NUOPT/SIMPLE マニュアル

株式会社 数理システム
Phone: 03-3358-1701
Fax: 03-3358-1727
Email: nuopt-support@msi.co.jp

2012/11/9

目次

Introduction

本マニュアルは最適化パッケージソフト NUOPT の利用経験者に対して,より強力な NUOPT の機能を説明するためのものです.

NUOPT には用途に応じて以下のマニュアルが準備されております.

（ア） NUOPT/SIMPLE マニュアル

（イ） NUOPT/SIMPLE チュートリアル

（ウ） NUOPT/SIMPLE Excel 連係マニュアル

（エ） NUOPT/SIMPLE 外部接続マニュアル

（オ） NUOPT/SIMPLE 例題集

（カ） NUOPT/DFO 利用ガイド

本マニュアル（ア）は， NUOPT の機能を詳細に説明するためのものです．読者にはある程度 NUOPT に馴染みがあることを想定しております.

初めて NUOPT をご利用の方は（イ） NUOPT/SIMPLE チュートリアルをご覧下さい.

NUOPT を Microsoft Excel との連係機能をご利用の方は（ウ） NUOPT/SIMPLE Excel 連係マニュアルをご覧下さい.

外部のプログラムから NUOPT を呼び出して利用されたい方は（エ） NUOPT/SIMPLE 外部接続マニュアルをご覧下さい.

典型的な数理計画問題に対する NUOPT/SIMPLE の記述例を知りたい方は（オ） NUOPT/SIMPLE 例題集をご参照下さい.

アドオンの NUOPT/DFO をご利用の方は（カ）NUOPT/DFO 利用ガイドをご覧下さい.

本マニュアル（ア）は NUOPT を実際に応用するにあたり必要十分な内容を含んでおりますが，NUOPT の機能全てを記載するものではありません．より細部の機能に関するご要望は nuopt-support@msi.co.jp までお寄せ下さい.

本マニュアルは以下のような内容から構成されています.

1 NUOPT の基本事項の解説

2 ～ 9 モデリング言語 SIMPLE の解説

10 ～ 15 求解ソルバ NUOPT の解説

16 ～ 17 特殊な数理計画問題と FAQ

付録 エラーメッセージ・アルゴリズムの解説，参考文献の紹介

1． NUOPT の基本事項

NUOPT を扱う上で必ず知っておくべき事項を，以下に列挙します.

1.1 NUOPT の構成

最適化パッケージソフト NUOPT は次の二つから構成されています.

- SIMPLE （数理計画問題を記述するためのモデリング言語）
- NUOPT （数理計画問題を解くための求解ソルバ）

本マニュアルで NUOPT と呼ぶ場合，上記二つをまとめたソフト全体を指すケースと，後者の求解ソルバのみを指すケースがありますのでご注意下さい.

1.2 NUOPT の利用法

NUOPT を利用するには， GUI を用いる方法とコマンドラインから起動させる方法との二種類があります． windows 版ではいずれの方法も提供されていますが， UNIX/Linux 版には GUI は無く，コマンドラインから起動させる方法のみが提供されています.

1.3 NUOPT の処理の流れ

1. SIMPLE で数理計画問題（モデル）を記述
2. 1 で記述されたモデルを実行形式に変換する（コンパイル）
3. 実行形式を（必要ならばデータファイルを引数として与えて）起動させる

GUI では上記手順の 2,3 の処理をダブルクリックでまとめて行いますが，コマンドラインでは 2,3 の処理を個別に行う必要があります.

windows では mknuopt.bat，UNIX/Linux では mknuopt と指定します.また windows では実行形式は [モデルファイル名].exe となりますが UNIX/Linux では実行形式は [モデルファイル名]となります．また，実行形式を起動させる際に UNIX/Linux では ./[モデルファイル名] と指定します．なお，モデルファイル名は全角空白・半角空白を含めることができません.

以下は windows 版での手順です.

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

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

以下は UNIX/Linux 版における手順です.

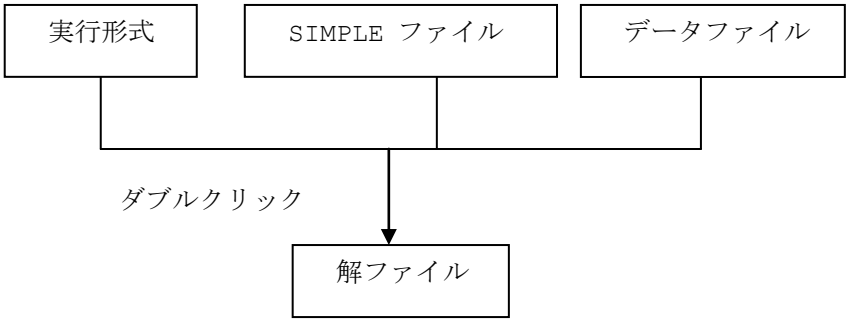
```
prompt% mknuopt [モデルファイル名].cc
```

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

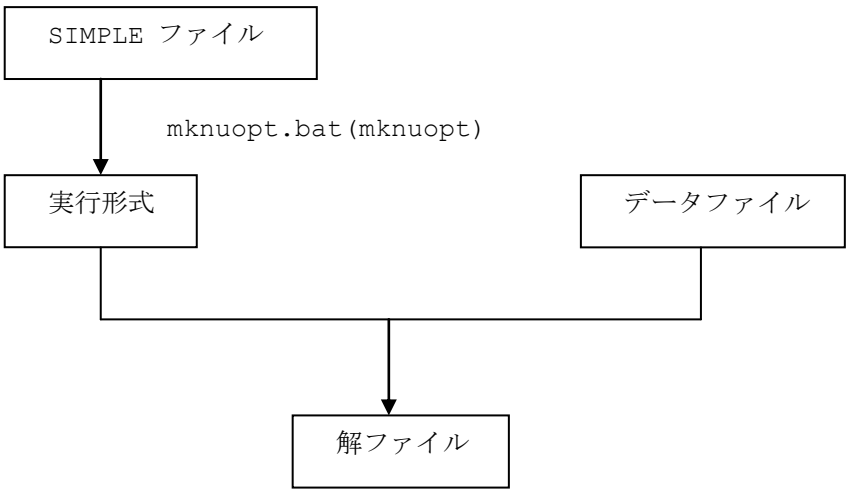
同じ数理計画問題に対してデータだけ異なる問題を解く場合には，再度 2 の手順を踏む必要は無く 3 だけの手順で事足ります.

上記の流れを図示すると，以下のようになります.

GUI の場合の処理の流れ



コマンドラインの場合の処理の流れ



1.3.1 分枝限定法の並列化

NUOPT windows 版において並列処理を伴う分枝限定法を実行することができます.

以下はその実行形式の作成及び使用する手順です.

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

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

「3．実行形式を（必要ならばデータファイルを引数として与えて）起動させる」については通常の Windows 版における手順と同様ですが「１で記述されたモデルを実行形式に変換する（コンパイル)」の手順において「-parallel」というオプションを付ける部分が異なります. 分枝限定法の並列化において必要なパラメータについては「14.3.4 整数計画法(simplex/asqp)/ 大域的最適化（ global ）に有効なパラメータ」を参考にしてください.

分枝限定法の並列化では Intel 社の並列化ライブラリ Intel® TBB を用いています．このため実行にあたっては，以下の動作環境が必要になります.

- NUOPT のインストール時に選択するコンパイラ
  - Microsoft Visual Studio 2005（ただし 「KB2538242・Microsoft Visual C++ 2005 Service Pack 1 再頒布可能パッケージ MFC のセキュリティ更新プログラム」を適用している必要があります)
  - Microsoft Visual Studio 2008
  - Microsoft Visual Studio 2010
- ハードウェア
  - インテル® Core(TM) 2 Duo プロセッサーまたはインテル® Xeon® プロセッサー以上
  - インテル® Pentium(R) 4 プロセッサー・ファミリー以上

2. モデリング言語 SIMPLE とは

SIMPLE はシステムの記述をなるべく人間に馴染みのある数学的な記述方法で簡単に行ない，実際のシミュレータやソルバなどが認識できるような表現に翻訳して所要の解析を行うことを目的とします.

本マニュアルでは SIMPLE で数理計画問題を記述したファイルを**モデルファイル**と呼びます.

SIMPLE はプログラミング言語 C++ を用いて実装されています. SIMPLE を用いるに際して C++ の知識を特別必要とすることはありませんが，一部 C++ を理解していないと利用が難しい機能もあります.

以下の説明は， C++ に関する知識をお持ちの方向けです.

SIMPLE の実体は C++ のクラスライブラリです. 式を特定のパーザプログラムによって解釈するのではなく, 記述をコンパイル, 実行して, ユーザのモデル記述内容の情報を取得します. その際に利用する仕組みは C++ の演算オーバーロード機能です. 具体的には SIMPLE 特有のクラスのオブジェクト間の演算を各クラス間の演算に対応して定義し，対応する特定の実装を適宜コールさせることによって，式の解釈を行います.

SIMPLE の式の記述の規則は，クラスオブジェクトの間の演算規則として記述され，コンパイラによって厳格にチェックを受けます. そのために, 定義された範囲外の演算を記述すると，まずコンパイル時にエラーとなります.

