



Nuorium Optimizer

C++SIMPLEマニュアル V26

株式会社NTTデータ数理システム

2024年6月

目次

第1章	モデリング言語 C++SIMPLE とは	1
第2章	数理最適化問題を記述する (チュートリアル)	3
2.1	目的関数・変数・制約	3
2.2	定数	7
2.3	集合・添字	9
2.4	集約・複数の添字	15
2.5	式	18
2.6	整数変数	19
2.7	結果出力関数	19
2.8	デバッグ出力関数	20
2.9	C++SIMPLE を記述する際の注意点	21
第3章	C++SIMPLE 基本事項	23
3.1	利用方法	23
3.2	C++SIMPLE による数理最適化計算の流れ	23
3.2.1	Windows 版における手順	23
3.2.2	UNIX/Linux 版における手順	23
3.2.3	備考	24
第4章	C++SIMPLE 一般事項	25
4.1	基本的な内容	25
4.1.1	行末のセミコロン;	25
4.1.2	半角空白文字と改行	25
4.1.3	構成要素の順序	26
4.1.4	モデルファイル内で利用できない文字	26
4.1.5	name 引数に利用できない文字	26
4.1.6	データファイル内で利用できない文字	27
4.2	高度な内容	27
4.2.1	外部接続における C++オブジェクトの宣言と代入	27
4.2.2	標準出力の抑制	27
4.2.3	UNIX/Linux 版におけるデータファイルの文字コード指定について	28
4.2.4	変数の上下限	28

第 5 章 数理最適化モデルの構成要素	31
5.1 変数クラス Variable	32
5.2 目的関数クラス Objective	32
5.3 制約式クラス Constraint	33
5.4 定数クラス Parameter	35
5.4.1 double 型への変換	37
5.4.2 char* 型への変換	38
5.5 整数変数クラス IntegerVariable	38
5.6 範囲演算関数 sum, prod	39
5.7 対称行列クラス SymmetricMatrix	40
5.8 式クラス Expression	44
5.9 添字クラス Element	45
5.10 集合クラス Set	46
5.10.1 追加操作 add	50
5.10.2 自動代入の禁止 lock()/unlock()	50
5.10.3 切断操作 slice	51
5.11 順序集合クラス OrderedSet	52
5.12 数列集合 Sequence	54
5.13 条件式	55
5.14 最小（大）値取得関数 min, max	57
5.15 数学関数	58
第 6 章 制約充足問題ソルバ wcsp	59
6.1 wcsp を用いる場合の注意点	59
6.2 目的関数クラス Objective	60
6.3 制約式クラス Constraint	61
6.3.1 ハード制約関数 hardConstraint	61
6.3.2 セミハード制約関数 semiHardConstraint	61
6.3.3 ソフト制約関数 softConstraint	62
6.3.4 求解オプション defaultConstraintWeight	63
6.4 整数変数クラス IntegerVariable	63
6.5 離散変数クラス DiscreteVariable	64
6.6 重複不能関数 alldiff	64
6.7 選択関数 selection	65
6.8 ブール関数 Boolean	66
6.9 最小（大）値取得関数 min, max	66
6.10 カウント関数 count	68
第 7 章 資源制約付きスケジューリング問題ソルバ rcpsp	69

7.1	rcpsp の構成要素	69
7.2	目的関数クラス Objective	70
7.3	制約式クラス Constraint	71
7.4	アクティビティクラス Activity	71
7.4.1	先行制約, 直前先行制約	72
7.4.2	Activity の要素	73
7.4.3	初期値の設定	73
7.5	必要資源クラス ResourceRequire	74
7.6	資源供給量クラス ResourceCapacity	75
7.7	モード順序関数 modeOrder	76
7.8	アクティビティ固定関数 fixActivity	77
7.9	アクティビティ固定解除関数 unfixActivity	78
7.10	資源制約付きスケジューリング問題の重みの設定	78
7.11	資源制約付きスケジューリング問題記述例	78
第 8 章	データファイル	87
8.1	データファイルの機能	87
8.2	dat 形式データファイル	87
8.3	csv 形式データファイル	90
第 9 章	最適化計算制御	97
9.1	solve 関数	97
9.2	求解オプション options	98
9.3	最適化計算結果 result	99
9.4	可変定数	101
9.5	初期値設定 SimpleSetInitialValues()	103
第 10 章	出力制御	105
10.1	出力対象	105
10.1.1	制約式に対する .val	106
10.2	print 関数	107
10.3	simple_printf 関数	110
10.4	simple_fprintf 関数	116
10.5	モデルの内容の表示 (showSystem 関数)	118
10.6	解ファイル出力制御	120
10.7	Nuorium/Excel アドインへの出力制御	120
第 11 章	その他の機能	123
11.1	Intel oneAPI Math Kernel Library のリンク	123
11.2	MPS ファイルおよび LP ファイルへの変換	123

11.2.1 変換方法	123
11.2.2 MPS ファイルへの変換機能使用時の注意	124
第 12 章 Nuorium Optimizer/SIMPLE FAQ	125
12.1 浮動小数点エラー	125
12.2 整数の割り算	125
12.3 添字付けに関するエラー	126
12.3.1 一次元の場合	126
12.3.2 二次元の場合	126
12.3.3 三次元以上の場合	126
12.3.4 一時オブジェクトの利用の禁止	127
12.4 for 文内部における宣言	127
12.5 宣言の引数に SIMPLE オブジェクトを含んだ式を入れる	128
12.6 複数の代入の禁止	128
12.7 .val と定義域外エラー	128
12.8 両辺が定数で満たされない制約式	129
付録 A SIMPLE/mknuopt のエラー/警告メッセージ	131
A.1 SIMPLE のエラー/警告メッセージ	131
A.2 mknuopt のエラー/警告メッセージ	165
索 引	167

第 1 章

モデリング言語 C++SIMPLE とは

モデリング言語 C++SIMPLE は C++ベースの数値最適化専用モデリング言語です。線形計画／二次計画／混合整数計画／非線形計画問題等多様な数値最適化問題をシームレスにモデリングし、解く環境を提供します。

モデリング言語 C++SIMPLE を用いると数式に近い自然な形で問題記述ができるほか、自動微分機能により、大規模で複雑な非線形計画問題に対しても高階の微係数の情報を活かした精度の高いアルゴリズムを適用することが可能です。モデルとデータが分離されており、大規模問題を簡潔に記述することができます。

モデリング言語 C++SIMPLE は C++のクラスライブラリとして実現されており、制御ループや出力などについては C++固有の機能を生かすことができるほか、他の C / C++プログラムとのリンクも可能です。C++SIMPLE を用いるにあたって C++の知識が必須ではありませんが、一部の高度な機能は C++の理解を前提としています。

第2章

数理最適化問題を記述する（チュートリアル）

本章では、具体的な例題を通して、モデリング言語 C++SIMPLE の文法を紹介します。

2.1 目的関数・変数・制約

次のような生産計画問題を考えます。

2つの油田 X, Y が存在し、それぞれ一日あたり重油・ガスを次の量だけ生産する。

生産量/日		
	重油	ガス
X	6t	4t
Y	1t	6t

また、重油・ガスの週あたりの生産ノルマが、次のように定められている。

ノルマ/週	
重油	12t
ガス	24t

油田 X, Y の日あたりの運転コストは、次のとおりである。

運転コスト/日	
X	180
Y	160

油田 X, Y とともに、最大で週5日まで運転可能である。ノルマを満たしながら運転コストを最小化するためには、それぞれの油田を週あたり何日運転すれば良いだろうか？

この問題を定式化すると、以下のようになります。

変数

x	油田 X の運転日数/週
y	油田 Y の運転日数/週

目的関数（最小化）

$180x + 160y$	運転コスト/週
---------------	---------

制約条件

$6x + y \geq 12$	重油ノルマ/週
$4x + 6y \geq 24$	ガスノルマ/週
$0 \leq x \leq 5$	油田 X の週あたりの運転日数制約
$0 \leq y \leq 5$	油田 Y の週あたりの運転日数制約

それでは、この問題を C++SIMPLE で記述した例を見てみましょう。

```
// 油田 X, Y の運転日数/週（変数）
Variable x(name = "油田 X の運転日数");
Variable y(name = "油田 Y の運転日数");

// 運転コスト（目的関数）
Objective cost(name = "全運転コスト", type = minimize);
cost = 180 * x + 160 * y;

// 製品ノルマ
6 * x + y >= 12;    // 重油ノルマ/週
4 * x + 6 * y >= 24; // ガスノルマ/週

// 各油田の日数制約
0 <= x <= 5;    // 油田 X の週あたりの運転日数制約
0 <= y <= 5;    // 油田 Y の週あたりの運転日数制約

// 求解
solve();

// 結果出力
x.val.print();
y.val.print();
cost.val.print();
```

この C++SIMPLE による記述を上から順に見ていきましょう。

```
// 油田 X, Y の運転日数/週（変数）
Variable x(name = "油田 X の運転日数");
Variable y(name = "油田 Y の運転日数");
```

この部分は変数（油田の運転日数）の宣言です。モデル中で使用する変数は、使用する前に宣言す

る必要があります。name = "... "の部分には変数の名前を指定します。name = "... "は省略可能ですが、出力などで使用されますので、なるべく記述した方が良いでしょう。

"/"から行の終わりまではコメントです。

```
// 運転コスト (目的関数)
Objective cost(name = "全運転コスト", type = minimize);
```

この部分は目的関数（運転コスト）の宣言です。目的関数の内容を定義する前に、宣言する必要があります。name = "... "の部分には目的関数の名前を指定します。変数の宣言同様、name = "... "は省略可能ですが、出力などに利用されますので、なるべく記述した方が良いでしょう。type = minimize で目的関数が最小化されるべきことを指示します。type = maximize とすれば、目的関数を最大化します。

```
cost = 180 * x + 160 * y;
```

この部分は目的関数（運転コスト）の内容定義です。=の左辺に目的関数を、右辺に目的関数の内容を記述します。*は積、+は和を表す演算子です。C++SIMPLE では四則演算や数学関数 (exp(), sin()...)などを式の記述に用いることができます。

```
// 製品ノルマ
6 * x + y >= 12;      // 重油ノルマ/週
4 * x + 6 * y >= 24; // ガスノルマ/週
```

この部分では制約式（生産ノルマ）を定義しています。関係演算子>=の左辺、右辺には、任意の式を記述できます。目的関数の内容定義の際と同様に、任意の式の中に演算子や数学関数を記述できます。左辺と右辺の関係を表す関係演算子には、以下のものを指定できます。

C++SIMPLE の関係演算子	定式化時の記述
>=	≥
<=	≤
==	=

```
// 各油田の日数制約
0 <= x <= 5;      // 油田 x の週あたりの運転日数制約
0 <= y <= 5;      // 油田 y の週あたりの運転日数制約
```

この部分は制約式（運転日数の上下限）を定義しています。ここでは変数の上下限を指定していますが、C++SIMPLE では一般の制約式と変数の上下限制約を区別しませんので、x, y の部分に任意の式を書くことが可能です。

以上で、問題の定義の記述は完了です。

次に、これまでに定義した問題の最適解を求め、結果を出力する部分を記述します。

```
// 求解
solve();
```

`solve()` は、定義したモデルについて最適解の計算を行う関数です。 `solve()` は、必ずモデル記述の後に記述する必要があります。

```
// 結果出力
x.val.print();
y.val.print();
cost.val.print();
```

この部分は、最適化計算結果の出力を指定しています。最適化計算後の値を出力するためには、最適化計算 `solve()` の後に記述する必要があります。

以上でこのモデルについての C++SIMPLE の記述は終了です。

次にこのモデルを実行してみます（実行方法については「[3.1 利用方法](#)」を参照してください）。すると、数理最適化モデルを解く経過が、以下のように出力されます。

```
[Expand Constraints and Objectives]
sample.smp:7:info: 展開中 目的関数 (1/5) name="全運転コスト"
sample.smp:10:info: 展開中 制約式 (2/5) name=""
sample.smp:11:info: 展開中 制約式 (3/5) name=""
sample.smp:14:info: 展開中 制約式 (4/5) name=""
sample.smp:15:info: 展開中 制約式 (5/5) name=""

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

[Problem and Algorithm]
PROBLEM_NAME                sample
NUMBER_OF_VARIABLES         2
NUMBER_OF_FUNCTIONS         3
PROBLEM_TYPE                 MINIMIZATION
METHOD                       HIGHER_ORDER

[Progress]
<preprocess begin>.....<preprocess end>
```

```

<iteration begin>
    res=4.0e+001 .... 2.8e-005  1.4e-007
<iteration end>

[Result]
STATUS                                OPTIMAL
VALUE_OF_OBJECTIVE                    750.0000021
ITERATION_COUNT                       6
FUNC_EVAL_COUNT                       9
FACTORIZATION_COUNT                   7
RESIDUAL                             1.402395924e-007
ELAPSED_TIME(sec.)                    0.20

```

最後に結果出力に対応する結果が以下のように出力されます。

```

油田 X の運転日数=1.5
油田 Y の運転日数=3
全運転コスト=750

```

=の左辺は指定した変数と目的関数の名前で、name = "... "に記述したものが出力されます。右辺には変数と目的関数の値が出力されています。

2.2 定数

現在は、モデル中に油田運転コストの値を直接記述しています。これを変更し、外部から任意の値を与えてみましょう。まず、定式化を以下のように変更します。

目的関数

$\text{costX} \cdot x + \text{costY} \cdot y$	運転コスト/週
---	---------

定数

costX	油田 X の運転コスト/日
costY	油田 Y の運転コスト/日

costX, costY はそれぞれ油田 X, Y の運転コスト/日を表す定数です。C++SIMPLE では、このような定数を使用した記述が可能です。

ここでは、定数を用いて、運転コストを以下のように変更します。

```
cost = 180 * x + 160 * y;
```

↓

```
Parameter costX(name="油田 X の運転コスト");  
Parameter costY(name="油田 Y の運転コスト");  
cost = costX * x + costY * y;
```

まず、Parameter で定数を宣言します。モデル中で使用する定数は、使用する前に宣言する必要があります。定数の値は、モデル中で定義せず外部からデータファイルで与えます。変数、目的関数の宣言と同様に、name = "... "には、定数名を指定します。定数名は、データファイル中のデータとの対応付けに使用されます。

次にモデルに定数を与えるために、以下のデータファイルを作成します。以下のデータファイルの拡張子は.dat となります。

```
"油田 X の運転コスト" = 180;  
"油田 Y の運転コスト" = 160;
```

=の左辺には、宣言時の name = "... "で与えたパラメータ名を記述します。右辺には、定数値を記述します。セミコロン; が定数データの区切りになります。データファイル中の"... "内にはないスペース、改行、タブは無視されます。

では、上記データファイルを入力として、実行してみます（実行方法については「[3.1 利用方法](#)」を参照してください）。

最適化経過の出力の後、次のような実行結果が得られます。

```
油田 X の運転日数=1.5  
油田 Y の運転日数=3  
全運転コスト=750
```

前回と同じ結果が得られています。Parameter とデータファイルを使用することで、データファイルの変更のみで違う問題を解くことができます。

では、データファイルを変更して実行してみましょう。以下のようにデータファイルを変更します。

```
"油田 X の運転コスト" = 100;  
"油田 Y の運転コスト" = 170;
```

実行すると、以下の結果が得られます。

```
油田 X の運転日数=5  
油田 Y の運転日数=0.666667  
全運転コスト=613.333
```

2.3 集合・添字

実は、ここまでのモデルでは、次のように各油田について同じ日数制約を定義しているので、冗長な記述になっていると言えます。

$$\begin{array}{l} 0 \leq x \leq 5 \quad \text{油田 X の週あたりの運転日数制約} \\ 0 \leq y \leq 5 \quad \text{油田 Y の週あたりの運転日数制約} \end{array}$$

そこで油田運転日数を一般的に記述することを考えてみましょう。まず油田運転日数 x, y をそれぞれ x_0, x_1 と変更し、定式化を次のように変更します。

集合	
$OilField = \{0, 1\}$	油田集合
変数	
$x_i, i \in OilField$	油田 i の運転日数/週
定数	
$costX$	油田 0 の運転コスト/日
$costY$	油田 1 の運転コスト/日
目的関数（最小化）	
$costX \cdot x_0 + costY \cdot x_1$	運転コスト/週
制約条件	
$6x_0 + x_1 \geq 12$	重油ノルマ/週
$4x_0 + 6x_1 \geq 24$	ガスノルマ/週
$0 \leq x_i \leq 5, \forall i \in OilField$	油田 i の週あたりの運転日数制約

運転日数の制約を一行で書き表すことができました。

対応する C++SIMPLE の記述は、次のようになります。

```
// 油田集合と添字の定義
Set OilField(name = "油田集合");
OilField = "0 1";
Element i(set = OilField);

// 油田 i の運転日数/週
Variable x(name = "油田の運転日数", index = i);
```

```
// 油田運転コスト/日
Parameter costX(name = "油田 X の運転コスト");
Parameter costY(name = "油田 Y の運転コスト");

// 運転コスト/週 (目的関数)
Objective cost(name = "全運転コスト", type = minimize);
cost = costX * x[0] + costY * x[1];

// 製品ノルマ
6 * x[0] + x[1] >= 12;    // 重油ノルマ
4 * x[0] + 6 * x[1] >= 24; // ガスノルマ

// 油田 i の週あたりの日数制約
0 <= x[i] <= 5;

// 求解
solve();

// 結果出力
x[i].val.print();
cost.val.print();
```

定式化と同様、日数制約を一行で書き表しています。

それでは、C++SIMPLE の記述の変更・追加点について、上から順に見ていきます。

```
Set OilField(name = "油田集合");
```

ここでは集合 (油田の集合) を宣言しています。C++SIMPLE で添字を使用する場合は、まず添字の属する集合を宣言する必要があります。変数、目的関数、定数と同様に、`name = "..."`の部分には集合名を指定します。`name = "..."`は省略可能ですが、内容を出力する際などで使用されますので、記述したほうが良いでしょう。

```
OilField = "0 1";
```

ここでは油田集合の内容を定義しています。先の定式化の添字範囲が $\{0, 1\}$ なので、0, 1 を集合の要素とします。

```
Element i(set = OilField);
```

ここでは集合 `OilField` の要素を表す添字 `i` を宣言しています。`set = ...` で添字が属する集合を定義します。


```
Variable x(name = "油田の運転日数", index = i);
```

ここでは油田の運転日数を、添字付き変数として宣言しています。index = i で添字を指定します。

```
cost = costX * x[0] + costY * x[1];
```

ここでは運転コストの内容定義をしています（制約式と内容定義は異なる点に注意してください）。添字付けは、x[添字] と記述します。

```
// 製品ノルマ
6 * x[0] + x[1] >= 12;
4 * x[0] + 6 * x[1] >= 24;
```

ここでは製品ノルマの制約を記述しています。以前に x, y と書いた変数部分を x[0], x[1] と置き換えただけです。

```
// 日数制約
0 <= x[i] <= 5;
```

ここでは日数制約を記述します。添字に i と指定することで、全ての $i \in OilField$ に関する日数制約を、かけたことになります。

```
// 結果出力
x[i].val.print();
```

結果出力も上記日数制約と同様に、添字に i と指定することで、全ての $i \in OilField$ について x[i] の値が出力されます。

次に実行してみます（実行方法については「[3.1 利用方法](#)」を参照してください）。データファイルは「[2.2 定数](#)」で使用した変更前のデータを使用してください。最適化経過が出力されたあと、x[i].val.print() に対応した、以下の出力が得られます。

```
油田の運転日数 [0]=1.5
油田の運転日数 [1]=3
```

変数名が添字つきで出力されているのが確認できます。

ここまでの記述の変更で、油田集合 OilField を導入し、各油田の運転日数を x[i] と簡略化することができました。次に、油田運転コスト costX, costY も添字 i を用いて簡略化してみます。運転コストを添字付けし、以下のように表すことにします。

定数

$costX_i, i \in OilField$	油田 i の運転コスト/日
---------------------------	---------------

costX₀, costX₁ はそれぞれ以前の costX, costY に対応する定数です。C++SIMPLE でも同様に定数の

添字付けを用いて、以下のように修正します。

```
Parameter costX(name = "油田 X の運転コスト");
Parameter costY(name = "油田 Y の運転コスト");
cost = costX * x[0] + costY * x[1];
```

↓

```
Parameter costX(name = "油田運転コスト", index = i);
cost = costX[0] * x[0] + costX[1] * x[1];
```

定数の添字付けは、変数の添字付けと同様に `index = i` と指定します。上記変更に合わせて、データファイルの内容を以下のように修正します。

```
"油田運転コスト" = [0] 180 [1] 160;
```

添字付きの定数値を指定する右辺は、

[添字] 値 [添字] 値 ...

と記述します。

では、実行してみましょう（実行方法については「[3.1 利用方法](#)」を参照ください）。最適化経過が出力された後、以下のように以前と同様の結果が得られます。

```
油田の運転日数 [0]=1.5
油田の運転日数 [1]=3
全運転コスト=750
```

ここで、油田集合とその要素について考えます。上記のデータファイル中には、運転コストの添字として 0, 1 が記述されています。そして C++SIMPLE の記述中で、運転コストの添字は油田集合の要素であると明示しています。以上より、C++SIMPLE はこのような油田集合の要素は 0, 1 からなると推定することができますので、実は、以下の油田集合の具体的な要素を与える記述は省略することができます。

```
OilField = "0 1";
```

この記述を削除して実行してみますと、前回と同様の結果が得られるのが確認できます。

このように C++SIMPLE では、添字と集合の関係から集合の内容を自動的に推定する機能があります。この機能を利用すれば集合の要素を C++SIMPLE で陽に記述する必要がなくなります。これにより、汎用的なモデル記述が可能となりますので、是非御活用ください。

次に、重油とガスの生産ノルマの値を外部から与えることを考えます。定式化において製品集合を導入して製品ノルマを以下のように記述します。

集合	
$Product = \{\text{重油}, \text{ガス}\}$	製品集合
定数	
$norma_j, j \in Product$	製品 j のノルマ/週

C++SIMPLE の記述においても同様に定数の添字付けを用いて表現し、ノルマに関する制約式を以下のように変更します。

```
6 * x[0] + x[1] >= 12;
4 * x[0] + 6 * x[1] >= 24;
```

↓

```
Set Product(name = "製品集合");
Element j(set = Product);
Parameter norma(name = "製品ノルマ", index = j);
6 * x[0] + x[1] >= norma["重油"];
4 * x[0] + 6 * x[1] >= norma["ガス"];
```

新たに製品集合の宣言を追加し、ノルマを製品を表す添字 j 付きの定数にします。上記のように文字列を添字に使用する場合は、文字列を "... " の中に記述する必要があります。次に、データファイルにノルマを与えるデータを追加しましょう。データファイルは以下のようになります。

```
"油田運転コスト" = [0] 180 [1] 160;
"製品ノルマ" = ["重油"] 12 ["ガス"] 24;
```

C++SIMPLE の記述中で、製品ノルマの添字は製品集合の要素であると明示しています。このことから、C++SIMPLE は製品集合の要素は "重油", "ガス" であると推定することができます。ゆえに、C++SIMPLE の記述中に製品集合の要素を書く必要はありません。このことは、油田集合の要素の推定と同様です。実行させると以前と同様の結果が得られます。

ここまでの変更をまとめて、集合、変数、定数、制約条件、目的関数を分類し整理すると、定式化と C++SIMPLE の記述は次のようになります。

集合	
$OilField = \{0, 1\}$	油田集合
$Product = \{\text{重油}, \text{ガス}\}$	製品集合
定数	
$costX_i, i \in OilField$	油田 i の運転コスト/日
$norma_j, j \in Product$	製品 j のノルマ/週

変数

$x_i, i \in OilField$	油田 i の運転日数/週
-----------------------	----------------

目的関数 (最小化)

$costX_0 \cdot x_0 + costX_1 \cdot x_1$	運転コスト/週
---	---------

制約条件

$6x_0 + x_1 \geq norma_{重油}$	重油ノルマ/週
$4x_0 + 6x_1 \geq norma_{ガス}$	ガスノルマ/週
$0 \leq x_i \leq 5, \forall i \in OilField$	油田 i の週あたりの運転日数制約

```
// 油田集合
Set OilField(name = "油田集合");
Element i(set = OilField);

// 製品集合
Set Product(name = "製品集合");
Element j(set = Product);

// 油田 i の運転コスト/日
Parameter costX(name = "油田運転コスト", index = i);

// 製品 j のノルマ/週
Parameter norma(name = "製品ノルマ", index = j);

// 油田 i の運転日数/週 (変数)
Variable x(name = "油田の運転日数", index = i);

// 運転コスト/週 (目的関数)
Objective cost(name = "全運転コスト", type = minimize);
cost = costX[0] * x[0] + costX[1] * x[1];

// 製品ノルマ
6 * x[0] + x[1] >= norma["重油"]; // 重油ノルマ/週
4 * x[0] + 6 * x[1] >= norma["ガス"]; // ガスノルマ/週

// 油田 i の週当たりの運転日数制約
```

```

0 <= x[i] <= 5;

// 求解
solve();
// 結果出力
x[i].val.print();
cost.val.print();

```

2.4 集約・複数の添字

コスト定義式

```
cost = costX[0] * x[0] + costX[1] * x[1];
```

は、すべての油田について運転コストの和をとるという意味なので、これを一般的に記述すると、以下ようになります。

$$cost = \sum_i costX_i \cdot x_i$$

対応する C++SIMPLE の記述は、以下ようになります。

```
cost = sum(costX[i] * x[i], i);
```

sum() は \sum に対応する関数で、

sum(和をとる式, 添字)

の書式を持ちます。

次にノルマ制約についても、sum() を適用したいと考えますが、旧記述では、

```

6 * x[0] + x[1] >= norma["重油"];
4 * x[0] + 6 * x[1] >= norma["ガス"];

```

と各油田の生産量が直接数値で記述されているので、一般化できません。そこで、定式化において定数 $prodX_{i,j}$ を導入し、制約式を次のように記述します。

制約条件

$$\sum_{i \in OilField} prodX_{i,j} \cdot x_i \geq norma_j, \forall j \in Product$$

製品 j のノルマ/週の制約式

定数

$prodX_{i,j}, i \in OilField, j \in Product$

油田 i の製品 j 生産量/日

 $norma_j, j \in Product$
製品 j のノルマ/週

対応する C++SIMPLE の記述は、以下のようになります。

```
Parameter prodX(name = "油田の生産量", index=(i, j));
sum(prodX[i, j] * x[i], i) >= norma[j];
```

複数の添字に依存する定数を宣言する際には、 $index = (i, j, \dots)$ と指定します。上記 $sum()$ は指定した添字 i のみの和をとります。 i, j について和をとる場合は、 $sum(\text{任意の式}, (i, j))$ 、と記述します。

次に油田の生産量の値を追加した以下のデータファイルを作成します。

```
"油田運転コスト" = [0] 180 [1] 160;
"製品ノルマ" = ["重油"] 12 ["ガス"] 24;
"油田の生産量" =
[0, "重油"] 6 [1, "重油"] 1
[0, "ガス"] 4 [1, "ガス"] 6
;
```

データファイル中の””に囲まれていない、スペース、タブ、改行は無視されます。従って、上記の“油田の生産量”のように、値を複数の行にわたって記述することができます。以上で、変更可能性のある全ての数値データをデータファイルから入力することができました。実行結果は以前と同様になります。

ここまでの変更をまとめて、集合、変数、定数、制約条件、目的関数を分類し整理すると、定式化と C++SIMPLE の記述は次のようになります。

集合

 $OilField = \{0, 1\}$

油田集合

 $Product = \{\text{重油}, \text{ガス}\}$

製品集合

定数

 $costX_i, i \in OilField$ 油田 i の運転コスト/日 $norma_j, j \in Product$ 製品 j のノルマ/週 $prodX_{i,j}, i \in OilField, j \in Product$ 油田 i の製品 j 生産量/日

変数

 $x_i, i \in OilField$ 油田 i の運転日数/週

目的関数（最小化）

$\sum_{i \in OilField} costX_i \cdot x_i$	運転コスト/週
---	---------

制約条件

$\sum_{i \in OilField} prodX_{i,j} \cdot x_i \geq norma_j, \forall j \in Product$	製品 j のノルマ/週の制約式
$0 \leq x_i \leq 5, \forall i \in OilField$	油田 i の週あたりの運転日数制約

```
// 油田集合
Set OilField(name = "油田集合");
Element i(set = OilField);

// 製品集合
Set Product(name = "製品集合");
Element j(set = Product);

// 油田 i の運転コスト/日
Parameter costX(name = "油田運転コスト", index = i);

// 製品 j のノルマ/週
Parameter norma(name = "製品ノルマ", index = j);

// 油田 i の製品 j 生産量/日
Parameter prodX(name = "油田の生産量", index = (i, j));

// 油田 i の運転日数/週（変数）
Variable x(name="油田の運転日数", index = i);

// 運転コスト/週（目的関数）
Objective cost(name = "全運転コスト", type = minimize);
cost = sum(costX[i] * x[i], i);

// 製品 j のノルマ/週の制約式
sum(prodX[i, j] * x[i], i) >= norma[j];

// 油田 i の週あたりの運転日数制約
0 <= x[i] <= 5;
```

```
// 求解
solve();

// 結果出力
x[i].val.print();
cost.val.print();
```

2.5 式

ここでは、これまでの結果出力（油田運転日数/週, 全運転コスト）に加えて、各製品の生産量/週も出力してみます。

生産量/週は一般的に以下のように記述できます。

$$prod_j = \sum_{i \in OilField} prodX_{i,j} \cdot x_i, \forall j \in Product \quad \text{製品 } j \text{ の生産量/週}$$

この式に対応する C++SIMPLE の記述は、以下のようになります。

```
Expression prod(name = "製品の生産量", index = j); // 式の宣言
prod[j] = sum(prodX[i, j] * x[i], i); // 式の定義
```

まず, Expression で式を宣言します。name, index の指定は、変数宣言時 (Variable) と同様に、name で名前を指定し、index で添字を指定します。prod[j] = ... で式の内容を定義します。Expression は、任意の変数を含む式に名前を付けるためのもので、数理最適化問題の変数の数が増加することはありません。

次に生産ノルマの記述を見てみます。

```
sum(prodX[i, j] * x[i], i) >= norma[j];
```

左辺は先ほど定義した prod[j] と全く同じ内容ですので、以下のように左辺を prod[j] に置き換えることができます。

```
prod[j] >= norma[j];
```

次に結果出力部分に以下のように prod[j] を追加します。

```
prod[j].val.print();
```

これで、製品の生産量/週が出力されるようになりました。生産量の出力結果は、以下のようになります。


```
製品の生産量 [ガス]=24
製品の生産量 [重油]=12
```

2.6 整数変数

ここまでは、運転日数を連続変数とみなして解いてきました。しかし実際には油田は1日単位でしか運転できません。そこで、運転日数を1日単位の整数変数とした、整数計画問題を解くことを考えます。そのために、変数（運転日数）の宣言を以下のように変更します。

```
Variable x(name = "油田の運転日数", index = i);
```

↓

```
IntegerVariable x(name = "油田の運転日数", index = i);
```

`IntegerVariable` で整数変数を宣言します。整数変数として宣言された変数は、値として整数のみを取ります。以上で変更完了です。

実行すると、以下の結果が得られます。

```
(solve() の最適化経過出力)
製品の生産量 [重油]=15
製品の生産量 [ガス]=26
油田の運転日数 [0]=2
油田の運転日数 [1]=3
全運転コスト=840
...
```

運転日数が整数になっているのが確認できます。このように変数を `IntegerVariable` で宣言するだけで、整数計画問題を記述することができます。

2.7 結果出力関数

ここまでは、結果の出力には `print()` を使用してきましたが、C++SIMPLE は他にも書式指定出力関数 `simple_printf()` があります。

以下、`simple_printf()` の機能を簡単に紹介します。なお、説明中に C++ 言語の機能にふれる記述があります。C++ 言語については、C++ 言語の参考書等を参照してください。

`simple_printf()` は書式を細かく指定できる出力関数です。

結果の確認程度の用途ならば `print()` で十分ですが、出力書式を細かく指定したい場合には `simple_printf()` を使用すると便利です。ここでは、運転日数の出力部を以下のように変更してみます。

```
x[i].val.print();
```

↓

```
simple_printf("油田 %d の最適運転日数 = %d\n", i, x[i]);
```

対応する実行結果出力は以下のようになります。

```
油田 0 の最適運転日数 = 2
```

```
油田 1 の最適運転日数 = 3
```

関数 `simple_printf()` の書式指定は、

```
simple_printf(出力書式指定, 出力対象 1, 出力対象 2, ...)
```

となります。

出力対象には、変数、式、定数、目的関数、添字、など集合以外の任意のものを任意の個数だけ指定できます。出力書式指定の指定方法は、C++言語の標準関数 `printf()` の書式指定と同様のものが指定できます。

2.8 デバッグ出力関数

数理最適化モデルが複雑になるほど、些細な記述ミスでも発見が困難になっていきます。そのようなミスを修正するための支援関数として `showSystem()` があります。`showSystem()` は、目的関数・制約式を実際のモデル内容に展開して出力します。

以下のように、`showSystem()` を最適化計算 `solve()` の直前に挿入してみます。

```
showSystem();
solve();
```

上記の位置に記述すれば、最適化計算を行うモデルの内容が出力できます。これを実行すると、`showSystem()` に対応した出力が以下のように得られます。

```
1-1 (sample.smp:26): 6*油田の運転日数 [0]+油田の運転日数 [1] >= 12
1-2 (sample.smp:26): 4*油田の運転日数 [0]+6*油田の運転日数 [1] >= 24
2-1 (sample.smp:29): 0 <= 油田の運転日数 [0] <= 5
2-2 (sample.smp:29): 0 <= 油田の運転日数 [1] <= 5
objective (sample.smp:23 name="全運転コスト"): 180*油田の運転日数 [0]+160*油田の運転日数 [1] (minimize)
```

