が必要となる場合に有効です．注意点として大規模な問題（特に分枝限定法の前処理や単体法に時間がかかる問題）を本手法で解かせると，他の Worker に探索させる分枝木のルート問題の生成に時間がかかるため，通常の分枝限定法よりも遅くなる可能性があります．このような問題の場合は Subtree よりも Racing が推奨されます.

並列分枝限定法を実行する際の求解オプションについては「整数計画法（simplex/asqp）に有効な求解オプション」の項に解説があります.

B.3 逐次二次計画（SQP）法

逐次二次計画法では，次のような等式・不等式制約つきの非線形最適化問題の解を求めることがで

きます[5].

$$\begin{aligned} \text{目的関数} \quad & f(x) \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x) = 0, j \in J_E \\ & g_j(x) \geq 0, j \in J_I \end{aligned}$$

SQP 法とは，元の問題を現在の反復点において二次計画問題で近似し，その二次計画問題の解を探索方向としながら解を求める手法です．Nuorium Optimizer では三通りの SQP 法を用いることができます．以下，それらについて説明します．なお，説明で用いる k は反復の回数を表します．

B.3.1 準ニュートン法を用いる方法

本手法では，元の問題を次のような二次計画問題で近似します．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}\Delta x^T B_k \Delta x + \nabla f(x_k)^T \Delta x \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x = 0, j \in J_E \\ & g_j(x_k) + \nabla g_j(x_k)^T \Delta x \geq 0, j \in J_I \end{aligned}$$

ここで B_k は元の問題のラグランジュ関数のヘッセ行列を準ニュートン法によって近似した行列です．この問題の解 Δx を探索方向とし，直線探索を行って，次の反復点を定める方法となります．

B.3.2 信頼領域法を用いる方法

本手法では，二つの二次計画問題を解くことで点列を生成していきます．まず，一つ目の二次計画問題は次の通りです．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}(\Delta x_k^{SD})^T D_k \Delta x_k^{SD} + \nabla f(x_k)^T \Delta x_k^{SD} \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} = 0, j \in J_E \\ & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} \geq 0, j \in J_I \end{aligned}$$

ここで D_k とは，要素が正の値であるような対角行列とします．この問題の解 Δx_k^{SD} は，本手法に大域的収束性を与える上で大きな役割を果たします．

この問題の解を求めたとき，効いている制約の集合を J_A^k とします．すなわち

$$J_A^k = \{j \in J_E \cup J_I \mid g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} = 0\}$$

となります．このとき，もう一つの二次計画問題は次のように定められます．

$$\begin{aligned} \text{目的関数} \quad & \frac{1}{2}(\Delta x_k^N)^T G_k \Delta x_k^N + \nabla f(x_k)^T \Delta x_k^N \quad \rightarrow \quad \text{最小化} \\ \text{制約条件} \quad & g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^N = 0, j \in J_A^k \end{aligned}$$

ここで G_k は元の問題のラグランジュ関数のヘッセ行列です．この問題の解 Δx_N は，反復点が元の問題の解に近づいたときに速い収束をするための方向になっています．また，この問題の KKT 条件は，

[5]内点法の説明の箇所でも説明しましたが，ここに挙げた数理最適化問題の定式化はアルゴリズム説明のために挙げた一例であり，Nuorium Optimizer は変数の上下限制約などを含んだより一般的な問題を扱うことができます．

次のように線形方程式系として表すことができます.

$$\begin{pmatrix} G_k & -\nabla g_{J_A^k}(x_k) \\ \nabla g_{J_A^k}(x_k)^T & 0 \end{pmatrix} \begin{pmatrix} \Delta x_k^N \\ y^N_{k+1,J_A^k} \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -g_{J_A^k}(x_k) \end{pmatrix}$$

ここで，y^N_{k+1,J_A^k} はこの問題の制約条件に対するラグランジュ乗数とします．一般に，問題規模が同程度であれば，線形方程式系は二次計画問題に比べ速く解くことができるので，この問題に対する計算コストは，先程の Δx_{SD} を求める問題と比べて小さいものと考えることができます.