3． SIMPLE の基本事項

3.1 一般事項

3.1.1 行末のセミコロン;

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 ;
```

3.1.2 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;
```

3.1.3 構成要素の順序

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，誤りです．

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

3.1.4 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）も使用することはできません．

3.1.5 name 引数に利用できない文字

GUI 版で NUOPT を起動する場合， name 引数で利用できない名前があります．具体的には，半角文字「¥/:*?"<>|」が利用できません．

コマンドラインから NUOPT を起動する場合は，この限りではありません．

3.2 windows 版と UNIX/Linux 版での差異

SIMPLE の規則の中で windows 版, UNIX/Linux 版によって異なる部分を解説します.

3.2.1 モデルファイルの拡張子

モデルファイルの拡張子は, windows 版では .smp, UNIX/Linux 版では .cc です.

3.2.2 UNIX/Linux 版でのモデル記述

UNIX/Linux 版でモデルファイルを記述する際には, ufun と呼ばれる void 型関数の中で数理計画モデルを定義する必要があります. ufun 関数を呼ぶには, ヘッダファイル simple.h のインクルードが必要です. より具体的には, 次のように記述する必要があります.

```
#include "simple.h"
void ufun(){
        ... // ここに数理計画モデルを記述
}
```

例えば次の数理計画問題を windows 版と UNIX/Linux 版で記述した場合, 以下のような差が出ます.

$$
\begin{aligned}
\text{最小化} \quad & x + y \\
\text{条件} \quad & x + 2y = 2 \\
& x \geq 0, \quad y \geq 0
\end{aligned}
$$

windows 版

```
Variable x;
Variable y;
Objective f(type=minimize);
f = x + y;
x + 2y == 2;
x >= 0;
y >= 0;
```

UNIX/Linux 版

```
#include "simple.h"
void ufun(){
  Variable x;
  Variable y;
  Objective f(type=minimize);
  f = x + y;
  x + 2y == 2;
  x >= 0;
  y >= 0;
}
```

windows 版では， ufun 関数を呼び出す必要はありません.

3.2.3 最大化/最小化の自動設定

目的関数を定義する際には引数に type=minimize あるいは type=maximize を指定する必要があります. これはそれぞれ扱う数理計画問題が最小化問題，最大化問題であることを意味します. この type= を省略した場合， windows 版では自動的に最小化問題であると認識しますが， UNIX/Linux 版ではコンパイルエラーとなります.

例えば以下の記述は， windows 版では正常に動作しますが， UNIX/Linux 版ではコンパイルエラーです.

```
Variable x;
Variable y;
Objective f;
f = x + y;
```

3.2.4 name 引数の自動設定

データファイルから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． UNIX/Linux 版では，必ず name を付ける必要があります.

4．数理計画モデルの構成要素

以下は SIMPLE を用いて数理計画モデル（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，大半の数理計画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	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
条件分岐関数	ifelse	区間によって定義が異なる関数を定める際に用いる
初等関数	exp, sin, cos, log ...	初等関数を表す

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	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	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
ガントチャート	Gantt	ガントチャートを出力する
ステータス	RcpspStatus	解の情報を保存する

4.1 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 と添え字クラス 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．

4.2 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には引数に type=minimize あるいは type=maximize を指定する必要があります．これはそれぞれ扱う数理計画問題が最小化問題，最大化問題であることを意味します．この type= を省略した場合， windows 版では自動的に最小化問題であると認識しますが， UNIX/Linux 版ではコンパイルエラーとなります．

目的関数に添え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];
```

4.3 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． 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下さい．**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;
```

$x-2y\le 0$ 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 と添え字クラス Element を用います．以下の例では， 3 個の制約式 $x_1-2y_1\le 0$， $x_2-2y_2\le 0$， $x_3-2y_3\le 0$ 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 より新たに導入された SDP ソルバ（lsdp, csdp, qnsdp）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， $X-2E\succeq 0$ であることを意味します.

```
X >= 2;
```

不等号 >= 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;
```

4.4 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． windows 版では 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");
```

UNIX/Linux 版では windows 版と異なり， name の自動解釈はなされません．従って Parameter の引数に必ず name を付ける必要があります．

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「7. データファイル」を参照下さい.

4.4.1 Parameter に対する漸化式による値定義

Parameter に対して漸化式により値を定義する事ができます. Parameter 以外に対しては, 漸化式による定義はできません. 下記は漸化式による Parameter の値定義記述の一例です.

モデルファイルに記述する方法

```
Set I(name="I"); Element i(set=I);
I = "1 .. 10";
Parameter P(name="P", index=I);

// 漸化式記述
startRecurrenceRelation();
P[1] = 1; P[2] = 1;
P[i+2] = P[i] + P[i+1], i+2 < I;
endRecurrenceRelation();
```

上記の例のように漸化式部分は漸化式開始指示関数 startRecurrenceRelation() と漸化式終了指示関数 endRecurrenceRelation() とで囲む必要があります. 上記の例の i+2<I のように Parameter の定義中の添字が添字集合を超えないように, 条件式で添字範囲を指定するのを忘れないでください. 単一の Parameter に対してだけではなく, 下記のように複数の Parameter が複合した複雑な漸化式も定義する事もできます.

複雑な漸化式記述

```
Set I(name=”I”);
Element i(set=I);
I = “1 .. 10”;
Parameter P(name=”P”, index=I);
Parameter Q(name=”Q”, index=I);

// 漸化式記述
startRecurrenceRelation();
P[1] = 1; Q[1] = 2;
P[i+1] = 2*P[i] - 5*Q[i], P[i] >= 0, i+1 < I;
P[i+1] = -3*P[i] + Q[i], P[i] < 0, i+1 < I;
Q[i+1] = -P[i] - 2*Q[i], Q[i] >= 0, i+1 < I;
Q[i+1] = P[i] - 3*Q[i], Q[i] < 0, i+1 < I;
endRecurrenceRelation();
```

漸化式で定義した Parameter は endRecurrenceRelation()以降普通の Parameter と同様に扱うことができます．なお漸化式定義中で下記に挙げる記述を行うことはできませんのでご注意ください．

- 定義が循環する記述(a[i] = a[i] のような記述や a[i] = b[i]かつ b[i] = a[i] のような記述)
- 漸化式定義対象のパラメータの値取得が必要な記述(例えば，パラメータ a が漸化式で定義されている最中の a.val.print() など)
- setOf 関数の使用
- 制約式の記述
- 目的関数の記述
- 漸化式の評価時に添字が添字集合を超えてしまうような記述(例えば a[i]=a[i+1]では評価時に i の値が上昇し続け i の添字集合の範囲を超えてしまいます)

上記に該当するような記述を行った場合実行時エラーとなります．

4.5 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);
```

4.6 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， 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;
```

4.7 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．以下の例をご覧下さい．

```
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 $X_1,..,X_{10}$ 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));
Set B(dim=2,superSet=(S,S));
Parameter a(name="a", index=A);
Parameter b(name="b", index=B);
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．

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．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．

上記の定式化を行った結果，|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．

4.8 行列クラス Matrix

一般の行列は Matrix というクラスで表現されます．対称行列と同様に，一般行列自体の定義と，一般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 2×3 の行列を定義したいとします．

$$X=\begin{pmatrix} x+y & 0 & 1 \\ -1 & 2 & z \end{pmatrix}$$

この場合，以下のように記述します．

```
Set S=”1 2”;
Set T=”1 2 3”;
Element i(set=S),j(set=T);
Matrix X((i,j));
Variable x,y,z;
X["1,1"] = x + y;
X["1,2"] = 0;
X["1,3"] = 1;
X["2,1"] = -1;
X["2,2"] = 2;
X["2,3"] = z;
```

i は行列の行の添字，j は行列の列の添字です．Matrix X((i,j))に表れる二重括弧を省略して Matrix X(i,j)と記述することはできません．

行列クラスを用いて，正方行列を定義することも可能です．ただし，対称行列とは異なり，上下三角部分いずれか一方に関して定義しても対称化されませんので，非対称な行列を定義することができます．

複数の行列を一括して定義することもできます．例えば，次の例では一般行列 $X_1,\ldots,X_{10}$ を一括して定義しています．

```
Set S="1 2";
Set T=”1 2 3”;
Element i(set=S),j(set=T);
Set N="1 .. 10";
Element n(set=N);
Matrix X(index=n,(i,j));
```

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

ここでは添字付けられた行列を行列族とよぶことにします．行列族の各々は行列ですが，行列族から行列を参照するには，丸括弧()か.at()関数を使います．

```
Matrix X(index=n,(i,j));
Matrix Z((i,j));
Z = X(1) + X(2);
Z = X.at(1) + X.at(2);
// Z = X[1] + X[2]; 違法
```

さらに行列の成分を参照する場合には，丸括弧や.at()の後に続けて角括弧[]を使って参照します．対称行列の場合と同様に，行列族の添字と行列の成分の添字を連結して角括弧[]だけで参照することもできます．

```
Matrix X(index=n,(i,j));
Matrix Z((i,j));
Z[“1,1”] = X(1)[“1,1”] + X(2)[“1,1”];
Z[“1,1”] = X.at(1)[“1,1”] + X.at(2)[“1,1”];
Z[“1,1”] = X[“1,1,1”] + X[“2,1,1”];
```

上の三つの代入文はどれも同じ意味に解釈されます．

行列クラスには行列演算が定義されています．二つの行列の型が整合すれば加算，減算，乗算を行うことができます.行列同士の加減乗にはそれぞれ,+,-,* 演算子が対応します.また，行列やベクトルやそれらの演算で表現された制約式も定義することができます．通常の制約式と同様に，等式制約には==を使い，不等式制約には，>=や<=を使います．行列クラスは添字を２つ内包した Expression クラスとみることができます．添字を２つ持った Expression を用いて行列の加減乗に相当する演算を定義すれば，行列クラスを使わなくても行列演算は可能ですが，行列クラスを使えば記述が容易に済みます.