1-1, 1-2 は次のノルマ制約式に対応しています。

```
prod[j] >= norma[j];
```

2-1, 2-2 は次の日数制約式に対応しています。

```
0 <= x[i] <= 5;
```

objective は、次のコスト定義式に対応しています。

```
cost = sum(costX[i] * x[i], i);
```

このように、showSystem() を使用することによって、定数値、添字等を実際の値に置き換えた後の目的関数・制約式を確認することができます。この機能を利用すれば、意図しない記述ミスを簡単に発見することができ、効率の良いモデル記述が可能になります。

2.9 C++SIMPLE を記述する際の注意点

モデリング言語 C++SIMPLE を用いる際に頻出する注意点を列挙します。

- 大文字と小文字は区別される
- 積演算子を省略してはならない
- 半角スペースは自由に入れてよい（等号/不等号の間は駄目）
- 改行は自由に入れてよい
- 文末には必ず半角セミコロン; を入れる
- // はコメントを意味する
- name = 引数はダブルクォート"で囲む
- minimize や maximize はダブルクォート"で囲まない
- name と type のように複数の設定を行うときはカンマで区切る
- name や type を設定する順番は変えて良い
- 等式付不等号<=と>=は使用できるが、等式なし不等号<と>は使用できない
- = は代入、== は等価を表わす

第 3 章

C++SIMPLE 基本事項

3.1 利用方法

C++SIMPLE を利用する方法には以下の二種類があります。

1. 数理最適化専用 GUI「Nuorium」を用いる（Windows 版のみ）
2. コマンドラインから起動する

本マニュアルでは「2. コマンドラインから起動する」のみ説明します。「1. 数理最適化専用 GUI「Nuorium」を用いる」については「Nuorium スタートガイド」をご参考ください。

3.2 C++SIMPLE による数理最適化計算の流れ

記述された数理最適化問題（モデル）を求解するには、以下の三段階の手順を実行します。

- [STEP1] テキストエディタ等を用い C++SIMPLE で数理最適化問題（モデル）を記述
- [STEP2] [STEP1] で記述されたモデルを実行形式に変換する（コンパイル）
- [STEP3] 実行形式を起動させる

3.2.1 Windows 版における手順

Nuorium Optimizer の提供するバッチコマンド mknuopt.bat でモデルファイル（smp ファイル）をコンパイルします。

```
prompt% mknuopt.bat [モデルファイル名].smp
```

生成される実行形式 [モデルファイル名].exe の引数にデータファイル（拡張子 csv あるいは dat）を与えて実行します。

```
prompt% [モデルファイル名].exe [データファイル]
```

3.2.2 UNIX/Linux 版における手順

Nuorium Optimizer の提供するシェルスクリプト mknuopt でモデルファイル（smp ファイル）をコンパイルします。

```
prompt% mknuopt [モデルファイル名].smp
```

生成される実行形式 [モデルファイル名] の引数にデータファイル（拡張子 csv あるいは dat）を与え

て実行します。

```
prompt% ./[モデルファイル名] [データファイル]
```

3.2.3 備考

- モデルファイル名に使える文字は、半角英数字および半角アンダースコア (_) のみです。
- 同じモデルファイルに対してデータファイルのみ変更して実行する場合は、「[STEP2] コンパイル」の手順は省略可能です。
- V22 以降の UNIX/Linux 版では、従来の ufun という名前の void 型関数を含んだ .cc ファイルはモデルファイルとして使用できなくなりました。Windows 版と同一の .smp 形式のモデルファイルをご利用ください。

第4章

C++SIMPLE 一般事項

4.1 基本的な内容

4.1.1 行末のセミコロン;

C++SIMPLE ではモデルファイルの行末に半角セミコロン; を付ける必要があります。全角セミコロン; を用いてはいけません。

正しい

```
x + y <= 1;
```

誤り

```
x + y <= 1 ;
```

4.1.2 半角空白文字と改行

半角空白文字（半角スペース）と改行はモデル中では任意に用いる事ができます。全角空白文字（全角スペース）を用いる事はできません。例えば、次の二つは同じ意味です。

```
x+y<=3;
```

```
x + y <= 3;
```

次の二つも同じ意味です。

```
Variable x;  
Parameter a;
```

```
Variable x;  
  
Parameter a;
```

意味のかたまりを区切ってしまう場合は、半角空白文字や改行を入れてはいけません。以下の例は誤りです。

```
Vari able x;
```

```
x + y <= 3;
```

```
Vari  
able x;
```

4.1.3 構成要素の順序

C++SIMPLE では、変数、定数や目的関数の定義順序に関する規則はありません。好きなタイミングで定義を行うことができます。例えば、以下の二つの記述は同じ意味です。

```
Variable x;  
Parameter a;
```

```
Parameter a;  
Variable x;
```

但し、定義していないものを先に使用する事は禁止されています。次の例は未定義の変数 x を用いて目的関数を定義しているのです、誤りです。

```
Objective f;  
f = x;  
Variable x;
```

4.1.4 モデルファイル内で利用できない文字

SIMPLE 内で既に予約されている文字列 (Variable, Parameter 等のクラス名) や、C++で既に使われている文字列 (class, enum など) を定義することはできません (このように、使用が禁止されている文字列を予約語と呼びます)。以下の例はいずれも誤りです。

```
IntegerVariable enum;
```

```
Parameter Variable;
```

また、全角空白文字 (全角スペース) や丸数字などの機種依存文字も使用することはできません。

4.1.5 name 引数に利用できない文字

name 引数として次の文字は利用できません。

- 二重引用符 (")
- 半角コンマ (,)
- 制御文字 (\r, \n, \t など)

- 機種依存文字（丸数字、波ダッシュ「～」など）

また、以下のような重複がある場合にはエラーとなります。

- name 引数の重複

以下はその例です。

```
Variable x(name = "a"), y(name = "a");
```

- name 引数とインスタンス名の重複

以下はその例です。

```
Variable x(name = "y");  
Variable y;
```

4.1.6 データファイル内で利用できない文字

データファイル内で機種依存文字（丸数字、波ダッシュ「～」など）は使用することができません。

4.2 高度な内容

4.2.1 外部接続における C++ オブジェクトの宣言と代入

外部接続を用いて、C++ プログラム内にモデリング言語 C++SIMPLE を記述する場合は、SIMPLE オブジェクト宣言時に代入文を書くことと意図通りに動作しません。例えば以下のように書くことはできません。

```
Parameter a = sum(x[i], i);
```

上記のように書きたい場合は、宣言と代入を別にして書く必要があります。

```
Parameter a;  
a = sum(x[i], i);
```

4.2.2 標準出力の抑制

実行時に引数として「-silent」を指定することにより標準出力が抑制されます。この設定は求解オプションでの設定等よりも優先されます。

```
prompt% ./[モデルファイル名] -silent
```

ただしユーザが `printf` や `std::cout` 等の C++ 関数を経由して行う標準出力は抑制されません。

4.2.3 UNIX/Linux 版におけるデータファイルの文字コード指定について

UNIX/Linux 版では、データファイルの文字コードを明示的に指定する必要があります。実行時に次のように指定してください。

- データファイルの文字コードが EUC-JP の場合

```
prompt% ./[モデルファイル名] -e [データファイル]
```

- データファイルの文字コードが Shift_JIS の場合

```
prompt% ./[モデルファイル名] -s [データファイル]
```

- データファイルの文字コードが UTF-8 の場合

```
prompt% ./[モデルファイル名] -w [データファイル]
```

なお、文字コードを指定しない場合、データファイルの文字コードは UTF-8 であるとみなされます。

```
prompt% ./[モデルファイル名] [データファイル 1] [データファイル 2]
```

上の場合、2つのデータファイルの文字コードは、ともに UTF-8 であるとみなされます。

異なる文字コードのデータファイルを同時に読み込むことも可能です。

```
prompt% ./[モデルファイル名] -e [データファイル 1] -s [データファイル 2] -w [データファイル 3]
```

上の場合、データファイル1の文字コードは EUC-JP、データファイル2の文字コードは Shift_JIS、データファイル3の文字コードは UTF-8 とみなされます。

```
prompt% ./[モデルファイル名] [データファイル 1] [データファイル 2] -s [データファイル 3]
```

上の場合、データファイル1とデータファイル2の文字コードは UTF-8、データファイル3の文字コードは Shift_JIS とみなされます。

同じ文字コードのデータファイルが続く場合、2つ目以降は文字コード指定を省略できます。

```
prompt% ./[モデルファイル名] -e [データファイル 1] [データファイル 2] [データファイル 3]
```

上の例の場合、3つのデータファイルの文字コードは、全て EUC-JP であるとみなされます。

4.2.4 変数の上下限

- 「変数の上下限」は、変数の上限および下限を表します。
- C++SIMPLE においては、「変数と定数の比較式」は「変数の上下限」として扱われます。注意すべき点は、これは制約式としては扱われない、という点です。
- 変数の上下限值は、下限であれば `x.lb.val.print()`、上限であれば `x.ub.val.print()` で確認で

きます。

例えば x, y を変数, A を定数とします。以下は「変数の上下限」となります。

```
x >= 0;
x >= a;
x[i] >= a[i];
```

以下は「変数の上下限」ではなく、制約式になります。

```
sum(x[i], i) >= a;
x >= y;
x + y >= a;
2*x >= a;
```

変数の上下限と制約式の違い

- 矛盾する「変数の上下限」が与えられた場合、**求解前の段階**でエラー検出がされます。
- 制約式については、矛盾していたとしても、求解は行われます。

例えば以下のようなモデルを記述します。

```
Variable x;
x >= 1;
x <= 0;
solve();
```

上記の場合、以下のようなエラー出力がされ求解は行われません。

(SIMPLE 1) 変数 x について矛盾した上下限が与えられました (上下限 $[-inf \leq * \leq 0]$ と $[1, inf]$ の共通部分を取ったとき)

もし求解まで行いたい場合は、「変数の上下限」を制約式に書き換えるなどの方策があります。たとえば先のモデルは以下のように書き換えます。

```
Variable x;
Variable dummy;
x + dummy >= 1;
x + dummy <= 0;
dummy == 0;
solve();
```


第5章

数理最適化モデルの構成要素

以下は C++SIMPLE を用いて数理最適化モデル（モデルファイル）を記述する際の構成要素の一覧です。ここでは全ての構成要素を列挙してはいませんが、大半の数理最適化モデルは以下の構成要素の組合せで記述することができます。

構成要素名	C++SIMPLE 内の名称	機能
変数	Variable	変数を表す
目的関数	Objective	目的関数を表す
制約式	Constraint	制約式を表す
定数	Parameter	定数を表す
整数変数	IntegerVariable	整数変数を表す
範囲演算関数	sum, prod	\sum や \prod に相当する
式	Expression	頻出する数式に対して、簡単な別の表現を与える
対称行列	SymmetricMatrix	対称行列を表す
添字	Element	添字を表す
集合	Set	添字の動く範囲を表す
順序集合	OrderedSet	要素間に順序を持つ集合を表す
数列集合	Sequence	等差数列からなる集合を表す
条件式		制約式や代入文を制限する
数学関数	exp, sin, cos, log ...	数学関数を表す

以下の構成要素はアルゴリズム wcsp を用いる場合にのみ使用が可能です（ソフト制約関数、ブール関数は rcpsp でも用いる事ができます）。

構成要素名	C++SIMPLE 内の名称	機能
離散変数	DiscreteVariable	離散変数を表す
重複不能関数	alldiff	重複不能を表す
選択関数	selection	限定選択を表す
ハード制約関数	hardConstraint	ハード制約を表す
セミハード制約関数	semiHardConstraint	セミハード制約を表す
ソフト制約関数	softConstraint	ソフト制約を表す
ブール関数	Boolean	制約式を引数として、0-1 を返す

以下の構成要素はアルゴリズム rcpsp を用いる場合にのみ使用が可能です。

構成要素名	C++SIMPLE 内の名称	機能
アクティビティ	Activity	アクティビティを表す
要求資源	ResourceRequire	要求資源を表す
資源供給量	ResourceCapacity	資源供給量を表す
同一モード順序選択関数	modeOrder	同一モードを選択する
アクティビティ固定関数	fixActivity	アクティビティを固定する
アクティビティ固定解除関数	unfixActivity	アクティビティの固定を解除する

5.1 変数クラス Variable

変数は Variable というクラスで表現されます。具体的に x という変数を定義するには以下のよう
に記述します。

```
Variable x;
```

複数の変数を一度に定義するには、集合クラス Set と添字クラス Element を用います。以下の例で
は、3 個の変数 $y[1], y[2], y[3]$ を一度に定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable y(index = i);
```

変数の初期値は=で設定できます。以下の例では x に初期値 3 を設定しています。

```
x = 3;
```

以下の例では $y[1], y[2], y[3]$ に初期値 5 をまとめて設定しています。

```
y[i] = 5;
```

明示的な指定が無い場合、変数の初期値はアルゴリズムに応じて自動的に決定されます。アルゴリ
ズムによっては初期値の設定を無視します。

5.2 目的関数クラス Objective

目的関数は Objective というクラスで表現されます。目的関数自身の定義と、目的関数の構造の定
義は別々に行う必要があります。例えば $2x + 3y$ という目的関数（最小化）を定義したい場合、次のよ
うに記述します。

```
Objective f(type = minimize);
f = 2 * x + 3 * y;
```

以下の記述はいずれも誤りです。

```
Objective 2 * x + 3 * y (type = minimize);
```

```
Objective f = 2 * x + 3 * y (type = minimize);
```

目的関数を定義する際、扱う問題が最小化問題である場合は引数に `type = minimize` を指定するか、`type =` を省略します。一方、最大化問題である場合は引数に `type = maximize` を指定します。

```
Objective f(type = minimize); // 最小化問題
Objective f; // 最小化問題
Objective f(type = maximize); // 最大化問題
```

目的関数に添字をつける事はできません。例えば、以下の記述は誤りです。

```
Objective f(index = i, type = minimize);
f[i] = 2 * x[i] + 3 * y[i];
```

`Minimize`, `Maximize` という糖衣構文もあります。

```
Minimize f; // Objective f(type = minimize); と同じ
Maximize f; // Objective f(type = maximize); と同じ
```

5.3 制約式クラス Constraint

制約式は `Constraint` というクラスで表現されます。C++SIMPLE で表現可能な制約式は、等式制約（`==` を使用）及び等号付不等式制約（`<=`, `>=` を使用）の二種類です。等式制約に用いる演算子は `=` ではなく、`==` であることに注意してください。具体的に $x + y = 1$ という制約式を定義するには次のように記述します。

```
x + y == 1;
```

$x - 2y \leq 0$ という制約式を定義するには次のように記述します。

```
x - 2 * y <= 0;
```

制約式自身の定義は必ずしも必要ではありませんが、以下の記述は上記と同じ意味です。解ファイルにおいて、制約式に対する双対変数等を取得したい場合には、制約式自身の定義を行うと、検索が容易です。

```
Constraint co;
co = x - 2 * y <= 0;
```

等号の付かない不等式を取り扱う事はできません。次の記述は誤りです。

```
x - 2 * y < 0;
```

複数の制約式を一度に定義するには、集合クラス `Set` と添字クラス `Element` を用います。以下の例

では、3 個の制約式 $x_1 - 2y_1 \leq 0$, $x_2 - 2y_2 \leq 0$, $x_3 - 2y_3 \leq 0$ を一度に定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
x[i] - 2 * y[i] <= 0;
```

制約式自身の定義を行う場合には、以下のようになります。

```
Set S;
S = "1 2 3";
Element i(set = S);
Constraint co(index = i);
co[i] = x[i] - 2 * y[i] <= 0;
```

不一致制約を表す演算子!=を用いることはできません。以下の記述は誤りです。

```
x + y != -1;
```

ただし、アルゴリズム wcsp を使用する際には不一致制約を用いることができます。

バージョン 10 より導入された SDP ソルバ (lsdp, trsdp) では、対称行列の半正定値制約を取り扱う事ができます。次の例では対称行列 X の半正定値制約を記述しています。

```
SymmetricMatrix X((i, j));
X >= 0;
```

対称行列 ≥ 0 を記述する事で、半正定値制約を表現できます。右辺には 0 以外のスカラー値を用いることもできます。この場合、右辺値は左辺行列の最小固有値を意味します。例えば、次の例は行列 X の最小固有値が 2 つまり、 $X - 2E \geq 0$ であることを意味します。

```
X >= 2;
```

不等号 \geq を逆向きに書く事はできません。例えば、次の記述は誤りです。

```
X <= 0;
```

右辺値に行列を記述することはできません。例えば次の記述は誤りです。

```
SymmetricMatrix X((i, j));
SymmetricMatrix Y((i, j));
X >= Y;
```

複数の半正定値制約を、一括して設定する事は可能です。

```
SymmetricMatrix X(index = n, (i, j));
X[n] >= 0;
```

半正定値制約からは、`solve()` 後に双対行列の値を取得することができます。下記の通り制約が定

められているとします.

```
SymmetricMatrix X((i, j));
Constraint c;
c = X >= 0;
```

このとき, solve() 後に下記の通り setDualMatrix 関数を呼ぶことで, 双対行列の値を得ることができます.

```
Parameter dM(index = (i, j));
dM.setDualMatrix(c);           //dM に双対行列の値が格納される
```

ただし, Parameter の添字が属する集合と SymmetricMatrix の添字が属する集合が一致していない場合や, 添字付きの半正定値制約を用いている場合, 双対行列の取得機能は利用できません. 例えば, 下記の用法はいずれも誤りです.

```
Parameter dM(index = (i, k)); // 添字 k の属する集合が異なる
dM.setDualMatrix(c); // 添字の集合が一致しないので不可
```

```
SymmetricMatrix X(index = k, (i, j));
Constraint c(index = k);
c[k] = X[k] >= 0;

Parameter dM(index = (i, j));
dM.setDualMatrix(c[0]); // 添字付きの半正定値制約は不可
```

```
Parameter dMs(index = (k, i, j));
dMs.setDualMatrix(c); // 添字の次元が一致しないので不可
```

5.4 定数クラス Parameter

定数は Parameter というクラスで表現されます. 具体的に a という定数を定義するには以下のよう
に記述します.

```
Parameter a;
```

複数の定数を一度に定義するには, 集合クラス Set と添字クラス Element を用います. 以下の例で
は, 3 個の定数 b[1], b[2], b[3] を一度に定義しています.

```
Set S;
S = "1 2 3";
Element i(set = S);
Parameter b(index = i);
```

以下の例では、6 個の定数 $c[1, p]$, $c[1, q]$, $c[2, p]$, $c[2, q]$, $c[3, p]$, $c[3, q]$ を一度に定義しています。

```
Set S;
Set T;
S = "1 2 3";
T = "p, q";
Element i(set = S);
Element j(set = T);
Parameter c(index = (i, j));
```

定数の値は=で設定します。定数の値を明示的に指定しない場合は、自動的に 0 が設定されます。以下の例では、定数 a に 3 を設定しています。

```
a = 3;
```

以下の例では定数 $b[1]$, $b[2]$, $b[3]$ に 5 をまとめて設定しています。

```
b[i] = 5;
```

個別に設定する場合は、以下のように記述します。

```
b[1] = 5;
b[2] = 5;
b[3] = 5;
```

添字が複数の場合は、まとめて設定する場合と個別に設定する場合の記述が異なります。以下の例では定数 $c[1, p]$, $c[1, q]$, $c[2, p]$, $c[2, q]$, $c[3, p]$, $c[3, q]$ に 6 をまとめて設定しています。

```
c[i, j] = 6;
```

個別に設定する場合は、添字をダブルクォート"で囲む必要があります。具体的には以下のように記述します。

```
c["1, p"] = 6;
c["1, q"] = 6;
c["2, p"] = 6;
c["2, q"] = 6;
c["3, p"] = 6;
c["3, q"] = 6;
```

定数の値は、モデルファイル内で設定する以外に、データファイルから設定する方法もあります。データファイルから設定する場合は、Parameter の引数に name を付ける必要があります。以下は、定数 a に 3 をデータファイル `foo.dat` から設定する場合の例です。

モデルファイル内

```
Parameter a(name = "cost");
```

foo.dat 内

```
cost = 3;
```

name で設定する名前はダブルクォート"で囲む必要があります。名前に空白や機種依存文字を用いる事はできません。name を省略した場合、モデルファイル内で定義された名前であるとみなされます。即ち、以下の二つは同等です。

```
Parameter a;
```

```
Parameter a(name = "a");
```

以下は、定数 b[1], b[2], b[3] に 5 をデータファイル foo.dat から設定する場合の例です。データファイルから値を設定する場合、まとめて設定する方法は無く、個別に設定する方法のみが存在します。モデルファイル内

```
Parameter b(name = "b");
```

foo.dat 内

```
b = [1] 5 [2] 5 [3] 5;
```

以下は、定数 c[1, p], c[1, q], c[2, p], c[2, q], c[3, p], c[3, q] に 6 をデータファイル foo.dat から設定する場合の例です。モデルファイルから設定する場合と異なり、添字をダブルクォートで囲む必要はありません。

モデルファイル内

```
Parameter c(name = "c");
```

foo.dat 内

```
c = [1, p] 6 [1, q] 6
      [2, p] 6 [2, q] 6
      [3, p] 6 [3, q] 6;
```

データファイルの種類や書式に関するより詳細な説明は「[8 データファイル](#)」を参照ください。

5.4.1 double 型への変換

定数 Parameter は値を double 型に変換できます。変換の際は val メソッドを通じて asDouble 関数を使います。

```
double d;
Parameter a;
d = a.val.asDouble(); // a の値を double 型にする.
```

これにより、例えば `Parameter` の値で求解オプションを設定できます。

```
Parameter maxtim(name = "最大計算時間");
options.maxtim = maxtim.val.asDouble();
```

5.4.2 char*型への変換

定数 `Parameter` は値を `char*` 型に変換できます。変換の際は `val` メソッドを通じて `asChar` 関数を使います。なお、`asChar` 関数は 1024 バイトまでの文字列に対応しています。

```
Parameter OutFileName(name = "OutFileName");
char *filename;
OutFileName.val.asChar(filename); // パラメータから文字列値の取り出し
...
delete [] filename; // 文字列値の取り出しのために確保した filename の領域を解放する
```

5.5 整数変数クラス IntegerVariable

整数変数は `IntegerVariable` というクラスで表現されます。具体的に `x` という整数変数を定義するには以下のように記述します。

```
IntegerVariable x;
```

複数の整数変数を一度に定義するには、集合クラス `Set` と添字クラス `Element` を用います。以下の例では、3 個の整数変数 `y[1]`, `y[2]`, `y[3]` を一度に定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
IntegerVariable y(index = i);
```

特に 0-1 のみの値を取る整数変数は、`type = binary` を引数に持たせる事で定義されます。以下では `z` という 0-1 変数を定義しています。

```
IntegerVariable z(type = binary);
```

複数の引数を持たせる場合、順序は任意です。以下の二表現は同様の意味を持ちます。

```
IntegerVariable z(type = binary, index = i);
```

```
IntegerVariable z(index = i, type = binary);
```

5.6 範囲演算関数 sum, prod

数式における \sum や \prod に類する機能として, C++SIMPLE では範囲演算関数として, sum 関数と prod 関数が提供されています. 次の例では, 制約式 $\sum_{i=1}^3 x_i = 10$ を記述しています.

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable x(index = i);
sum(x[i], i) == 10;
```

上の記述を sum 関数を使わずに書くと次のようになります.

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable x(index = i);
x[1] + x[2] + x[3] == 10;
```

次の例では, 制約式 $\prod_{i=1}^3 x_i = 20$ を記述しています.

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable x(index = i);
prod(x[i], i) == 20;
```

上の記述を prod 関数を使わずに書くと次のようになります.

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable x(index = i);
x[1] * x[2] * x[3] == 20;
```

sum 関数は複数の添字に対して適用する事もできます. 次の例では, 制約式 $\sum_{i=1}^3 \sum_{j=1}^2 a_i b_j y_{ij} = 10$ を記述しています.

```
Set S = "1 2 3";
Set T = "1 2";
```

```

Element i(set = S);
Element j(set = T);
Variable y(index = (i, j));
Parameter a(index = i);
Parameter b(index = j);
sum(a[i] * b[j] * y[i, j], (i, j)) == 10;

```

次のように記述することも可能です。

```

Set S = "1 2 3";
Set T = "1 2";
Element i(set = S);
Element j(set = T);
Variable y(index = (i, j));
Parameter a(index = i);
Parameter b(index = j);
sum(sum(a[i] * b[j] * y[i, j], j), i) == 10;

```

条件式を用いて、和や積を取る範囲を制限する事もできます。次の例では、制約式 $\sum_{i=3}^5 x_i = 10$ を記述しています。

```

Set S;
S = "1 2 3 4 5";
Element i(set = S);
Variable x(index = i);
sum(x[i], (i, i >= 3)) == 10;

```

次の例では、制約式 $\sum_{i \in T} x_i = 10$, $\sum_{i \notin T} x_i = 20$ を記述しています。

```

Set S = "p q r s";
Set T(superSet = S);
T = "p r";
Element i(set = S);
Variable x(index = i);
sum(x[i], (i, i < T)) == 10;
sum(x[i], (i, i > T)) == 20;

```

5.7 対称行列クラス SymmetricMatrix

対称行列は `SymmetricMatrix` というクラスで表現されます。対称行列自体の定義と、対称行列の構造の定義は別々に行う必要があります。例えば、次のような二次正方対称行列を定義したいとします。

$$X = \begin{pmatrix} x+3 & 4y+1.5z \\ 4y+1.5z & 2x+10y \end{pmatrix}$$

この場合、以下のように記述します。

```
Set S = "1 2";
Element i(set = S), j(set = S);
Variable x, y, z;
SymmetricMatrix X((i, j));
X["1, 1"] = x + 3;
X["1, 2"] = 4 * y + 1.5 * z;
X["2, 2"] = 2 * x + 10 * y;
```

対角要素以外の成分の定義は上下三角部分いずれか一方に関してのみ定義してください。上記の例では [1,2] 成分が定義されているので、[2,1] 成分を定義する必要はありません。

対称成分が重複して定義された場合は、先に定義された方は無視されます。例えば、

```
X["1, 1"] = x + 3;
X["1, 2"] = 1.5 * y + 4 * z; // 次の [2,1] 要素の定義で打ち消される
X["2, 1"] = 4 * y + 1.5 * z;
X["2, 2"] = 2 * x + y;
```

は、次の行列を定義している事になります。

$$X = \begin{pmatrix} x+3 & 4y+1.5z \\ 4y+1.5z & 2x+y \end{pmatrix}$$

Variable x,y,z に添字を付けて、係数に対しても Parameter を導入すれば、行列の定義を一括して行う事もできます。以下の例をご覧ください。

```
Set S = "1 2";
Set T = "1 2 3";
Element i(set = S), j(set = S);
Element k(set = T);
Variable x(index = k);
Parameter a(index = (k, i, j));
Parameter b(index = (i, j));
SymmetricMatrix X((i, j));
// 上三角部分を定義
a["1, 1, 1"] = 1;
a["1, 2, 2"] = 2;
a["2, 1, 2"] = 4;
a["2, 2, 2"] = 1.5;
```

```
a["3, 1, 2"] = 10;
b["1, 1"] = 3;
X[i, j] = sum(a[k, i, j] * x[k], k) + b[i, j], i <= j;
```

この例では、行列 $X = \begin{pmatrix} x_1 + 3 & 4x_2 + 10x_3 \\ 4x_2 + 10x_3 & 2x_1 + 1.5x_2 \end{pmatrix}$ を定義している事になります。

対称行列に対しては、制約として半正定値制約を課す事ができます。半正定値制約を課す場合、右辺に ≥ 0 を記述する必要があります。次の例では対称行列 X の半正定値制約を記述しています。

```
SymmetricMatrix X((i, j));
X >= 0;
```

右辺には0以外のスカラー値を用いることもできます。この場合、右辺値は左辺行列の最小固有値を意味します。例えば、次の例は行列 X の最小固有値が2つまり、 $X - 2E \geq 0$ であることを意味します。

```
X >= 2;
```

次の記述は誤りです。

```
SymmetricMatrix X((i, j));
SymmetricMatrix Y((i, j));
X >= Y;
```

複数の対称行列を一括して定義する事もできます。例えば、次の例では対称行列 X_1, \dots, X_{10} を一括して定義しています。

```
Set S = "1 2";
Element i(set = S), j(set = S);
Set N = "1 .. 10";
Element n(set = N);
SymmetricMatrix X(index = n, (i, j));
```

対称行列自体の添字 n には `index =` を付ける必要があります。個別の対称行列内部の添字 (i, j) には `index =` を付与してはいけません。

複数の対称行列に対して一気に半正定値制約を設定するには、次のように記述します。

```
SymmetricMatrix X(index = n, (i, j));
X[n] >= 0;
```

大規模な行列を考える場合、定数並びに行列成分を疎形式で定義することで高速な求解が可能です。次の例は少し難解ですが、Parameter a, b が出現する添字に絞って行列成分を定義しています。

```
Set S, T;
Element i(set = S), j(set = S);
Element k(set = T);
```



```

Variable x(index = k);
Set A(dim = 3, superSet = (T, S, S));
Element m(set = A);
Set B(dim = 2, superSet = (S, S));
Element n(set = B);
Parameter a(name = "a", index = m);
Parameter b(name = "b", index = n);
A = A | "1" * B;
B = A.slice(2, 3);
S = A.slice(1);
T = A.slice(2) | A.slice(3);
SymmetricMatrix X((i, j));
X[i, j] = sum(a[k, i, j] * x[k], (k, (k, i, j) < A)) + b[i, j], (i, j) < B;

```

上記の例では Parameter a, b の値を外部ファイルから与えるケースを想定しています。外部ファイルから自動代入によって定まる a, b の添字の範囲がそれぞれ集合 A, B であると定められています。行列内部での疎形式定義の為に、集合 A, B をそれぞれ加工します。また、変数の範囲を示す集合 T, 行列内の添字の範囲を示す集合 S は、いずれも集合 A から作成されています。

具体的に以下のようなデータファイルが与えられたとします。

```

a =
[1, 1, 1] 1
[1, 2, 3] 3
[2, 2, 2] 2
[3, 2, 2] 0.5
[3, 3, 3] 5
;
b =
[1, 1] 10
[1, 2] 4
;

```

これは、次の対称行列を定義している事に相当します。

$$X = \begin{pmatrix} x_1 + 10 & 4 & & \\ & 4 & 2x_2 + 0.5x_3 & 3x_1 \\ & & 3x_1 & 5x_3 \\ & & & \end{pmatrix}$$

自動代入の段階では、集合 A の要素は

$$\{(1, 1, 1), (1, 2, 3), (2, 2, 2), (3, 2, 2), (3, 3, 3)\}$$

集合 B の要素は

$$\{(1, 1), (1, 2)\}$$

です。これに対して以下の加工を施す事により、

```
A = A | "1" * B;
B = A.slice(2, 3);
```

集合 A の要素は

$$\{(1, 1, 1), (1, 1, 2), (1, 2, 3), (2, 2, 2), (3, 2, 2), (3, 3, 3)\}$$

集合 B の要素は

$$\{(1, 1), (1, 2), (2, 2), (2, 3), (3, 3)\}$$

となります。なお、"1" * B では、1 個の要素 (1) からなる集合と集合 B の直積を求めています。また、A | "1" * B により集合 A と (直積) 集合 "1" * B の和集合を求めています。

関数 slice() は、集合の一部を射影して切り出す機能を有しています。なお、関数 slice() については引数を最大 5 個とることができます。

上記の例では、集合 A の第二第三成分を切り出して、集合 B に渡しています。この結果 $|A| = 6$, $|B| = 5$ となりましたが、特に何の工夫もしない場合 $|A| = 27$, $|B| = 9$ となり、内部で余分な領域を必要とします。特に行列の次元が大きい場合には、パフォーマンスにかなりの違いが生じます。

5.8 式クラス Expression

Variable や Parameter を含んだ式は Expression というクラスで表現することができます。例えば、Variable である x, y に対して $2x + 3y$ という式を定義したい場合、次のように記述します。

```
Expression g;
g = 2 * x + 3 * y;
```

以下のように記述をすることも可能です。

```
Expression g = 2 * x + 3 * y;
```

複数の式を一度に定義するには、集合クラス Set と添字クラス Element を用います。以下の例では、3 個の式 $g[1]$, $g[2]$, $g[3]$ を一度に定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
Expression g(index = i);
g[i] = 2 * x[i] + 3 * y[i];
```

Expression を使う事によりモデルの記述を簡略化することができます。同じ式が何度も出現するモデルにおいて特に有効です。Expression はあくまで記述の簡略化を目的としたもので、Expression の導入の有無により最適化計算結果が異なる事はありません。

5.9 添字クラス Element

添字は Element というクラスで表現されます。添字とは数式 x_i の i に相当するものを意味します。添字と集合を対応させるには、引数 `set` を用います。頭文字の `s` は小文字である事に注意してください。集合クラス `Set` と併用することで、変数 `Variable` を集合の要素ごとに設定できます（制約式 `Constraint`、定数 `Parameter`、整数変数 `IntegerVariable`、式 `Expression` についても同様です）。以下の例では、3 個の変数 `y[1], y[2], y[3]`、3 個の定数 `b[1], b[2], b[3]` を定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable y(index = i);
Parameter b(index = i);
```

添字は複数導入することも可能です。次の例では 6 個の変数 `x["1, p"], x["1, q"], x["2, p"], x["2, q"], x["3, p"], x["3, q"]` を定義しています。