本手法では，Δx_{SD} と Δx_N の凸結合

$$\Delta x_k(\nu_k) = \nu_k \Delta x_k^{SD} + (1-\nu_k)\Delta x_k^N$$

を探索方向とし，定められた信頼領域の中を探索し，次の反復点を生成します.

B.4 制約充足問題ソルバ wcsp

このアルゴリズムは離散変数，0-1 整数変数を変数とした制約充足問題：

$$\begin{aligned} &\text{変数} \quad x_j \in X_j, && j = 1, \cdots, n \\ &\text{条件} \quad c_i^U \geq g_i(x_1, \cdots, x_j, \cdots, x_n) \geq c_i^L, && i = 1, \cdots, m \end{aligned}$$

をメタヒューリスティクスの解法の一つであるタブー・サーチを用いて解くものです．ここで，X_j は各変数の定義域に対応する有限集合です.

本アルゴリズムは，各制約の違反量の合計を最小化する問題を解きます．近似解法であるため，制約式をすべて満たす解が存在しない場合でもできるだけ制約を満たす解を得ることができます．各制約の違反量の合計を計算する際，各制約の違反量には「重み」と呼ばれる正の定数を掛けて合算します.

この際，重みの大きな制約式は優先的に満たされるように解の探索が行われます.

重要度の高い制約式の重みを大きくすることで，より有用な解が期待できます.

制約には重みを ∞ として設定することもできます．このように設定された制約は，何よりも優先して満たすべき制約として扱われます．重みが ∞ に設定されたものをハード制約，正の有限の値に設定されたものをソフト制約と呼びます.

制約の他に最大化，最小化すべき目的関数 $f(x_1, \cdots, x_j, \cdots, x_n)$ を定義することもできますが，その際には目標値 μ を定めて

- 最小化問題であれば $\mu - f(x) \geq 0$
- 最大化問題であれば $f(x) - \mu \geq 0$

というソフト制約を定義して目的関数を扱います．つまり，目的関数自身も，あくまで満たすべき制約式のひとつとして扱われるため，目的関数に対しても重みを設定する必要があります.

制約充足問題ソルバは，以下のいずれかの条件を満たせば停止します.

1. すべてのハード制約およびソフト制約を満たす解が求まった（目的関数に関しては目標値を満たす解が求まった）
2. 反復回数が指定の上限を超えた
3. 計算時間が指定の上限を超えた

4. ユーザが停止を命じた

本アルゴリズムは，京都大学「問題解決エンジン」グループの開発によるものです．詳細については [9], [10] をご参照ください．

B.5 タブー・サーチによる資源制約スケジューリング問題解法

このアルゴリズムは以下の資源制約付きスケジューリング問題:

- 限られた資源の下，仕事の処理に用いる資源の分配，作業の開始時刻を決定する

をタブー・サーチを用いたリストの探索を用いて解くものです．

アルゴリズムは wcsp と同じ京都大学「問題解決エンジン」グループによるものです．詳細については [16] をご覧ください．

B.6 重み付き局所探索法 WLS

このアルゴリズムは整数変数を変数とした最適化問題を局所探索により近似的に解くものです．目的関数が二次関数や線形関数である整数計画問題を扱うことができます．近似解法のため得られた解の最適性の保証はありません．

本アルゴリズムは各制約式に疑似的に「重み」をもたせ，制約違反量の合計を目的関数と合わせて最小化します．

WLS の対象となる問題を以下のように定義します．

$$
\begin{aligned}
\text{minimize} \quad & c^\top x + \frac{1}{2} x^\top Q x \\
\text{subject to} \quad & f_i(x) \le b_i, && i \in M_L, \\
& f_i(x) \ge b_i, && i \in M_G, \\
& f_i(x) = b_i, && i \in M_E, \\
& l_j \le x_j \le u_j, \ \ x_j \in \mathbb{Z}, && j \in N.
\end{aligned}
$$