```
Set S="1 2 3";
Set T="1 2 3";
Element i(set=S),j(set=T);
Matrix X((i,j)), Y((i,j)), Z((i,j));
Z = X + Y;          // 行列の足し算
Z = X - Y;          // 行列の引き算
Z = X * Y;          // 行列の掛け算
Z = 2 * X;          // 行列の定数倍
Z = X / 2.0;        // 行列の全成分を 2.0 で割る
```

行列の足し算や引き算では行列のサイズは変わりませんが，掛け算ではサイズが変わることがあります．行列の掛け算の結果を行列に代入する際には行列の宣言時に与えた行列のサイズに注意します.

行列クラスには，加減乗の他に行列固有の演算が関数として用意されています．行列のトレース（対角和）には tr()関数，行列の転置には trans()関数，行列同士の内積には inprod()関数が対応します．関数の戻り値はそれぞれ，Expression，Matrix，Expression になります.

次は，これらの関数の使用例です．行列の演算では添字を陽に書かないことに注意します.

```
Set S="1 2 3";
Set T="1 2 3";
Element i(set=S),j(set=T);
Matrix X((i,j)), Y((i,j)), Z((i,j));
Expression e,d;
e = tr(X);         // 行列のトレース
Z = trans(X);      // 行列の転置
d = inprod(X, Y); // 行列の内積
```

行列クラスには次のような操作関数が用意されています.

.nrow()　　行数を int で返す
.ncol()　　列数を int で返す
.row(const Element& i)　　i 行目を Vector で取得する
.col(const Element& j)　　j 列目を Vector で取得する
.subRect(const Set&, const Set&)　行列の一部分を Matrix で取得する
.val.print() 現在の値を表示する
.printDim()　行列のサイズを表示する

また，零行列や単位行列を表現したい場合には，Matrix クラスのサブクラスである，ZeroMatrix クラスや UnitMatrix を使います. ZeroMatrix クラスと UnitMatrix クラスは Matrix クラスとは違い，値を設定することなく零行列や単位行列として使うことができます．また，オブジェクトを宣言することなく zeros(const Set& S, const Set& T) や unit(const Set& S, const Set& T) といった関数で零行列や単位行列を表すこともできます.

```
Set S="1 2 3";
Element i(set=S), j(set=S);
Matrix X((i,j));
ZeroMatrix O(i);    // 零行列
UnitMatrix I(i);    // 単位行列
X - I >= 0;         // 半正定値の意味
```

$n \times n$の正方行列 >= 0 という記述をすると，行列のすべての成分が零以上であるという意味ではなく，行列の半正定値制約という意味に解釈されます．このとき，行列の対称性はチェックされませんので, かならず, 対称になるように値を設定しておく必要があります.（$n \times n$の正方行列 <= 0 という記述をすると，-1倍した行列の半正定値制約という意味に解釈されます.）

$n \times 1$の行列 >= 0 という記述をすると，行列のすべての成分が零以上であるという意味に解釈されます．この場合，右辺に 0 以外の値を持ってくることはできません.（$n \times 1$の行列 <= 0 という記述をすると，行列のすべての成分が零以下であるという意味に解釈されます.）

Matrix には外部ファイルから値を設定することができます．SIMPLE データ形式でデータを与えることができ，添字なしの Matrix であれば，添字を２つ持った SIMPLE のデータ形式で与え，添字付けられた Matrix であれば，行列族の添字の次元に２を加えた次元の SIMPLE のデータ形式で与えます.

4.9 ベクトルクラス Vector

ベクトルは Vector というクラスで表現されます．ベクトルは行列の特殊なかたち，すなわち，$n\times 1$の行列と同等なものとして扱います．$n\times 1$の行列に対して合法な行列演算はn次元ベクトルに対しても有効です．対称行列や一般行列と同様に，ベクトル自体の定義と，ベクトルの構造の定義は別々に行う必要があります．例えば，次のような三次元のベクトルを定義したいとします．

$$v=\begin{pmatrix} x \\ 2+y \\ z \end{pmatrix}$$

この場合，以下のように記述します．

```
Set S="1 2 3";
Element i(set=S);
Vector v(i);
Variable x,y,z;
v["1"] = x;
v["2"] = 2+y;
v["3"] = z;
```

i はベクトルの添字です．Matrix とは異なり，二重括弧は不要で Vector v(i)のように記述します．

複数のベクトルを一括して定義することもできます．例えば，次の例ではベクトル$v_1,\ldots,v_{10}$を一括して定義しています．

```
Set S="1 2 3";
Element i(set=S);
Set N="1 .. 10";
Element n(set=N);
Vector v(index=n,i);
```

ベクトル自体の添字 n には index= を付ける必要があります．個別のベクトル内部の添字 i には index= を付与してはいけません．

ここでは添字付けられたベクトルをベクトル族とよぶことにします．Matrix クラスと同様に，ベクトル族の添字を指定してベクトルを参照する場合には丸括弧()や.at()関数を使い，ベクトルの成分を参照する場合には角括弧[]を使います．

SIMPLE ではn次元ベクトルと$n\times 1$の行列を同等に扱います．また，1 次元ベクトルと 1×1 の行列とスカラーを同等に扱います．行列やベクトルの型が整合すれば，行列同士の演算，ベクトル同士の演算，行列とベクトルの演算が可能です．

零ベクトルや全ての要素が 1 のベクトルを表現したい場合には，zeros()関数や ones()関数を使います．

```
Set S="1 2 3";
Element j(set=S);
Vector v(j);
v <= ones(S);              // 全ての要素が 1 のベクトル
v >= zeros(S);             // 零ベクトル
```

diag(const Vector&)関数を使えば Vector を行列の対角に並べた正方行列（対角行列）を作ることができます．

```
Set S="1 2 3";
Element i(set=S);
Element j(set=S);
Vector v(j);
Matrix X((i,j));
X = diag(v);
```

1 次元の Vector や 1×1 の Matrix をスカラーに変換したい場合には，scalar(const Matrix&)関数を使います．

```
Set S="1";
Element i(set=S);
Element j(set=S);
Vector v(j);
Matrix X((i,j));
Expression e,d;
e = scalar(v); // ベクトルをスカラーに変換
d = scalar(X);  // 行列をスカラーに変換
```

次は Matrix クラスと Vector クラスを用いて二次計画問題を記述した例です.

```
Set S;
Set T;
Element i(set=S);
Element j(set=T);
Element k(set=T);
Vector x(j);                          // 変数
Variable v(index=j);
x[j] = v[j];
Matrix V((j,k));            // 定数
Matrix A((i,j));            // 定数
Vector b(i);                // 定数
Vector c(j);                // 定数

Objective f(type=minimize);
f = 0.5* inprod(x,V*x) + inprod(c, x);
A*x == b;
x >= 0;
```

次は上のモデルに対するデータファイルの具体例です.

```
S = 1 .. 2;
T = 1 .. 4;
V =
[1,1] 1.0 [1,2] 0.5 [1,3] 0.0 [1,4] 0.0
[2,1] 0.5 [2,2] 1.0 [2,3] 0.0 [2,4] 0.0
[3,1] 0.0 [3,2] 0.0 [3,3] 0.0 [3,4] 0.0;
A =
[1,1]  1 [1,2]  2 [1,3]  1 [1,4]  0
[2,1]  1 [2,2]  1 [2,3]  0 [2,3]  1;
b = [1] 7 [2] 5;
c = [1] -10 [2] -11 [3]  0 [4]  0;
```

行列クラスやベクトルクラスを用いて記述したモデルに対して showSystem()関数を呼ぶと行列の成分ごとに展開された制約式が表示されますので，記述したモデルが正しく記述できているか確認することができます.

4.10 式クラス Expression

式は Expression というクラスで表現されます．式自体の定義と，式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;
```

以下の記述は誤りです．

```
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. 同じ数式が何度も出現するモデルは Expression を用いると，大幅に見易くなります. 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求解結果が異なる事はありません．

4.11 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．集合クラス Set と併用することで，変数 Variable，制約式 Constraint ，定数 Parameter ，整数変数 IntegerVariable ，式 Expression の次元を設定できます．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下さい. 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);
```

添字の引数 index には Element を指定するかわりに，その Element が含まれる Set を指定することもできます．次の例は，上の記述と同じ意味です．

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

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);
Variable x(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え字は自動展開され，制約式が一括して複数定義されます．（**添え字の自動展開機能**）

```
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.