```
Set S;
Set T;
S = "1 2 3";
T = "p q";
Element i(set = S);
Element j(set = T);
Variable x(index = (i, j));
```

一つの集合に対して複数の添字を定める事もできます。次の例では 12 個の定数 `a["1, p, p"], a["1, p, q"], a["1, q, p"], a["1, q, q"], a["2, p, p"], a["2, p, q"], a["2, q, p"], a["2, q, q"], a["3, p, p"], a["3, p, q"], a["3, q, p"], a["3, q, q"]` を定義しています。集合 `T` に対して 2 つの添字 `j, k` が定められています。

```
Set S;
Set T;
S = "1 2 3";
T = "p q";
Element i(set = S);
Element j(set = T);
Element k(set = T);
Parameter a(index = (i, j, k));
```

複数の添字を持つ対象を個別に記述する場合は、添字部分をダブルクォート"で囲む必要があります。

```
y["1, p"] >= b["1, p"] + 3;
```

また以下のようにダブルクォートで囲まないと添字は自動展開され、制約式が一括して複数定義されます。（添字の自動展開機能）

```
y[i, j] >= b[i, j] + 3;
```

添字は、属する集合が整数値を取る場合には次のような演算子を用いることができます。

```
, (順序対)  + (和)  - (差)  / (商)  * (積)
% (余り)  ceil (切り上げ)  floor (切り下げ)
```

次の例では、定数 $a[1], a[2], a[3]$ に初期値 2, 4, 6 を設定しています。

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(index = i);
a[i] = 2 * i;
```

次の例では、制約式 $x_2 + x_4 + x_6 \leq 5$ を記述しています。制約式の左辺を定義するために偶数番目の項のみの和を取得しています。

```
Set S;
S = "1 2 3 4 5 6";
Element i(set = S);
Variable x(index = i);
sum(x[i], (i, i % 2 == 0)) <= 5;
```

次の例では、漸化不等式 $x_i \leq x_{i+1}$ を定義しています。

```
Set S = "1 2 3 4";
Element i(set = S);
Variable x(index = i);
x[i] <= x[i + 1], i != 4;
```

5.10 集合クラス Set

集合は Set というクラスで表現されます。以下の例では、自然数 1, 2, 3 を要素とする集合 S を定義しています。

```
Set S;
S = "1 2 3";
```

集合の要素間は、半角スペースで区切る必要があります。また、集合自体の定義と、構成要素の定義を同時に行う事ができます。以下の記述は上の記述と同じ意味です。

```
Set S = "1 2 3";
```

添字クラス Element と併用することで、変数 Variable を集合の要素ごとに設定できます（制約式 Constraint, 定数 Parameter, 整数変数 IntegerVariable, 式 Expression についても同様です）。以下の例では 3 個の変数 $y[1], y[2], y[3]$, 3 個の定数 $b[1], b[2], b[3]$ を定義しています。

```
Set S;
S = "1 2 3";
Element i(set = S);
Variable y(index = i);
Parameter b(index = i);
```

集合の要素には自然数だけでなく、文字列も使用することができます。以下の例では 2 個の整数変数 $z[p], z[q]$, 2 個の式 $g[p], g[q]$ を定義しています。

```
Set T;
T = "p q";
Element j(set = T);
IntegerVariable z(index = j);
Expression g(index = j);
g[j] = 2 * x[j] + 3 * y[j];
```

要素の文字列は必ずしも一文字である必要はありません。

```
Set T = "before after";
```

要素の文字列には半角英数字, 半角記号_と全角文字を使用できます。

```
Set T = "前_before 後_after";
```

集合の要素に文字列を使用した場合は, 対象を個別に記述する際に, 添字部分にダブルクォート"を用いる必要があります。

```
-1 <= z["p"] <= 2;
```

一括して記述する場合にはダブルクォートで囲んではいけません。

```
-1 <= z[j] <= 2;
```

集合の要素に自然数を用いる場合は, .. を用いる事で途中の要素を自動的に補間することが可能です。以下の二つの例はいずれも自然数 1 から 10 で構成される集合 S を定義しています。

```
Set S = "1 .. 10";
```

```
Set S = "1 2 3 4 5 6 7 8 9 10";
```

要素に文字列を用いる場合は, 上記の自動補間機能を用いることはできません。次の記述は誤りです。

```
Set T = "a .. k";
```

集合の要素は、モデルファイル内で定義する以外に、データファイルから与える方法もあります。以下は自然数 1, 2, 3 を要素とする集合 S の定義を、データファイル foo.dat から与える例です。

モデルファイル内

```
Set S;
```

foo.dat 内

```
S = "1 2 3";
```

集合クラスの構成要素は、明示的に定義しなくとも、モデルファイルやデータファイルの情報から自動的に定義されます。これを SIMPLE の**自動代入機能**と呼びます。以下の例では、自動代入機能により、集合 S の要素は 1, 2, 3 であると判断されます。

```
Set S;
Element i(set = S);
Parameter a(index = i);
a[1] = -1;
a[2] = -1;
a[3] = 1;
```

以下のように、データファイル (foo.dat) 内で a の値を定めた場合も同様です。

モデルファイル内

```
Set S;
Element i(set = S);
Parameter a(index = i);
```

foo.dat 内

```
a = [1] -1 [2] -1 [3] 1
```

部分集合を定義したい場合は、引数 superSet を用います。以下の例では集合 T が集合 S の部分集合であることを記述しています。

```
Set S;
Set T(superSet = S)
```

ある集合に対して定義された添字は、その部分集合に対しても自動的に定義されます。

添字を部分集合のみ（あるいは部分集合以外）で走らせたい場合は、集合と添字の包含関係を表す演算子<, >を利用します。以下の例では、定数 a[1], a[2] に-1 を、a[3] に 1 を設定しています。

```
Set S;
S = "1 2 3";
Set T(superSet = S);
T = "1 2";
```

```

Element i(set = S);
Parameter a(index = i);
a[i] = -1, i < T; // i が T に含まれる場合
a[i] = 1, i > T; // i が T に含まれない場合

```

多次元集合（要素の組の集合）を定義する場合には、引数 `dim` で次元を指定します。以下の例では、`I` の要素と `J` の要素の組を要素とする集合 `IJ` を定義し、二次元の添字をもつ変数 `x` を定義しています。`I` の要素と `J` の要素のすべての組み合わせについて変数を定義したいわけではない場合などに、多次元集合を使用します。

```

Set I;
Element i(set = I);
Set J;
Element j(set = J);
Set IJ(dim = 2, superSet = (I, J));
Element ij(set = IJ);
IJ = "a 1 b 2"; // 集合の要素を"a, 1"と"b, 2"とする
Variable x(index = ij); // x["a, 1"] と x["b, 2"] が定義される
x[i, j] >= 0, (i, j) < IJ; // x["a, 1"] と x["b, 2"] に下限を設定する

```

条件式から部分集合を取得するには、関数 `setOf` を使用します。`setOf` 関数の第一引数は添字、第二引数は条件式である必要があります。関数の返り値は集合です。なお、得られた集合について、通常の集合と異なり自動代入機能を利用することは出来ません。以下の例では、集合 `S` の中で条件 `a[i] > 0` を満たす要素のみから、集合 `T` を取得しています。

```

Set S = "1 2 3";
Set T;
Element i(set = S);
Parameter a(index = i);
a[1] = -1;
a[2] = -1;
a[3] = 1;
T = setOf(i, a[i] > 0); // 要素 3 のみからなる集合 T が作成される。

```

集合の要素数を取得するには、`card` 関数を使用します。`card` 関数の返り値は `int` 型です。以下の例では、整数 `n` に集合 `S` の要素数を格納しています。

```
int n = S.card();
```

集合には他にも様々な演算がありますが、使用頻度の低い用法は本マニュアルでは除外しています。詳細はお問い合わせください。

5.10.1 追加操作 add

```
add(追加される要素オブジェクト [, 条件式]);
```

- 集合 Set の操作 add() は集合に要素を追加します。
- add() の引数は、単一の要素を入れることも、添字を与えることも可能です。
- 引数に添字が与えられた場合、添字展開された要素が追加されます。
- 第 2 引数の条件式は省略可能です。

条件式を 2 番目の引数として与えると、引数である要素オブジェクトの展開される範囲が限定されます。

以下に add() を用いた例を示します。

```
Set S = "1 2 3";
Set T = "4 5 6";
Element i(set = S);
S.add(7); // S には 7 が追加される
T.add(i); // T には {1 2 3 7} が追加される
```

条件式を 2 番目の引数として与えると、引数である要素オブジェクトの展開される範囲が限定されます。

例えば以下の例では集合 U に追加する添字 i の範囲を $i < 3$ という条件で限定しています。

```
Set S = "1 2 3";
Element i(set = S);
Set U;
U.add(i, i < 3); // U には "1 2" が追加される
```

5.10.2 自動代入の禁止 lock()/unlock()

```
lock();
unlock();
```

集合の自動代入は集合のデータを明示的に与える必要が省かれ効率的に記述ができますが、一方で誤ったデータを与えた際、集合に予期せぬ変更が加わる場合があります。

例えば以下のように、モデルにとって悪影響を与える場合があります。

```
Set S = "1 2 3";
Element i(set = S);
Parameter c(index = i);
c = "[1] 7.2 [2] 8.2 [5] 1.2"; // 誤って添字 5 を与える
```



```
// これにより S の内容は "1 2 3 5" になってしまう

Variable x(index = i);

sum(x[i], i) >= 5; // 和の取られる範囲は "1 2 3" ではなく "1 2 5" となる.
```

そのような場合に有効なのが集合に対する lock() という操作です. 集合に対して lock() を呼び出すと, 以降の自動代入による要素の追加を禁じます.

例えば上記の例で lock() を記述すると lock() 以降で集合に要素が追加されるとエラーとなります.

```
Set S = "1 2 3";
S.lock(); // 集合をロック
Element i(set = S);
Parameter c(index = i);
c = "[1] 7.2 [2] 8.2 [5] 1.2"; // 誤って添字 5 を与える
// ここで S は "1 2 3 5" になってしまうのでエラーとなる
```

上記の様にデータのチェックが容易となるため特にモデル開発時に役立ちます.

操作 lock() の逆操作 (自動代入を許す) として unlock() という操作があります. 以下利用例です.

```
Set S = "1 2 3";
S.lock(); // 集合をロック
Element i(set = S);
Parameter c(index = i);
S.unlock();
c = "[1] 7.2 [2] 8.2 [5] 1.2"; // 誤って添字 5 を与える
// ここで S は "1 2 3 5" になってしまうが unlock() により許容される.
```

5.10.3 切断操作 slice

```
slice(次元 [, ..., 次元])
```

- Set の操作 slice() は, 多次元集合を指定の次元で切り出し, 新しい集合を構成します.
- 引数は 1 からはじまる整数で, Set の次元以下である必要があります.

集合の要素を構成する直積のメンバを左から数えた順番を表します. 以下に slice() を用いた例を示します.

```
Set a(dim = 3); // 次元が 3 となる集合 a を設定する
a = "1,1,1 1,3,1 b,2,1 1,3,1 1,4,a 1,1,1 1,5,1";
Set a1; // 次元が 1 となる集合 a1 を設定する
a1 = a.slice(1); // 集合 a1 は a の要素の 1 番目のメンバで構成される
```

```

a1.val.print()           // 結果: a1=(1 b)
Set a12(dim = 2);        // 次元が 1 となる集合 a1 を設定する
a12 = a.slice(1, 2);     // 集合 a12 は a の要素の 1, 2 番目のメンバで構成される
a12.val.print()          // 結果: a12=(1 1 1 3 b 2 1 4 1 5)
Set a121(dim = 3);
a121 = a.slice(1, 2, 1); // 要素の 1,2,1 番目のメンバで構成される集合
a121.val.print();        // 結果: a121=(1 1 1 1 3 1 b 2 b 1 4 1 1 5 1)
Set a13(dim = 2);
a13 = a.slice(1, 3);     // 要素の 1,3 番目のメンバで構成される集合
a13.val.print();         // a13=(1 1 b 1 1 a)

```

5.11 順序集合クラス OrderedSet

集合内の要素に順序が定められた順序集合は OrderedSet クラスで表現されます。集合の要素にわたるループを表現する場合には、この OrderedSet クラスが有用です。OrderedSet クラスは Set クラスの機能を全て有しており、加えて次の関数可以使用です。

- first 関数 順序集合の最初の要素を返す
- last 関数 順序集合の最後の要素を返す
- next 関数 次の要素を返す
- prev 関数 前の要素を返す
- position 関数 要素の位置（何番目かを意味する整数）を返す
- elementAt 関数 位置（何番目かを意味する整数）にある要素を返す

C++の関数としてのフォーマットは以下ようになります。

```

// OrderedSet のメンバ関数
Element first(); // 最初の要素を返す
Element last(); // 最後の要素を返す
Element next(const Element& i); // 要素 i の次の要素を返す
Element prev(const Element& i); // 要素 i の前の要素を返す
int position(const Element& i); // 要素 i の位置を返す
Element elementAt(int p); // 場所 p にある要素を返す

```

OrderedSet クラスを利用する事で、漸化式や漸化不等式を取り扱うことが可能です。

次の例では漸化不等式 $x_p \leq x_q, x_q \leq x_r, x_r \leq x_s$ を OrderedSet を利用して記述しています。

```

OrderedSet S = "p q r s";
Element i(set = S);
Variable x(index = i);
x[i] <= x[S.next(i)], i != S.last();

```

最後の条件式 $i \neq S.\text{last}()$ は $i == s$ の場合を除外するためです。上記の例では `next` 関数と `last` 関数を利用しましたが、以下のように `prev` 関数と `first` 関数を利用することもできます。

```
OrderedSet S = "p q r s";
Element i(set = S);
Variable x(index = i);
x[S.prev(i)] <= x[i], i != S.first();
```

集合の要素が整数ではない場合、上記のように `OrderedSet` を用いる必要があります。しかし集合の要素が整数の場合は、次のように `OrderedSet` を用いない記述も可能です。

```
Set S = "1 2 3 4";
Element i(set = S);
Variable x(index = i);
x[i] <= x[i + 1], i != 4;
```

同様に次の記述も可能です。

```
Set S = "1 2 3 4";
Element i(set = S);
Variable x(index = i);
x[i - 1] <= x[i], i != 1;
```

整数以外の要素からなる集合を利用する場合、条件式において $i + 1, i - 1$ 等の要素間の演算が使用できない事が、`OrderedSet` に頼らざるを得ない主な理由です。

次の例では、定数 $a[p], a[q], a[r]$ にそれぞれ 2, 4, 6 (2 ずつ増加) を設定します。

```
OrderedSet S = "p q r";
Element i(set = S);
Element j;
Parameter a(index = i);
for(j = S.first(); j < S; j = S.next(j)){
    a[j] = 2 * S.position(j);
}
```

以下のように記述しても同じ意味です。

```
OrderedSet S = "p q r";
Element i(set = S);
Element j;
Parameter a(index = i);
for(int p = 1; p < S.card() + 1; p++){
    j = S.elementAt(p);
```

```
a[j] = 2 * p;
}
```

集合の要素が整数である場合は、次のように `OrderedSet` を用いない記述も可能です。以下の例では、定数 `a[1]`, `a[2]`, `a[3]` にそれぞれ 2, 4, 6 (2 ずつ増加) を設定します。

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(index = i);
a[i] = 2 * i;
```

`Set` の操作の一部は `OrderedSet` でも利用できます。例えば `add` は `OrderedSet` に対して適用可能です。 `Set` から `OrderedSet` を構成する場合は操作 `add` を用いるのが簡便です。

```
Set S;
Element i(set = S);
S = "1 2 3";
OrderedSet T;
// i を T に追加する。これにより S と T は同じ要素で構成される。
T.add(i);
```

5.12 数列集合 Sequence

数列集合 `Sequence` は、等差数列が成す集合を表現するためのものです。引数に `from`, `to`, `by` を指定することで、`from` から `to` までの間に `by` 刻みの要素が作成されます。次の例では、1 から 9 までの 2 刻みの要素を作成しています。

```
Sequence S(from = 1, to = 9, by = 2);
// Set S = "1 3 5 7 9"; と同等
```

刻み幅が 1 でない大規模な集合を扱う際に有用です。ただし、数式集合 `Sequence` については集合 `Set` と異なり自動代入機能は利用できません。

数列集合 `Sequence` は順序集合 `OrderedSet` で利用可能な関数

```
// Sequence のメンバ関数
Element first();    // 最初の要素を返す
Element last();     // 最後の要素を返す
Element next(const Element& i); // 要素 i の次の要素を返す
Element prev(const Element& i); // 要素 i の前の要素を返す
int position(const Element& i); // 要素 i の位置を返す
Element elementAt(int p); // 場所 p にある要素を返す
```

を利用することができます。

5.13 条件式

条件式は、添字を含む制約式および代入文において、その添字の動く範囲を制限する機能です。

条件式は制約式や代入文の右側に記述します。この際、「, (半角コンマ)」で繋げます。書式は以下の通りです。

制約式, 条件式
代入文, 条件式

次の例では、定数 $a[1], a[2]$ に -1 を、 $a[3]$ に 1 を設定しています。

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(index = i);
a[i] = -1, i <= 2;
a[i] = 1, i >= 3;
```

条件式に使用することができる演算子は以下です。

- 等号 ==
- 等号付不等号 <=, >=
- 不等号 <, >
- 不一致 !=

次の例では、定数 $asum$ の値を、定数 $a[i]$ の中で 0 より大きいものの和 $\sum_{a_i > 0} a_i$ で定めています。

```
Set S;
Element i(set = S);
Parameter a(index = i);
Parameter asum;
asum = sum(a[i], (i, a[i] > 0));
```

不等号 <, > は添字に対する集合の所属有無を表現する演算子としても使用されます。

以下の例では、定数 $a[1], a[2]$ に -1 を、 $a[3]$ に 1 を設定しています。

```
Set S;
S = "1 2 3";
Set T(superSet = S);
T = "1 2";
Element i(set = S);
Parameter a(index = i);
a[i] = -1, i < T; // i が T に含まれる場合
a[i] = 1, i > T; // i が T に含まれない場合
```

条件式は `setOf` 関数を使って部分集合を構成する際にも使用されます。以下の例では、集合 S の中

で条件 $a[i] > 0$ を満たす要素のみから、集合 T を取得しています。

```
Set S = "1 2 3";
Set T;
Element i(set = S);
Parameter a(index = i);
a[1] = -1;
a[2] = -1;
a[3] = 1;
T = setOf(i, a[i] > 0); // 要素 3 のみからなる集合 T が作成される。
```

条件式同士を次のような演算子「!」、「&&」、「, (半角コンマ)」、「||」で連結することができます。それぞれ、not, and, and, or を意味します。次の例では、定数 $b[1], b[4]$ に -1 を、 $b[2], b[3]$ に 1 を設定しています。

```
Set S = "1 2 3 4";
Element i(set = S);
Parameter b(index = i);
b[i] = -1, (i <= 1 || i >= 4);
b[i] = 1, (i >= 2 && i <= 3);
```

最後の一行は、以下の各例のように記述する事もできます。

```
b[i] = 1, !(i <= 1 || i >= 4);
```

```
b[i] = 1, i >= 2, i <= 3;
```

条件式に変数や整数変数を用いることはできません。次の記述は誤りです。

```
Variable x;
Variable y;
3 * x + 2 * y == 0, x >= 0;
3 * x + 2 * y <= 0, x < 0;
```

条件式は「添字を含んだ」制約式や代入に対する機能です。添字を含んでいない場合は記述することができません。例えば以下のような記述は誤りです。

```
Parameter a;
Variable x;

// a が 2 以上であれば制約式  $x \geq 1$  を成立させる
// 不正な書き方
x >= 1, a >= 2;
```

上のような記述を実現したい場合は if 文を使います。

```

Parameter a;
Variable x;

// a が 2 以上であれば制約式  $x \geq 1$  を成立させる
if(a >= 2){
    x >= 1;
}

```

ここからは C++SIMPLE 上級者向けの説明です.

条件式同士を演算子「&&」や「, (半角コンマ)」で連結する場合, 左から順に解釈される点に注意してください.

例として, 以下のように記述された Parameter d に関して $d[i + 1] == d[i]$ となっている箇所のみ出力することを考えます.

```

Set S = "1 2 3 4 5 6";
Element i(set = S);
Parameter d(index = i);
d[1] = 5;
d[2] = 4;
d[3] = 2;
d[4] = 2;
d[5] = 2;
d[6] = 3;

```

以下の記述は $d[i + 1] == d[i]$ が $i + 1 < S$ より先に解釈されるため, 範囲外である $d[7]$ を参照してしまうことからエラーとなります.

```
simple_printf("d[%d] = d[%d] = %f\n", i + 1, i, d[i], d[i + 1] == d[i], i + 1 < S);
```

以下のように解釈の順番を逆にすることにより, 正しく表示ができます.

```
simple_printf("d[%d] = d[%d] = %f\n", i + 1, i, d[i], i + 1 < S, d[i + 1] == d[i]);
```

5.14 最小（大）値取得関数 min, max

最小値取得関数 min 及び最大値取得関数 max は, 添字付けされた定数式の中から最小（大）のものを返す関数です.

```

// 添字の範囲にわたる最小値
Parameter    min(定数式, 範囲指定並び)    // 戻り値は定数式

```

```
// 添字の範囲にわたる最大値
Parameter    max(定数式, 範囲指定並び)    // 戻り値は定数式
```

上のように記述した場合には, Nuorium Optimizer で最適化計算を行う前に値の取得を行います. このため全てのアルゴリズムで 사용할 ことが可能です.

5.15 数学関数

C++SIMPLE では次の演算と数学関数が定義されています. それぞれの意味はプログラミング言語 C/C++におけるものと 同 じです.

+	-	/	*		
sin	cos	tan	asin	acos	atan
sec	csc	cot	asec	acsc	acot
sinh	cosh	tanh	sech	coth	csch
atan2	hypot	erf			
exp	log	log10	pow	sqrt	
ceil	floor	fabs	fmod		

次の例では, 制約式 $4x^3 \leq 11$ を記述しています.

```
4 * pow(x, 3) <= 11;
```

累乗関数 pow を用いると, 例え次数が 2 であっても二次計画問題とはみなされず, 一般の非線形計画問題と認識されます. 二次計画問題専用のアルゴリズム asqp を利用する場合は累乗関数 pow を用いないでください.

バージョン 9 よりガウスの誤差関数 erf が追加されました. 誤差関数は次のように定義される関数です.

$$\operatorname{erf}(x) \equiv \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt, \quad x \in [-\infty, \infty]$$

第6章

制約充足問題ソルバ wcsp

6.1 wcsp を用いる場合の注意点

アルゴリズムとして制約充足問題ソルバ wcsp を用いる際には、wcsp は近似解法であり必ずしも厳密解が求まるわけではない点に注意してください。

また、他のアルゴリズムと異なり幾つかの制限があります。1つは**実数変数 Variable** を用いることができないこと。もう1つは**制約式や目的関数において小数点以下が切り捨てられる**ことです。以下で詳しく説明します。

wcsp は変数として 0-1 整数変数 (type = binary とした IntegerVariable) と離散変数 (DiscreteVariable) を用いて定式化します。連続変数や、0-1 整数変数でない整数変数は用いることが出来ません。以下、wcsp で用いることができる構成要素を説明します。

構成要素名	C++SIMPLE 内の名称	機能
目的関数	Objective	目的関数を表す
制約式	Constraint	制約式を表す
定数	Parameter	定数を表す
整数変数	IntegerVariable	整数変数を表す
範囲演算関数	sum	\sum に相当する
式	Expression	頻出する数式に対して、簡単な別の表現を与える
添字	Element	添字を表す
集合	Set	添字の動く範囲を表す
順序集合	OrderedSet	要素間に順序を持つ集合を表す
数列集合	Sequence	等差数列からなる集合を表す
条件式		制約式や代入文を制限する
数学関数	exp, sin, cos, log ...	数学関数を表す
離散変数	DiscreteVariable	離散変数を表す
重複不能関数	alldiff	重複不能を表す
選択関数	selection	限定選択を表す
ハード制約関数	hardConstraint	ハード制約を表す
セミハード制約関数	semiHardConstraint	セミハード制約を表す
ソフト制約関数	softConstraint	ソフト制約を表す
ブール関数	Boolean	制約式を引数として、0-1 を返す

冒頭で述べた小数点以下が切り捨てられる点について説明します。wcsp は制約式や目的関数をペナ

ルティとして認識し、ペナルティが小さくなる答えを探すアルゴリズムです。Nuorium Optimizer の内部でこのペナルティを評価する際に小数点以下を切り捨て、整数として評価を行います。

例えば、 $2x + 0.9y \geq 4.5$ という制約式があったとします。このとき、 $x = 1, y = 1$ における制約式の違反量は $4.5 - (2 \cdot 1 + 0.9 \cdot 1) = 1.6$ に対して小数点以下の切り捨てが適用され、1 となります。

このように、「評価した後に切り捨てが適用される」というのが特徴です。定式化の段階で小数点以下も考慮する必要がある目的関数や制約式に関しては、該当する式（制約式ならば両辺）に大きな値（1000 ならば小数第三桁まで有効になる）をかけておく必要があります¹。

6.2 目的関数クラス Objective

wcsp 利用時にも目的関数クラス Objective を用いますが、引数 target で目標値を指定する必要があります。目的関数が target で指定された値を上回っている分（最大化問題の場合は下回っている分）が「ソフト制約としての制約違反量」に加算されます。次の例では、target に 10 を設定しています。

```
Objective f(type = minimize, target = 10);
```

target の初期設定は 0 で、明示的に target を指定しない場合は target は 0 であると解釈されます。すなわち、次の二つは同じ意味です。

```
Objective f(type = minimize, target = 0);
```

```
Objective f(type = minimize);
```

target の値は、定数 Parameter から設定する事もできます。

```
Parameter p;
p = 10;
Objective f(type = minimize, target = p);
```

target の値は、求解オプション defaultObjectiveTarget で指定することもできます。

```
options.defaultObjectiveTarget = 5;
```

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。

目的関数の重みは求解オプション defaultObjectiveWeight で指定します。次の例では、目的関数の重みに 5 を指定しています。

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

求解オプション defaultObjectiveWeight の初期値は 1 です。

¹ただし値が極端に大きくならないよう注意してください。

6.3 制約式クラス Constraint

wcsp 利用時には、全ての制約式は次の3つの種類に分類されます。

- ハード制約式
- セミハード制約式
- ソフト制約式

ハード制約式とは、最も優先して満たすべき制約式（必ず満たさなければならない制約式）のことです。

セミハード制約式とは、ハード制約式の次に優先して満たすべき制約式のことです。通常、必ず満たさなければならない制約式の一部をセミハード制約式とし、実行不可能性の原因をセミハード制約に押し付けるという使い方をします。

ソフト制約式は、優先度がもっとも低く、必ずしも満たす必要はないが、できるだけ満たして欲しい制約式のことです。その際、ソフト制約式は、各制約式の違反量からペナルティ量を計算し、その総ペナルティ量が最も小さくなることをもってできるだけ制約式を満たしたと解釈します²。

初期設定では、全ての制約式はハード制約式として扱われます。

6.3.1 ハード制約関数 hardConstraint

hardConstraint 関数を使用すると、その行以降に出現した制約式は全てハード制約として扱われます。

```
hardConstraint();  
sum(a[i] * x[i], i) == 3; // ハード制約となる
```

hardConstraint 関数、semiHardConstraint 関数、softConstraint 関数が混在する場合は、後の行に記述されたものが優先されます。

6.3.2 セミハード制約関数 semiHardConstraint

semiHardConstraint 関数を使用すると、その行以降に出現した制約式は全てセミハード制約として扱われます。

```
semiHardConstraint();  
sum(a[i] * x[i], i) == 3; // セミハード制約となる
```

hardConstraint 関数、semiHardConstraint 関数、softConstraint 関数が混在する場合は、後の行に記述されたものが優先されます。

²制約式の違反量からのペナルティ量の計算方法は後述します。

6.3.3 ソフト制約関数 softConstraint

softConstraint 関数を使用すると、その行以降に出現した制約式をソフト制約として扱います。さらに、softConstraint 関数へ与えた引数（求解オプション）により制約式の違反量の扱いの設定もできます。

一般的なソフト制約への求解オプションの設定は、次のように 3 つの引数の値により行われます。

```
softConstraint(int weight, double a, double b);
```

ただし、引数は全て非負で設定します。この時、ソフト制約の違反量を x とすると、そのソフト制約のペナルティ量 p は、

```
p = (a * x * x + b * x) * weight;
```

という式により定義します。

また、softConstraint 関数の第 2 引数と第 3 引数は省略することができ、次のような規則により解釈されます。

- 第 2, 3 引数の省略

```
softConstraint(weight) ==> softConstraint(weight, 0, 1)[p = x * weight と等価]
```

- 第 3 引数の省略

```
softConstraint(weight, a) ==> softConstraint(weight, a, 0)[p = a * x * x * weight  
と等価]
```

即ち、

```
softConstraint(1); // softConstraint(1, 0, 1) と等価  
softConstraint(1, 2); // softConstraint(1, 2, 0) と等価
```

となります。

softConstraint 関数は、第 2, 3 引数を省略する時のみ引数に -1, -2 を設定できます。それぞれハード制約、セミハード制約と解釈されます。

```
softConstraint(-1);  
sum(a[i] * x[i], i) == 3; // ハード制約となる
```

```
softConstraint(-2);  
sum(a[i] * x[i], i) == 3; // セミハード制約となる
```

hardConstraint 関数、semiHardConstraint 関数、softConstraint 関数が混在する場合は、後の行に記述されたものが優先されます。

バージョン 8, 9 の softConstraint 関数は、softConstraint(int weight) という書き方のみ可能でした。バージョン 10 から、上記のような形式になっています。なお、バージョン 8, 9 で作成した wcsp モデルにおいて、softConstraint(weight) のような記述は、バージョン 10 の省略規則により同一なものとして計算されます。そのため、バージョン 8, 9 で作成したモデルの softConstraint 関数

はバージョン 10 以降でも変更する必要はありません。

6.3.4 求解オプション defaultConstraintWeight

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。

```
sum(a[i]*x[i], i) == 3;    // 重み 12 のソフト制約となる
options.defaultConstraintWeight = 12;
sum(b[i] * x[i], i) <= 20; // 重み 12 のソフト制約となる
sum(c[i] * x[i], i) <= 100; // 重み 12 のソフト制約となる
sum(d[i] * x[i], i) >= 15; // 重み 12 のソフト制約となる
```

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。

求解オプション 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。

```
sum(a[i] * x[i], i) == 3;    // 重み 12 のソフト制約となる
options.defaultObjectiveWeight = 12;
sum(b[i] * x[i], i) <= 20; // 重み 12 のソフト制約となる
hardConstraint();
sum(c[i] * x[i], i) <= 100; // ハード制約となる
semiHardConstraint();
sum(d[i] * x[i], i) <= 100; // セミハード制約となる
softConstraint(5);
sum(e[i] * x[i], i) >= 15; // 重み 5 のソフト制約となる
```

6.4 整数変数クラス IntegerVariable

wcsp 使用時には、0-1 整数変数のみ利用が可能です。即ち IntegerVariable を利用する際には、必ず引数に type=binary を付ける必要があります。通常の整数変数を用いることはできません。通常の整数変数を利用したい場合は、離散変数 DiscreteVariable を用いて記述する必要があります。例えば、 $1 \leq x \leq 10$ を満たす整数変数を定めたい場合、通常の数理最適化モデルでは次のように記述します。

```
IntegerVariable x;
1 <= x <= 10;
```

一方、離散変数を用いた場合、以下のような記述になります。

```
Set S = "1 .. 10";
DiscreteVariable x(dom = S);
```

6.5 離散変数クラス DiscreteVariable

離散変数クラス `DiscreteVariable` は、`wcsp` でのみ利用可能な構成要素です。必ず引数に定義域 `dom` を持つ必要があります。引数 `dom` が集合 `Set`、順序集合 `OrderedSet`、数列集合 `Sequence` を指すことにより、その離散変数が取り得る値の範囲を定めます。

次の例では、1 から 3 までの値を取る離散変数 `x` を定義しています。

```
Set S = "1 2 3";
DiscreteVariable x(dom = S);
```

離散変数 `DiscreteVariable` の定義域は、必ずしも整数である必要はありません。次の例では、`open` あるいは `closed` のいずれかを取る離散変数 `y` を定義しています。

```
Set S = "open closed";
DiscreteVariable y(dom = S);
```

離散変数 `DiscreteVariable` は添字を取る事もできます。次の例では `open` 又は `closed` を取る離散変数 `y[1]`, `y[2]`, `y[3]` を定義しています。

```
Set S = "open closed";
Set T = "1 2 3";
Element i(set = T);
DiscreteVariable y(dom = S, index = i);
```

6.6 重複不能関数 alldiff

重複不能関数 `alldiff` は、添字付きの離散変数 `DiscreteVariable` を引数に取り、「それぞれの値が全て異なる」という制約を与えることができます。

次の例では、添字と定義域が同じ集合を対象とする、離散変数 `y` を考えます。`alldiff` 関数により、`y[1]`, ..., `y[10]` は全て異なる値 (1, ..., 10 のどれか) を取ります。

```
Set S = "1 .. 10";
Element i(set = S);
DiscreteVariable y(dom = S, index = i);
alldiff(y[i], i);
```