ここで，$f_i(x)$ は線形式あるいは二次式を表します．

上の問題に対して重み w^+, w^- を導入し，以下のような緩和問題を考えます．

$$
\begin{aligned}
\text{minimize} \quad & c^\top x + \frac{1}{2} x^\top Q x + \sum_{i \in M_L \cup M_E} w_i^+ y_i^+ + \sum_{i \in M_G \cup M_E} w_i^- y_i^- \\
\text{subject to} \quad & f_i(x) - y_i^+ \le b_i, && i \in M_L, \\
& f_i(x) + y_i^- \ge b_i, && i \in M_G, \\
& f_i(x) - y_i^+ + y_i^- = b_i, && i \in M_E, \\
& l_j \le x_j \le u_j, \ \ x_j \in \mathbb{Z}, && j \in N, \\
& y_i^+ \ge 0, && i \in M_L \cup M_E, \\
& y_i^- \ge 0, && i \in M_G \cup M_E.
\end{aligned}
$$

アルゴリズムの各反復では，重み w^+, w^- を当該制約式の違反量が大きいものほど大きくなるよう内部で自動調整します．探索時は自動調整された重み w^+, w^- に基づき，上記緩和問題において目的関数

が減少する方向へ解が遷移します．これにより制約条件に違反している解から違反を解消する方向や目的関数値が減少する方向へ解が遷移し，より良い解が得られます．

本アルゴリズムは緩和問題に対して局所探索を行うため，制約式をすべて満たす解が存在しない場合においてもできるだけ制約を満たす解が得られます．

本アルゴリズムはソフト制約付きの最適化問題を扱えます．ここでソフト制約とは，制約違反に対して所与の重みに比例したペナルティがかかる制約のことです．ソフト制約に違反した解も実行可能解となります．したがってソフト制約は通常の制約に比べ満たすべき優先度が低く，ソフト制約同士であれば所与の重みが大きいほど優先度が高いと解釈できます．本アルゴリズムでは所与の重みを重み w^+, w^- の自動調整の際の上限値として扱うことで，上記の優先度に合わせた制約違反の解消を実現します．

類似アルゴリズムである制約充足問題ソルバ wcsp との関連に触れながら，技術的な詳細を簡単に説明します．WLS と wcsp はいずれも局所探索の性能向上のための工夫をもつ手法です．wcsp はタブー・サーチと呼ばれるアルゴリズムを用いることで得られる局所最適解の質の向上を狙います．一方で WLS は近傍操作の種類を増やすことで同様の効果を狙います．各近傍操作において wcsp は 1 つの離散変数の値しか変化させないのに対し，WLS は最大 4 つの整数変数の値を変化させます．wcsp が複数回の近傍操作をしないと得られないような解も WLS は一回の近傍操作で得られます．

WLS は近傍操作の計算時間の削減を「隣接リスト」というデータ構造を用いて実現します．ここで隣接リストとは，変数同士の関連度の推定値に基づき，強く関連する変数の組を保持するリストです．2 つ以上の整数変数の値を変化させる際にはこの隣接リストを用い，改善解を得る見込みが薄い近傍操作をスキップします．

本アルゴリズムは，係数が 1 である線形な制約式に対する高速化手法を取り入れています．したがってそのような制約式が多い最適化問題，特に大規模な集合被覆問題や集合分割問題に対して高い性能を発揮します．

本アルゴリズムの詳細については [19] をご参照ください．

付録 C 使い方に関するサポート

C.1 ユーザーサポートのページ

ユーザーサポートのページ（https://www.msi.co.jp/nuopt/user/index.html）にお客様からよく寄せられるご質問をまとめました．お問い合わせの前に，是非一度ご確認ください．

C.2 使い方サポートサービス

年間保守にご加入の方は，使い方サポートサービスをご利用いただけます．以下のページの「製品サポート」フォームからお問い合わせください．

https://reg34.smp.ne.jp/regist/is?SMPFORM=mgsb-ldrfmf-42ca1c7610237b14b4c1133097b27515

なお，データおよびプロジェクトファイルをお送りいただく場合には，いったんお送りいただく旨をフォームの通信欄にてお知らせいただければこちらよりセキュアなデータ転送サービスご利用についてご案内をいたします．