```
, （直積） + （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;
```

4.12 集合クラス Set

集合は Set というクラスで表現されます．添字クラス Element と併用することで，変数 Variable，制約式 Constraint ， 定数 Parameter ， 整数変数 IntegerVariable ， 式 Expression などの次元を設定できます. 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";
```

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";
```

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保管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
```

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．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下さい.

4.13 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，加えて次の関数が使用できます.

- 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のようになります.

```
// 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 $x_p \le x_q$ ， $x_q \le x_r$ ， $x_r \le x_s$ 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 != S.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);
Parameter a(index=i);
for(i=S.first(); i<S; i=S.next(i)){
        a[i] = 2*S.position(i);
}
```

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);
Parameter a(index=i);
for(int p=1; p<S.card()+1; p++){
        i = elementAt[p];
        a[i] = 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;
```

添字を付ける事で，集合族を定義することもできます．次の例では，集合族 M を定義しています． M[1] は a,b,c から， M[2] は d,e,f から構成されています.

```
Set S="1 2";
Element i(set=S);
Set M(index=i);
M[1]="a b c";
M[2]="d e f";
```

4.14 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.

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.

4.15 条件式

条件式は，制約式や代入文（ Variable の初期値設定， Parameter の値設定，Expression の構造定義）の右側に記述され，添字の動く範囲を制限する機能を有します．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;
```

条件式には等号 == ，等式付不等号 <= >= ，不等号 < > ，不一致 != 演算子が使用できます（制約式に使用できるのは，等号と等式付不等号のみです）．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));
```

< > は集合に対する所属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!, &&, || で連結することができます．それぞれ，not，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，以下のように記述する事もできます．

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

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;
```

4.16 条件分岐関数 ifelse

条件分岐関数は，区間によって定義が異なる目的関数 Objective や式 Expression を定義するためのものです．次の例では，目的関数 $f = \begin{cases} x^2, x \geq 0 \\ 0, x \leq 0 \end{cases}$ を ifelse 関数を使用して定義しています．

```
Variable x;
Objective f(type=minimize);
f = ifelse(x>=0, x*x, 0);
```

条件分岐関数を用いる場合，**対象の関数は領域全体で連続かつ微分可能である必要があります**．折れ線関数は連続ですが微分可能でないため，条件分岐関数を用いて記述することはできません．以下の例は誤りです．

```
Variable x;
Objective f(type=minimize);
f = ifelse(x>=0, x, -x);
```

4.17 初等関数

SIMPLE では次の演算と初等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    asinh   acosh   atanh
sech    coth    asech   acsch   acoth
atan2   hypot   erf
exp     log     log10   pow     sqrt
ceil    floor   fabs    fmod
```

次の例では，制約式 $4x^3 <= 11$ 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 を用いないで下さい.

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.

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

5．制約充足問題ソルバ wcsp

5.1　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, 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	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
条件分岐関数	ifelse	区間によって定義が異なる関数を定める際に用いる
初等関数	exp, sin, cos, log ...	初等関数を表す
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．NUOPT 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．よって,「 0 >= 1 」のような制約式は制約違反（ペナルティ 1 ）となりますが,「 0 >= 0.99 」というような制約式は制約を満たしている（ペナルティ 0 ）となります．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]．

5.2 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 で終了条件を指定する必要があります．目的関数が target で指定された値を下回った場合（最大化問題の場合は上回った場合）， wcsp は求解動作を終了します．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，パラメータ 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，パラメータ 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．

目的関数の重みはパラメータ 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．

[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．

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

パラメータ defaultObjectiveWeight の初期値は 1 です.

5.3 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.

5.3.1 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.

5.3.2 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.

[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.

```
        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.

5.3.3 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(パラメータ)により制約式の違反量の計算パラメータの設定もできます.

一般的なソフト制約へのパラメータの設定は, 次のように 3 つのパラメータにより行われます.

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

第 3 引数の省略

softConstraint(w,a)⇒ softConstraint(w,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, 上記のようなパラメータの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(w) 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 でも変更する必要はありません.

5.3.4 パラメータ defaultConstraintWeight

パラメータ 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.

パラメータ 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
```

5.4 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，$0 \le x \le 10$ を満たす整数変数を定めたい場合，通常の数理計画モデルでは次のように記述しますが，

```
IntegerVariable x;
0 <= x <= 10;
```

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);
```

5.5 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);
```

5.6 重複不能関数 alldiff

重複不能関数 alldiff は，添字つきの離散変数 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定めています．

```
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定めています.

```
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));
```

5.7 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.

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

[3] selection 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.

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);
```

5.8 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．

```
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],...,x[10] を考えます．以下の式は，x[1],...,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．

5.9 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，添え字付けされた式及び定数項の中から最小（大）のものを返す関数です．以下の仕様になっています．

```
// 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
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
Expression   max(式, 範囲指定並び)       // 戻り値は一般の式
Parameter    max(定数式, 範囲指定並び)   // 戻り値は定数式
```

min 関数，max 関数は他の関数と異なり範囲指定並びの部分に変数を利用することができます.

第一引数が定数式であり,かつ範囲指定並びの箇所で変数が用いられていない場合は NUOPT で最適化計算を行う前に値の取得を行います. この場合は wcsp 以外のアルゴリズムで使用することも可能です.

範囲指定並びに変数が入る場合や，第一引数が式の場合には，最適化計算速度自体に影響を及ぼします．特に式が線形でなおかつ範囲指定並びに変数が含まれない場合は内部で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((y[j] -1 )*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] <= (y[j] -1 )*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@msi.co.jp までお問い合わせください．

5.10 最小（大）値を取る式を取得する関数 argmin，argmax

最小値を取る式を取得する関数 argmin, 最大値を取る式を取得する関数 argmax は添え字付けされた式及び定数項の中から最小(大)を取る添え字を返す関数です．以下の仕様になっています．

```
// 添字の範囲にわたる最小値を取る式
Element   argmin(線形式, 範囲指定並び)   // 戻り値は特定の添え字
Element   argmin(定数式, 範囲指定並び)   // 戻り値は特定の添え字
// 添字の範囲にわたる最大値を取る式
Element   argmax(線形式, 範囲指定並び)   // 戻り値は特定の添え字
Element   argmax(定数式, 範囲指定並び)   // 戻り値は特定の添え字
```

argmin 関数, argmax 関数は他の関数と異なり範囲指定並びの部分に変数を利用することができます.また，最小（大）値を取る式が複数あった場合にはその中の添え字をどれか１つ返します.

argmin, argmax は以下に挙げる使用方法はサポートしておりません.

- WCSP 以外のアルゴリズム
- 変数との積
- 変数の中の添え字
- 定数の中の添え字として使用した場合にその定数と変数の積

5.11 カウント関数 count

条件を満たす式の数を取得するカウント関数 count は，添え字付けられた式及び定数項の中で与えられた条件（線形の不等式）を満たす個数を返す関数です．以下の仕様になっています.

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

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.

```
Variable x(index = I,type = binary);

Objective obj(type = minimize);
obj = count( 3 <= x[i] <= 5 ,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.

```
Variable x(index = I,type = binary);
Parameter rand(index = I);
Parameter M;
Objective obj(type = minimize);
obj = count(3 <= x[i] <= 5,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リング値です．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下さい.

6. 資源制約付きスケジューリング問題ソルバ rcpsp

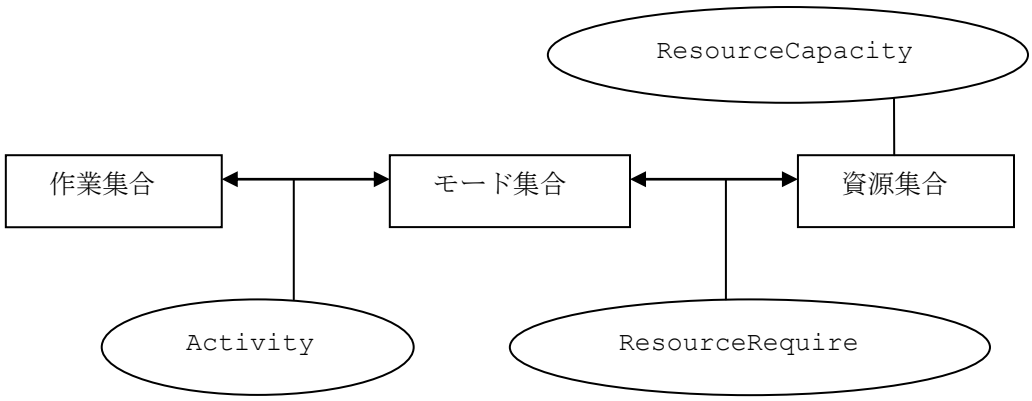
資源制約付きスケジューリング問題とは

- 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． NUOPT 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．

6.1 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	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
ガントチャート	Gantt	ガントチャートを出力する
ステータス	RcpspStatus	解の情報を保存する

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.

6.2 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 = 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 = 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．目的関数の重みはパラメータ 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;
```

パラメータ 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事はできません．

6.3 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 関数あるいはパラメータ 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．

6.4 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]);
```

6.4.1 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 << 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";
```

6.4.2 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る．

6.4.3 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，以下の点にご注意下さい.

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

6.5 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;
```

6.6 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]);
```

6.7 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,i!="b",i!="a"]) == modeOrder(x["a"]);
```

6.8 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);
```

6.9 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");
```

6.10 ガントチャートクラス Gantt

ガントチャートクラス Gantt を用いる事で， rcpsp で解いた結果を， Excel を用いたガントチャートに表示させる事が出来ます．出力対象となる作業を， add 関数に Activity 関数を引数で与えることで制限できます．ガントチャート出力機能を用いるには， NUOPT の Excel 連係機能を用いる必要があります.