第二引数の添字は、省略することもできます。

```
alldiff(y[i]);
```

alldiff 関数の引数には、条件式を与えることもできます。これにより、alldiff 関数が作用する範囲を制限することができます。次の例では、 $y[1]$, ..., $y[5]$ までは、全て異なる値を取るようになっています。

```
Set S = "1 .. 10";
Element i(set = S);
DiscreteVariable y(dom = S, index = i);
alldiff(y[i], (i, i <= 5));
```

条件式を付与した場合は、第二引数の添字を省略することはできません。例えば、次の例は誤りです。

```
alldiff(y[i], i <= 5);
```

次の例では、 $y[1]$, $y[2]$, $y[3]$ が重複せずに a, b, c のいずれかを取るようになっています。

```
Set S = "1 .. 10";
Set T = "a b c";
Element i(set = S);
DiscreteVariable y(dom = T, index = i);
alldiff(y[i], (i, i <= 3));
```

6.7 選択関数 selection

選択関数 selection は、添字つき 0-1 整数変数の中で一つだけを 1 に固定したい場合に用います。同様の記述は sum 関数を用いる事でも可能ですが、wcsp を利用する際には selection 関数を用いた方が効率的です³。

次の例では、3 つの 0-1 整数変数 $z[1]$, $z[2]$, $z[3]$ のうち一つだけを 1 にするよう指定しています。

```
Set S = "1 2 3";
Element i(set = S);
IntegerVariable z(type = binary, index = i);
selection(z[i], i);
```

第二引数の添字は省略することもできます。

```
selection(z[i]);
```

sum 関数を利用した場合、次のようになります。

```
sum(z[i], i) == 1;
```

selection 関数の引数には、条件式を指定することもできます。次の例では、 $z[1]$, $z[2]$ のうち一つだけを 1 にするよう指定しています。

³selection 関数を用いた場合、内部的には複数の 0-1 整数変数を用意する替わりに一つの離散変数を用意するため、内部処理が高速化されます。また、実行時に標準出力に出力される NUMBER_OF_VARIABLES の値は、内部的な変数の数になります。

```
Set S = "1 2 3";
Element i(set = S);
IntegerVariable z(type = binary, index = i);
selection(z[i], (i, i <= 2));
```

引数に条件式を指定する場合には、第二引数の添字を省略することはできません。例えば、次の例は誤りです。

```
selection(z[i], i <= 2);
```

6.8 ブール関数 Boolean

ブール関数 Boolean は、引数に与えた制約式に対して、その真偽を判定し 0（偽の場合）か 1（真の場合）を返す関数です。

なお、macOS 版 Nuorium Optimizer ではブール関数には対応していません。

```
Boolean(制約式); // 0 か 1 を返す
```

次の例では、定義域が 1 から 4 の離散変数 $x[1], \dots, x[10]$ を考えます。以下の式は、 $x[1], \dots, x[10]$ の中で、3 より大きい値を取る事ができるのは、最大でも一つであることを示しています。

```
Set S = "1 .. 10";
Set T = "1 2 3 4";
Element i(set = S);
DiscreteVariable x(dom = T, index = i);
sum(Boolean(x[i] >= 3), i) <= 1;
```

次の例では、3 人の工員 ryu, ken, guy に割り振られる仕事 a, b, c, d（離散変数 $x[a], x[b], x[c], x[d]$ ）を定義し、ken に割り振られる仕事の数は 2 つであると定めています。

```
Set Workers = "ryu ken guy";
Set Tasks = "a b c d";
Element j(set = Tasks);
DiscreteVariable x(dom = Workers, index = j);
sum(Boolean(x[j] == "ken"), j) == 2;
```

集合の要素が文字列である場合は、ダブルクォート"で囲む必要があります。

6.9 最小（大）値取得関数 min, max

wcsp 使用時には、最小値取得関数 min 及び最大値取得関数 max は、第一引数を定数式ではなく式にすることもできます。


```
// 添字の範囲にわたる最小値
Expression min(式, 範囲指定並び) // 戻り値は一般の式
// 添字の範囲にわたる最大値
Expression max(式, 範囲指定並び) // 戻り値は一般の式
```

ここでいう「範囲指定並び」は、関数の適用範囲となる添字の範囲です。添字そのもの、あるいは条件づけられた添字が入ります。

第一引数が式の場合には、内部で wcsp 専用の特別な処理を行うため良好なパフォーマンスが期待されます。

次の例では施設配置問題の「最寄りの施設に収容される」という制約を min 関数を用いて表現しています。

```
Set Mesh; // メッシュ集合
Element i(set = Mesh);
Set Facility; // 施設集合
Element j(set = Facility);

// メッシュ i が施設 j に収容されるならば 1 そうでないならば 0
IntegerVariable x(type = binary, index = (i, j));
// 施設 j が建設されるならば 1 されないならば 0
IntegerVariable y(type = binary, index = j);
// メッシュ i から施設 j までの距離
Parameter dist(index = (i, j));

// 制約, 各メッシュは 1 つの施設に対応される
selection(x[i, j], j);

// メッシュ i から収容される施設までの距離
Expression res_dist(index = i);
res_dist[i] = sum(x[i, j] * dist[i, j], j);

// メッシュから収容される施設は, 建設された施設の中では
// 最寄のものとする
Parameter M; // 十分大きい数
M = 1000000;
res_dist[i] <= min((1 - y[j]) * M + dist[i, j], j);
```

```
// 以下は min 関数を用いないで定式化する場合の記述方法
// res_dist[i] <= (1 - y[j]) * M + dist[i, j];
```

wcsp 以外のアルゴリズム選択時で、式に対し min 関数や max 関数を用いるような定式化を行いたい場合には、上記の例題記述においてコメントで示されているように非線形な記述を用いない等価な書き換えが存在します。min/max 関数の定式化や求解について、より詳細には nuopt-support@ml.msi.co.jp までお問い合わせください。

6.10 カウント関数 count

カウント関数 count は、条件を満たす式の数を取得します。

添字付き変数を含んだ制約式に対して制約式を満たす添字の個数を返す関数です。定数 Parameter のみを含んだ式に対しては適用できません。

```
// 添字の範囲にわたる最小値
Expression count(制約式, 範囲指定並び) // 戻り値は一般の式
```

ここでいう「範囲指定並び」には、count 関数の適用範囲となる添字を指定します。添字に条件をつけることはできないため注意してください。

count 関数によって、中間変数を用いることなく個数を数え上げることができるので問題規模の増加を防ぐことができます。目的関数や softConstraint で用いる場合にはメタヒューリスティクスの性質によりノイズを加えることによって速度を向上させることが可能です。以下簡単に説明します。

```
IntegerVariable x(index = i, type = binary);

Objective obj(type = minimize);
obj = count(x[i] == 1, i);
```

上記のような問題の場合にノイズの導入は有効です。以下のように目的関数を書き換えます。

```
IntegerVariable x(index = i, type = binary);
Parameter rand(index = i);
Parameter M;
Objective obj(type = minimize);
obj = count(x[i] == 1, i) * M + sum(x[i] * rand[i], i);
```

上記パラメータ rand は適当な乱数を想定しています。M は目的関数の第二項が第一項に影響を及ぼさないようにするためのスケール値です。このようにすると大幅に速度向上する場合がありますのでお試しください。

第7章

資源制約付きスケジューリング問題 ソルバ rcpsp

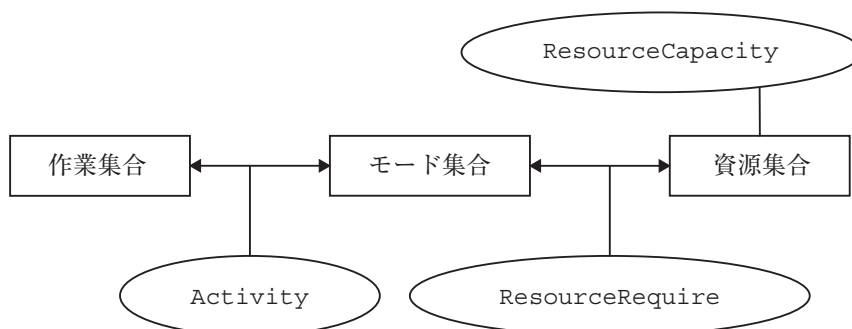
資源制約付きスケジューリング問題とは

- 幾つかの作業が存在し、一定の資源下でそれらの最後の作業の完了時刻を最小化する問題
- 納期のある幾つかの作業が存在し、一定の資源下でそれらの納期遅れを最小化する問題

のことを指します。Nuorium Optimizer では資源制約付きスケジューリング問題ソルバ rcpsp を用いてこれらの問題を解く事ができます。

7.1 rcpsp の構成要素

資源制約付きスケジューリング問題ソルバ rcpsp を利用する際には、必ず次の3つの構成要素 Activity, ResourceRequire, ResourceCapacity を定義しなければなりません。この3つの構成要素の関係を表すと、以下のような図になります。



作業集合は、「必ず実施しなければならない作業」から構成される集合です。作業集合の要素には、実施する必要の無い作業が含まれてはいけません。モード集合は、「作業に対する対処方法」から構成される集合です。「作業に対する対処方法」の事を、rcpsp ではモードと呼びます。資源集合は、「モードの利用に必要な資源」から構成される集合です。rcpsp を利用する際には、まずこの3種類の集合を定義する必要があります。

次に、「どの作業をどのモードで処理するか」に相当する変数 Activity を定義します。rcpsp が決定するのは、この Activity の値です。さらに、「各モードはどの資源をどの程度必要とするか」に相当する定数 ResourceRequire を定めます。最後に、「資源はどれだけ利用できるか」に相当する定数 ResourceCapacity を定めます。

なお、rcpsp では完了時刻最小化問題と、納期遅れ最小化問題を扱うことができます。どちらを扱うかは、目的関数で指定します。

rcpsp で利用することのできる構成要素は以下の通りです。

構成要素名	C++SIMPLE 内の名称	機能
目的関数	Objective	目的関数を表す
制約式	Constraint	制約式を表す
定数	Parameter	定数を表す
範囲演算関数	sum	\sum に相当する
式	Expression	頻出する数式に対して、簡単な別の表現を与える
添字	Element	添字を表す
集合	Set	添字の動く範囲を表す
順序集合	OrderedSet	要素間に順序を持つ集合を表す
数列集合	Sequence	等差数列からなる集合を表す
条件式		制約式や代入文を制限する
ソフト制約関数	softConstraint	ソフト制約を表す
ブール関数	Boolean	制約式を引数として、0-1 を返す
アクティビティ	Activity	必要な作業がどのモードを用いるかという変数
必要資源	ResourceRequire	モードの利用に必要な資源を表す
資源供給量	ResourceCapacity	利用可能な資源の限界値を表す
モード順序関数	modeOrder	モード順序を同一に設定する
アクティビティ固定関数	fixActivity	アクティビティを固定する
アクティビティ固定解除関数	unfixActivity	アクティビティの固定を解除する

以降、rcpsp でのみ利用可能な構成要素，あるいは rcpsp で用いる場合に注意を要する構成要素に関してのみ説明します。

7.2 目的関数クラス Objective

rcpsp では，最後の作業の完了時刻，納期遅れの 2 種類が目的関数 Objective に設定出来ます。自動的に最小化問題として扱われ，最大化問題として記述する事はできません。

最後の作業の完了時刻最小化問題を扱うには，次のように定めます

```
Objective f;
f = completionTime;
```

納期遅れ最小化問題を扱うには，次のように定めます。

```
Objective f;
f = tardiness;
```

最後の作業の完了時刻最小化問題を扱う場合は，目的関数に重みを設定することができます。目的

関数の重みは求解オプション `defaultObjectiveWeight` で指定します。次の例では、目的関数の重みに 5 を指定しています。

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

求解オプション `defaultObjectiveWeight` の初期値は 1 です。

納期遅れ最小化問題を扱う場合は、目的関数に重みを設定することはできません。

7.3 制約式クラス Constraint

rcpsp を利用する際には、以下のようなものが制約式として扱われます。

- 先行制約
- 直前先行制約
- Activity の要素による制約
- 同一モード順序選択関数による制約
- 固定関数による制約

取り扱う問題が最後の作業の完了時刻最小化問題（完了時刻最小化問題）の場合、制約式に重みを設定することができます。ハード制約、セミハード制約を設定することはできません。制約式に対してソフト制約を設定するには、`softConstraint` 関数あるいは求解オプション `defaultConstraintWeight` を利用します。

納期遅れ最小化問題を扱う場合には、制約式に重みを設定することはできず、全ての制約がハード制約として扱われます。

7.4 アクティビティクラス Activity

どの作業をどのモードで行うかを定めるアクティビティは、Activity で表現されます。Activity は rcpsp 利用時の変数に相当します。モード集合は、引数 `mode` で与えられます。

次の例では 4 つの作業 a, b, c, d に対するアクティビティ `x[a]`, `x[b]`, `x[c]`, `x[d]` を定めています。それぞれの作業にはモード 1, 2, 3 のいずれかが割り当てられます。

```
Set A = "a b c d";
Set M = "1 2 3";
Element i(set = A);
Activity x(index = i, mode = M);
```

納期遅れ最小化問題を扱う場合には、各 Activity に対する納期（定数 Parameter で表現されます）を引数 `duedate` で指定する必要があります。次の例では、4 つの作業 a, b, c, d に対して、納期 3, 5, 10, 7 を設定しています。

```
Set A = "a b c d";
Set M = "1 2 3";
```

```

Element i(set = A);
Parameter due(index = i); // 納期を示す定数
Activity x(index = i, mode = M, duedate = due[i]);
due["a"] = 3;
due["b"] = 5;
due["c"] = 10;
due["d"] = 7;

```

各作業に対して割り当て可能なモード集合が異なる場合は、引数 `mode` にモード集合族を与えます。次の例では、作業 `a` はモード 1, 2 作業 `b` はモード 1, 3 作業 `c` はモード 2 作業 `d` はモード 3 を取ることができます。

```

Set A = "a b c d";
Set M = "1 2 3";
Set M2(index = i); // モード集合族
M2["a"] = "1, 2";
M2["b"] = "1, 3";
M2["c"] = "2";
M2["d"] = "3";
Activity x(index = i, mode = M2[i]);

```

7.4.1 先行制約, 直前先行制約

先行制約とは、ある作業が必ず別の作業より先に実施されていなければならない、という制約のことです。先行制約は、アクティビティ `Activity` 間の不等式 `<` で表現されます。次の例では、作業 `a` は作業 `b` に優先することを記述しています。

```

Set A = "a b c d";
Activity x(index = i, mode = M);
x["a"] < x["b"];

```

先行制約の後ろには、条件式を付ける事もできます。次の例では、作業 `a, b, c` は作業 `d` に優先することを記述しています。

```

Set A = "a b c d";
Activity x(index = i, mode = M);
x[i] < x["d"], i != "d";

```

先行制約の後ろには、先行する期間を定数 `Parameter` で指定できます。以下の例では、作業 `a` は作業 `b` に 2 期間先行することを記述しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
Parameter p = 2;
x["a"] < x["b"], p;
```

簡略して、次のように書くこともできます。

```
x["a"] < x["b"], 2;
```

直前先行制約は、特定の資源を消費する状況ではある作業が別の作業の直後に来る、という制約を表現します。直前先行制約は、アクティビティ Activity 間の不等式 \ll で表現されます。直前先行制約は、完了時刻最小化問題でのみ使用することができます。

次の例では、作業 a, b いずれも資源 X を用いる場合（どちらも資源 X が必要なモードを取得した場合）には作業 a は作業 b に優先することを記述しています。

```
x["a"] << x["b"], "X";
```

7.4.2 Activity の要素

以下の要素の定数倍を足し合わせて一般の制約式を記述することができます。

```
Activity.startTime // 作業の開始時刻
Activity.endTime // 作業の終了時刻
Activity.processTime // 作業の所要期間
Boolean(Activity == 文字列) と Activity.startTime との積
Boolean(Activity == 文字列) と Activity.endTime との積
```

次の例では、作業 a の所用期間を 2 以下と定めています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
x["a"].processTime <= 2;
```

また Activity には、全てのアクティビティに対して先行するアクティビティ `sourceActivity`、全てのアクティビティに対して後続するアクティビティ `sinkActivity` が、各々自動的に定義されます。

7.4.3 初期値の設定

探索における Activity の初期値を明示的に与える事が出来ます。rcpsp の初期値として与える事の出来るものは、以下の 2 つです。

```
// 処理モード：
Activity = 文字列[, 条件式]
Activity = Parameter[, 条件式]
// 作業リスト：
Activity.order = 整数値
Activity.order = Parameter
```

上記 2 つは、`solve()` がコールされる前に記述します。

作業リストとは、全作業の順列であり、スケジュールが生成される際の基になるものです。rcpsp では、この順序に従って、開始時刻が順に決定されています。ただし、以下の点にご注意ください。

1. 初期値として与えているものは、作業リスト内の順番であり、初期化は全てのアクティビティに対して行わなければならない
2. 先行制約「作業 $i <$ 作業 j 」が存在する場合には、作業リストの中で、 i は j よりも先に位置しなければならない
3. 直前先行制約で関連づけられた作業の集合は、作業リストの中で連続して現れなければならない

7.5 必要資源クラス ResourceRequire

各モードに対する必要資源の量は `ResourceRequire` クラスで設定します。`ResourceRequire` は rcpsp 利用時に必要な定数の一つです。モード集合が引数 `mode` で、資源集合が引数 `resource` で与えられます。また、モード開始時からの経過時間を表す経過時間集合が引数 `duration` で与えられます。これら 3 つの引数は全て指定する必要があります。

次の例では、モード集合 M 、資源集合 R 、経過時間集合 D に対する必要資源 `req` を定義しています。

```
Set M; // モード集合
Set R; // 資源集合
Set D; // 経過時間集合
ResourceRequire req(mode = M, resource = R, duration = D);
```

次の例では、モード `aonly`, `bonly`, `both` それぞれに対して必要な資源 a , b を定めています。モードは全て期間 1 で終わり、モード `aonly` は資源 a が 1 期間、モード `bonly` は資源 b が 1 期間、モード `both` は資源 a , b の両方が 1 期間必要であることを示しています。

```
Set M = "aonly bonly both";
Set R = "a b";
Set D = "1"
ResourceRequire req(mode = M, resource = R, duration = D);
req["aonly, a, 1"] = 1;
req["aonly, b, 1"] = 0; // 記述しなくても良い
req["bonly, a, 1"] = 0; // 記述しなくても良い
```



```
req["bonly, b, 1"] = 1;
req["both, a, 1"] = 1;
req["both, b, 1"] = 1;
```

必要資源 ResourceRequire の値は、何も設定しない場合 0 が設定されます。上記の例では、特に設定する必要の無い行が二行あります。

初期設定値を 0 以外の値にするには、引数 defaultval を用います。次の例では、初期設定値を 1 にしているため、上記の例で 0 を設定していた箇所のみ設定する必要があります。

```
Set M = "aonly bonly both";
Set R = "a b";
Set D = "1"
ResourceRequire req(mode = M, resource = R,
    duration = D, defaultval = 1); // 初期値を 1 にした
req["aonly, b, 1"] = 0;
req["bonly, a, 1"] = 0;
```

次の例では、モード aonly は資源 a が 3 期間、モード bonly は資源 b が 3 期間、モード both は資源 a, b の両方が 1 期間必要であることを示しています。

```
Set M = "aonly bonly both";
Set R = "a b";
Set D = "1 2 3" // 期間を表す
Element i(set = D);
ResourceRequire req(mode = M, resource = R, duration = D);
req["aonly, a", i] = 1, 1 <= i <= 3;
req["bonly, b", i] = 1, 1 <= i <= 3;
req["both, a, 1"] = 1;
req["both, b, 1"] = 1;
```

7.6 資源供給量クラス ResourceCapacity

各資源の利用可能限界値を意味する資源供給量クラスは、ResourceCapacity で表現されます。資源集合が引数 resource で、スケジューリング全体の期間を表す期間集合が引数 timeStep で表現されます。どちらの引数も必要です。

次の例では、資源集合 R、期間集合 T に対する資源供給量 cap を定義しています。

```
Set R; // 資源集合
Set T; // 期間集合
ResourceCapacity cap(resource = R, timeStep = T);
```

次の例では、期間 0 から 10 に対して、資源 a, b はいずれも毎日 1 だけ利用可能であることを記述しています。期間集合は 0 始まりの集合でなければなりません。

```
Set R = "a b"; // 資源集合
Set T = "0 .. 10"; // 期間集合
Element j(set = T);
ResourceCapacity cap(resource = R, timeStep = T);
cap["a", j] = 1, 1 <= j <= 10;
cap["b", j] = 1, 1 <= j <= 10;
```

資源供給量の初期設定値は 0 です。

資源供給量には重みを設定する事ができます。重みは引数 `weight` で与えます。次の例では、一律に重み 10 を設定しています。

```
Set R; // 資源集合
Set T; // 期間集合
ResourceCapacity cap(resource = R, timeStep = T, weight = 10);
```

引数 `weight` には定数 `Parameter` を与える事もできます。次の例では、資源 a に対する資源供給量には重み 10 を、資源 b に対する資源供給量には重み 20 を与えています。

```
Set R = "a b"; // 資源集合
Set T = "0 .. 10"; // 期間集合
Element i(set = R);
Element j(set = T);
Parameter w(index = i);
w["a"] = 10;
w["b"] = 20;
ResourceCapacity cap(resource = R, timeStep = T, weight = w[i]);
```

7.7 モード順序関数 modeOrder

モード順序関数 `modeOrder` は `Activity` を引数に取り、`Activity` のモード順序が同一である、という制約を表現します。

次の例は、作業 a がモード 1 を取った場合には作業 b はモード 2 を取る事。また、作業 a がモード 3 を取った場合は、作業 b はモード 1 を取ることを記述しています。

```
Set A = "a b";
Element i(set = A);
Set M(index = i);
M["a"] = "1 3";
```

```
M["b"] = "2 1";
Activity x(index = i, mode = M[i]);
modeOrder(x["a"]) == modeOrder(x["b"]);
```

次の例は、作業 a と作業 b のモードが同じであることを記述しています。但し、この記述は作業 a, b に対するモード集合が同一でなければできません。

```
Set A = "a b c d";
Element i(set = A);
Activity x(index = i, mode = M);
modeOrder(x["a"]) == modeOrder(x["b"]);
```

Activity の添字には、条件式を付与することもできます。次の例は、作業 b 以外の全ての作業のモードが作業 a のモードと同じであることを記述しています。

```
Set A = "a b c d";
Element i(set = A);
Activity x(index = i, mode = M);
modeOrder(x[i]) == modeOrder(x["a"]), i!="a", i!="b";
```

7.8 アクティビティ固定関数 fixActivity

rcpsp における変数である、Activity, Activity.startTime, Activity.endTime の値は、アクティビティ固定関数 fixActivity を用いる事で、直前に代入されている値で固定する事が出来ます。

次の例では、作業 a の開始時刻を 5 に固定しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
x["a"].startTime = 5;
fixActivity(x["a"].startTime);
```

次の例では、作業 b 以外の全ての作業の終了時刻を 10 に固定しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
x[i].endTime = 10, i != "b";
fixActivity(x[i].endTime, i != "b");
```

fixActivity 関数の第二引数で、重みを設定する事ができます。次の例では、作業 a の開始時刻を 5 に固定し、その重みを 100 に設定しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
```

```
x["a"].startTime = 5;
fixActivity(x["a"].startTime, 100);
```

7.9 アクティビティ固定解除関数 unfixActivity

アクティビティ固定解除関数 `unfixActivity` を用いることで、アクティビティ固定関数 `fixActivity` で固定された内容を解除することができます。

次の例では、固定した作業 a の開始時刻を解除しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
x["a"].startTime = 5;
fixActivity(x["a"].startTime);
unfixActivity(x["a"].startTime);
```

次の例では、固定した作業 b 以外の終了時刻を解除しています。

```
Set A = "a b c d";
Activity x(index = i, mode = M);
x[i].endTime = 10, i != "b";
fixActivity(x[i].endTime, i != "b");
unfixActivity(x[i].endTime, i != "b");
```

7.10 資源制約付きスケジューリング問題の重みの設定

rcpsp の利用時には以下に対して重みを設定する事が可能です。

- 目的関数
- 資源の利用可能量
- 制約式

重みの値は、正の整数値を設定する必要があります。具体的な設定方法に関しては、それぞれの節を参照してください。

最後の作業の完了時刻最小化問題を扱う際にはソフト制約のみ、納期遅れ最小化問題を扱う際にはハード制約のみ扱うことができます。

7.11 資源制約付きスケジューリング問題記述例

ここでは、資源制約付きスケジューリング問題の一種である人員スケジューリング問題を、C++SIMPLE を用いて記述する方法を紹介します。

具体的には、次のような人員スケジューリング問題を考えます。

■ 例題 1 全体の作業完了時刻最小化

6つの仕事 (1, ..., 6) を A, B, C の3人に割り振ろうとしている。各人は同時に2つ以上の仕事はできず、A, B, Cの習熟度により、各人が仕事の完成に必要な日数は異なっている。6つの仕事それぞれは均質であるので、すべての仕事について各人の所要時間は以下のように考えるとよい。

仕事 1-6 の所要時間

所要時間	
A	6 日
B	8 日
C	11 日

この時、すべての仕事が完成するまで最短で何日程度所要するか、また、その際の A, B, C への仕事の割り当てはどのようにすればよいか。なお、すべての仕事を終えるまでの所要時間は最大で 40 日までとする。

この問題に対する C++SIMPLE の定式化は以下のようになります。

```
//
// 例題 1 (全体の作業完了時刻最小化)
//
Set M = "A_does B_does C_does"; // モード
Element m(set = M);
Set R = "A B C"; // 資源
Element r(set = R);
Set D = "1 .. 11"; // 各モードの作業時間 (日単位で最大が 11 日である)
Element d(set = D);
// モードと資源消費の連関
ResourceRequire req(mode = M, resource = R, duration = D);
req["A_does, A", d] = 1, 1 <= d <= 6;
req["B_does, B", d] = 1, 1 <= d <= 8;
req["C_does, C", d] = 1, 1 <= d <= 11;
// アクティビティ
Set J = "1 .. 6";
Element j(set = J);
Activity act(name = "act", index = j, mode = M); // 作業 (j = 1, ..., 6)
// 利用可能な資源の定義
Set T = "0 .. 40"; // スケジューリング全体の時間 (日単位で最大 40 日とする)
Element t(set = T);
ResourceCapacity cap(resource = R, timeStep = T);
```

```

cap[r, t] = 1;
Objective f(type = minimize);
f = completionTime; // 最後の作業の完了時刻最小化
options.maxtim = 2;
// 求解
solve();
// 解の表示
simple_printf("job = %d %s %2d %2d %2d\n", j, act[j], act[j].startTime, act[j].endTime,
    act[j].processTime);

```

それでは、このモデルに対する定式化の手順を見ていきましょう。

例題において実施しなければならない作業は 6 つの仕事です。そこでこれらを作業集合 J として定義します。

```
Set J = "1 .. 6";
```

それぞれの作業には、{A に任せる, B に任せる, C に任せる} の三種類のモード（対処方法）が存在します。そこでこれらをモード集合 M として定義します。

```
Set M = "A_does B_does C_does";
```

モードが利用する資源は、A, B, C の 3 人のみですから、これらを資源集合 R として定義します。

```
Set R = "A B C";
```

上記を整理すると次のようになります。

■ 例題 1 の rcpsp による表現のための整理

1. 作業

各仕事 j ($j=1, \dots, 6$) に対応してアクティビティが存在し、各仕事の作業モードと開始時刻、終了時刻を決定したい。

2. モード

各仕事 j には以下の 3 つのモードが対応付けられる。

仕事 1-6 に対応するモード		
モード種別	所要時間	消費資源
A_does	6 日	A を各日について 1
B_does	8 日	B を各日について 1
C_does	11 日	C を各日について 1

3. 資源

各人に対応する A, B, C があり、次の量が利用可能である。

資源	利用可能量
A	全時間ステップで 1
B	全時間ステップで 1
C	全時間ステップで 1

次に、これら 3 つの集合の関連付けを行います。

全ての作業は、モード集合のいずれかのモードで行われる事を示します。そのため、Activity の引数には作業集合 J の添字と、モード集合 M を与えます。

```
Set J = "1 .. 6";
Element j(set = J);
Set M = "A_does B_does C_does";
Activity act(index = j, mode = M);
```

モードに対応する資源を与える定数 ResourceRequire は次のように定義されます。引数には、モード集合 M と資源集合 R 以外に、新たに作業時間集合 D を定義する必要があります。今回の例では資源を用いる最大期間が 11 日なので、D = "1 .. 11" と定めています。

```
Set M = "A_does B_does C_does";
Set R = "A B C";
Set D = "1 .. 11";
Element d(set = D);
ResourceRequire req(mode = M, resource = R, duration = D);
```

モード A_does は資源 A を 6 日間、モード B_does は資源 B を 8 日間、モード C_does は資源 C を 11 日間用いるので、各々の ResourceRequire の値は、次のように設定します。

```
ResourceRequire req(mode = M, resource = R, duration = D);
req["A_does, A", d] = 1, 1 <= d <= 6;
req["B_does, B", d] = 1, 1 <= d <= 8;
req["C_does, C", d] = 1, 1 <= d <= 11;
```

次は資源供給量 ResourceCapacity の設定です。各人は同時に 2 つ以上の仕事をすることはできないので、資源の上限値は、最後の期間まで全て 1 です。スケジューリングの期間は全体で 40 日なので、期間集合 T は T = "0 .. 40" で定めます。なお、期間集合は 0 はじまりでなければなりません。これらをまとめると、次のように設定されます。

```
Set R = "A B C";
Element r(set = R);
Set T = "0 .. 40";
Element t(set = T);
```

```
ResourceCapacity cap(resource = R, timeStep = T);
cap[r, t] = 1; // 資源の上限値
```

次に問題の種類を指定します。rcpsp で扱う事の出来る問題は、

- 幾つかの作業が存在し、一定の資源下で最後の作業の完了時刻を最小化する問題
 - 納期のある幾つかの作業が存在し、一定の資源下でそれらの納期遅れを最小化する問題
- の二種類ですが、ここで扱う問題は前者です。これは目的関数で以下のように指定します。

```
Objective f(type = minimize);
f = completiontime; // 完了時刻最小化を示す
```

最後に、終了条件を指定します。今回は終了条件として、計算時間 2 秒を設定します。

```
options.maxtim = 2;
```

以上をまとめると、本例題の定式化が完了します。

C++SIMPLE モデルを実行させると、次のような実行結果が得られます。

```
job = 1 "A_does" 6 12 6
job = 2 "B_does" 8 16 8
job = 3 "A_does" 12 18 6
job = 4 "C_does" 0 11 11
job = 5 "A_does" 0 6 6
job = 6 "B_does" 0 8 8
```

この出力は、最適化の実行（直前の solve () 呼び出し）が終わった後の

```
// 解の表示
simple_printf("job = %d %s %2d %2d %2d\n", j, act[j], act[j].startTime, act[j].endTime,
    act[j].processTime[j]);
```

に対応するもので、rcpsp が求めた各仕事についてのモード、作業開始時刻、終了時刻、作業所要時間が表示されています。この表示から、例えば仕事 1 は A に実施させ (A_does というモードを適用)、作業開始は 6 日目、終了は 12 日目で、作業所要時間は 6 という解となっていることがわかります。細かな点ですが、仕事 1 の場合、作業開始は 6 日目のスタートで、作業終了は 12 日目が始まる直前（すなわち 11 日目一杯まで）と解釈してください。このように解釈すると、作業所要時間は作業終了時刻から作業開始時刻を引いたものになります。

なお、この例題は作業完了時刻最小化問題ですので、制約に重みを設定することが可能です。いま、各モードで仕事を行った際にコストが発生するとします。そのもとで、コストが 0 を超過した分に制約の重みをかけたペナルティと完了時刻の値を足した全体を最小化することを考えます。

先程のモデルの solve() 以前に、


```

Parameter cost(index = m);
cost["A_does"] = 5; cost["B_does"] = 2; cost["C_does"] = 1;
Expression costTotal;
costTotal = sum(Boolean(act[j] == m) * cost[m], (j, m)); // 全コスト
softConstraint(1); // 制約式の重み
costTotal <= 0; // コストの最小化に対応する制約式

```

を追加します。A_does, B_does, C_does 各々のモードを選択したときにコストは各々 5, 2, 1 かかるものとし、コスト全体を costTotal で表現しています。

この実行結果は次のようになります。

```

job = 1 "C_does" 0 11 11
job = 2 "C_does" 11 22 11
job = 3 "B_does" 8 16 8
job = 4 "A_does" 0 6 6
job = 5 "B_does" 0 8 8
job = 6 "B_does" 16 24 8

```

先程の問題は作業完了時刻が 18 であったのに対し、今回の問題の完了時刻は 24 となり、完了するまでに時間を要するようになりました。その分、コストの小さいモード C_does で行う仕事が増加し、コストの大きいモード A_does で行う仕事が減少しております。

次に、納期遅れ最小化問題を考えます。実際のプロジェクトスケジューリングにおいては、「各仕事に納期が設定されており、納期遅れを最小化したい」という問題も多く存在します。

■ 例題 2 納期遅れ最小化

例題 1 の状況において、各仕事について次のような納期が設定されているとき、納期遅れを最小化するようなスケジュールを出力せよ。

仕事	納期
1	10 日
2	10 日
3	10 日
4	17 日
5	17 日
6	6 日

これは Activity の定義に納期情報を設定し、目的関数に納期遅れ最小化を定義することによって可能です。

```
Set J = "1 .. 6";
Element j(set = J);
Parameter due(index = j);
Activity act(index = j, mode = M, duedate = due[j]);
```

目的関数には次のようにして納期遅れを設定します。

```
// 納期遅れ最小化
Objective f(type = minimize);
f = tardiness;
```