「製品サポート」フォームをご利用いただけない場合，下記アドレスに E-Mail でお問い合わせください．

nuopt-support@ml.msi.co.jp

E-Mail でのお問い合わせの際には下記を明記してください．

- ご利用の製品名
- バージョン
- シリアル ID
- ご登録者様のお名前
- ご質問事項

ご質問に関わるデータやプロジェクトファイルなどは，直接メール添付をしないようお願いいたします．（容量により，エラーとなる場合がございます．）

データおよびプロジェクトファイルをお送りいただく場合には，いったんお送りいただく旨を E-Mail にてお知らせいただければこちらよりセキュアなデータ転送サービスご利用についてご案内をいたします．

フォームおよび E-Mail でのお問い合わせについては，回答は一営業日以内に行います．もし回答がない場合，送信いただいた E-Mail がエラーとなっている等の場合があります．お手数ではございます

が，今一度，宛先のメールアドレス等をご確認ください．どうしても原因が分からない場合は，下記までお電話にてご連絡下さい．（使い方のご質問そのものは，お電話ではお受けしておりませんので，ご注意ください．）

（株）NTT データ数理システム　営業部　03-3358-6681

参考文献

[1] J. E. Beasley(ed.), Advances in Linear and Integer Programming, Oxford University Press, 1996.

[2] I. Bongartz, A. Conn, N. Gould and Ph. L. Toint, CUTE: Constrained and Unconstrained Testing Environment, Research Report RC 18860, IBM, T. J. Watson Research Center, Yorktown, U.S.A., 1993.

[3] V・フバータル著/阪田省二朗・藤野和建訳，線形計画法（上/下），啓学出版，1983.

[4] I. S. Duff and J. K. Reid, The Multifrontal solution of indefinite sparse symmetric linear systems, ACM Transaction on Mathematical Software, Vol.9,No.3 ,302-325,1983.

[5] P. E. Gill, W. Murray, M.A.Saunders and M.H.Wright, A practical anti-cycling procedure for linearly constrained optimization, Mathematical Programming, 45:437-474, 1989.

[6] P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright, Inertia-controlling methods for general quadratic programming, SIAM Review, 33:1-36, 1991.

[7] W. Hock and K. Shittkowski, Test examples for nonlinear programming codes, Springer Verlag, 1981.

[8] 田辺隆人，山下浩，主双対外点法とそのパラメトリック最適化への応用，2005 年日本オペレーションズ・リサーチ学会秋季研究発表会アブストラクト集 50-51.

[9] K. Nonobe and T. Ibaraki, A tabu search approach for the constraint satisfaction problem as a general problem solver, European Journal of Operational Research 106, 599-623, 1998.

[10] K. Nonobe and T. Ibaraki, An improved tabu search method for the weighted constraint satisfaction problem, INFOR 39, 131-151, 2001.

[11] 伊理正夫，今野浩，刀根薫監訳，最適化ハンドブック，朝倉書店，1995.

[12] 矢部博，八巻直一，非線形計画法，朝倉書店，1999.

[13] H. Yamashita, A globally convergent primal-dual interior point method for constrained optimization, Technical Report, Mathematical Systems Inc., Tokyo, Japan, April 1992 (revised May 1992).

[14] H. Yamashita and T. Tanabe, A primal-dual interior point trust region method for large scale constrained optimization, Technical Report, Mathematical Systems Inc., Tokyo, Japan, October 1993.

[15] H. Yamashita and H. Yabe, Superlinear and quadratic convergence of primal-dual interior point methods for constrained optimization, Technical Report, Mathematical Systems Institute Inc., Tokyo, Japan, June 1993.

[16] K. Nonobe and T. Ibaraki, Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: C.C. Ribeiro and P. Hansen (eds.): Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, pp.557-588, 2002.

[17] T. Ralphs, et al., Parallel solvers for mixed integer linear optimization, In Handbook of parallel constraint reasoning, Springer, 2018, pp. 283-336.

[18] Y. Shinano, et al., Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In: Parallel and Distributed Processing Symposium, 2016 IEEE International. IEEE. 2016, pp. 770-779.

[19] S. Umetani, Exploiting variable associations to configure efficient local search in large-scale set partitioning problems, Twenty-Ninth AAAI Conference on Artificial Intelligence, February 2015.

索　引

記号・数字

A

B

C

D

E

F

G

O

P

R

S

T

U

V

W

に

は

ひ

ふ

へ

ま

め

も

ゆ

ら