次の例では，全ての作業をガントチャートで出力させています.

```
Set A="a b c d";
Activity x(index=i, mode=M);
Gantt g;   // 宣言：
g.add(x[i], i);   //出力要素の追加：
g.dump();  //出力：
```

次の例では，作業 b 以外の全ての作業をガントチャートで出力させています.

```
Set A="a b c d";
Activity x(index=i, mode=M);
Gantt g;   // 宣言：
g.add(x[i], i!="b");   //出力要素の追加：
g.dump();  //出力：
```

添え字と条件式の記述方法は範囲演算関数と同様になります.

6.11 ステータスクラス RcpspStatus

ステータスクラス RcpspStatus を用いる事により，探索終了後の解をファイルにセーブし，次回の探索時に，そのファイルを読み込む事により，探索終了後の解から探索を開始する事が出来ます.

```
RcpspStatus オブジェクト名; // 宣言
オブジェクト名.save(ファイル名の文字列); // 保存
オブジェクト名.load(ファイル名のモード集合);// 読み込み
```

ただし，保存時と読み込み時で，条件に変更があった場合や，ファイルの内容に変更があった場合の動作については，保証されません.

6.12 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.

6.13 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，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なると考えてよい．

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 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].startTim
 e,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,...,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.

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 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].startTim
 e,act[j].endTime,act[j].processTime);
```

実行結果は, 以下のようになります.

```
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
```

7. データファイル

7.1 データファイルの機能

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理計画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．

7.2 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．dat 形式データファイルの拡張子は .dat である必要があります．

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");
```

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 [] を用います．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;
```

7.3 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
```

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際には，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;
```

8．出力制御

SIMPLE で記述された数理計画モデルは， NUOPT 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． GUI 版では，ポップアップにも解の情報が出力されます．

この章では，出力情報の追加に用いられる 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．

8.1 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
Set	val	現在値
OrderedSet	val	現在値

8.2 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
```

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
```

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
```

8.3 simple_printf 関数

simple_printf 関数は,プログラミング言語 C/C++ の関数である printf を SIMPLE 用に拡張したものです． printf 関数は int, double, char 型の変数を出力しますが, simple_printf 関数は替わりにモデルファイルの構成要素の情報を出力します．

simple_printf 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 , ベクトル Vector, 行列 Matrix に関する情報を，任意のフォーマ

ットで出力させる機能を有しています． 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．

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, ..);
```

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，浮動小数点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.val);
```

出力

```
x の値 = 3.000000
```

以下は浮動小数点形式で出力する例です．浮動小数点形式で出力するには %e を用います．

```
simple_printf("x の値 = %e¥n", x.val);
```

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.val);
        simple_printf("x の値 = %.2e¥n", x.val);
```

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.val);
        simple_printf("x の値 = %15e¥n", x.val);
```

出力

```
        x の値 =        3.000000
        x の値 =   3.000000e+000
```

上記の両方の記述をまとめることもできます．

```
        simple_printf("x の値 = %15.2f¥n", x.val);
        simple_printf("x の値 = %15.2e¥n", x.val);
```

出力

```
        x の値 =            3.00
        x の値 =      3.00e+000
```

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
simple_printf("x の値 = %f¥n", x.val);
```

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.val);
```

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.val);
```

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].val);
```

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].val, i<3);
```

出力

```
x[1]の値 = 3
x[2]の値 = 3
```

simple_printf 関数の引数には制限が無いので，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（ .val は省略しても大丈夫な性質を利用しています).

```
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
```

対称行列 SymmetricMatrix の情報を表示させたい場合，対称行列自身の添字と，行列成分の添字を意識して区別する必要はありません．例えば，次の例では 3 つの対称行列 X_1, X_2, X_3 の値を表示させています．行列要素を示す添え字は自動的に出力されます.

```
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, X[n]);
        // [] 内の%d,%d に対応する引数並びは自動的に補われる
```

出力

```
X1[1,1] = 111;
X1[1,2] = 112;
X1[2,2] = 122;
X2[1,1] = 211;
X2[1,2] = 212;
X2[2,2] = 222;
X3[1,1] = 311;
X3[1,2] = 312;
X3[2,2] = 322;
```

simple_printf 関数は, 求解結果を表す要素 result の情報も出力させる事ができます. 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	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	解における最適性条件の残差
elapseTime	double	所要計算時間
errorCode	int	終了時ステータス （成功時：０，失敗時：エラー番号） エラー番号は付録 A.2.1 のものに従います

次のモデルに対する出力は，以下のようになります.

```
Variable x,y;
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);
```

出力

```
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
```

8.4 simple_fprintf 関数

simple_fprintf 関数は, 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 関数と全く同じ機能を有しています. 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.val);
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].val);
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");
```

8.5 出力情報の抑制

デフォルト設定の NUOPT は，標準出力に求解情報を表示し，解ファイル（モデル名.sol）を作成します．ここでは，これらを抑制する方法を紹介します.

標準出力による求解情報の表示を抑制するには，次のように記述します.

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

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_";
```

9．その他の機能

9.1 solve 関数

solve 関数は NUOPT 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．

しかし，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$x-y$ 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 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事があります． 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,type=binary);
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[1]): x[1] <= 23
1-2 (a.smp:8[2]): x[2] <= 5
1-3 (a.smp:8[3]): x[3] <= 4
1-4 (a.smp:8[4]): x[4] <= 12
1-5 (a.smp:8[5]): x[5] <= 1

f<objective>:13*x[1]+7*x[2]+201*x[3]+14*x[4]+23*x[5] (maximize)
```

showSystem の出力では，名前が name= によって与えられない変数の名前は〝〟となります．オブジェクトに name= で名前を与えてから行うと，表示にその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,type=binary);
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 (co[1]): valx[1] <= 23
1-2 (co[2]): valx[2] <= 5
1-3 (co[3]): valx[3] <= 4
1-4 (co[4]): valx[4] <= 12
1-5 (co[5]): valx[5] <= 1

obj<objective>:
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．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 (co[1]): x[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 (co[1]): x[1] <= 23
1-2 (co[2]): x[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

```
SymmetricMatrix{
elem#0 X[1,1] 2*(x[1])
elem#1 X[1,2] 1
elem#2 X[2,2] 1*(x[2])
}
>= 0
```

9.3 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 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);
}
```

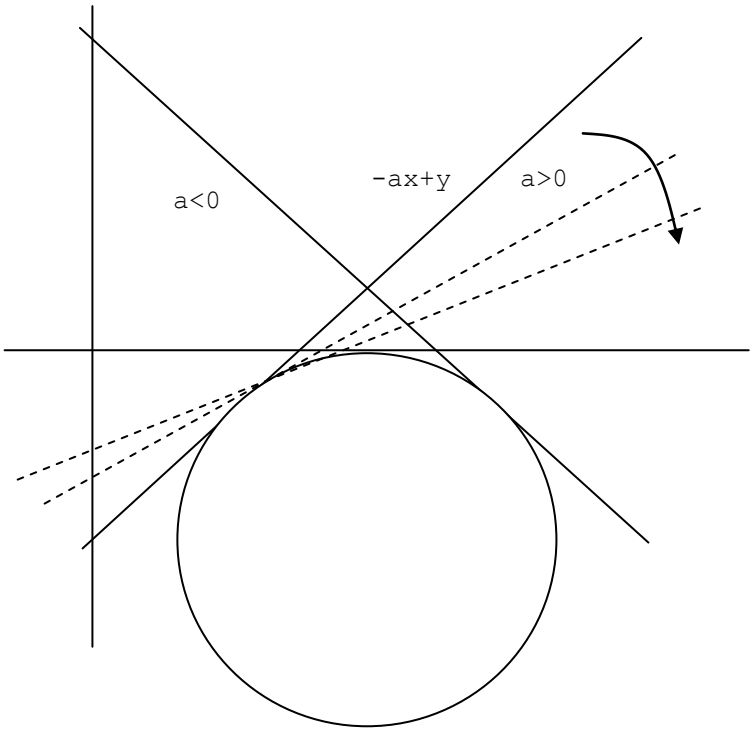
上記のように VariableParameter は通常の Parameter と全く同様にモデル定義に使用することができます.

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 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下さい. また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.



10. 最適化ソルバ NUOPT とは

最適化ソルバ NUOPT はモデリング言語 SIMPLE や MPS ファイル形式で記述された数理計画問題を解くための装置です. NUOPT は豊富なアルゴリズムを有しており，幅広い数理計画問題を扱うことができます.

本マニュアル第二部では，最適化ソルバ NUOPT が出力する情報や，最適化ソルバ NUOPT の動作を制御する各種機能を紹介します.

11. 標準出力

数理計画問題が NUOPT で解かれた場合，最終的な解情報の一部が標準出力に出力されます. 以下はその一例です.