これらを反映した例題 2 の C++SIMPLE モデルは次のようになります。

```
//
// 例題 2 (納期遅れ最小化)
//
Set M = "A_does B_does C_does"; // モード
Element m(set = M);
Set R = "A B C"; // 資源
Element r(set = R);
Set D = "1 .. 11"; // 各モードの作業時間の最大
Element d(set = D);
ResourceRequire req(mode = M, resource = R, duration = D);
req["A_does, A", d] = 1, 1 <= d <= 6;
req["B_does, B", d] = 1, 1 <= d <= 8;
req["C_does, C", d] = 1, 1 <= d <= 11;
Set J = "1 .. 6";
Element j(set = J);
Parameter due(index = j);
due[j] = 10, 1 <= j <= 3;
due[j] = 17, 4 <= j <= 5;
due[j] = 6, j == 6;
Activity act(name = "act", index = j, mode = M, duedate = due[j]);
Set T = "0 .. 40"; // スケジューリング全体の時間 (日単位)
Element t(set = T);
ResourceCapacity cap(resource = R, timeStep = T);
cap[r, t] = 1;
Objective f(type = minimize);
f = tardiness; // 納期遅れ最小化
options.maxtim = 2;
```

```
solve();  
// 解の表示  
simple_printf("job = %d %s %2d %2d %2d\n", j, act[j], act[j].startTime, act[j].endTime,  
             act[j].processTime);
```

実行結果は、以下ようになります。

```
job = 1 "A_does"  6 12  6  
job = 2 "C_does"  0 11 11  
job = 3 "B_does"  0  8  8  
job = 4 "B_does"  8 16  8  
job = 5 "A_does" 12 18  6  
job = 6 "A_does"  0  6  6
```


第8章

データファイル

8.1 データファイルの機能

C++SIMPLE では、定数 `Parameter` の値、集合 `S` の要素、変数 `Variable` の初期値をデータファイルと呼ばれる外部ファイルから与える事ができます。データファイルには `dat` 形式データファイル（拡張子 `.dat`）、`csv` 形式データファイル（拡張子 `.csv`）の二種類が存在し、それぞれ利用方法が異なります。

データファイルを用いることで、数値のみが異なる数値最適化問題を簡便に扱う事ができます。

一つのモデルファイルは、複数のデータファイルを利用する事ができます。複数のデータファイルを利用する場合、それらの形式を統一する必要はありません（`dat` 形式と `csv` 形式が混在していても問題ありません）。以下は、モデルファイル `model.smp` とデータファイル `data1.dat`, `data2.csv` をコマンドラインから使用する例です（Windows 版）。

```
prompt% mknuopt model.smp
```

```
prompt% model.exe data1.dat data2.csv
```

データファイルを引数に与える順番は任意です。すなわち、次のコマンドは上記と等価です。

```
prompt% model.exe data2.csv data1.dat
```

同じ対象に対して複数のデータファイルから値を設定した場合、エラーとなります。

8.2 dat 形式データファイル

`dat` 形式データファイルでは、定数 `Parameter` の値、集合 `S` の要素、変数 `Variable` の初期値が設定できます。拡張子が `.dat` であるデータファイルは `dat` 形式データファイルと解釈されます。

定数の値や変数の初期値を設定する場合は、`name` 引数によって定数や変数の名前を定めることができます。特に `name` 引数を付けない場合には、モデルファイルで定義された名称そのものが `name` だと認識されます。つまり、以下の二つの例は同等です。

```
Parameter a;
```

```
Parameter a(name = "a");
```

`dat` 形式データファイルの行末には半角セミコロン `;` を付ける必要があります。

次の例では、`dat` 形式データファイルに定数 `a(name = "aa")` の値 `10` を設定しています。

モデルファイル内

```
Parameter a(name = "aa");
```

データファイル内 (dat 形式)

```
aa = 10;
```

次の例では、dat 形式データファイルに変数 x(name = "xx") の初期値 4 を設定しています。

モデルファイル内

```
Variable x(name = "xx");
```

データファイル内 (dat 形式)

```
xx = 4;
```

次の例では、dat 形式データファイルで集合 S の要素 1 2 3 を設定しています。データファイル内で定義する場合は、モデルファイル内で定義する場合と異なり、ダブルクォート"で囲ってはいけません。

モデルファイル内

```
Set S;
```

データファイル内 (dat 形式)

```
S = 1 2 3;
```

一つの dat 形式データファイルには、まとめて複数の設定を記述することができます。以下の例では、定数 a(name = aa) の値 10、変数 x(name = xx) の初期値 4、集合 S の要素 1 2 3 を全て設定しています。

モデルファイル内

```
Parameter a(name = "aa");
```

```
Variable x(name = "xx");
```

```
Set S;
```

データファイル内 (dat 形式)

```
aa = 10;
```

```
xx = 4;
```

```
S = 1 2 3;
```

dat 形式データファイル内では、任意に改行を挟むことができます。

```
aa = 10;
```

```
xx = 4;
```

```
S = 1 2 3;
```

//はじまりのコメント文を付与することもできます。

```
// 定数の設定
aa = 10;

// 初期値設定
xx = 4;

// 集合の定義
S = 1 2 3;
```

添字を持つ定数 `Parameter` の値や、変数 `Variable` の初期値を設定するには、以下のように `[]` を使います。次の例では、定数 `a[1]`, `a[2]`, `a[3]` に対して、初期値 1, 0.5, -1 を与えています。

モデルファイル内

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(name = "aa", index = i);
```

データファイル内 (dat 形式)

```
aa = [1] 1 [2] 0.5 [3] -1;
```

行末の半角セミコロン;`;` は、一つの設定データの最後に記述します。改行は自由なので、データファイル部分は、以下のように記述する事もできます。

```
aa = [1] 1
      [2] 0.5
      [3] -1;
```

これは、モデルファイル内で以下のように記述する場合と同じ意味です。

```
a[1] = 1;
a[2] = 0.5;
a[3] = -1;
```

データファイル内で値を設定する場合は、添字部分にダブルクォート"は付けません。

次は、変数 `x["1, p"]`, `x["1, q"]`, `x["2, p"]`, `x["2, q"]` に初期値 1, 3, 5, 7 を与える例です。

モデルファイル内

```
Set S = "1 2";
Set T = "p q";
Element i(set = S);
Element j(set = T);
Variable x(name = "xx", index = (i, j));
```

データファイル内 (dat 形式)

```
xx = [1, p] 1 [1, q] 3
      [2, p] 5 [2, q] 7;
```

これは、モデルファイル内で以下のように記述する場合と同じ意味です。

```
xx["1, p"] = 1;
xx["1, q"] = 3;
xx["2, p"] = 5;
xx["2, q"] = 7;
```

集合クラスの構成要素は、明示的に定義しなくとも、モデルファイルやデータファイルの情報から自動的に定義されます。これを SIMPLE の **自動代入機能** と呼びます。以下の例では、自動代入機能により、集合 S の要素は 1, 2, 3 であると判断されます。

モデルファイル内

```
Set S;
Element i(set = S);
Parameter a(index = i);
```

データファイル内 (dat 形式)

```
a = [1] -1 [2] -1 [3] 1;
```

以下のように、csv 形式データファイルで $a[1]$, $a[2]$, $a[3]$ の値を定めた場合も同様です。

```
i, a
1, -1
2, -1
3, 1
```

以下のように、モデルファイル内で $a[1]$, $a[2]$, $a[3]$ の値を定めた場合も同様です。

```
Set S;
Element i(set = S);
Parameter a(index = i);
a[1] = -1;
a[2] = -1;
a[3] = 1;
```

8.3 csv 形式データファイル

csv 形式データファイルでは、定数 `Parameter` の値、変数 `Variable` の初期値が設定できます。dat 形式データファイルと異なり、集合 `Set` の要素を設定することはできません。csv 形式データファイル

の拡張子は.csv でなければなりません。

定数の値や変数の初期値を設定する場合は、name 引数によって定数や変数の名前を定める必要があります。Windows 版では、特に name 引数を付けない場合には、モデルファイルで定義された名称そのものが name だと認識されます。つまり、以下の二つの例は同等です。

```
Parameter a;
```

```
Parameter a(name = "a");
```

csv 形式のデータファイルは半角コンマ, で区切られた行から構成されています。

次の例では、csv 形式データファイルに定数 a(name = "aa") の値 10 を設定しています。

モデルファイル内

```
Parameter a(name = "aa");
```

データファイル内 (csv 形式)

```
aa
```

```
10
```

なお、csv 形式データファイルではヘッダー行のみの場合は読み込まれません。以下の例では定数 a には値が設定されませんのでご注意ください。

データファイル内 (csv 形式)

```
aa, 10
```

次の例では、csv 形式データファイルに変数 x(name = "xx") の初期値 4 を設定しています。

モデルファイル内

```
Variable x(name = "xx");
```

データファイル内 (csv 形式)

```
xx
```

```
4
```

一つの csv 形式データファイルには、まとめて複数の設定を記述することができます。以下の例では、定数 a(name = "aa") の値 10、変数 x(name = "xx") の初期値 4 を両方設定しています。

モデルファイル内

```
Parameter a(name = "aa");
```

```
Variable x(name = "xx");
```

データファイル内 (csv 形式)

```
aa, xx
```

```
10, 4
```

//はじまりのコメント文を付与することもできます。

```
// 定数と初期値の設定
aa, xx
10, 4
```

改行を挟む事はできません。

添字のある定数 `Parameter` の値や、変数 `Variable` の初期値を設定するには、以下のようにします。次の例では、定数 `a[1]`, `a[2]`, `a[3]` に対して、初期値 1, 0.5, -1 を与えています。

モデルファイル内

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(name = "aa", index = i);
```

データファイル内 (csv 形式)

```
i, aa
1, 1
2, 0.5
3, -1
```

これは、モデルファイル内で以下のように記述する場合と同じ意味です。

```
a[1] = 1;
a[2] = 0.5;
a[3] = -1;
```

csv 形式データファイルではまとめて複数の設定を記述できますが、添字を持つ定数や変数をまとめて設定する場合は、それらが属する添字が同一でなければなりません。次の例では、定数 `a[1]`, `a[2]`, `a[3]` に対して、初期値 1, 0.5, -1 を、定数 `b[1]`, `b[2]`, `b[3]` に対して初期値 3, 5, 7.1 を与えています。

モデルファイル内

```
Set S = "1 2 3";
Element i(set = S);
Parameter a(name = "aa", index = i);
Parameter b(name = "bb", index = i);
```

データファイル内 (csv 形式)

```
i, aa, bb
1, 1, 3
2, 0.5, 5
3, -1, 7.1
```

次のように、定数 `a` と `b` の添字が異なる場合は、一つの csv 形式データファイルで `a`, `b` 両方の値を設定することはできません。

モデルファイル内

```
Set S = "1 2 3";
Set T = "p q";
Element i(set = S);
Element j(set = T);
Parameter a(name = "aa", index = i);
Parameter b(name = "bb", index = j);
```

csv 形式データファイルで 2 次元の添字を持つ定数 `Parameter` や変数 `Variable` の初期値を設定するには、二通りの方法があります。一つは添字を全て縦に左に記述する方法で、**1D 書式**と呼ばれます。もう一つは、最後の添字を横一行目に記述する方法です。これは **2D 書式**と呼ばれます。

以下の例では、変数 `x["1, p"]`, `x["1, q"]`, `x["2, p"]`, `x["2, q"]` に初期値 1, 3, 5, 7 を 1D 書式で与える場合と、2D 書式で与える場合両方を記述しています。

モデルファイル内

```
Set S = "1 2";
Set T = "p q";
Element i(set = S);
Element j(set = T);
Variable x(name = "xx", index = (i, j));
```

データファイル内 (csv 形式 1D 書式)

```
i, j, xx
1, p, 1
1, q, 3
2, p, 5
2, q, 7
```

データファイル内 (csv 形式 2D 書式)

```
xx, p, q
1, 1, 3
2, 5, 7
```

これは、モデルファイル内で以下のように記述する場合と同じ意味です。

```
xx["1, p"] = 1;
xx["1, q"] = 3;
xx["2, p"] = 5;
xx["2, q"] = 7;
```

1D 書式の場合、定数や変数の添字が同じであれば、同時に複数の設定を行うことが可能です。しか

し、2D 書式の場合は一つの csv 形式データファイルに対して一つの定数あるいは変数しか設定する事ができません。

以下の例では、変数 `x["1, p"]`, `x["1, q"]`, `x["2, p"]`, `x["2, q"]` に初期値 1, 3, 5, 7 を、定数 `a["1, p"]`, `a["1, q"]`, `a["2, p"]`, `a["2, q"]` に値 2, 4, 6, 8 を 1D 書式で与えています。

モデルファイル内

```
Set S = "1 2";
Set T = "p q";
Element i(set = S);
Element j(set = T);
Variable x(name = "xx", index = (i, j));
Parameter a(name = "aa", index = (i, j));
```

データファイル内 (csv 形式 1D 書式)

```
i, j, xx, aa
1, p, 1, 2
1, q, 3, 4
2, p, 5, 6
2, q, 7, 8
```

集合クラスの構成要素は、明示的に定義しなくとも、モデルファイルやデータファイルの情報から自動的に定義されます。これを SIMPLE の**自動代入機能**と呼びます。以下の例では、自動代入機能により、集合 S の要素は 1, 2, 3 であると判断されます。

モデルファイル内

```
Set S;
Element i(set = S);
Parameter a(index = i);
```

データファイル内 (csv 形式)

```
i, a
1, -1
2, -1
3, 1
```

以下のように、dat 形式データファイルで `a[1]`, `a[2]`, `a[3]` の値を定めた場合も同様です。

```
a = [1] -1 [2] -1 [3] 1;
```

以下のように、モデルファイル内で `a[1]`, `a[2]`, `a[3]` の値を定めた場合も同様です。

```
Set S;
Element i(set = S);
```

```
Parameter a(index = i);
a[1] = -1;
a[2] = -1;
a[3] = 1;
```

パラメータ定義時の添字の順序と csv 形式の添字の順序は一致している必要があります。例を使って説明します。

モデルファイル内

```
Set S;
Set T;
Element i(set = S);
Element j(set = T);
Parameter a(name = "aa", index = (i, j));
```

このモデルに対し以下のように設定したいとします。

```
aa["1, p"] = 1;
aa["1, q"] = 3;
aa["2, p"] = 5;
aa["2, q"] = 7;
```

以下はモデルファイル内の aa と添字の順序が異なるため無効となります。

データファイル内 (csv 形式)

```
j, i, aa
p, 1, 1
q, 1, 3
p, 2, 5
q, 2, 7
```

以下のように設定してください。

データファイル内 (csv 形式)

```
i, j, aa
1, p, 1
1, q, 3
2, p, 5
2, q, 7
```


第9章

最適化計算制御

9.1 solve 関数

solve 関数は Nuorium Optimizer に最適化計算の実行を命令する関数です。以下の書式で記述されます。

```
solve();
```

モデルファイルに明示的に記述しない場合は、モデルファイルの最後に solve 関数が書かれた場合と同じ意味になります。従って、通常の場合はモデルファイルに solve 関数を明示的に記述する必要はありません。なお、求解オプションの設定によって、この自動的な実行を避けることができます。以下のように記述すると自動実行が抑制できます。このオプションは C++SIMPLE の挙動を素早く確認したい時などに有効です。

```
options.noDefaultSolve = 1;
```

一方、以下のような場合は、どこで最適化計算を行っているかを明示的に指示する必要があります。

- 解いた後の構成要素の内容を表示させる。
- 複数の目的関数について連続して最適化を行う。
- モデルの定義を変更しながら複数回の最適化を行う。

以下は、解いた後の構成要素を表示させるモデルファイルの記述例です。

モデルファイル

```
Variable x;  
Objective f(type = minimize);  
f = 2 * x;  
x >= 5; //制約  
x = 10; //初期値設定  
solve();  
x.val.print();
```

出力

```
x=5
```

明示的に solve 関数を記述しないと、次のように解く前の値が出力されてしまいます。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
x.val.print();
```

出力

```
x=10
```

次のモデルでは複数の目的関数を定義しています.

```
Variable x(name = "x"), y(name = "y");
Objective f(name = "f", type = maximize);
Objective g(name = "g", type = maximize);
f = x + y; // 目的関数 f の定義
g = x - y; // 目的関数 g の定義
pow(x - 1, 2) + pow(y - 1, 2) <= pow(0.5, 2);
solve(); // 最後に代入された g に関する最大化を行う
solve(f); // 目的関数 f に関する最大化を行う
solve(g); // 目的関数 g に関する最大化を行う
```

最初の `solve()` は目的関数の指定を省略した最適化の実行で、この場合には最後に代入された目的関数について最適化が行われます。その後、明示的に目的関数を指定した最適化が実行されます。`solve(f)` で、目的関数 `f` に関する最適化が、`solve(g)` で目的関数 `g` に関する最適化がそれぞれ行われます。`solve()` の呼び出しによって最適化が行われるのは、その時点以前に定義されたモデルの内容に対してです。このことを利用して、モデルを逐次変更しながら最適化を行うことができます。

次の例では、制約式 $x - y \geq 0.5$ を 1 つ加える前と後で最適化を行っています。

```
Variable x(name = "x"), y(name = "y");
Objective f(name = "f", type = maximize);
f = x + y; // 目的関数 f の定義
pow(x - 1, 2) + pow(y - 1, 2) <= pow(0.5, 2);
solve(); // 上記までのモデルを解く
x - y >= 0.5;
solve(); // 上の制約式も加えたモデルを解く
```

9.2 求解オプション options

プログラム内で最適化計算の動作を制御するパラメータを設定することができます。C++SIMPLE

モデルに対してパラメータを設定するには、モデルやドライバ中に以下のように記述します。

```
options. パラメータ名 = 値;
```

例えば計算時間上限を 300 秒に設定する場合は以下のように記述します。

```
options.maxtim = 300;
```

オプション設定の詳細については Nuorium Optimizer マニュアルの「5 求解オプション設定」をご参考ください。

9.3 最適化計算結果 result

最適化計算の結果は result というオブジェクトが保持しています。以下は、result が保有する情報の一覧です。

名称	型	意味
nvars	int	変数の数
nfunc	int	関数の数
iters	int	内点法の反復回数
fevals	int	内点法の関数評価回数
optValue	double	目的関数値
tolerance	double	内点法の収束判定値
residual	double	内点法の解における最適性条件の残差
elapsedTime	double	所要計算時間
peakPhysicalMemoryUsed	size_t	物理メモリの最大使用量
peakVirtualMemoryUsed	size_t	仮想メモリの最大使用量
hardPenalty	double	制約充足問題ソルバ WCSP によるハードペナルティ
semiHardPenalty	double	制約充足問題ソルバ WCSP によるセミハードペナルティ
softPenalty	double	制約充足問題ソルバ WCSP によるソフトペナルティ
errorCode	int	終了時ステータス (成功時：0, 失敗時：エラー番号) エラー番号は Nuorium Optimizer マニュアルの付録に記載の ものに従います
simpleErrorCode	int	終了時ステータス (成功時：0, 失敗時：エラー番号) エラー番号は付録 A.1 のものに従います
isFeasible()	bool	実行可能解が得られているかどうか (true：得られている, false：得られていない)

以下は最適化計算結果を出力するモデルファイルの記述例です。

```

Variable x, y;
Objective f(type = minimize);
f = 2 * x + 3 * y;
x + 2 * y == 15;
x >= 0;
y >= 0;
solve();

simple_printf("関数の数           %d\n", result.nfunc);
simple_printf("内点法の反復回数 %d\n", result.iters);
simple_printf("関数評価回数      %d\n", result.fevals);
simple_printf("目的関数値        %e\n", result.optValue);
simple_printf("収束判定条件      %e\n", result.tolerance);
simple_printf("最適性条件残差    %e\n", result.residual);
simple_printf("所要計算時間      %d\n", result.elapsedTime);
simple_printf("終了時ステータス %d\n", result.errorCode);

```

以下出力例です.

```

関数の数           2
内点法の反復回数  5
関数評価回数      8
目的関数値        2.250000e+001
収束判定条件      1.000000e-008
最適性条件残差    3.978422e-008
所要計算時間      0
終了時ステータス  0

```

result オブジェクトには、アルゴリズム毎に有効なメソッドもあります.

最適化計算を行った場合、実行可能解が得られたかどうかを表す isFeasible() 関数を使用できます. 返回值は bool 型です.

以下記述例です.

```

Variable x;
Variable y;
Variable z;
Variable s1;
Variable s2;
Variable s3;
IntegerVariable delta1(type = binary);

```

```

IntegerVariable delta2(type = binary);
IntegerVariable delta3(type = binary);
Parameter M = 10000;
Objective cost(type = minimize);
cost = delta1 + delta2 + delta3;
x + y + z == 12 + s1;
5 * x + 5 * y + 5 * z == 60 + s2;
0 <= s1 <= 1.0e-3;
0 <= s2 <= 1.0e-3;
-M * delta1 <= x <= M * delta1;
-M * delta2 <= y <= M * delta2;
-M * delta3 <= z <= M * delta3;
options.plevel = 0;
options.maxnod = 1;
options.method = "simplex";
solve();
simple_printf("feasible = %d\n", result.isFeasible());

```

以下は、上記例の `simple_printf` 関数における出力例です。

```
feasible = 1
```

制約充足問題ソルバ WCSP で最適化計算を行った場合、ハード制約・セミハード制約・ソフト制約のペナルティを `hardPenalty`, `semiHardPenalty`, `softPenalty` に保持します。

以下記述例です。

```

IntegerVariable x(type = binary);
x - 2 >= 0;
options.maxtim = 0.1;
options.method = "wcsp";
solve();
simple_printf("hard penalty = %f\n", result.hardPenalty);

```

以下は、上記例の `simple_printf` 関数における出力例です。

```
hard penalty = 1.000000
```

9.4 可変定数

モデル中の定数 `Parameter` は、モデルを展開する段階で固定され、展開後には変更不可能となります。しかし、パラメトリック最適化等の場合には変更可能な定数を含めてモデルを定義して、後で変

更しながら解析を行う場合が生じます。そのために C++SIMPLE は `VariableParameter` という名称で可変定数を提供しています。

次は簡単な可変定数の使用例です。線形な目的関数の係数を変更しながら、円の内部が実行可能領域である二次計画問題を逐次的に解きます。この問題に対するモデル定義とシステム制御は次のようになります。

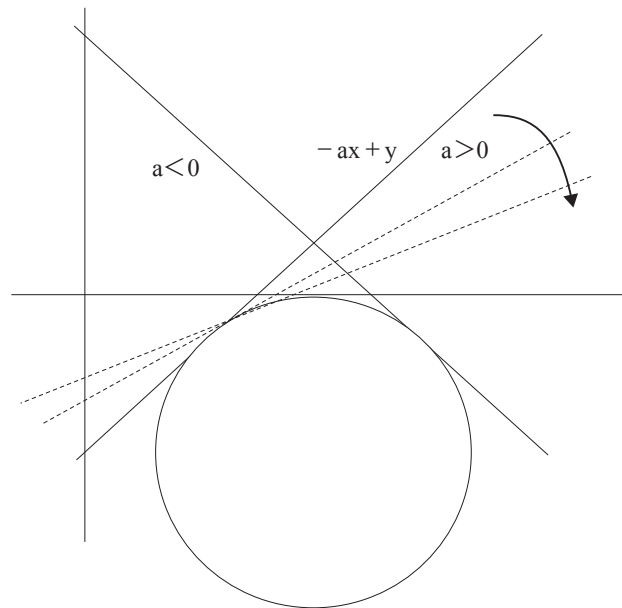
```
Variable x, y;
VariableParameter a; // 可変定数
Objective f(type = maximize);
f = -a * x + y;
pow(x - 1, 2) + pow(y + 0.5, 2) <= 0.25;
options.outputMode = "silent"; // 出力抑制
for(int i = -5; i < 5; i++){
    a = i;
    solve();
    simple_printf("a = %d, x = %f, y = %f, f = %f\n",
        a, x, y, f);
}
```

上記のように `VariableParameter` は通常の `Parameter` と全く同様にモデル定義に使用することができます。

最適値は定円の接線（傾き a ）の y 切片となりますので、目的関数として各 a に対応する y 切片の値が得られます。

以下は、上記のモデルに対する出力結果です。

```
a = -5, x = 1.490290, y = -0.401942, f = 7.049510
a = -4, x = 1.485071, y = -0.378732, f = 5.561553
a = -3, x = 1.474342, y = -0.341886, f = 4.081139
a = -2, x = 1.447214, y = -0.276393, f = 2.618034
a = -1, x = 1.353553, y = -0.146447, f = 1.207107
a = 0, x = 1.000000, y = -0.000000, f = -0.000000
a = 1, x = 0.646447, y = -0.146447, f = -0.792893
a = 2, x = 0.552786, y = -0.276393, f = -1.381966
a = 3, x = 0.525658, y = -0.341886, f = -1.918861
a = 4, x = 0.514929, y = -0.378732, f = -2.438447
```



VariableParameter には以下の注意点があります。

- VariableParameter は通常の Parameter と比べて、メモリを多く所要します。モデルを定義するときに、必要以上の Parameter を VariableParameter とすることは避けてください。
- アルゴリズムの自動選択機能に影響を及ぼす可能性がありますので注意してください。
- 制約充足問題ソルバ WCSP では VariableParameter を用いることはできません。

9.5 初期値設定 SimpleSetInitialValues()

複数求解を行う場合、最適化計算結果をそのまま初期値として利用したい場合があります。そのような場合は関数 SimpleSetInitialValues() が便利です。本関数を用いると最適化計算結果を初期値として利用できます。以下記述例です。

```
solve();
SimpleSetInitialValues(); // 最適化計算結果が初期値に設定される
solve();                  // 最適化計算結果を初期値として計算が行われる
```


第 10 章

出力制御

C++SIMPLE で記述された数理最適化モデルは、Nuorium Optimizer で求解されます。その際には求解情報が標準出力に、より細かい解情報が解ファイル（モデル名.sol）に出力されます。

この章では、出力情報の追加に用いられる C++SIMPLE の関数 `print`, `simple_printf`, `simple_fprintf` ならびに、出力情報を抑制する記述方法について説明します。

10.1 出力対象

後述する `print` 関数, `simple_printf` 関数, `simple_fprintf` 関数では、以下の構成要素に対する情報を適宜取得することができます。

構成要素	情報	意味
Variable	val	現在値
	init	初期値
	dual	双対変数値
	ub	上限値
	lb	下限値
Objective	val	現在値
	init	初期値
Constraint	val	現在値
	init	初期値
	dual	双対変数値
	ub	上限値
	lb	下限値
Parameter	val	現在値
IntegerVariable	val	現在値
	init	初期値
	dual	双対変数値
	ub	上限値
	lb	下限値
Expression	val	現在値
	init	初期値
SymmetricMatrix	val	現在値
	init	初期値

構成要素	情報	意味
Set	val	現在値
OrderedSet	val	現在値

現在値 val は solve 関数が呼ばれる前であれば初期値, 呼ばれた後には最適化計算結果が入っています.

例えば以下のモデルファイルでは `x.val.print()`; は初期値 10 を出力します (print 関数については次節を参照してください).

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
x.val.print(); // 初期値 10
```

一方, 以下では `x.val.print()`; は最適化計算結果 5 を出力します.

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
solve();
x.val.print(); // 最適化計算結果 5
```

また, val は `simple_printf` 関数および `simple_fprintf` 関数では省略可能です.

10.1.1 制約式に対する .val

制約式に対する値は, 左辺値の現在値になります. 例えば以下の制約式を考えます.

```
Variable x;
Constraint co;
co = x + 1 >= 1;
```

このとき `co.val` の値は `x.val + 1` に等しくなります.

ただし, いくつかの制約式については `.val` の値は不定となります. このような場合は, `.val` を参照することは非推奨です.

具体的には, 以下を含んだ制約式に対して不定です.

- Boolean
- DiscreteVariable

- SymmetricMatrix
- alldiff
- count
- max
- min
- selection

10.2 print 関数

print 関数は、変数 Variable, 目的関数 Objective, 制約式 Constraint, 定数 Parameter, 整数変数 IntegerVariable, 式 Expression, 対称行列 SymmetricMatrix, 集合 Set, 順序集合 OrderedSet に関する情報を、決まったフォーマットで出力させる機能を有しています。

print 関数の書式は、以下のように定められています。

```
構成要素.情報.print();
```

変数の現在値を出力するには、次のように記述する必要があります。

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

目的関数の初期値を出力するには、次のように記述する必要があります。

```
Objective f;  
f.init.print();
```

制約式の双対変数値を出力するには、次のように記述する必要があります。

```
Constraint Co;  
Co.dual.print();
```

定数の現在値を出力するには、次のように記述する必要があります。

```
Parameter a;  
a.val.print();
```

整数変数の下限値を出力するには、次のように記述する必要があります。

```
IntegerVariable z;  
z.lb.print();
```

対称行列の現在値を出力するには、次のように記述する必要があります。

```
SymmetricMatrix X((i, j));  
X.val.print();
```

式の初期値を出力するには、次のように記述する必要があります。

```
Expression g;
g.init.print();
```

集合の現在値を出力するには、次のように記述する必要があります。

```
Set S;
S.val.print();
```

順序集合の現在値を出力するには、次のように記述する必要があります。

```
OrderedSet O;
O.val.print();
```

求解関数 `solve` の前に `print` 関数を記述すると、求解前の初期状態の情報が記述されます。求解関数 `solve` は、明示的に記述されない場合、モデルの最後尾にあるものと認識されます。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。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
x.val.print();
```

出力

```
x=10
```

`solve` 関数の後に `print` 関数を用意した場合、出力は以下のようになります。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
solve();
x.val.print();
```

出力

```
x=5
```

`solve` 関数の前後に `print` 関数を用意した場合、出力は以下のようになります。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
x.val.print();
solve();
x.val.print();
```

出力

```
x=10
x=5
```

添字が付いている場合は、全体を出力します。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。変数 $x[1]$, $x[2]$, $x[3]$ の現在値が全て出力されます。

モデルファイル

```
Set S = "1 2 3";
Element i(set = S);
Variable x(index = i);
Objective f(type = minimize);
f = 2 * sum(x[i], i);
x[i] >= 5; //制約
x[i] = 10; //初期値設定
solve();
x.val.print();
```

出力

```
x[1]=5
x[2]=5
x[3]=5
```

出力範囲を、条件式で制限する事も可能です。以下のようにした場合、変数 $x[1]$, $x[2]$ の現在値のみが出力されます。

```
(x[i].val, i < 3).print();
```

出力

```
x[1]=5
x[2]=5
```

次の例では、添字付きの対称行列を出力させています。

モデルファイル

```
Set S = "1 2";
Element i(set = S), j(set = S);
Set N = "1 2 3";
Element n(set = N);
Variable x, y;
x = 10;
y = 2;
SymmetricMatrix X(index = n, (i, j));
X[n, i, j] = 100 * n + x * i + y * j, i <= j; // 上三角部分のみ定義
X.val.print();
```

出力

```
X[1,1,1]=112
X[1,1,2]=114
X[1,2,2]=124
X[2,1,1]=212
X[2,1,2]=214
X[2,2,2]=224
X[3,1,1]=312
X[3,1,2]=314
X[3,2,2]=324
```

10.3 simple_printf 関数

simple_printf 関数は、以下のような特徴を持っています。

- simple_printf 関数は、SIMPLE オブジェクトの情報をユーザ指定のフォーマットで出力することができます。
- フォーマットの形式はプログラミング言語 C/C++ の標準関数 printf に従います。
- 条件式を記述して、出力する添字範囲を指定することができます。
- 対象となる SIMPLE オブジェクトは以下です。
 - 変数 Variable
 - 目的関数 Objective
 - 制約式 Constraint
 - 定数 Parameter
 - 整数変数 IntegerVariable
 - 式 Expression

- 対称行列 SymmetricMatrix
- 集合 Set や順序集合 OrderedSet に関する情報を取得することはできません。
- 引数の数は 34 個までです。

simple_printf 関数の書式は以下のように定められています。

```
simple_printf(出力指定書式, 出力対象 1, 出力対象 2, ...);
```

条件式を引数の最後に記述することが可能です。

```
simple_printf(出力指定書式, 出力対象 1, 出力対象 2, ..., 条件式);
```

条件式を複数記述することも可能です。

```
simple_printf(出力指定書式, 出力対象 1, 出力対象 2, ..., 条件式 1, 条件式 2, ...);
```

次の例では、変数の現在値を整数形式で出力させています。整数形式で出力させるには%dを用います。

```
Variable x;  
x = 3;  
simple_printf("%d\n", x.val);
```

これに対する出力は以下のようになります。

```
3
```

次のように記述すると、出力は以下のようになります。

```
simple_printf("x の値 = %d\n", x.val);
```

出力

```
x の値 = 3
```

simple_printf 関数の引数では、.val を省略する事ができます。従って、次のように記述しても出力は同じです。

```
simple_printf("x の値 = %d\n", x);
```

simple_printf 関数では、整数以外にも小数や、指数形式で値を出力させる事ができます。以下は小数を出力する例です。小数を出力するには%fを用います。

```
simple_printf("x の値 = %f\n", x);
```

出力

```
x の値 = 3.000000
```

以下は指数形式で出力する例です。指数形式で出力するには%eを用います。

```
simple_printf("x の値 = %e\n", x);
```

出力

```
x の値 = 3.000000e+000
```

表示させる桁数を指定することもできます。以下の例では、小数点以下二桁のみが出力されるよう記述しています。

```
simple_printf("x の値 = %.2f\n", x);
simple_printf("x の値 = %.2e\n", x);
```