```
PROBLEM_NAME                                          a
NUMBER_OF_VARIABLES                                   2
NUMBER_OF_FUNCTIONS                                   2
PROBLEM_TYPE                                 MINIMIZATION
METHOD                                      HIGHER_ORDER
<preprocess begin>.....<preprocess end>
<iteration begin>
    res=2.4e+001 .... 4.0e-008
<iteration end>
STATUS                                           OPTIMAL
VALUE_OF_OBJECTIVE                            22.50000001
ITERATION_COUNT                                       5
FUNC_EVAL_COUNT                                       8
FACTORIZATION_COUNT                                   6
RESIDUAL                                  3.97842202e-008
ELAPSED_TIME(sec.)                                 0.01
SOLUTION_FILE                                     a.sol
```

11.1 アルゴリズム共通の出力

PROBLEM_NAME は「扱うモデルのファイル名」です. この例では a というモデルを解いています. NUMBER_OF_VARIABLES は「変数 Variable の数」です. この例では変数が 2 個あります. NUMBER_OF_FUNCTIONS は「関数の数」です. ここで言う関数

とは，目的関数 Objective ，制約式 Constraint ，式 Expression 全てを意味します．この例では，関数が 2 個（目的関数 1 個，制約式 1 個）あります． PROBLEM_TYPE は問題が最小化問題（ MINIMIZATION ）なのか最大化問題（ MAXIMIZATION ）なのかを表示します． METHOD は「最適化計算に用いたアルゴリズムの種類」です．この例では線形計画専用内点法（ HIGHER_ORDER ）を用いています． STATUS は「 NUOPT の計算終了時の状態」です．ここでは OPTIMAL（最適）と出力されています． VALUE_OF_OBJECTIVE は「目的関数の値」です． RESIDUAL は「反復終了時の最適性条件」の残差， ELAPSED_TIME(sec.) は「求解に要した時間」， SOLUTION_FILE は「出力される解ファイルの名前」を意味します．出力される解ファイルの名前は，環境や設定により異なります．

11.2 内点法における出力

内点法を用いたアルゴリズムでは，実行経過の出力が以下のようになります．

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

<preprocess begin> と <preprocess end> の間は収束計算に入る前の処理の進行を， <iteration begin> と <iteration end> の間は収束計算の進行を示しています．計算の進行中に表示される数字（ 2.4e+001，2.7e-005 など）は最適性条件の残差で，この表示はそれが計算の進行とともに減少していく様子を示しています．

11.3 単体法，有効制約法，クロスオーバーにおける出力

単体法，有効制約法，クロスオーバーを用いた場合には，実行経過の出力が以下のようになります．

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

ドットは単体法の反復の進行を示しています(1 つのドットにつき，数回の反復を示しています)．また，文字 1 はフェーズ(最適化実行可能解探索状態)からフェーズ(最適解探索状態)への遷移の様子を示しています．

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

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

のように，内点法の経過表示の後に単体法の経過表示が現れます.

11.4 制約充足問題ソルバ (wcsp)における出力

制約充足問題ソルバ(wcsp)を用いた場合には，実行経過に「最良解におけるハード制約」,「ソフト制約のペナルティ値」が逐次表示されます.

```
<iteration begin>
(hard/soft) penalty= 364/1157, time= 0.00(s)
<greedyupdate begin>.........<greedyupdate end>
greedyupdate time= 0.01(s)
(hard/soft) penalty= 91/1295, time= 0.01(s), iteration= 1
(hard/soft) penalty= 70/1314, time= 0.01(s), iteration= 2
(hard/soft) penalty= 52/1320, time= 0.01(s), iteration= 3
(hard/soft) penalty= 38/1282, time= 0.01(s), iteration= 4
(hard/soft) penalty= 19/1259, time= 0.01(s), iteration= 5
             ...
(hard/soft) penalty= 0/4, time= 1.16(s), iteration= 7425
(hard/soft) penalty= 0/3, time= 14.13(s), iteration= 98738
(hard/soft) penalty= 0/1, time= 14.17(s), iteration= 99061
(hard/soft) penalty= 0/0, time= 39.10(s), iteration= 267557
```

```
# (hard/soft) penalty= 0/0
# cpu time = 39.10/39.10(s)
# iteration = 267557/267557
<iteration end>
```

項目の意味は次の通りです.

表示	意味
(hard/soft)	発見された解に関する
penalty=値 1/値 2	値 1：ハード制約違反量，値 2：ソフト制約違反量

time=値 3，iteration=値 4	値 3：経過時間，値 4：反復回数
#penalty=...	最良解のペナルティ値
#cpu time=...	最良解発見時の経過時間
#itereation=...	最良解発見時の反復回数
#status=...	反復停止の理由

最初の項目で説明されている表示は最良解が更新される度に現れます．解が更新されないと表示は停止しますが，計算は行われています．メタ・ヒューリスティクスによる探索の一般的な性質として，計算の後半には解の更新は鈍くなります．

制約にセミハード制約が付随している場合は，次のような出力になります．

```
<iteration begin>
(hard/soft) penalty= 364/1157, time= 0.00(s)
<greedyupdate begin>.........<greedyupdate end>
greedyupdate time= 0.01(s)
(hard/semihard/soft) penalty= 0/91/1295, time= 0.01(s), iteration= 1
(hard/semihard/soft) penalty= 0/70/1314, time= 0.01(s), iteration= 2
(hard/semihard/soft) penalty= 0/52/1320, time= 0.01(s), iteration= 3
(hard/semihard/soft) penalty= 0/38/1282, time= 0.01(s), iteration= 4
(hard/semihard/soft) penalty= 0/19/1259, time= 0.01(s), iteration= 5
              ...
(hard/semihard/soft) penalty= 0/0/4, time= 1.06(s), iteration= 7425
(hard/semihard/soft) penalty= 0/0/3, time= 13.72(s), iteration= 98738
(hard/semihard/soft) penalty= 0/0/1, time= 13.77(s), iteration= 99061
(hard/semihard/soft) penalty= 0/0/0, time= 37.46(s), iteration= 267557
```

```
# (hard/semiard/soft) penalty= 0/0/0
# cpu time = 37.46/37.46(s)
# iteration = 267557/267557
<iteration end>
```

penalty 値に，セミハード制約の違反量も出力されます．

表示	意味
(hard/semihard/soft) penalty=値 1/値 2/値 3	発見された解に関する 値 1：ハード制約違反量，値 2：セミハード制約違反量 値 3:ソフト制約違反量

11.5 分枝限定法における出力

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

```
<iteration begin>
          .1.2B
    up:       1e+50 lo:  8.6617e+05 time:  0.1s:mem(Mb)=0
                                        llen:1 #prob:10 #piv:210
#1  up:  9.0859e+05 lo:  8.6655e+05 gap: 42036 time:  1.6s:mem(Mb)=2
                                        llen:359 #prob:1009 #piv:4223
#2  up:  9.0746e+05 lo: 8.6655e+05 gap: 40901 time:  6.6s:mem(Mb)=4
                                        llen:1002#prob:2652 #piv:6493

#3  up:  9.0713e+05 lo: 8.6655e+05 gap: 40574 time: 12.4s:mem(Mb)=8
                                        llen:1876          #prob:4776
#piv:9105
#4  up:  9.0692e+05 lo: 8.6655e+05 gap: 40361 time: 17.9s:mem(Mb)=12
                                      llen:2756#prob:6896#piv:11802
                     (中略)
                                      llen:2680#prob:58708iv:65012
#47 up:  8.7972e+05 lo:8.6655e+05 gap: 13161 time:174.4s:mem(Mb)=43
                                      llen:2167#prob:59374 piv:65851
    up:  8.7972e+05 lo: 8.6655e+05 gap: 13161 time:180.1s:mem(Mb)=43
                                      len:2169#prob:60990#piv:67529
#48    up:       8.7907e+05    lo:       8.6655e+05    gap:    12518
time:184.3s:mem(Mb)=43
                                      llen:1470#prob:62102 piv:68561
#49   up:            8.79e+05   lo:      8.6655e+05   gap:    12446
time:190.4s:mem(Mb)=43
                                      llen:1396#prob:64450#piv:71062
#50    up:       8.7843e+05    lo:       8.6655e+05    gap:    11876
time:192.2s:mem(Mb)=43
```

各行は,

- 新しい暫定解が得られた
- 所要メモリが 50Mb 以上変動した
- 60 秒経過した

のいずれかの条件を満たせば出力されます.

新しい暫定解が得られた場合は，行頭に"#"が表示されます．"#"に続く番号は暫定解の番号です．それ以外の場合には#の項目がありません.

他の項目の意味は次の通りです.

表示	意味
up	目的関数の上界値
lo	目的関数の下界値
gap	上下界ギャップ：(上界値 － 下界値)
time	経過時間（秒）
mem(Mb)	全使用メモリ / 実メモリ上にあるメモリ量
avail(Mb)	現在実行している計算機で使用できるプロセスメモリ最大量/実メモリの残り最大量
llen	分枝限定法に対するリストの長さ
#prob	通算の子問題数
#pivot	通算のピボット数

実行環境によっては"mem"の表示の「実メモリにあるメモリ量」を示す部分，および"avail"項は表示されません.

この問題では合計 50 個の暫定解が求まり，最後の暫定解の上下界ギャップは 11876 であったことがわかります.

11.6 資源制約付きスケジューリング問題ソルバ(rcpsp)における出力

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

11.6.1 完了時刻最小化

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

項目の意味は次の通りです.

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

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

11.6.2 納期遅れ最小化

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

項目の意味は次の通りです.

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

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

11.7 実行不可能性要因検出機能（ iisDetect ）の出力

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

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

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

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

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

この問題を解かせたとき，以下のような出力がなされます．

それぞれの出力の意味は，以下のようになります.