出力

```
x の値 = 3.00
x の値 = 3.00e+000
```

出力の幅を指定することもできます。以下の例では、半角 15 文字に出力が収まるように記述しています。

```
simple_printf("x の値 = %15f\n", x);
simple_printf("x の値 = %15e\n", x);
```

出力

```
x の値 =          3.000000
x の値 =   3.000000e+000
```

桁数と出力幅の両方をまとめて記述することもできます。

```
simple_printf("x の値 = %15.2f\n", x);
simple_printf("x の値 = %15.2e\n", x);
```

出力

```
x の値 =          3.00
x の値 =   3.00e+000
```

求解関数 `solve` の前に `SIMPLE` オブジェクトを出力すると、求解前の初期状態の情報が記述されます。求解関数 `solve` は、明示的に記述されない場合、モデルの最後尾にあるものと認識されます。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。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
```

```
x >= 5; //制約
x = 10; //初期値設定
simple_printf("x の値 = %f\n", x);
```

出力

```
x の値 = 10.00000
```

solve 関数の後に simple_printf 関数を記述すると次のようになります。

モデルファイル

```
Variable x;
Objective f(type = minimize);
f = 2 * x;
x >= 5; //制約
x = 10; //初期値設定
solve();
simple_printf("x の値 = %f\n", x);
```

出力

```
x の値 = 5.000000
```

添字を持つ対象も出力させる事ができます。次の例では、添字を持つ変数の現在値を整数形式で出力させています。

```
Set S = "1 2 3";
Element i(set = S);
Variable x(index = i);
x[i] = 3;
simple_printf("%d\n", x);
```

これに対する出力は以下のようになります。

```
3
3
3
```

次のように記述すると、以下のように出力されます。

```
simple_printf("x[%d] の値 = %d\n", i, x[i]);
```

出力

```
x[1] の値 = 3
x[2] の値 = 3
x[3] の値 = 3
```

引数の最後に条件式を指定することで、添字の範囲を制限させる事ができます。次の例では、`x[1]`, `x[2]` の値のみを表示させています。

```
simple_printf("x[%d] の値 = %d\n", i, x[i], i < 3);
```

出力

```
x[1] の値 = 3
x[2] の値 = 3
```

同じ添字の対象であれば、同時に複数出力することができます。次の例では、変数 `x[1]`, `x[2]`, `x[3]` 定数 `a[1]`, `a[2]`, `a[3]` の値を同時に出力させています（メソッド `val` は省略しています）。

```
Set S = "1 2 3";
Element i(set = S);
Variable x(index = i);
Parameter a(index = i);
x[i] = 3;
a[i] = 5;
simple_printf("x[%d] = %f, a[%d] = %f\n", i, x[i], i, a[i]);
```

出力

```
x[1] = 3.000000, a[1] = 5.000000
x[2] = 3.000000, a[2] = 5.000000
x[3] = 3.000000, a[3] = 5.000000
```

多次元集合の添字について、複雑な条件を課して出力を行いたい場合、複雑な条件を記述する集合を事前に作成しておくことが推奨されます。以下は具体例です。二次元集合 `IJ`などを定義します。

```
Set I;
Element i(set=I);
I = "1 .. 7";
Set J;
Element j(set=J);
J = "1 .. 4";
Set IJ(dim=2);
IJ = "1,1, 2,2 3,1";
```



```
Parameter b(index=(i,j));
b[i,j] = i+j, (i,j) < IJ;
```

上記の集合について、定数 b が 4 となる変数の組を出力する例を示します。以下のように二次元集合 B を定義します。

```
Set B(dim=2);
Element ij(set=B);
B = setOf( (i,j), b[i,j] == 4);
```

上記集合 B を用いると、意図通りの記述ができます。

```
/*
  以下が出力されます
    2,2
    3,1
*/
simple_printf("%d,%d\n", ij, b[ij] == 4);
```

次の例では 3 つの対称行列 (SymmetricMatrix) X_1, X_2, X_3 の値を表示させています。

```
Set S = "1 2";
Element i(set = S), j(set = S);
Set N = "1 2 3";
Element n(set = N);
SymmetricMatrix X(index=n, (i, j));
X[n, i, j] = 100 * n + 10 * i + j, i <= j; // 上三角部分のみ定義
simple_printf("X%d[%d, %d] = %f\n", n, i, j, X[n, i, j]);
```

出力

```
X1[1, 1] = 111.000000
X1[1, 2] = 112.000000
X1[2, 1] = 112.000000
X1[2, 2] = 122.000000
X2[1, 1] = 211.000000
X2[1, 2] = 212.000000
X2[2, 1] = 212.000000
X2[2, 2] = 222.000000
X3[1, 1] = 311.000000
X3[1, 2] = 312.000000
```

```
X3[2, 1] = 312.000000
X3[2, 2] = 322.000000
```

simple_printf 関数は、最適化計算結果 result の情報も出力させる事ができます。以下は最適化計算結果を出力するモデルファイルの記述例です。

```
Variable x,y;
Objective f(type = minimize);
f = 2 * x + 3 * y;
x + 2 * y == 15;
x >= 0;
y >= 0;
solve();

simple_printf("関数の数           %d\n", result.nfunc);
simple_printf("内点法の反復回数 %d\n", result.iters);
simple_printf("関数評価回数      %d\n", result.fevals);
simple_printf("目的関数値        %e\n", result.optValue);
simple_printf("収束判定条件      %e\n", result.tolerance);
simple_printf("最適性条件残差    %e\n", result.residual);
simple_printf("所要計算時間      %d\n", result.elapseTime);
simple_printf("終了時ステータス %d\n", result.errorCode);
```

以下出力例です。

```
関数の数           2
内点法の反復回数  5
関数評価回数      8
目的関数値        2.250000e+001
収束判定条件      1.000000e-008
最適性条件残差    3.978422e-008
所要計算時間      0
終了時ステータス  0
```

10.4 simple_fprintf 関数

simple_fprintf 関数は、関数は以下のような特徴を持っています。

- 標準出力ではなくファイルに対して出力をするための関数です。
- 出力先が違うという点以外は、simple_printf 関数と同等の機能を有しています。
- simple_printf 関数と同様引数の数は 34 個までです。

simple_printf 関数の書式は以下のように定められています。出力先ファイルを指定するための第1引数以外は、simple_printf 関数と同様の書式です。

```
simple_fprintf(ファイルポインタ, 出力指定書式, 出力対象 1, ...);
```

出力対象は「構成要素. 情報」の形式である必要があります。

次の例では、変数の現在値を整数形式で出力させています。出力ファイルとして、output.txt を指定しています。

```
Variable x;  
FILE* fp; // ファイルポインタの設定  
fp = fopen("output.txt", "w"); // ファイルを開く  
x = 3;  
simple_fprintf(fp, "%d\n", x);  
fclose(fp); // ファイルを閉じる
```

これに対する出力ファイル output.txt への出力は以下のようになります。

```
3
```

次の例では、添字つきの変数の現在値を出力させています。出力先ファイルは result ディレクトリ（フォルダ）以下の data.txt です。

```
Set S = "1 2 3";  
Element i(set = S);  
Variable x(index = i);  
FILE* fp; // ファイルポインタの設定  
fp = fopen("result/data.txt", "w"); // ファイルを開く  
x[i] = 3;  
simple_fprintf(fp, "x[%d] = %f\n", i, x[i]);  
fclose(fp); // ファイルを閉じる
```

これに対する出力ファイル result/data.txt への出力は以下のようになります。

```
x[1] = 3.000000  
x[2] = 3.000000  
x[3] = 3.000000
```

ファイルに上書きではなく、追加をしたい場合はファイルを開く際の引数を"a"とする必要があります。

```
fp = fopen("result/data.txt", "a");
```

10.5 モデルの内容の表示 (showSystem 関数)

モデルファイルとデータファイルを分離して記述した場合、目的関数や制約式の具体的な情報が記述されないため、記述の誤りをそのまま見過ごしてしまうことがあります。showSystem 関数は、分離されたモデルファイルとデータファイルから最終的に構成される目的関数 Objective, 制約式 Constraint, 対称行列 SymmetricMatrix の具体的な情報を出力します。showSystem 関数は以下の書式で記述されます。

```
showSystem();
```

次のようなモデルを考えます。

```
Set S;
Element i(set = S);
Parameter c(index = i);
Parameter a(index = i);
Variable x(index = i);
Objective f(type = maximize);
f = sum(c[i] * x[i], i);
x[i] <= a[i];
showSystem();
```

データファイルは dat 形式で次のように与えられています。

```
c = [1] 13 [2] 7 [3] 201 [4] 14 [5] 23;
a = [1] 23 [2] 5 [3] 4 [4] 12 [5] 1;
```

この場合、次のような出力がなされます。

```
1-1 (a.smp:8): x[1] <= 23
1-2 (a.smp:8): x[2] <= 5
1-3 (a.smp:8): x[3] <= 4
1-4 (a.smp:8): x[4] <= 12
1-5 (a.smp:8): x[5] <= 1
objective (a.smp:7 name="f"): 13*x[1]+7*x[2]+201*x[3]+14*x[4]+23*x[5] (maximize)
```

オブジェクトに name = で名前を与えてから showSystem でモデル情報を出力すると、表示に指定した名前が使われますのでわかりやすくなります。上記の例に name を付与した次のモデルに対しては、以下のように出力されます。

```
Set S(name = "S");
Element i(set = S);
Parameter c(name = "c", index = i);
Parameter a(name = "a", index = i);
```

```
Variable x(name = "valx", index = i);
Objective f(name = "obj", type = maximize);
f = sum(c[i] * x[i], i);
Constraint co(name = "co", index = i);
co[i] = x[i] <= a[i];
showSystem();
```

出力

```
1-1 (a.smp:9 name="co[1]"): valx[1] <= 23
1-2 (a.smp:9 name="co[2]"): valx[2] <= 5
1-3 (a.smp:9 name="co[3]"): valx[3] <= 4
1-4 (a.smp:9 name="co[4]"): valx[4] <= 12
1-5 (a.smp:9 name="co[5]"): valx[5] <= 1
objective (a.smp:7 name="obj"): 13*valx[1]+7*valx[2]+201*valx[3]+14*valx[4]+23*valx[5]
(maximize)
```

showSystem 関数は、引数に制約式を与えることで、その制約式のみを出力できます。name を付与したモデルに関して、次のように showSystem 関数の引数を変更した場合、以下が出力されます。

```
showSystem(co[1]);
```

出力

```
1-1 (a.smp:9 name="co[1]"): valx[1] <= 23
```

引数に条件式を与える事で、出力させる制約式の範囲を制限させることもできます。次の例では、co[1], co[2] の情報のみを出力させています。

```
showSystem(co[i], i < 3);
```

出力

```
1-1 (a.smp:9 name="co[1]"): valx[1] <= 23
1-2 (a.smp:9 name="co[2]"): valx[2] <= 5
```

半正定値制約に対して showSystem 関数を用いた場合、以下が出力されます。

```
SymmetricMatrix X((i, j));
X["1, 1"] = 2 * x[1];
X["1, 2"] = 1;
X["2, 2"] = x[2];
X >= 0;
showSystem();
```

出力

```
1-1 (a.smp:8): 2*(x[1]) (sdpconselem)
1-2 (a.smp:8): 1 (sdpconselem)
1-3 (a.smp:8): x[2] (sdpconselem)
```

10.6 解ファイル出力制御

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。解ファイルの詳細については Nuorium Optimizer マニュアルの「3. 解ファイル」をご参考ください。

ここでは、コマンドライン上で実行した場合の、解ファイル出力制御について説明します。

デフォルトではモデルファイル名で解ファイルが出力されます。例えばモデルファイル名が「model.smp」の場合、解ファイルは「model.sol」という名前で出力されます。

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ができます。モデルファイル内で options で変更する場合は以下です。

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

求解オプションファイル nuopt.prm で変更する場合は以下です。

```
output:name = ファイル名
```

上記のように設定すると「ファイル名.sol」という名前で解ファイルが出力されます。

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抑制することができます。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。

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

解オプションを用いて抑制するには、次のように記述します。

```
output:name = _NULL_
```

10.7 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
```

SIMPLE オブジェクトの内, 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;
```

このような出力制御は Parameter/Expression の出力に時間を要する場合などに有効です.

第 11 章

その他の機能

11.1 Intel oneAPI Math Kernel Library のリンク

Intel oneAPI Math Kernel Library (以下, oneMKL) は, Intel Corporation が開発している, BLAS, LAPACK, FFT を含む数値計算ライブラリです⁴. oneMKL は以下の URL からダウンロードして利用することができます.

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html>

oneMKL がインストールされている 64bit コンパイラ環境では, mknuopt で oneMKL をリンクした実行可能ファイルをビルドすることで, 計算速度を高速化することができる場合があります. mknuopt で oneMKL をリンクする場合, 環境変数 NUOPT_MKL_LIBPATH に oneMKL ライブラリへのパスを設定してください.

以下に oneMKL をデフォルトのパスにインストールした場合の設定例を示しますが, インストール時の設定を変更した場合, 適宜読み替えてください.

- Windows 環境で oneMKL がデフォルトのパスにインストールされている場合, 次のように設定してください.

`C:\Program Files (x86)\Intel\oneAPI\mkl\latest\lib\intel64`

- Linux 環境で oneMKL がデフォルトのパスにインストールされている場合, 次のように設定してください.

`/opt/intel/oneapi/mkl/latest/lib/intel64/`

11.2 MPS ファイルおよび LP ファイルへの変換

11.2.1 変換方法

モデリング言語 C++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.

⁴Intel はアメリカ合衆国およびその他の国における Intel Corporation の商標です.

```
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
```

11.2.2 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 x_1, x_2, \dots 、関数名は F_1, F_2, \dots 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 12 章

Nuorium Optimizer/SIMPLE FAQ

本章では、ある程度 Nuorium Optimizer に慣れた方が陥りやすい誤りをまとめました。より初歩的な FAQ に関しては、「Nuorium Optimizer/SIMPLE チュートリアル」をご参照ください。

12.1 浮動小数点エラー

モデルの中に変数の割り算などが入っていないでしょうか。変数は初期値を与えないと初期値 0 と解釈されますので変数の割り算は浮動小数点エラーの原因となります。

```
Variable x;  
1 / x >= 5; // 浮動小数点エラーの原因
```

log 関数に 0 を与えてしまった場合も、浮動小数点エラーとなってしまいます。

```
Variable x;  
log(x) >= 3; // 浮動小数点エラーの原因
```

なお、変数に初期値を与える方法に関しては [5.1 変数クラス Variable](#) や [8 データファイル](#) を参照してください。

12.2 整数の割り算

C++SIMPLE は C++ を用いて実装されているため、「整数/整数」は切り捨てられた整数と解釈されてしまいます。例えば、次の例では定数 a に 1/6 を設定しようとしていますが、実際には 0 が与えられてしまいます。

```
Parameter a;  
a = 1 / 6; // 0 が設定される
```

1/6 を与えるには、小数で記述する必要があります。

```
Parameter a;  
a = 1.0 / 6.0;
```

12.3 添字付けに関するエラー

12.3.1 一次元の場合

文字列が集合の要素である場合は、個別に代入する際ダブルクォート"で囲む必要があります。

```
Set S = "p q"; // 文字列が要素
Element i(set = S);
Parameter a(index = i);
a["p"] = 2;
```

12.3.2 二次元の場合

二次元の添字を持つ場合、個別に代入する際には対象の集合の要素が文字列であるかないかにかかわらず、全体をダブルクォート"で囲む必要があります。

一部に Element が直接付随している場合は全体をダブルクォート"で囲む必要はありません。

```
Set S = "p q";
Set T = "1 2 3";
Element i(set = S), j(set = T);
Parameter a(index = (i, j));
a["p, 1"] = 2; // "で囲む
a["p", j] = 3; // p のみ"で囲む
a[i, j] = 4;   // "で囲まなくて良い
```

12.3.3 三次元以上の場合

三次元以上の添字を持つ場合、一部に Element が直接付随している場合であっても、残りの部分に二次元以上の箇所が残る場合は、ダブルクォート"で囲む必要があります。

```
Set S = "p q";
Set T = "1 2 3";
Set U = "r s"
Element i(set = S), j(set = T), k(set = U);
Parameter a(index = (i, j, k));
a["p, 1, r"] = 2; // 全体を"で囲む
a["p, 1", k] = 3; // 一部"で囲む
a["p", j, "r"] = 4; // 文字のみ"で囲む
a[i, "1, r"] = 5; // 一部"で囲む
```

```
a[i, j, "r"] = 6;    // 文字のみ"で囲む
a[i, 1, k] = 7;     // "で囲まなくて良い
a["p", j, k] = 8;   // 文字のみ"で囲む
a[i, j, k] = 9;     // "で囲まなくて良い
```

12.3.4 一時オブジェクトの利用の禁止

添字付けにおいて、宣言時に index 引数に対して setOf などで作成した一時オブジェクトを渡すのは非推奨です。以下は非推奨および推奨の記述例です。

```
Set I = "0 1 2";
Element i(set=I);
Variable x(index=i);
0 <= x[i] <= 1;

// 以下は非推奨の記述です.
// Expression y(index=setOf(i, i==0));

// 以下は推奨される記述です.
// 集合を事前に作成します.
Set tmpS;
tmpS = setOf(i, i==0);
Element tmps(set=tmpS);
Expression y(index=tmps);
```

12.4 for 文内部における宣言

C++SIMPLE は C++ との親和性が高いため、例えば for 文などを記述することができます。ただしこのような制御構文の内部に SIMPLE オブジェクトを定義する場合は注意が必要です。例えば for 文を記述すると for 文内部で宣言された SIMPLE オブジェクトは、for 文内部の回数分だけされます。

以下は具体例です。

```
for(int a = 0; a < 3; ++a)
{
    Variable x;
}
```

上記のように記述すると Variable オブジェクトである x が三つ定義されます。

制御構文である for 文や while 文を用いる場合は、これらの外において SIMPLE オブジェクトを定義

することが推奨されます。ただし定数 `Parameter` については、制御構文内部で利用されることが想定されている「可変定数 `VariableParameter`」があります。詳細は [9.4 可変定数](#) を参照してください。

12.5 宣言の引数に SIMPLE オブジェクトを含んだ式を入れる

SIMPLE オブジェクトの宣言ではしばしば引数をとりますが、この時に引数に SIMPLE オブジェクトを含んだ式をいれると評価を誤る可能性があります。

例えば以下のような書き方は、誤って評価される可能性があります。

```
Parameter a(index=i);
Sequence Seq(from=0, to=sum(a[i], i), by=1);
```

上述のように書く場合は、一度 `double` などに変換することがおすすめです。

```
Parameter a(index=i);
Parameter b;
b = sum(a[i], i);
// double 型への変換
double c = b.val.asDouble();

Sequence Seq(from=0, to=c, by=1);
```

12.6 複数の代入の禁止

C++SIMPLE においては、カンマ「`,`」は条件式という特別な意味を持ちます。そのため、以下のような記述をすることはできません。

```
Parameter c(index=i);
c[1] = 1, c[2] = 2;
```

上記のように記述したい場合は二文に分ける必要があります。セミicolon「`;`」で区切ることで一行におさめることも可能です。

以下は複数の代入を一行におさめている例です。

```
Parameter c(index=i);
c[1] = 1; c[2] = 2;
```

12.7 `.val` と定義域外エラー

変数の値を `.val` で参照するとき、添字の範囲外となります。例えば以下のようなケースです。

```

Set A;
Element a(set=A);
Set B;
Element b(set=B);
Set AB(dim=2, superSet=(A, B));
Element ab(set=AB);
AB="A1 B1 A2 B2";

Variable x(index=AB);
0 <= x[ab] <= 1;

options.method="simplex";
solve();

x[a,b] = x[a,b].val, (a,b)<AB; // NG

```

このようなときは、変数の添字が範囲内におさまるように記述する必要があります。以下は正しい記述例です。

```

x[ab] = x[ab].val; // OK

```

12.8 両辺が定数で満たされない制約式

添字を伴う制約式を展開したときに、添字が空集合の場合、制約式は定数となります。以下のような例を考えます。

```

Set S;
S = "1 2 3";
Element i(set=S);
Parameter a(index=i);
IntegerVariable x(type=binary, index=i);
Expression e;
e = sum(a[i]*x[i], i);
e >= 1;

```

上記の場合、定数 a は 0 のため e は 0 となります。したがって、 $0 \geq 1$ という必ず満たされない制約式になります。このため以下のようなエラー出力がされます。

```

(SIMPLE 215) 制約式 1 は以下の式に等価です "0 >= 1" (常に満たされない).

```

定数となる制約式がソフト制約あるいはセミハード制約で、解法が制約充足ソルバ wcsp の場合、該

当の制約式は無視されます。具体的には以下のようなモデルを考えます。

```
Set S;  
S = "1 2 3";  
Element i(set=S);  
Parameter a(index=i);  
IntegerVariable x(type=binary, index=i);  
Expression e;  
softConstraint(1);  
e = sum(a[i]*x[i], i);  
e >= 1;  
options.method = "wcsp";
```

このようなモデルの場合、 $e \geq 1$ は無視されます。

付録 A

SIMPLE/mknuopt のエラー/警告メッセージ

A.1 SIMPLE のエラー/警告メッセージ

次表は SIMPLE の実行時のエラー/警告メッセージ一覧です。メッセージは英語（UNIX 版）あるいは日本語（Windows 版）で、下の表では両方が並べて記述されています。

エラー 番号	エラーメッセージ
	説明
1	(SIMPLE 1) Infeasible bound for variable XX(YY) (SIMPLE 1) 変数 XX について矛盾した上下限が与えられました (YY)
	変数 XX に与えられた上下限が矛盾しています（モデル全体を通して上下限が結合された結果についてこのチェックが行われます）。
3	(SIMPLE 3) Infeasible bound for constraint XX (YY) (SIMPLE 3) 制約 XX について矛盾した上下限が与えられました。 : (YY)
	制約式 XX に与えられた上下限が矛盾しています（モデル全体を通して上下限が結合された結果についてこのチェックが行われます）。
5	(SIMPLE 5) In datafile, Around c's definition: Expected element instead of d. (SIMPLE 5) データファイルの c の定義の付近において d とあるところには添字がなくてはなりません。
	使われたオブジェクトの添字の数が一致していません。（モデル中のオブジェクトの定義の際の添字の次元（index = ?）とそれが使われた時点（[i, j] 等）での添字の次元が一致しているかどうかを確認してください。）
6	(SIMPLE 6) In datafile, Around c's definition: Expected element instead of d. (SIMPLE 6) データファイルの c の定義の付近において d とあるところには添字がなくてはなりません。
	添字の現われるべき所に無効な文字（"[", "]", "... " など）が使われました。
10	(SIMPLE 10) In datafile, Around c's definition: Expected element instead of d. (SIMPLE 10) データファイルの c の定義の付近において d とあるところには添字がなくてはなりません。
	データファイルにおいて、添字付きオブジェクトに代入される内容（等号の右側）に "[]" が現れていません。

エラー 番号	エラーメッセージ
	説明
11	(SIMPLE 11) In datafile, Around c's definition: Expected data instead of d. (SIMPLE 11) データファイルの c の定義の付近において d とあるところにはデータがなく てはなりません。
	データファイルにおいて、添字付きオブジェクトに代入される内容（等号の右側）に値が 現れていません。
15	(SIMPLE 15) Internal problem in SIMPLE. (SIMPLE 15) システム内部の問題が起きました。
	SIMPLE の内部エラーが起きました。（nuopt-support@ml.msi.co.jp へお知らせください。）
18	(SIMPLE 18) Internal problem in SIMPLE. (SIMPLE 18) システム内部の問題が起きました。
	SIMPLE の内部エラーが起きました。（nuopt-support@ml.msi.co.jp へお知らせください。）
19	(SIMPLE 19) No auto-assignment performed for constant set. (SIMPLE 19) 自動代入によって定義されない集合があります。
	（警告）通常なら自動追加が行われる場合ですが、自動追加先が定数集合であるので行わ れません。（Set を定義する時、superSet に定数集合等を指定するとこの警告メッセージ が現れます。）
20	(SIMPLE 20) In datafile, Around c's definition: Can't match the pattern "from ... to". (SIMPLE 20) データファイルの c の定義の付近において "from ... to" 形式の文法が正しく ありません。
	データファイルに from/to の省略記号"..."が現れましたが、from/to がペアになってい ません。
22	(SIMPLE 22) In datafile, Around c's definition: Number of values not matched with the index dimension of set. (SIMPLE 22) データファイルの c の定義の付近において添字の数とモデル中の集合の次元 が合致しません。
	データファイル中で、あるオブジェクトに代入される内容の"[]"の中に現れる Element の 数がそのオブジェクトの定義時の index の数と合っていない。
23	(SIMPLE 23) In datafile, Around c's definition "..." or ".." appeared in the head or the tail. (SIMPLE 23) データファイルの c の定義の付近において "..." または ".." が定義の先頭か末 尾に現われました。
	データを表す文字列の先頭または末尾に from/to の省略記号"..."/".."が現われました。
24	(SIMPLE 24) Attempt to find the maximum of an empty set. (SIMPLE 24) 空集合から最大要素を求めようとしてしました。
	空集合から最大要素を求めようとしています。

エラー 番号	エラーメッセージ
	説明
36	(SIMPLE 36) Sequence is empty.
	(SIMPLE 36) Sequence が空となっています。
	列 Sequence が空です。
37	(SIMPLE 37) Only one-dimensional data can be specified as Sequence.
	(SIMPLE 37) Sequence の指定は 1 次元のデータに限って有効です。
	列 Sequence の定義（宣言）時に dim = 1 以外を与えました。
39	(SIMPLE 39) Sequence data not matched with the format: "first .. last, step".
	(SIMPLE 39) Sequence の入力データ形式は "開始要素 .. 終了要素 , 増分" でなければなりません。
	列 Sequence の定義（宣言）で、最初の値、列の最後の値、増分（default = 1）のいずれかが抜けています。
40	(SIMPLE 40) Inappropriate operation on Sequence.
	(SIMPLE 40) Sequence に対する正しくない演算が行なわれました。
	列 Sequence に対して無効な演算を行おうとしました。
41	(SIMPLE 41) Only one-dimensional data can be specified as Sequence.
	(SIMPLE 41) Sequence の指定は 1 次元のデータに限って有効です。
46	(SIMPLE 46) Attempt to find the maximum of an empty sequence.
	(SIMPLE 46) 空の Sequence から最大要素を求めようとしてしました。
	空列から最大要素を求めようとしてしました。
47	(SIMPLE 47) Note: No auto-assignment performed for Sequence.
	(SIMPLE 47) Sequence に対する自動代入は行なわれません。
	（警告）通常なら自動追加が行われる場合ですが、自動追加先が Sequence なので行われません。
49	(SIMPLE 49) A call has been made to an inappropriate function.
	(SIMPLE 49) 正しくない関数呼び出しが行なわれました。
	無効な関数呼び出しが行なわれました。
53	(SIMPLE 53) Operation between elements of different dimension.
	(SIMPLE 53) 要素が異なる次元を持つ集合同士で演算が行なわれました。
	属している集合の次元が異なる Element の間に演算が行なわれました。
54	(SIMPLE 54) Invalid operation on multi-dimensional elements.
	(SIMPLE 54) 多次元の要素に対して、不正な演算がされました。
	次元が 2 以上の要素に対して加算・乗算などの演算は禁止されています。

エラー 番号	エラーメッセージ
	説明
55	(SIMPLE 55) The operation prev/next is only valid for elements in OrderedSet. (SIMPLE 55) 順序集合 (OrderedSet) に含まれない要素に対して prev/next 演算は使用できません.
	OrderedSet クラスのメンバ関数である prev 関数や next 関数は引数として Element を取りますが、この際 OrderedSet に含まれない Element を指定しています.
58	(SIMPLE 58) In operation s1-s2, s1 doesn't include s2. (SIMPLE 58) 集合の引算 s1-s2 で、s2 は s1 に含まれていませんでした.
	集合の差分 (s1 - s2) 演算において、s2 は s1 の部分集合になっていません.
59	(SIMPLE 59) Fixed value (b) out of defined range of (a). (SIMPLE 59) 要素 (a) に対して定義域外の値 (b) を代入しようとしてしました.
	Element がその定義範囲外の値に固定されようとしています.
60	(SIMPLE 60) Inappropriate character[s] included in subscript. (SIMPLE 60) 添字に正しくない文字が含まれています.
	添字として使うことができない文字が含まれています.
62	(SIMPLE 62) Comparison between elements of different dimension. (SIMPLE 62) 異なる次元を持つ要素の間に比較が行なわれました.
	属している集合の次元が異なる Element 同士間で比較が行われました.
64	(SIMPLE 64) Constraint: subscript not matched. (SIMPLE 64) 制約式 (Constraint) の添字が合致しません.
	制約式の添字エラーです.
65	(SIMPLE 65) Internal problem in SIMPLE. (SIMPLE 65) システム内部の問題が起きました.
	SIMPLE の内部エラーが起きました. (nuopt-support@ml.msi.co.jp へお知らせください.)

エラー 番号	エラーメッセージ
	説明
67	(SIMPLE 67) Index error in reference of "XX" with index of dimension YY but should be with index of dimension ZZ. (SIMPLE 67) 参照オブジェクト "XX" の添字付けに誤りがあります。次元 ZZ の添字を付けるべきですが、次元 YY の添字が付けられています。
	代入の右辺や計算式の中にあるオブジェクト XX に正しい添字が与えられていません。 XX が添字なしなのに添字付けられている XX に添字をつけるべきなのに添字付けられていない XX の添字の次元が違う というケースが該当します。 例えば付けられた添字の次元が誤っている場合に出力されます。例を挙げると Variable <code>x(index = (i, j));</code> と 2 次元で宣言されているにも関わらず、 <code>x[1] == 1;</code> というように 1 次元の添字が付けられていると本エラーが出力されます。ただし、一見 2 次元の添字を付けているように見える場合にも本エラーが出力されることがあります。例えば先の例ですと、 <code>x[1, 1] == 1;</code> と記述すると出力されます。このような int 型の値の並びは、SIMPLE の仕様で 2 次元の添字とはみなされません。この場合は <code>x["1, 1"] == 1;</code> と記述します。
70	(SIMPLE 70) XX used inappropriate manner. (SIMPLE 70) XX の使い方に誤りがあります。
	以下の例のように添字の扱いに問題がある場合生じます。 代入の左辺と右辺で現れる添字が違う場合 <code>sum(x[i], i) == p[i];</code> のように記述した場合（左辺で和をとるために使った添字 <code>i</code> は右辺で使えない） <code>x[i] >= 0, j > 3;</code> のように記述した場合（制約の条件式 <code>j > 3</code> に制約式 <code>x[i] >= 0</code> に現れない添字 <code>j</code> が含まれており不適切）
72	(SIMPLE 72) Index of LHS causes ambi : guilty (index value should be unique). (SIMPLE 72) 代入の左辺の添字の値に重複があるので意味が曖昧です。
	<code>a[i % 2] = 1</code> というように代入時において添字の値に重複がある場合生じます。
74	(SIMPLE 74) Dependent subscript used inappropriate manner. (SIMPLE 74) 式や宣言において他の添字に依存している添字は、その依存している添字と一緒に現れねばなりません。
	添字が他の添字に依存しているにも関わらず、単独でしかも固定されないまま使われました。（例えば <code>Element j(set = S[i]);</code> として宣言された <code>Element</code> が <code>i</code> を伴わず、固定されずに使われた場合に生じます。）

エラー 番号	エラーメッセージ
	説明
75	(SIMPLE 75) XX cannot be indexed. (SIMPLE 75) "XX" に添字は付けられません。
	代入の左辺にあるオブジェクト XX は添字なしで宣言されていますが添字を付けて値の設定や代入が成されました。
76	(SIMPLE 76) In assignment, dimension of object "XX" conflict. (SIMPLE 76) 代入されている "XX" の添字の次元に矛盾があります。
	代入の左辺にあるオブジェクト XX が宣言された集合の次元と違う次元の添字を付けて値の設定や代入が成されました。
77	(SIMPLE 77) Index required in assignment to "XX" (SIMPLE 77) "XX" の代入には添字が必要です。
	XX は N 個の添字を付けて定義されていますが、スカラーのように添字付けせずに値を設定しようとしています。
78	(SIMPLE 78) Unindexed assignment to "a" or unindexed constraint "a" is done ignoring condition. Use "if" to condition. (SIMPLE 78) 式 "a" の代入あるいは制約式 "a" は添字付けられていないので、条件式の成立にかかわらず行われます。条件付けするには if 文を使ってください。
	(警告) 条件式が、添字付けられていない Expression の代入や制約式に現れています。条件付けをするには if 文を使う必要があります。
79	(SIMPLE 79) Condition never satisfied because dimation of element mismatch. (SIMPLE 79) 添字の次元が違うので決して充足されない条件式が現れました。
	次元の違う集合と要素に関する条件式が検出されました。
80	(SIMPLE 80) Undefined Element in Condition. (SIMPLE 80) 条件式に値が未定義の添字が使われました。
81	(SIMPLE 81) # of subscript mismatch. (SIMPLE 81) 添字の数が合致しません。
	添字に関するエラーが検出されました。
82	(SIMPLE 82) Subscript "XX" of "YY" out of range. (SIMPLE 82) YY の添字が定義域外である "XX" となりました。
	オブジェクトに付けられた添字の値が、その定義の際に (index = ?) 設定された添字の値の範囲をはみ出しました。
91	(SIMPLE 91) Operation between sets of different dimension. (SIMPLE 91) 異なる次元の要素を持つ集合の間に演算が行なわれました。
	要素の次元 (dim) が異なる集合間で演算が行なわれた。

エラー 番号	エラーメッセージ
	説明
92	(SIMPLE 92) No auto assignment performed for result of operation of Sets. (SIMPLE 92) 演算結果から生成された集合に対する自動代入は行なわれません。
	(警告) 通常なら自動追加が行われる場合ですが、自動追加先が集合演算の結果（和集合）であるので行われません。
93	(SIMPLE 93) Internal problem in SIMPLE. (SIMPLE 93) システム内部の問題が起きました。
	SIMPLE の内部エラーが起きました。（nuopt-support@ml.msi.co.jp へお知らせください。）
97	(SIMPLE 97) No auto-assignment performed for sets made by "setOf". (SIMPLE 97) "setOf" で作った集合に対する自動代入は行なわれません。
	通常なら自動追加が行われる場合ですが、自動追加先が setOf の結果であるので行われません。
98	(SIMPLE 98) "from ... to" has been defined before data file reading. (SIMPLE 98) "from ... to" がデータファイルを読み込む前に、定義されました。
102	(SIMPLE 102) Set assignment: dimension conflict. (SIMPLE 102) 集合に対する代入で等号の右辺と左辺の集合の要素の次元が合致しません。
	集合同士の代入（またはデータファイルからの読み込み）の場合、右辺の集合の要素の次元と左辺の集合の要素の次元が合っていない。
103	(SIMPLE 103) Set and superSet should have same dimention. (SIMPLE 103) 異なる次元を持つ集合が "superSet" として使用されました。
	要素の次元の違う集合同士に包含関係を定義しようとした。 (Set T(dim = 2); Set S(dim = 1, superSet = T); とした場合等.)
104	(SIMPLE 104) XX cannot be a superset of YY. (SIMPLE 104) XX は YY の superSet として使用することができません。
	集合 XX は YY の superSet になることができません。例えば XX が数列集合 Sequence の時 YY の superSet として設定することはできません。
105	(SIMPLE 105) Argument error: a of b and c . (SIMPLE 105) 引数の問題 a の b と c
	オブジェクト定義時の引数エラーです。
110	(SIMPLE 110) Argument: inappropriate use of "index". (SIMPLE 110) "index=" が誤った場所に使用されています。
	オブジェクト定義時に属性引数 index が無効な場所に現れました。
111	(SIMPLE 111) Assignment: rhs includes free subscript. (SIMPLE 111) 代入の右辺に決定できない添字がありました。
	代入の右側に不定になる添字 (Element) が現れました。

エラー 番号	エラーメッセージ
	説明
113	(SIMPLE 113) Inappropriate assignment. (SIMPLE 113) 正しくない代入が行なわれました。
	その他の代入に関するエラーが検出されました。
114	(SIMPLE 114) Argument: Inappropriate use of "set". (SIMPLE 114) "set=" が誤った場所に使用されています。
	オブジェクト定義時に属性引数 <code>set</code> が無効に使われました。
119	(SIMPLE 119) Assignment: "[" or "]" occurred both in lhs and rhs of = when assigning a set by a string. (SIMPLE 119) オブジェクト [添字] = 文字列という代入文で、文字列の中に "[" または "]" が現れました
	文字列を集合へと代入する際、文字列の中身である集合の値と代入の左辺両方が添字付けされていました。(文字列による集合への代入を行う際には、等号の両側に同時に "[" または "]" が現われることを禁止しています。例えば、Set S; S[1] = "[1] a 1 2 [1] 3"; はエラーになります。)
120	(SIMPLE 120) Operators "+=", "-=", "*=", "/=", "++", and "--" are not allowed here. (SIMPLE 120) 演算子 "+=", "-=", "*=", "/=", "++", と "--" は使用することができません。
	演算子 += -= /= ++ -- を不適切なオブジェクトに適用しました。
121	(SIMPLE 121) Inappropriate constraint specification: ("parameter <= expr ==> parameter" is not allowed). (SIMPLE 121) 有効でない制約式 ("parameter <= expr ==> parameter") が使用されました。
	SIMPLE が解釈できない制約式の形 (定数式<= 式>= 定数式, 定数式>= 式<= 定数式) が現れました。
123	(SIMPLE 123) In datafile, Around c's definition: Syntax error(around "d"). (SIMPLE 123) データファイルの c の定義の付近において文法的な誤りがあります (問題のありそうな文字列 "d").
	データファイルの文法エラーです。データファイルを定義するときにデータの区切の ";" を忘れている、またはデータファイルの文字コードに問題がある可能性があります。Nuorium Optimizer Windows 版で利用可能なデータファイルの文字コードは Shift_JIS (SJIS) のみです。異なる文字コードの場合は事前に SJIS に変換してください。Nuorium のファイルメニューにある「文字コードを指定して保存」から SJIS と指定して変換することも可能です。
127	(SIMPLE 127) String uncompleted (missing a '"' mark). (SIMPLE 127) 不完全な String (" が足りません)。
	値としての文字列の " がペアになっていません。

エラー 番号	エラーメッセージ
	説明
128	(SIMPLE 128) String is empty.
	(SIMPLE 128) データ文字列が空です.
	値としての文字列が空です.
129	(SIMPLE 129) String contains space(s).
	(SIMPLE 129) データ文字列に空白が含まれています.
	値としての文字列の名前に空白が含まれています.
130	(SIMPLE 130) Inappropriate cast from character to int.
	(SIMPLE 130) 文字から数字へのキャストが行なわれました.
	演算等で、文字を数字 <code>int</code> へキャストする必要が生じましたが、失敗しました.
131	(SIMPLE 131) Inappropriate cast from character to double.
	(SIMPLE 131) 文字から <code>double</code> へのキャストが行なわれました.
	演算等で、文字を数字 <code>double</code> へキャストする必要が生じましたが、失敗しました.
132	(SIMPLE 132) In datafile, Around c's definition: Syntax error(around "d").
	(SIMPLE 132) データファイルの <code>c</code> の定義の付近において文法的な誤りがあります (問題のありそうな文字列 "d").
133	(SIMPLE 133) In datafile, Around c's definition<> and following data unmatched.
	(SIMPLE 133) データファイルの <code>c</code> の定義の付近において<> と 引き続くデータが整合していません.
135	(SIMPLE 135) Syntax error occurred within [...] .
	(SIMPLE 135) データファイルの <code>c</code> の定義の付近においてデータ書式の [...] の中に文法の問題が発生しました.
	データファイルにおける"[]"の中の定義が文法エラーとなっています.
136	(SIMPLE 136) Syntax error occurred within <...> .
	(SIMPLE 136) データファイルの <code>c</code> の定義の付近においてデータ書式の<...> の中に文法の問題が発生しました.
	データファイルにおける"< >"の中の定義が文法エラーとなっています.
137	(SIMPLE 137) In datafile, Around c's definition: Wild card in [] and following data mismatch.
	(SIMPLE 137) データファイルの <code>c</code> の定義の付近において [] の中のワイルドカードとその後にあるデータが不整合です.
138	(SIMPLE 138) In datafile, Around c's definition: Wild card in [] and following data mismatch.
	(SIMPLE 138) データファイルの <code>c</code> の定義の付近において [] の中のワイルドカードとその後にあるデータが不整合です.

エラー 番号	エラーメッセージ
	説明
139	(SIMPLE 139) The result dimension of parameter must be 1. (SIMPLE 139) Parameter の演算結果は次元 1 でなければなりません.
148	(SIMPLE 148) Argument: inappropriate use of "superSet". (SIMPLE 148) "superSet=" が誤った場所に使用されています.
	オブジェクト定義の属性引数 superSet が無効に使われました.
151	(SIMPLE 151) Set auto-assignment failed. (SIMPLE 151) 集合に対する自動代入はできません.
	集合に対する自動追加が失敗しました. (ある集合の自動追加を行わねば集合の包含関係が矛盾となることがわかりましたが, その集合は定数集合等である等の理由で自動追加ができない場合に生じます.)
159	(SIMPLE 159) No dual value: the model has not being solved or has been changed since last solving. (SIMPLE 159) dual 値が存在しません. モデルを解いていないか前回解を求めた後, モデル定義が変わりました.
160	(SIMPLE 160) Empty data can't be casted to double. (SIMPLE 160) 空データから double へのキャストが行なわれました.
	オブジェクトの参照可能な値を評価した結果空であるので, double へキャストすることができません.
163	(SIMPLE 163) No current value for empty constraint. (SIMPLE 163) 制約式が空なので制約式の値 (.val) が存在しません.
	宣言したのみで定義されていない制約式に対して現状値を調べようとしてしました.
164	(SIMPLE 164) Constraint "a" is empty and related value set to zero. (SIMPLE 164) 空の制約式"a"に関連する値は 0 として出力されます
	(警告) 宣言したのみで定義されていない制約式に対して初期値を調べようとしてしました.
165	(SIMPLE 165) Dual value of Constraint "a" is assumed to be zero (Constraint is empty or model is not solved). (SIMPLE 165) 制約式 "a" の双対変数は 0 として出力されます (一度も求解を行っていません).
	(警告) 求解していない状態では, Constraint の双対変数値は 0 として出力されます.

エラー 番号	エラーメッセージ
	説明
167	(SIMPLE 167) Leftmost part of Constraint "XX" is used for output. contains more than two constraints.
	(SIMPLE 167) 制約式 "XX" については最も左の部分が出力されます (二つ以上の制約式を含んでいます).
	(警告) 制約式 XX が複数回代入された場合や二つ以上に分解される様な定義 ($co = x[i] \leq y[i] \leq z[i]$ 等) を行った場合, その参照値の出力をすると最初の代入の内容かもっとも左の式に対する参照値が出力されます.
168	(SIMPLE 168) Objective can only be assigned once.
	(SIMPLE 168) 目的関数 (Objective) に対する代入は一度のみ可能です.
169	目的関数への代入を複数回行おうとした. (Objective には 1 回の代入のみが許されています)
	(SIMPLE 169) Argument: "type" Error!
	(SIMPLE 169) "type=" が誤って使われました.
171	オブジェクト定義時の属性引数 type の指定が無効である.
	(SIMPLE 171) Inappropriate assignment to Objective: only "Objective=Expression" is allowed.
	(SIMPLE 171) 目的関数 (Objective) に対する正しくない代入が行われました. "Objective = Expression" のみ有効です.
172	目的関数に対して変数を含まない式が代入された.
	(SIMPLE 172) Objective has not been assigned.
	(SIMPLE 172) 目的関数 (Objective) に対する代入が行なわれていません.
173	モデルを解く (solve() 関数) に未定義の (代入が行われていない) 目的関数を渡しました.
	(SIMPLE 173) Dual member only exists in Constraints and Variables.
	(SIMPLE 173) 双対変数値を制約式と変数以外について参照しようとしてしました.
174	制約式と変数以外のオブジェクトの dual 値を照会しました.
	(SIMPLE 174) Set data cannot be transformed to Parameter by .val .
	(SIMPLE 174) 集合の値 (.val) をパラメータ (Parameter) に変換しようとしてしました.
177	集合の現状値 val を定数 Parameter に変換しようとしてしました. (例外として集合の現状値のみは Parameter と等価に扱うことはできません. 表示したりダンプしたりするのみです.)
	(SIMPLE 177) No lower bound for empty constraint.
	(SIMPLE 177) 制約式の下限は存在しません. 制約式が空です.
178	宣言したのみで定義されていない制約式に対して下限値を調べようとしてしました.
	(SIMPLE 178) No upper bound for empty constraint.
	(SIMPLE 178) 制約式の上限は存在しません. 制約式が空です.
178	宣言したのみで定義されていない制約式に対して上限値を調べようとしてしました.

エラー 番号	エラーメッセージ
	説明
180	(SIMPLE 180) Argument: "from", "to" and "by" must be an integer type.
	(SIMPLE 180) 引数 "from", "to" および "by" は、int 型でなければなりません。
	列 Sequence の定義の際に引数 from,to,by はいずれも整数値を設定する必要があります。
181	(SIMPLE 181) Argument: "from" is required in defining a Sequence.
	(SIMPLE 181) Sequence を定義する時には、"from=" 指定が必須です。
	列 Sequence の定義に属性引数 from が現れていません。
182	(SIMPLE 182) Argument: "to" is required in defining a Sequence.
	(SIMPLE 182) Sequence を定義する時には、"to=" 指定が必須です。
	列 Sequence の定義に属性引数 to が現れていません。
186	(SIMPLE 186) Argument: "index" is ignored in the Interval/Sequence definition.
	(SIMPLE 186) Interval/Sequence を定義する時に、"index=" は指定できません。
	(警告) 範囲 Interval または列 Sequence の定義に属性引数 index が現れました。
187	(SIMPLE 187) Argument: "dim" is ignored in the Interval/Sequence definition.
	(SIMPLE 187) Interval/Sequence を定義する時に、"dim=" は指定できません。
	(警告) 範囲 Interval または列 Sequence の定義に属性引数 dim が現れました。
188	(SIMPLE 188) Argument: "left, right, oleft, oright" are ignored in the Sequence definition.
	(SIMPLE 188) Sequence を定義する時には、"left=, right=, oleft=, oright=" は指定できません。
	(警告) 列 Sequence を定義するとき、属性引数 left などは無視されます。
191	(SIMPLE 191) No assignment is allowed to Sets having both index and superSet.
	(SIMPLE 191) 添字と superSet を同時に持つような集合に対する直接の代入はできません。 (データファイル経由のみが許されます)
	添字付きで親集合を持つ集合に対する代入を行おうとしました。(実装の都合上、このケースでの直接の代入は禁止されており、自動追加のみが許されます)
192	(SIMPLE 192) Problem in solve() before solution process (no result)
	(SIMPLE 192) アルゴリズム実行時の問題 (結果がでません)
	Nuorium Optimizer が前処理でエラーを起こしました。
193	(SIMPLE 193) Problem in solve(): XX
	(SIMPLE 193) アルゴリズム実行時の問題 XX
	Nuorium Optimizer が計算途中にエラーを起こしました。[解出力あり]
194	(SIMPLE 194) Problem in solve() (no result) XX
	(SIMPLE 194) アルゴリズム実行時の問題 (結果がでません) XX
	Nuorium Optimizer が計算途中にエラーを起こしました。[解出力なし]
195	(SIMPLE 195) Index with SuperSet error.
	(SIMPLE 195) Index と SuperSet の定義の問題。
	Superset と添字つき集合の添え字付けに矛盾があります。

エラー 番号	エラーメッセージ
	説明
196	(SIMPLE 196) Objective function is constant. (SIMPLE 196) 目的関数がコンスタントとなっています。
	(警告) 目的関数が定数であることがモデルの解釈によって明らかになりました。 目的関数に変数を含まない場合、このような警告が出力されます。例えば、以下のような記述をした場合です。 <code>Objective obj;</code> <code>obj = 1;</code> このような警告が出たときは、データがモデルに適切に渡っていないことが多いです。データが適切に渡っているか確認することをお勧めします。
197	(SIMPLE 197) Set of fixed element cannot be a superset. (SIMPLE 197) メンバーが固定している集合を <code>superSet</code> として使用することはできません。
	代入によって固定した要素を <code>superSet</code> として使おうとしました。
198	(SIMPLE 198) The specified dimension in the <code>.slice</code> function is out of range. (SIMPLE 198) <code>slice</code> 関数で指定された次元は範囲外です。
	関数 <code>slice</code> の引数は <code>slice</code> しようとしている集合の次元数以下である必要がありますが、それを超えています。
199	(SIMPLE 199) Character-value("XX") appeared in constraint/objective definition. (SIMPLE 199) 文字列値 ("XX") が制約式や目的関数の定義に現れています。
	文字列 XX に対する演算が制約式や目的関数の定義中に現れました。Parameter の値で不適切な個所に文字列が現れている可能性があります。
200	(SIMPLE 200) <code>simple_[f]printf()</code> ignored Set object in the arglist. (SIMPLE 200) <code>simple_[f]printf()</code> は引数並び中の Set オブジェクトを無視しました
	(警告) <code>simple_[f]printf</code> は Set の出力を行いません。
201	(SIMPLE 201) You cannot write constraint in <code>simple_[f]printf()</code> 's arglist. (SIMPLE 201) <code>simple_[f]printf()</code> の引数並び中に変数を含む式の定義は記述できません。
	制約式を <code>simple_[f]printf()</code> の引数並びに記述しました。
202	(SIMPLE 202) {...} appears inappropriate position. (SIMPLE 202) {...} が正しくない場所に現れています。
	データファイルや文字列中で自然数の連続を示す... の現れる場所が不正です。
203	(SIMPLE 203) Insufficient # of Data after {...}...{...} expected XX but found YY. (SIMPLE 203) {...}...{...} にひきつづくデータ個数が不正です (XX 個必要ですが YY 個あります)。
	データファイルの省略形 "{...}" において、期待されるデータの個数が異なります。
204	(SIMPLE 204) Try to unlock Set with noname. (SIMPLE 204) 名前なしの集合を <code>unlock()</code> しようとしてしました。
	集合演算の結果など、陽に宣言されていない集合に対して <code>unlock()</code> を呼びました。

エラー 番号	エラーメッセージ
	説明
205	(SIMPLE 205) Any Set without name is already locked. (SIMPLE 205) 名前なしの集合は常に lock() された状態ですので lock() のコールは不要です.
	(警告) 集合演算の結果など、陽に宣言されていない集合に対して lock() を呼びました. もともと lock() されているという仕様なので lock() のコールは不要です.
206	(SIMPLE 206) Locked Set "XX" cannot be assigned. (SIMPLE 206) lock() された集合 "XX" に代入が行われました.
	集合 XX を lock() によってロックしているのに、代入が行われようとした.
207	(SIMPLE 207) Auto-assignment mechanism try to add some element[s] to locked Set "XX" In setting object "YY". (SIMPLE 207) データ "YY" の設定の際に、自動代入で lock されている集合 "XX" に要素が追加されようとした.
	集合 XX を lock() によってロックしている際に、自動代入によって新しい要素が追加されようとしています. YY へのデータ設定の添字に問題があります.
208	(SIMPLE 208) Note: Skip auto-assignment to locked base Set in assignment to XX (SIMPLE 208) XX の添字集合 (lock 済) への自動代入は抑制されました.
	(警告) 集合 XX を lock() によってロックしている際に、自動代入によって新しい要素が追加されようとしています (設定によって自動代入がチェックのみのモードになっている場合の出力).
209	(SIMPLE 209) Note: Skip assignment to locked superSet XX (SIMPLE 209) XX の superSet(lock 済) への代入は抑制されました.
	(警告) 集合 XX を lock() によってロックしているのに、代入が行われようとした (設定によって自動代入がチェックのみのモードになっている場合の出力).
212	(SIMPLE 212) User Termination(at Model Expansion) (SIMPLE 212) ユーザによる中断 (モデル展開)
	式の展開の途中にユーザによる中止が命令されました.
213	(SIMPLE 213) Warning from solve(): XX (SIMPLE 213) モデル実行時警告
	(警告) Nuorium Optimizer より警告 XX が出ました.
214	(SIMPLE 214) Warning constraint#XX reduce to "YY" (always satisfied). (SIMPLE 214) 制約式 XX は以下の式に等価です "YY" (常に満たされる).
	(警告) 式 XX の展開の結果, YY という形の常に満たされる制約式が現れました.
215	(SIMPLE 215) constraint#XX reduce to "YY" (never satisfied). (SIMPLE 215) 制約式 XX は以下の式に等価です "YY" (常に満たされない).
	制約式 (番号: XX) は「YY」に等価で、決して満たされないことがわかりました. 216 と同時に現れます.

エラー 番号	エラーメッセージ
	説明
216	(SIMPLE 216) Trivial and Infeasible constraint appeared. (SIMPLE 216) 常に Infeasible な制約式が現れました。
	式 XX の展開の結果, YY という形の明らかに満たすことのできない制約式が現れました (式の解釈の結果, 問題が実行不可能であることがわかりました). 215 と同時に現れます.
217	(SIMPLE 217) Internal problem XX in SIMPLE. (SIMPLE 217) システム内部の問題 XX が起きました。
	SIMPLE の内部エラー XX が起きました. (nuopt-support@ml.msi.co.jp へお知らせください.)
218	(SIMPLE 218) The header column have XX fields, but row#YY has ZZ fields. (SIMPLE 218) (ヘッダー行には XX 個のフィールドがありますが, YY 番目の行は ZZ 個の フィールドがあります).
	データファイルとして与えられた CSV ファイルのフィールド数エラーです. CSV ファイルのフィールド数はすべての行で同一でなければなりません.
219	(SIMPLE 219) Only the header row and no data row exist. (SIMPLE 219) 与えられた CSV ファイルにはヘッダー行しかありません。
	データファイルとして与えられた CSV ファイルにはヘッダー行しかありません。
220	(SIMPLE 220) No data row exist. (SIMPLE 220) 与えられた CSV ファイルには有効行がありません。
	データファイルとして与えられた CSV ファイルにはデータとして解釈できる部分がまっ たくありません。
221	(SIMPLE 221) Duplicate name "XX" (field# YY and ZZ) in the header row. (SIMPLE 221) 名前 "XX" がヘッダー行で重複しています (フィールド番号 YY と ZZ に現 れています).
	データファイルとして与えられた CSV ファイルのヘッダーが示す行名前は重複してはい けません。
222	(SIMPLE 222) Inappropriate field string "XX" at row# YY (SIMPLE 222) フィールドデータとして不適切な文字 "XX" が現れています。
	データファイルとして与えられた CSV ファイルに違法な文字が混ざっています。
223	(SIMPLE 223) Empty field at row#XX, field# YY. (SIMPLE 223) XX 行目の YY 番目のフィールドが空です。
	データファイルとして与えられた CSV ファイルに空のフィールドが混ざっています。
224	(SIMPLE 224) In reading scalar "YY" from CSV file "XX", found too many (ZZ) lines for scalar. (SIMPLE 224) CSV ファイル "XX" からスカラデータ "YY" を読もうとしましたが, この ファイルには合計 ZZ 行の無駄な行があります。
	(警告) データファイルとして与えられた CSV ファイルからスカラを与えようとしている 場合には, 行はヘッダ行以外は一行でなければなりません, それ以上の行があります。

エラー 番号	エラーメッセージ
	説明
225	(SIMPLE 225) Try to read data "XX" (with M index) from CSV file "YY" but it has too few(N) preceding column(s).
	(SIMPLE 225) データ "XX" (添字の数 M) を CSV ファイル "YY" から読もうとしています が、該当列の前に添字となるはずの列が N 列しかありません。 データファイルとして与えられた CSV ファイルの添え字の数が足りません（添え字の数は読み込もうとしているオブジェクトの定義から判定されますが、それから考えて足りません）。
227	(SIMPLE 227) Multiple data entry "XX" found in YY and ZZ
	(SIMPLE 227) データ "XX" についての記述が YY と ZZ に複数見つかりました。 XX というオブジェクトの内容をデータファイル YY と ZZ において二回以上定義しました。二回以上定義されているオブジェクトを発見するたびに現れます。231 番のメッセージとともに現れます。
228	(SIMPLE 228) Field#N of the first row (index) is empty. In reading data "XX" (with M index , 2D format) from CSV file "YY" .
	(SIMPLE 228) データ "XX" (添字の数 M, 2D 書式) を CSV file "YY" から読み込もうとしましたが、最初の行のフィールド N (添字として使われる) が空です。 XX というオブジェクトを 2D 書式で呼んでいるときに、最初の行には添え字の並びが来なければいけません、フィールド N が空になっています。
229	(SIMPLE 229) Error from XX, this object YY is not scalar.
	(SIMPLE 229) XX のコールを行ったオブジェクト YY はスカラではありません。 スカラでないオブジェクトに asDouble() (double 値への変換) のコールを行いました。
230	(SIMPLE 230) In datafile, Around XX's definition: dimension of element [YY] should be same as others.
	(SIMPLE 230) データファイルの XX の定義の付近において添字 [YY] の次元が他と異なっています。 添字付きオブジェクト XX に対するデータの並びで、YY という添字記述のみ次元が異なっています。 a = [1] 3.0 [2 3] 4.0 [5] 5.0; (モデル内の定義文字列に発見された場合にもこのエラーが出力されます)。
231	(SIMPLE 231) Multiple data definition found. This may cause performance deterioration. See the message window duplicate items.
	(SIMPLE 231) 同一のデータ定義が二つ以上あり、パフォーマンス下落を招く可能性があります。重複した品目についてはメッセージを参照してください。 データの二重定義が一つでもあると現れます。 options.multDataPolicy を 0 (デフォルト) ならばエラーになります。0 以外に設定すると警告の意味となり、実行は停止しません。

エラー 番号	エラーメッセージ
	説明
232	(SIMPLE 232) Proxy object is used before set.Can be used only after declaration. (SIMPLE 232) オブジェクトが宣言前に使われています。オブジェクトは宣言した後に利用しなければなりません。
	宣言する前のオブジェクト (Parameter, Variable など) を式の定義に利用しました。
233	(SIMPLE 233) Floating point arithmetic error. (SIMPLE 233) 浮動小数点例外が発生しました。
	<p>ゼロ除算等の理由により浮動小数点演算において例外が発生しました。</p> <p>モデルの中に変数の割り算または log などが入っていないでしょうか。変数は初期値を与えないと初期値 0 と解釈されますので変数の割り算や log は浮動小数点エラーの原因となります。</p> <pre>Variable x(index = i); 1 / x[i] >= 5; // 0 割りによるエラー sum(log(x[i]), i) == 1; // log(0) によるエラー</pre> <p>このような場合には変数に $x[i] = 1$; のようにして初期値を与えます。solve 関数を呼ぶ場合は、初期値は solve(); より前に与えてください。</p>
234	(SIMPLE 234) XX is inappropriate as element in domain of DiscreteVariable. (SIMPLE 234) XX は DiscreteVariable の値として不適切です。
	<p>DiscreteVariable とその domain に含まれない値の間の条件式を定義した場合に出力されます。</p> <p>例えば x の domain が $\{a, b, c\}$ のとき、</p> <pre>Boolean(x == "d") ..</pre> <p>のように書いた場合。</p>
235	(SIMPLE 235) Problem in domain of DiscreteVariable XX (should contain only positive integer or string). (SIMPLE 235) DiscreteVariable XX の domain が不適切です。
	DiscreteVariable の domain の値は非負の整数か文字列である必要があります。
236	(SIMPLE 236) DiscreteVariable XX 's domain has no element. (SIMPLE 236) DiscreteVariable XX の domain が空集合です。
237	(SIMPLE 237) = is inappropriately used.== instead of = should be used to define equality constraint. (SIMPLE 237) = が不適切に使われています。おそらく == の誤りです。(等式制約を定義するには = (代入) ではなく == を用います)。
	<pre>x + y = z;</pre> <p>のように等式制約の定義に=を誤って用いた場合に出力されます。</p>

エラー 番号	エラーメッセージ
	説明
238	(SIMPLE 238) = is inappropriately used in definition of Set. (SIMPLE 238) 集合演算において = が不適切に使われています。
239	(SIMPLE 239) DiscreteVariable XX has no "dom" argument. (SIMPLE 239) DiscreteVariable XX の dom 引数が設定されていません。
240	(SIMPLE 240) DiscreteVariable XX 's domain should not be indexed. (SIMPLE 240) DiscreteVariable XX の domain が添字付けられています。
241	(SIMPLE 241) Table/Parameter a is indexed by more than 3 DiscreteVariables. (SIMPLE 241) Table/Parameter a が 4 つ以上の DiscreteVariable で添字付けられています。
242	(SIMPLE 242) ResourceRequire "XX"'s argument "duration" have to be Set of integer (SIMPLE 242) ResourceRequire "XX" の引数 "duration" は整数の集合でなければなりません。
243	(SIMPLE 243) Constraint XX can't apply to rcpsp. (SIMPLE 243) 制約 XX は rcpsp では扱う事は出来ません。 rcpsp では扱えない制約 alldiff, valgrop 等が定義された場合に出力されます。
244	(SIMPLE 244) ResourceRequire is not defined for rcpsp. (SIMPLE 244) rcpsp を適用するのに必要な ResourceRequire が定義されていません。
245	(SIMPLE 245) ResourceCapacity is not defined for rcpsp. (SIMPLE 245) rcpsp を適用するのに必要な ResourceCapacity が定義されていません。
246	(SIMPLE 246) All ResourceCapacity's arguments "timeStep" are not same. (SIMPLE 246) 異なる ResourceCapacity の引数 "timeStep" に与えられている集合が同一ではありません。