```
ERROR_TYPE                          <<NUOPT 11>> infeasible.
DETECTED_IIS_SIZE                                        3（追加）
(#IIS_RELATED_VAR)                                       3（追加）
INFEASIBILITY_OF_IIS                                     1（追加）
```

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

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

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

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

出力

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

11.8 標準出力内容一覧

以下は，標準出力に出力される内容の一覧です.

タイトル	解説	備考
PROBLEM_NAME	問題名	SIMPLE 版:モデル名 MPS 版:TITLE の内容
NUMBER_OF_VARIABLES	変数の総数	
NUMBER_OF_FUNCTIONS	関数 (目的関数を含む) の総数	
PROBLEM_TYPE	minimize（最小化） maximize（最大化）	
METHOD	適用した最適化手法	
ERROR_TYPE	発生したエラーの種類	エラー発生時のみ
STATUS	optimal（最適） non_optimal （最適でない）	
VALUE_OF_OBJECTIVE	目的関数値	
ITERATION_COUNT	反復回数	内点法のみ
FUNC_EVAL_COUNT	関数の評価回数	内点法のみ
FACTORIZATION_COUNT	行列の分解回数	内点法のみ
RESIDUAL	最適性条件の充足度合	分枝限定法以外
SIMPLEX_PIVOT_COUNT	単体法の反復回数	単体法のみ
PARTIAL_PROBLEM_COUNT	部分問題数	分枝限定法のみ
DUAL_SIMPLEX_PIVOT_COUNT	双対単体法の反復回数	分枝限定法のみ
PENALTY	重みつき制約違反量	wcsp/rcpsp 適用時のみ
TERMINATE_REASON	計算終了理由	wcsp/rcpsp 適用時のみ
DETECTED_IIS_SIZE	検出された IIS に含まれる行の数	IIS 特定成功時のみ
(#IIS_RELATED_VAR)	IIS に含まれている行に含まれる変数の数	IIS 特定成功時のみ
INFEASIBILITY_OF_IIS	IIS 全体での実行不可能性	IIS 特定成功時のみ

NO_IIS_FOUND_BY	IIS 検出失敗の原因	IIS 特定失敗時のみ
(#NONLINEAR_CONSTR.)	IIS 検出失敗の原因の可能性がある非線形制約の数	IIS 特定失敗時のみ
NUMBER_OF_ACTIVITIES	アクティビティの総数	rcpsp のみ
NUMBER_OF_RESOURCES	資源の総数	rcpsp のみ
NUMBER_OF_PRECEDENCE	先行制約の総数	rcpsp のみ
NUMBER_OF_IMPRECEDENCE	直前先行制約の総数	rcpsp のみ
NUMBER_OF_GENERAL_CONSTRAINT	一般の考慮制約の総数	rcpsp のみ
NUMBER_OF_MODES	モードの総数	rcpsp のみ

11.9 標準出力の抑制

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

一つは，パラメータファイル nuopt.prm に記述する方法です． nuopt.prm 内に以下のように記述します．

```
output:mode = silent
```

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

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

12. 解ファイル

NUOPT は最適化計算の詳細情報を解ファイルというファイルに出力します. コマンドラインで NUOPT を起動した場合, モデル名.sol という解ファイルが作成されます. GUI を用いて NUOPT を起動した場合, result フォルダ以下に solfile.txt という解ファイルが作成されます.

具体例として, 次の例題に対する解ファイルを見ていくことにします.

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

12.1 冒頭部分

解ファイルの冒頭部分には標準出力に出力される内容と, 同等の情報が出力されます.

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

```
NUMBER_OF_VARIABLES                                    3
NUMBER_OF_FUNCTIONS                                    4
PROBLEM_TYPE                                  MINIMIZATION
 METHOD                                        HIGHER_ORDER
STATUS                                             OPTIMAL
VALUE_OF_OBJECTIVE                                   -10.5
ITERATION_COUNT                                        6
FUNC_EVAL_COUNT                                        9
FACTORIZATION_COUNT                                    7
RESIDUAL                                 1.354127444e-008
ELAPSED_TIME(sec.)                                  0.01
```

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

```
NUMBER_OF_VARIABLES                                    3
NUMBER_OF_FUNCTIONS                                    4
PROBLEM_TYPE                                  MINIMIZATION
METHOD                                            SIMPLEX
STATUS                                            OPTIMAL
VALUE_OF_OBJECTIVE                                 -10.5
SIMPLEX_PIVOT_COUNT                                    3
RESIDUAL                                               0
ELAPSED_TIME(sec.)                                  0.00
```

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

```
NUMBER_OF_VARIABLES                                    3
NUMBER_OF_FUNCTIONS                                    4
PROBLEM_TYPE                                  MINIMIZATION
METHOD                                            SIMPLEX
STATUS                                            OPTIMAL
VALUE_OF_OBJECTIVE                                    -10
SIMPLEX_PIVOT_COUNT                                    4
PARTIAL_PROBLEM_COUNT                                  3
DUAL_SIMPLEX_PIVOT_COUNT                               6
ELAPSED_TIME(sec.)                                 0.017
```

12.2 解ファイルの変数値表示部

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

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

```
%%
%% VARIABLES
%%
     NAME      VALUE      STATUS [           BOUND TYPE           ]
V# 1  x1       2.499999998  FREE [     0      <=  x1              ]
V# 2  x2       1.499999999  FREE [     0      <=  x2              ]
V# 3  x3  1.223480816e-009 LOWER [     0      <=  x3               ]
```

VALUE は「変数の値」， STATUS は「変数値が上下限のいずれに付着しているかの状態」， BOUND は「変数の上下限」を示しています.

例えば，V# 1 から始まる行は x1 という名前の変数の値(2.5)と，それが下限にも上限にも等しくなっていないこと("FREE")，続いて X1 に課された制約の種類([]内)が示されています.

変数の状態を示す STATUS の意味を以下に示します.

状態を示す文字列	状態
LOWER	下限制約に付着(下限制約が active)
UPPER	上限制約に付着(上限制約が active)
FREE	上下限，いずれにも付加していない
INFS	上下限を違反している

INFS は数値的な性質の悪い問題や最適化が途中で失敗した際に出力されます．これが含まれる場合には，エラーが出力されます．問題は正常に解けていません.

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

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

例えば，

```
%%
%% ACTIVITIES
%%
        NAME          MODE               [             START/END TIME             ]
V# 1 sourceActivity     "DummyMode"   ACT [     0     <= sourceActivity <=     0     ]
V# 2    act[1]           "A_does"    ACT [     6     <=     act[1]     <=     12     ]
V# 3    act[2]           "C_does"    ACT [     0     <=     act[2]     <=     11     ]
V# 4    act[3]           "B_does"    ACT [     0     <=     act[3]     <=     8      ]
```

のような出力があった場合には， MODE が アクティビティが処理されるモード，START/END TIME がアクティビティの開始時刻，終了時刻を表します.

12.3 解ファイルの関数値表示部

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

で始まる部分には最適化アルゴリズム停止時における関数[4]とその双対変数（シャドウプライス）の値が記述されています.

```
   %%
   %% FUNCTIONS
   %%
        NAME    VALUE     STATUS [   CONSTRAINT/OBJECTIVE TYPE    ]
F# 1  f            -10.5  FREE [      OBJECTIVE (MINIMIZE)       ]
F# 2  g1       3.999999999 UPPER [              g1  <=     4      ]
F# 3  g2       4.999999998 UPPER [              g2  <=     5      ]
F# 4  g3       6.499999998  FREE [             g3  <=     7      ]
```

VALUE が「関数の値」，STATUS が「関数値が上下限のいずれに付加しているかの状態」，CONSTRAINT/OBJECTIVE TYPE が「関数の上下限」を示しています.

F# 1 から始まる行には F という名前の関数(目的関数)の値(-10.5)と，それが最小化された("MINIMIZE")目的関数である旨が示されています. F# 2 から始まる行 G1 という名前の制約式の値が 4 であり，それが上限に等しくなっていること("UPPER")，続いて G2 に課された制約の種類([] 内)が示されています.

[4]目的関数と制約式を総称してこう呼んでいます.

関数(制約式)の状態を示す STATUS の意味を以下に示します.

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

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

12.4 解ファイルの上下限，制約と対応する双対変数表示部

解ファイルの

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

から続く部分には変数の上下限と双対変数，また

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

から続く部分には制約式の上下限と双対変数がそれぞれ表示されます.

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

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

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

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

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

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

目的関数の双対変数値に対応する部分には零が出力されます.

12.5 解ファイルの実行不可能性要因出力部

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

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

解ファイル出力

```
%%
%% IIS
%%
--------------------------------------------------------------------------------
#2     lp.smp:5     :        y - x
                                                  <=         0   (        0)
--------------------------------------------------------------------------------
#3     lp.smp:6     :       -1 + x - z
                                                  <=         0   (        0)
--------------------------------------------------------------------------------
#4     lp.smp:7 INFS :        2 + z - y
                                                  <=         0   (        1)