247	(SIMPLE 247) resourceXXdefined at ResourceRequire is not defined at ResourceCapacity. (SIMPLE 247) ResourceRequire で定義されている資源 XX が ResourceCapacity に定義されていません。
248	(SIMPLE 248) Activity is not defined for rcpsp. (SIMPLE 248) rcpsp を適用するのに必要な Activity が定義されていません。

エラー 番号	エラーメッセージ
	説明
249	(SIMPLE 249) Immediate precedence use undefined resource "XX". (SIMPLE 249) 定義されていない資源 "XX" が直前先行制約で使われています.
250	(SIMPLE 250) mode "XX" used at Boolean is not defined. (SIMPLE 250) Boolean で指定しているモード "XX" は定義されていません.
251	(SIMPLE 251) Boolean times Boolean can't use for rcpsp (SIMPLE 251) rcpsp では一般の制約式において Boolean 同士の積は記述できません.
252	(SIMPLE 252) Activity is not different to startTime, endTime, processTime in Boolean (SIMPLE 252) 一般の制約式の Boolean と startTime, endTime, processTime の積の Activity が異なります
253	(SIMPLE 253) Activities are same in precedence (SIMPLE 253) 先行制約において同じ Activity に対し先行関係が与えられています.
255	(SIMPLE 255) can't use Activity for general constraints. (SIMPLE 255) 一般の制約式に Activity は用いられません. startTime, endTime, processTime に対して記述してください
258	(SIMPLE 258) modeXXnot defined at ResourceRequire set to Activity (SIMPLE 258) ResourceRequire で設定されていないモード XX が Activity に与えられています.
259	(SIMPLE 259) sourceActivity can use only defined Activity. (SIMPLE 259) sourceActivity は Activity を定義した時のみ使用出来ます.
260	(SIMPLE 260) sinkActivity can use only defined Activity. (SIMPLE 260) sinkActivity は Activity を定義した時のみ使用出来ます.
261	(SIMPLE 261) Activity has index "XX" but mode has index "YY". (should be the same) . (SIMPLE 261) Activity "XX" の index と引数 mode "YY" の index とが異なります.
262	(SIMPLE 262) Activity has index "XX" but mode is not defined. (SIMPLE 262) Activity "XX" の引数に mode が設定されていません.

エラー 番号	エラーメッセージ
	説明
263	(SIMPLE 263) Activity "XX"'s argument due date "YY"'s index is inappropriate . (SIMPLE 263) Activity "XX" の引数 due date "YY" の index が異なります。
264	(SIMPLE 264) two Activities' index are inappropriate in precedence. (SIMPLE 264) 先行制約の 2 つの Activity 間で index が異なります。
265	(SIMPLE 265) Can't set weight for precedence. convert to hard constraint (SIMPLE 265) 先行制約に重みを設定する事は出来ません。常に hard 制約として扱われます
	(警告)
266	(SIMPLE 266) Can't set weight for immediate precedence. convert to hard constraint (SIMPLE 266) 直前先行制約に重みを設定する事は出来ません。常に hard 制約として扱われます
	(警告)
267	(SIMPLE 267) can't use character for precedence's time (SIMPLE 267) 先行制約の時間指定に文字列が使われています
268	(SIMPLE 268) index error between imprecendene and precedence. (SIMPLE 268) 直前先行制約の index と指定された資源の index が異なります。
269	(SIMPLE 269) define resource for immediate precedence. (SIMPLE 269) 直前先行制約に資源が指定されていません
270	(SIMPLE 270) index error between precedence and precedece's time. (SIMPLE 270) 先行制約の index と時間指定の index が異なります
271	(SIMPLE 271) index error between precedence . (SIMPLE 271) 先行制約の添字に他の添字に依存するものがありますが他の添字と同時に使われていません。
272	(SIMPLE 272) defined multi resources for one immediate precedence (SIMPLE 272) 1 つの直前先行制約に複数の資源が指定されています
273	(SIMPLE 273) resource is not defined at ResourceCapacity "XX"'s argument. (SIMPLE 273) ResourceCapacity "XX" の引数に resource が与えられていません

エラー 番号	エラーメッセージ
	説明
274	(SIMPLE 274) timeStep is not defined at ResourceCapacity "XX"'s argument. (SIMPLE 274) ResourceCapacity "XX" の引数に timeStep が与えられていません
275	(SIMPLE 275) ResourceCapacity "XX" 's index is not same as argument weight "YY"'s index (SIMPLE 275) ResourceCapacity "XX" の index と 引数 weight "YY" の index が異なります
276	(SIMPLE 276) mode is not defined at ResourceRequire "XX"'s argument. (SIMPLE 276) ResourceRequire "XX" の引数に mode が与えられていません
277	(SIMPLE 277) resource is not defined at ResourceRequire "XX"'s argument. (SIMPLE 277) ResourceRequire "XX" の引数に resource が与えられていません
278	(SIMPLE 278) duration is not defined at ResourceRequire "XX"'s argument. (SIMPLE 278) ResourceRequire "XX" の引数に duration が与えられていません
279	(SIMPLE 279) duration is negative at ResourceRequire "XX"'s argument. (SIMPLE 279) ResourceRequire の引数 duration に負の値が用いられています
280	(SIMPLE 280) ResourceRequire has no data. (SIMPLE 280) ResourceRequire にデータが設定されていません
281	(SIMPLE 281) mode XX is not defined at ResourceRequire. (SIMPLE 281) モード XX が ResourceRequire に設定されていません (警告) 予期せぬ動作の原因になります.
282	(SIMPLE 282) mode XX defined at ResourceRequire is not used. (SIMPLE 282) モード XX が ResourceRequire に定義されていますが使われていません. (警告) 予期せぬ動作の原因になります.
283	(SIMPLE 283) resource XX 's ResourceCapacity value is 0 at through TimeStep. (SIMPLE 283) 資源 XX が全スケジュール期間において ResourceCapacity の値が 0 となっています. (警告) 初期解の失敗の原因になります.
284	(SIMPLE 284) can't use hard Constraint for rcpsp. (SIMPLE 284) rcpsp は一般の制約式を hard 制約として扱えません (十分大きな重みを与える必要があります).

エラー 番号	エラーメッセージ
	説明
285	(SIMPLE 285) general constraint has negative weight for rcpsp. (SIMPLE 285) 一般の制約式に負の重みが与えられています。
286	(SIMPLE 286) can't set weight for general constraint when use tardiness. (SIMPLE 286) 納期遅れ最小化では一般の制約式に重みを与える事は出来ません (全て hard 制約として扱われます).
	(警告)
288	(SIMPLE 288) Objective completionTime has negative weight. (SIMPLE 288) 目的関数 (最後の作業の完了時刻最小化) の重みに負の値が与えられています
289	(SIMPLE 289) can't treat as hardConstraint for Objective completionTime. (SIMPLE 289) 目的関数 (最後の作業の完了時刻最小化) は hard 制約として扱う事は出来ません
290	(SIMPLE 290) can't set weight to tardiness. (SIMPLE 290) 納期遅れ最小化に対して重みを設定する事は出来ません
	(警告)
291	(SIMPLE 291) ResourceCapacity's weight have to be positive value. (SIMPLE 291) 再生資源制約の重みは 0 より大きい値のみ与えられます (-1 は hard 制約とみなされます)
292	(SIMPLE 292) can't set weight to ResourceCapacity when use tardiness. (SIMPLE 292) 納期遅れ最小化時は資源制約に重みを設定する事は出来ません。 全て hard 制約として扱われます
	(警告)
293	(SIMPLE 293) use constraint which constructed by different Activitie's precedence or only Boolean when use tardiness. (SIMPLE 293) 納期遅れ最小化時の一般の制約式は異なる 2 つの Activity の先行関係か Boolean のみからなる制約以外の制約は扱えません
294	(SIMPLE 294) can't use equality constraint. use two in equality constraint. (SIMPLE 294) 納期遅れ最小化時は等式制約を扱う事は出来ません。 不等式制約を連立させます
	(警告)

エラー 番号	エラーメッセージ
	説明
295	(SIMPLE 295) can't define immediate precedence when use tardiness (SIMPLE 295) 納期遅れ最小化時は直前先行制約は定義出来ません
296	(SIMPLE 296) can't use constarint constructed by Boolean and Activity when use tardiness. (SIMPLE 296) 納期遅れ最小化時は一般の制約式で Boolean と Activity を混合する事は出来ません.
298	(SIMPLE 298) exist not indexing in rcpsp 's Object(Activity, ResourceRequire, ResourceCapacity). (SIMPLE 298) rcpsp のオブジェクト (Activity, ResourceRequire, ResourceCapacity) に添字が与えられていないものがあります.
299	(SIMPLE 299) resource XX defined at ResourceCapacity is not used at ResourceRequire. (SIMPLE 299) ResourceCapacity で定義されている資源 XX が ResourceRequire で使われていません.
	(警告)
300	(SIMPLE 300) ResourceRequire "XX"'s argument "timeStep" 's dim have to be one. (SIMPLE 300) ResourceRequire "XX" の引数 "timeStep" に与える集合は 1 次元でなければなりません.
301	(SIMPLE 301) ResourceRequire "XX"'s argument "timeStep" 's val have to be integer. (SIMPLE 301) ResourceRequire "XX" の引数 "timeStep" に与える集合の要素は整数でなければなりません
302	(SIMPLE 302) ResourceRequire "XX"'s argument "timeStep" 's val have to start 0. (SIMPLE 302) ResourceCapacity の引数 "timeStep" に与える集合の要素は 0 始まりでなければなりません
303	(SIMPLE 303) ResourceRequire "XX"'s argument "timeStep" 's val step is one. (SIMPLE 303) ResourceCapacity の引数 "timeStep" に与える集合の要素は 1 刻みでなければなりません.
304	(SIMPLE 304) ResourceCapacity "XX"'s val is real. (SIMPLE 304) ResourceCapacity "XX" の値が実数です. 整数に切り捨てます.
	(警告)

エラー 番号	エラーメッセージ
	説明
305	(SIMPLE 305) ResourceCapacity "XX"'s weights are real. (SIMPLE 305) ResourceCapacity "XX" に与えられた重みに実数のものがあります。整数に切り捨てます。
	(警告)
306	(SIMPLE 306) ResourceRequire "XX"'s weights are real. (SIMPLE 306) ResourceRequire "XX" に与えられた重みに実数のものがあります。整数に切り捨てます。
	(警告)
307	(SIMPLE 307) General Constraint "XX"'s weights are real. (SIMPLE 307) 一般の制約式 "XX" に与えられた重みに実数のものがあります。整数に切り捨てます。
	(警告)
308	(SIMPLE 308) CompletionTime's weights are real. (SIMPLE 308) 目的関数 (最後の作業の完了時刻最小化) に与えられた重みに実数のものがあります。整数に切り捨てます。
	(警告)
310	(SIMPLE 310) multi Objectives are defined for rcpsp. (SIMPLE 310) rcpsp において目的関数が複数定義されています。
311	(SIMPLE 311) Objective type is minimize for rcpsp. (SIMPLE 311) rcpsp において Objective に type=maximize が与えられています。 目的関数は最小化のみ扱えます。
312	(SIMPLE 312) warning: Objective function and general constraint are not defined. (SIMPLE 312) rcpsp において目的関数も一般の制約式も定義されていません。
	(警告)
313	(SIMPLE 313) Activity "XX" 's argument mode is empty. (SIMPLE 313) Activity "XX" の引数 mode に与えられた集合に空のものがあります。
314	(SIMPLE 314) can't define resource before condition in imprecendence constraint. (SIMPLE 314) 直前先行制約において条件式の前に資源を指定する事は出来ません
315	(SIMPLE 315) tardiness only can use when define Activity. (SIMPLE 315) tardiness は Activity を定義した時のみ使用出来ます。

エラー 番号	エラーメッセージ
	説明
316	(SIMPLE 316) ResourceRequire "XX" 's argument default is negative. (SIMPLE 316) ResourceRequire "XX" の引数 default に負の値が与えられています
317	(SIMPLE 317) only completionTime and tardiness are set to Objective for rcpsp. (SIMPLE 317) completionTime, tardiness 以外は Objective に設定する事は出来ません。 rcpsp を用いる場合のみ
318	(SIMPLE 318) ResourceRequirie "XX"'s val have to be non-negative (SIMPLE 318) ResourceRequirie "XX" に負の値が与えられています。
319	(SIMPLE 319) ResourceCapacity "XX"'s val have to be non-negative (SIMPLE 319) ResourceCapacity "XX" に負の値が与えられています。
320	(SIMPLE 320) ResourceCapacity "XX"'s weight have to be non-negative. (SIMPLE 320) ResourceCapacity "XX" の重みに負の値が与えられています。 ただし, -1 は hard な資源制約の意味になります。
321	(SIMPLE 321) ResourceCapacity "XX"'s mode "YY" is uninitialized. (SIMPLE 321) ResourceCapacity "XX" のモード YY の処理時間が明確ではありません。(初期化してください)
322	(SIMPLE 322) Tardiness is set to Objective but Activity isn't set duedate. tardiness is canceled. (SIMPLE 322) 目的関数に tardiness が設定されていますが, Activity に duedate が与えられていません。納期遅れ最小化は行われません。 (警告)
323	(SIMPLE 323) (immediate)precedence contradicts. (SIMPLE 323) (直前) 先行制約の先行関係に矛盾があります。
324	(SIMPLE 324) multiple resource are given to immediate precedence. (SIMPLE 324) 直前先行制約に複数の資源が与えられています。
325	(SIMPLE 325) cannot use sourceActivity when use tardiness. (SIMPLE 325) 納期遅れ最小化時には sourceActivity は用いる事は出来ません。
326	(SIMPLE 326) cannot use sinkActivity when use tardiness. (SIMPLE 326) 納期遅れ最小化時には sinkActivity は用いる事は出来ません。

エラー 番号	エラーメッセージ
	説明
327	(SIMPLE 327) cannot use DummyMode when use tardiness. (SIMPLE 327) 納期遅れ最小化時には DummyMode は用いる事は出来ません.
328	(SIMPLE 328) Activity times Activity can't use for rcpsp (SIMPLE 328) rcpsp では一般の制約式において Activity 同士の積は記述できません.
329	(SIMPLE 329) Activity times Activity.startTime can't use for rcpsp (SIMPLE 329) rcpsp では一般の制約式において Activity と Activity.startTime の積は記述できません.
330	(SIMPLE 330) Activity.startTime times Activity.endTime can't use for rcpsp (SIMPLE 330) rcpsp では一般の制約式において Activity.startTime と Activity.endTime の積は記述できません.
331	(SIMPLE 331) Activity.startTime times Activity.startTime can't use for rcpsp (SIMPLE 331) rcpsp では一般の制約式において Activity.startTime と Activity.startTime の積は記述できません.
332	(SIMPLE 332) Activity.endTime times Activity.endTime can't use for rcpsp (SIMPLE 332) rcpsp では一般の制約式において Activity.endTime と Activity.endTime の積は記述できません.
333	(SIMPLE 333) Activity times Activity.endTime can't use for rcpsp (SIMPLE 333) rcpsp では一般の制約式において Activity と Activity.endTime の積は記述できません.
334	(SIMPLE 334) completionTime only can use when define Activity. (SIMPLE 334) completionTime は Activity を定義した時のみ使用できます.
335	(SIMPLE 335) undefined Value XX at ResourceRequire is given to Activity. (SIMPLE 335) ResourceRequire で定義されていない XX が Activity の初期値に与えられました

エラー 番号	エラーメッセージ
	説明
336	(SIMPLE 336) undefined Value XX at ResourceRequire is given to Activity.
	(SIMPLE 336) ResourceRequire で定義されていない XX が Activity の初期値に与えられました
337	(SIMPLE 337) can't fix mode for undefined Activity "XX"
	(SIMPLE 337) 初期化されていない "XX" に対してモードの固定を行なおうとしました.
338	(SIMPLE 338) can't give weight to fixActivity when set due date
	(SIMPLE 338) 納期遅れ最小化時は fixActivity に重みを与える事は出来ません
339	(SIMPLE 339) can't fix endTime when give weight to fixActivity
	(SIMPLE 339) fixActivity に重みを与えた場合は終了時刻の固定は出来ません
340	(SIMPLE 340) Mode YY written in Boolean is not in the Activity XX's domain.
	(SIMPLE 340) Boolean で指定された XX はモード YY を取る事が出来ません
341	(SIMPLE 341) can't give to modeOrder. convert to hardConstraint.
	(SIMPLE 341) modeOrder 制約に重みを設定する事は出来ません. 常に hard 制約として扱われます (警告)
343	(SIMPLE 343) two Activities' index are inappropriate in modeOrder.
	(SIMPLE 343) modeOrder 制約の 2 つの Activity 間で index が異なります.
344	(SIMPLE 344) index error between modeOrder .
	(SIMPLE 344) modeOrder 制約の添字に他の添字に依存するものがありますが他の添字と同時に使われていません.
345	(SIMPLE 345) can't define modeOrder when use tardiness
	(SIMPLE 345) 納期遅れ最小化時は modeOrder 制約を定義出来ません
346	(SIMPLE 346) two Activitie's mode num given to moderOrder is different
	(SIMPLE 346) modeOrder 制約に与えられた 2 つの Activity XX, YY の取り得るモードの数が異なります modeOrder 制約に与えられた Activity の取りうるモードの数は等しい必要があります.

エラー 番号	エラーメッセージ
	説明
347	(SIMPLE 347) can't give same Activity to moderOrder (SIMPLE 347) 同一の Activity を modeOrder 制約に与える事は出来ません.
348	(SIMPLE 348) can't fix XX's endTime zero (SIMPLE 348) 終了時刻を 0 に固定する事は出来ません (XX).
349	(SIMPLE 349) can't fix endTime negative value (XX). (SIMPLE 349) 開始時刻を負の値で固定する事は出来ません (XX).
350	(SIMPLE 350) can't fix endTime negative value (XX). (SIMPLE 350) 終了時刻を負の値で固定する事は出来ません (XX).
351	(SIMPLE 351) can't fix both startTime and endTime (XX). (SIMPLE 351) 開始時刻と終了時刻の両方を固定する事は出来ません (XX).
352	(SIMPLE 352) can't fix startTime over timeStep (XX). (SIMPLE 352) timeStep を越える時刻の固定は行なう事が出来ません (XX).
353	(SIMPLE 353) Activity list is not initialized. (SIMPLE 353) 作業リストが与えられていない Activity があります. 自動で初期値設定は行われません.
	(警告)
354	(SIMPLE 354) initialized Activity XX can't be YY (SIMPLE 354) 初期値に与えられた YY を XX は取り得ません
355	(SIMPLE 355) can't give negative weight for fixActivity (SIMPLE 355) fixActivity に負の重みが与えられています
357	(SIMPLE 357) XX's time is fixed to a real number. omit to integer number. (SIMPLE 357) XX の時刻が実数の値で固定されています. 整数に丸められます.
	(警告)
358	(SIMPLE 358) can't set initial value to "XX". (SIMPLE 358) XX に初期値を代入する事は出来ません.

エラー 番号	エラーメッセージ
	説明
359	(SIMPLE 359) Inappropriate order's are given to initial value. give order's which start from "1" to all Activity. (SIMPLE 359) 初期値に与えられた順番が不正です (順番は 1 始まりで全ての Activity に対し与えて下さい).
361	(SIMPLE 361) can't set defaultval for ResourceCapacity "XX". (SIMPLE 361) ResourceCapacity "XX"に defaultval を設定する事は出来ません.
	defaultval は ResourceRequire クラスにのみ有効です.
362	(SIMPLE 362) can't decide timeStep.given empty set to timeStep (SIMPLE 362) スケジュール期間が定まりません. timeStep に与えられた集合が空集合です.
363	(SIMPLE 363) There exist some ResourceRequire whose value of mode is unset. (SIMPLE 363) 全てのモードに対して設定されていない ResourceRequire が存在します.
370	(SIMPLE 370) The index dim of the Matrix "XX" should be 2. (SIMPLE 370) 行列 "XX" の宣言に与えた要素の添字が 2 次元ではありません.
371	(SIMPLE 371) Inappropriate index dim of the Matrix "XX" or its Element. Now its index is dim YY, but should be ZZ (as Matrix) or MM (as Element). (SIMPLE 371) 行列 "XX" (添字の次元 ZZ) に, 次元 YY の添字は不適合です. 行列全体を指すのなら, 添字の次元は ZZ に, 要素を指すのなら, 添字の次元は MM でなければなりません.
372	(SIMPLE 372) The Matrix "XX" is null. The SDP constraint on this matrix is ignored. (SIMPLE 372) 行列 "XX" は空です. この行列に対する正定値制約は無視されます.
	(警告)
373	(SIMPLE 373) Weight coefficients of the constraint "a" is inappropriate. (a: Quadratic b: Linear) = (XX YY). a and b both should be non-negative. (SIMPLE 373) 制約式 "a" のウエイト係数が不正です. (a 二次 b 一次) = (XX YY). a,b ともに零または正でなければなりません.

エラー 番号	エラーメッセージ
	説明
375	(SIMPLE 375) count/max/min/argmax/argmin contains DiscreteVariable. (SIMPLE 375) count/max/min/argmax/argmin が DiscreteVariable を含んでいます.
	count/max/min/argmax/argmin が DiscreteVariable を含む表現を制約式/目的関数の定義に使うことはできません. 0-1 の IntegerVariable (type = binary) のみ可能です.
376	(SIMPLE 376) 1st argument of count is inappropriate type. (SIMPLE 376) count の最初の引数が, [定数 <=] 式 [<= 定数] あるいは [定数 >=] 式 [>= 定数] 以外の形をしています.
	count の最初の引数の式の形が想定しているものと異なります. 定数 <= 式 <= 定数 定数 >= 式 >= 定数 のみが可能です (上, 下限に相当する定数のいずれか一つは省略可).
377	(SIMPLE 377) Format string of simple_[f]printf running out. (SIMPLE 377) simple_[f]printf の書式指定に対して引数が足りなくなりました.
	simple_[f]printf の引数に対してフォーマット文字列が足りません. このメッセージにひきつづいてフォーマット文字列と足りなくなった箇所が表示されます.
378	(SIMPLE 378) Uninitialized object passed to simple_[f]printf. (SIMPLE 378) simple_[f]printf に初期化されていないオブジェクトが渡されました.
	宣言前のオブジェクト, あるいは実行されない if 文の中で宣言されたオブジェクトが simple_[f]print の引数として渡されました.
379	(SIMPLE 379) Try to convert uninitialized object to Parameter. (SIMPLE 379) 初期化されていないオブジェクトを Parameter に変換しようとしてしました.
	宣言前のオブジェクト, あるいは, 実行されない if 文の中で宣言されたオブジェクトが, 式の中で定数値として使われました.
380	(SIMPLE 380) Try to display uninitialized object. (SIMPLE 380) 初期化されていないオブジェクトを表示しようとしてしました.
	宣言前のオブジェクト, あるいは, 実行されない if 文の中で宣言されたオブジェクトを .val.print() などで表示しようとしてしました.
381	(SIMPLE 381) The specified dimension in the .at function is out of range. (SIMPLE 381) at 関数で指定された次元は範囲外です.

エラー 番号	エラーメッセージ
	説明
382	(SIMPLE 382) simple_[f]printf with no indexed argument for output is done ignoring condition. Use "if" to condition.
	(SIMPLE 382) simple_[f]printf にある条件式は添字付きの出力引数がないので無視されます。条件付けするには if 文を使ってください。
	(警告) 添字のない Parameter a を用いて simple_printf("1\n", a > 0); のように simple_[f]printf の条件式を記述した場合条件式は無視されます。if(a > 0){simple_printf("1\n");}のように記述してください。
400	(SIMPLE 400) the dimension of reference object XX and reference object YY are not suitable.
	(SIMPLE 400) 参照オブジェクト XX と参照オブジェクト YY の次元が整合しません。
401	(SIMPLE 401) reference object XX cannot be converted into vector.
	(SIMPLE 401) 参照オブジェクト XX はベクトルに変換できません。
402	(SIMPLE 402) reference object XX cannot be converted into scalar.
	(SIMPLE 402) 参照オブジェクト XX はスカラーに変換できません。
403	(SIMPLE 403) because the dimension of reference object XX and reference object YY are not suitable, inner product is not computable.
	(SIMPLE 403) 参照オブジェクト XX と参照オブジェクト YY の次元が整合しないため、内積は計算できません。
404	(SIMPLE 404) because the dimension of reference object XX and reference object YY are not suitable, addition and subtraction are not computable.
	(SIMPLE 404) 参照オブジェクト XX と参照オブジェクト YY の次元が整合しないため、加減演算はできません。
405	(SIMPLE 405) because the dimension of reference object XX and reference object YY are not suitable, multiplication is not computable.
	(SIMPLE 405) 参照オブジェクト XX と参照オブジェクト YY の次元が整合しないため、乗算は計算できません。
406	(SIMPLE 406) reference object XX and scalar YY are not compareable.
	(SIMPLE 406) 参照オブジェクト XX とスカラー YY は比較できません。

エラー 番号	エラーメッセージ
	説明
407	(SIMPLE 407) because reference object XX is not square matrix, the trace of it is not computable. (SIMPLE 407) 正方行列でないため、参照オブジェクト XX のトレースは計算できません。
410	(SIMPLE 410) function sum of reference object XX is not computable. (SIMPLE 410) 参照オブジェクト XX の sum は計算できません。
411	(SIMPLE 411) the multiplication of reference object XX and indexed parameter is not computable. (SIMPLE 411) 参照オブジェクト XX と添字付き Parameter の乗算は計算できません。
413	(SIMPLE 413) substitution for indexed reference object XX is prohibited. (SIMPLE 413) 添字付きの参照オブジェクト XX に対する代入は禁止されています。
449	(SIMPLE 449) error occurred by matrix or vector "XX" arithmetic operation. Please try to write by using Expression. (SIMPLE 449) 行列またはベクトル "XX" の演算で問題が発生しました。Expression を用いて記述してください。
450	(SIMPLE 450) Try to operate "YY" on empty set "XX" (SIMPLE 450) 空の集合 "XX" に対して 操作 "YY" を行いました。 (警告)
451	(SIMPLE 451) Try to evaluate null-element. (SIMPLE 451) 空の集合の要素の値が式の評価の際に参照されました。
452	(SIMPLE 452) Element "a" binded to Set, so cannot be fixed (new feature since V20). (SIMPLE 452) 要素 "a" は定義集合を持っているので固定できません (V20 からの新しい仕様)。
453	(SIMPLE 453) Element "a" passed to OrderedSet::position unfixed. (SIMPLE 453) 要素 "a" が OrderedSet::position に固定されないで渡されました。
454	(SIMPLE 454) Null valued element passed to OrderedSet::position. (SIMPLE 454) 空の要素が OrderedSet::position に渡されました。
455	(SIMPLE 455) The specified index in the .elementAt function is out of range. (SIMPLE 455) elementAt 関数で指定された要素番号は範囲外です。

エラー 番号	エラーメッセージ
	説明
456	(SIMPLE 456) Argument: "dim" 's val have to be an integer greater than or equal to 1. (SIMPLE 456) 引数 "dim" の値は 1 以上でなければなりません.
502	(SIMPLE 502) Inappropriate class type in "[" function. (SIMPLE 502) 関数 "[" の引数のタイプが正しくありません.
511	(SIMPLE 511) Internal problem in SIMPLE. (SIMPLE 511) システム内部の問題が起きました.
	SIMPLE の内部エラーが起きました. (nuopt-support@ml.msi.co.jp へお知らせください.)
514	(SIMPLE 514) No such file: "XX". (SIMPLE 514) 次のファイルは存在しません. "XX".
515	(SIMPLE 515) Must be Minimize/Maximize Objective. (SIMPLE 515) ここは Minimize/Maximize (目的関数) でなければなりません.
516	(SIMPLE 516) No Minimize/Maximize Objective exists. (SIMPLE 516) Minimize/Maximize (目的関数) が存在しません.
517	(SIMPLE 517) Only Expanded Variable's current value can be changed. (SIMPLE 517) 変数の current 値を展開前に求めようとしてしました.
520	(SIMPLE 520) Max operation was applied to empty set. (SIMPLE 520) max 操作が空集合に対して適用されました.
	(警告) max 関数を使用した際, 範囲指定並びが空集合のため最大値の取得ができない場合に表示されます.
521	(SIMPLE 521) Min operation was applied to empty set. (SIMPLE 521) min 操作が空集合に対して適用されました.
	(警告) min 関数を使用した際, 範囲指定並びが空集合のため最小値の取得ができない場合に表示されます.
522	(SIMPLE 522) Empty selection appeared. (SIMPLE 522) 空の selection が現れました.
	(警告) selection 関数を使用した際, 選択候補となる 0-1 整数変数が存在しない場合に表示されます.

エラー 番号	エラーメッセージ
	説明
523	(SIMPLE 523) Empty alldiff appeared. (SIMPLE 523) 空の alldiff が現れました.
	(警告) alldiff 関数を使用した際, 制約を与える離散変数が存在しない場合に表示されます.
524	(SIMPLE 524) The argument of selection should be indexed Variable. (SIMPLE 524) selection の引数となる変数は添字付きであることが必要です.
525	(SIMPLE 525) The argument of alldiff should be indexed Variable. (SIMPLE 525) alldiff の引数となる変数は添字付きであることが必要です.
550	(SIMPLE 550) Elements of a empty Set are expanded. (SIMPLE 550) 空集合に対して添字の展開をしようとしてしました.
	(警告)
551	(SIMPLE 551) Inappropriate semidefinite constraint on "XX". RHS must not have movable index. (SIMPLE 551) 行列 "XX" についての半正定値制約の右辺が一意に定まりません.
	半正定値制約の右辺に a[i] など, 動ける index を伴ったパラメータが表れると出るエラーです. a[4] などの場合には出ません.
552	(SIMPLE 552) SymmetricMatrix "XX" has index "YY" out of its dimSet. (SIMPLE 552) 対称行列 "XX" のインデクス "YY" が dim で与えられた範囲の外です.
	半正定値制約の定義の添字が dim で与えられた範囲をはみだしています. 行列要素の代入については自動代入が適用されません.
553	(SIMPLE 553) In a recurrence relation, evaluation of "XX" is circular. (SIMPLE 553) 漸化式において, オブジェクト "XX" の評価が循環しました.
	漸化式で定義されたオブジェクトの値評価の途中で定義漸化式が循環を含んでいることが判明しました. 漸化式として不整合です.
554	(SIMPLE 554) Inappropriate evaluation in a recurrence relation. (SIMPLE 554) 漸化式定義中に正しくないオブジェクトの評価が行われました.
	漸化式によるオブジェクトの値の定義途中で, そのオブジェクト自身の評価が要請されました. 漸化式で定義されるオブジェクトの値評価は, 漸化式定義最中には行えません.
555	(SIMPLE 555) In a recurrence relation, to use "setOf" is forbidden. (SIMPLE 555) 漸化式定義中に "setOf" を使うことはできません.
	漸化式によるオブジェクトの値の定義途中で setOf 関数が使用されました. 漸化式定義中では setOf 関数を使用できません.

エラー 番号	エラーメッセージ
	説明
556	(SIMPLE 556) In a recurrence relation, to define constraints or an objective is forbidden. (SIMPLE 556) 漸化式定義中に制約式や目的関数の定義はできません。
	漸化式によるオブジェクトの値の定義途中で、制約式や目的関数の定義がなされました。 漸化式定義中では制約関数や目的関数の定義はできません。
557	(SIMPLE 557) No data in dat file. (SIMPLE 557) 与えられた dat ファイルに有効なデータがありません。
	(警告)
558	(SIMPLE 558) It is not possible to specify the "type=maximize" in Minimize. (SIMPLE 558) Minimize では "type=maximize" を指定することはできません。
559	(SIMPLE 559) It is not possible to specify the "type=minimize" in Maximize. (SIMPLE 559) Maximize では "type=minimize" を指定することはできません。
560	(SIMPLE 560) The weight of the constraint XX is overwritten. (SIMPLE 560) 制約式 XX の重みが上書きされました。
	制約式オブジェクトには重みを一つのみ設定できます。
562	(SIMPLE 562) The addition of Set exceeded the limit. (SIMPLE 562) 集合への追加が上限を超えました。
	集合に対する追加 (add 操作) の回数が上限を超えました。

A.2 mknuopt のエラー/警告メッセージ

次の表はモデルファイルのビルド時のエラー/警告メッセージ一覧です。

エラー 番号	エラーメッセージ
	説明
1	(MKNUOPT 1) モデルファイルが空です。
	与えられたモデルファイルにモデル記述がありません。
2	(MKNUOPT 2) 文法エラーがありました。
	与えられたモデルファイルに文法エラーがありました。
3	(MKNUOPT 3) ～ の宣言で ～ が重複しています。
	SIMPLE オブジェクトの宣言時に name などが 2 回以上指定されました。
4	(MKNUOPT 4) 予期しないトークン "～" です。
	文法上正しくないトークンが現れました。
5	(MKNUOPT 5) ～ に対応する閉括弧がありません。
	開括弧に対応する閉括弧がありません。

エラー 番号	エラーメッセージ
	説明
6	(MKNUOPT 6) モデルファイルに利用できない文字を含んでいます.
	モデルファイル名に利用できない文字を含んでいます.
7	(MKNUOPT 7) ～ と同じ名前の SIMPLE 型変数が複数回宣言されています.
	同じ名前の SIMPLE オブジェクトが複数回宣言されました.
8	(MKNUOPT 8) Variable ～ の宣言で type=binary となっています.
	連続変数 Variable の宣言で type = binary と指定されました.
9	(MKNUOPT 9) ～ と同じ name の SIMPLE 型変数が複数回宣言されています.
	同じ name の SIMPLE オブジェクトが複数宣言されました.
10	(MKNUOPT 10) ～ の宣言で name に不正な文字が含まれています :
	SIMPLE オブジェクトの name に不正な文字が含まれています. name には, 「¥」「"」「,」を使うことはできません.
11	(MKNUOPT 11) ～ の宣言で ～ が不定です.
	SIMPLE オブジェクトの宣言時に name などが = 指定されずに現れました.
12	(MKNUOPT 12) SIMPLE 型のポインタを宣言しています.
	SIMPLE 型オブジェクトのポインタを宣言しています.
13	(MKNUOPT 13) ～ の宣言で name がリテラルではありません.
	SIMPLE オブジェクトの宣言時に指定された name が文字列リテラルではありません.

索引

記号・数字

<=	5, 21
>=	21
.csv	87
.dat	87
==	21
0-1 変数	38
1D 書式	93
2D 書式	93

A

Activity	69, 71
add	50
alldiff	64
asChar	38
asDouble	37
asqp	58

B

binary	38, 63
Boolean	66

C

card	49
Constraint	33, 63
count	68
csv 形式	87, 90, 93

D

dat	8
dat 形式	87

defaultConstraintWeight	63, 71
defaultObjectiveTarget	60
defaultObjectiveWeight	60, 71
defaultval	75
duration	74

E

Element	45
elementAt	52
endTime	77
erf	58
Expression	18, 44

F

FAQ	125
first	52, 53
fixActivity	77

H

hardConstraint	61
----------------	----

I

index	11, 12, 16, 18
IntegerVariable	19, 38
isFeasible	100

L

last	52, 53
lock	51
lpout	123

M

max 57, 66
 Maximize 33
 maximize 33
 min 57, 66
 Minimize 33
 minimize 5, 21, 33
 mode 71, 76
 modeOrder 76
 mpsout 123
 mpsout_e 123

N

name 5, 7, 8, 10, 18, 21, 26, 88
 next 52, 53

O

Objective 32, 60
 OrderedSet 52

P

Parameter 8, 35, 37, 38
 position 52
 pow 58
 prev 52, 53
 print 11, 19, 20, 105, 107
 prod 39

R

rcpsp 69
 resource 74, 75
 ResourceCapacity 69, 75
 ResourceRequire 69, 74

S

selection 65
 semiHardConstraint 61
 Sequence 54

Set 46
 setDualMatrix 35
 setOf 49, 55
 showSystem 20, 21, 118
 simple_fprintf 105, 116
 simple_printf 105, 110
 SimpleSetInitialValues 103
 slice 44, 51
 softConstraint 62, 63
 solve 6, 20, 97
 startTime 77
 sum 15, 16, 39
 SymmetricMatrix 40, 115

T

target 60
 timeStep 75
 type 5, 21, 33, 38, 63

U

unfixActivity 78
 unlock 51

V

Variable 32, 64
 VariableParameter 102

W

wcsp 59, 63, 64, 66
 weight 76

あ

アクティビティ 71
 アクティビティ固定解除関数 78
 アクティビティ固定関数 77

え

演算子 5, 21

か

解ファイル	33
ガウスの誤差関数	58
カウント	68
拡張子	87
可変定数	102
完了時刻	69

き

期間集合	76
------------	----

け

経過時間集合	74
--------------	----

こ

コメント文	88, 91
-------------	--------

さ

最後の作業の完了時刻	70
最後の作業の完了時刻最小化	70
最小（大）値取得関数	57, 66
最小固有値	34, 42
最適解	5, 6

し

資源供給量	75
資源集合	69
資源制約付きスケジューリング問題	69, 78
自動代入機能	48, 90, 94
自動展開機能	46
自動補間機能	47
集合	9-13, 16, 20, 46
重複不能関数	64
出力関数	19
上下限制約	5
条件式	53, 55, 72
人員スケジューリング問題	78

す

数学関数	5, 58
数理最適化問題	18
数列集合	54

せ

整数計画問題	19
整数変数	19, 38
制約式	5, 11, 13, 15, 17, 20, 21, 33, 70, 144, 145
制約充足問題ソルバ	59
制約条件	4, 9, 14, 15, 17
セミハード制約	61
全角空白文字	25
漸化式	52
漸化不等式	46, 52
先行制約	71, 72
選択関数	65, 71

そ

双対行列	34
双対変数	33
添字	10-13, 15, 16, 18, 20, 21, 45
疎形式	42
ソフト制約	61, 62

た

対称行列	40, 107, 110, 115
------------	-------------------

ち

直前先行制約	71, 73
--------------	--------

て

定数	7-13, 15, 16, 20, 21, 35
データファイル	8, 12, 13, 16, 87, 93, 145

と

等式制約	33
------------	----

の

納期遅れ	69, 70
納期遅れ最小化	69-71, 78

は

ハード制約	61
範囲演算関数	39
半角空白文字	25
半角セミコロンの	25, 87, 89
半正定値制約	42

ひ

標準出力	105
------------	-----

ふ

不一致制約	34
-------------	----

ブール関数	66
不等式制約	33
浮動小数点エラー	125
部分集合	48, 55

へ

変数	3-5, 7-14, 16, 18-20, 32, 63
----------	------------------------------

も

モード	69, 71, 76
目的関数 ...	3, 5, 7-10, 13, 14, 16, 17, 20, 21, 32, 60
モデル	4, 6-9, 12, 20, 21
モデルファイル	93

り

離散変数	64
------------	----