--------------------------------------------------------------------------------
```

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

INFEASIBILITY_OF_IIS

という値です．

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

12.6 解ファイルのハード制約，セミハード制約およびソフト制約表示部

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

解ファイル出力

```
%%
%% WCSP_PENALTY
%%
          NAME               TYPE    VALUE   BOUND    AMOUNT WEIGHT PENALTY
F#   246   model.smp:83[6]      HARD    0    >=    1       1
F#   432   model.smp:91[13]     S.HARD  0    >=    1       1
F#   946   model.smp:119[17,E]  S.HARD  7    <=    6      1
F#  7080   model.smp:152[1](u)  SOFT    3    <=    2       1     100    100
F#  7586   model.smp:156[7,3]   SOFT    5    <=    2       3      10     30
F#  7675   model.smp:156[21,6]  SOFT    3    <=    2       1      10     10
F#  7756   model.smp:156[2,10]  SOFT    4    <=    2       2      10     20
F#  7780   model.smp:156[1,11]  SOFT    3    <=    2       1      10     10
F#  8293   model.smp:162[19,25] SOFT    0    ==    5       5       1      5
                            . . .
```

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

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

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

windows 版では，同一の内容が ”wcspout” という名前の解オブジェクトとして出力され， GUI から参照することができます．

12.7 解ファイルの対称行列成分値表示部

```
%%
%% MATRICES
%%
```

で始まる部分には，最適化アルゴリズム停止時における対称行列の各成分の値が記述されています．この部分は，モデルで対称行列を定義した場合にのみ出現します．冒頭の例では，半正定値制約が存在しないので，解ファイルにはこの項目は出現しません．

新たな具体例として，次の例題に対する解ファイルを考えます．

最小化　$x_1 + x_2$

条件

$$X = \begin{pmatrix} 2x_1 & 1 \\ 1 & x_2 \end{pmatrix} \succeq 0$$

この問題に対する解ファイルのうち，対称行列に関する部分は次のように出力されます．

```
%%
%% MATRICES
%%
     NAME      VALUE
M# 1  X[1,1]  1.414215
M# 2  X[1,2]  1.000000
M# 3  X[2,1]  0.707107
```

VALUE は「各行列成分値」を示しています．

13. NUOPT の適用範囲とアルゴリズム

本章では， NUOPT で扱う事のできる数理計画問題の範囲， NUOPT が備えているアルゴリズムを列挙し，それらの対応関係を与えます．また，アルゴリズムの設定方法も示します．

13.1 数理計画問題一覧

NUOPT では，以下のような数理計画問題を取り扱うことができます．

- LP (Linear Programming:線形計画問題)
  目的関数と制約式がすべて線形である問題で，整数変数を含まないものです．

- MILP (Mixed Integer Linear Programming:混合整数計画問題)
  目的関数と制約式がすべて線形で，整数変数を含むものです． MIP (Mixed Integer Programming)と呼ばれることも多いです．

- MIQP (Mixed Integer Quadratic Programming:混合整数二次計画問題)
  制約式がすべて線形，目的関数が二次関数で，整数変数を含むものです．

- MINLP (Mixed Integer Nonlinear Programming:混合整数非線形計画問題)
  制約式および目的関数が非線形で整数変数を含むものです．

- CQP (Convex Quadratic Programming:凸二次計画問題)
  目的関数が凸な二次関数，制約式がすべて線形であるもの（ただし，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できるもの）です．

- CP (Convex Programming:凸計画問題)
  目的関数，制約式に非線形なものが含まれていますが，実行可能領域が凸で，目的関数の符号の変更で下に凸な目的関数の最小化に帰着できる問題です．
  ここでは整数変数は含まないものを言います．
  半正定値計画問題も凸計画問題の一部ですが，ここには含めません．

- NLP (Nonlinear Programming:非線形計画問題)
  上記以外で，整数変数を含まない一般の非線形計画問題です．

- SDP (SemiDefinite Programing:半正定値計画問題)
  行列の半正定値制約を含む線形計画問題です.

- NLSDP (NonLinear SemiDefinite Programing:非線形半正定値計画問題)
  行列の半正定値制約を含み, かつ目的関数・制約式に非線形項が含まれる問題です.

- WCSP (Weighted Constraint Satisfaction Problem:重み付き制約充足問題)
  各々重みの付いた制約条件をなるべく満足するためには値をどのように割り当てると良いかを決定する問題です. 制約充足問題ソルバ wcsp により高速に解を得ることができます.

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

13.2 アルゴリズム一覧

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

- simplex :単体法(SIMPLEX)
  線形計画法の解法として古くから知られている方法です. 大規模問題では内点法に速度的に劣りますが，可能基底解が求まり原理的に内点法/外点法よりも高精度です.
  整数変数を含む問題に対して指定すると，単体法を分枝限定法(Branch and bound method)という枠組のなかで繰り返し行って，最適性の保証のある整数解を求めます．大規模問題において基底解が必要な場合には，"cross:on"と指定して内点法からのクロスオーバーを用いるのが有利です.

- asqp :有効制約法(ACTIVE_SET_QP)
  単体法と同様，古典的な凸二次計画問題の厳密解法です．1 万変数以上の大規模問題では，一般に内点法（直線探索法 (Line Search Method)）に劣りますが，
  ・変数に比べて制約式の数が非常に少ない（1/10 以下）場合

・目的関数のヘッセ行列が密行列である場合
には内点法よりも高速かつ高精度です．また，整数計画法に対応しているので，整数変数が含まれている 凸二次計画問題を解くことができます． "cross:on"と指定することで内点法からのクロスオーバーを用いることができるので，大規模問題に対して高精度な解を求めることができます．

◆ higher :線形計画問題専用内点法(HIGHER_ORDER)
線形計画法に特化した内点法で，大規模な線形計画問題の解法としては最も高速です．単体法と違い，可能基底解は求まりません．

◆ lipm :（新版）直線探索法(LINE_SEARCH_IPM)
◆ lepm :直線探索外点法(LINE_SEARCH_EPM)
◆ line :（旧版）直線探索内点法(LINE_SEARCH)
一般の凸計画問題に適用可能な内点法・外点法です．問題が凸であることがわかっている場合には信頼領域法よりも高速です．幅広い範囲の問題に対して有効なのが内点法（lipm），外点法（lepm）は問題に対して比較的良い初期値が得られている場合に有効であることが示されています．旧版の内点法(line)は，以前のバージョンとの整合を取る場合にご利用ください(Ver.7 以前の内点法 line と Ver.8 以降の内点法 lipm では，メリット関数の定義が若干異なっておりますので，新版と旧版では異なる結果を与える場合があります)．

◆ bfgs :準ニュートン法(BFGS_LINE_SEARCH)
準ニュートン法によって二階微係数を求める内点法です．”bfgs”はヘッセ行列の近似行列を密行列として保持しますので，小規模(50 ～500 変数以下)な非線形計画問題に適しています．

◆ tipm :（新版）信頼領域内点法（TRUST_REGION_IPM）
◆ tepm :信頼領域外点法(TRUST_REGION_EPM)
◆ trust :（旧版）信頼領域法内点法(TRUST_REGION)
大規模なものを含む一般の非線形計画問題に適用可能な内点法・外点法です．幅広い範囲の問題に対して有効なのが内点法（tipm），外点法（tepm）は問題に対して比較的良い初期値が得られている場合に有効であることが示されています．旧版の内点法(trust)は，以前のバージョンとの整合を取る場合にご利用ください（Ver.7 以前の内点法 trust と Ver.8 以降の内点法 tipm では，メリット関数の定義が若干異なっておりますので，新版と旧版では異なる結果を与える場合がございます)．

- lsqp :直線探索法に基づく逐次二次計画法(LINE_SEARCH_SQP)
  準ニュートン法によって二階微係数を求める逐次二次計画法です．小規模(50～100 変数以下)な非線形計画問題に適しています
  問題によっては直線探索内点法/外点法（lipm/lepm/line）よりも安定的により精度の良い解を導くことができます.

- tsqp :信頼領域法に基づく逐次二次計画法(TRUST_REGION_SQP)
  二階微係数をそのまま用いる逐次二次計画法です．大規模なものを含む一般の非線形計画問題に適用可能な方法です．一般に内点法よりも低速ですが，問題によっては内点法よりも安定的に，より精度の良い解を導くことができます.
  変数の数よりも制約式数が多い場合には内点法/外点法（tipm/tepm/trust）よりも高速な場合があります.

- slpsqp :逐次線形二次計画法 (SLP-SQP)
  ペナルティ関数を用いない逐次線形二次計画法です．ある程度大規模なものを含む一般の非線形計画問題に適用可能です．大域的収束性を保証する原理が内点法/外点法(tipm/tepm/trust)や従来の逐次二次計画法(lsqp/tsqp)のものとは異なるため，他の方法で収束しない問題に対して有効な場合があります．また，制約式の勾配が小さい反復点にいる時，実行可能領域に接近するという仕組みが組み込まれていますので複雑な制約においても安定的な動作が期待できます.

- lsdp :線形半正定値計画問題に対する主双対内点法
  線形の半正定値計画問題に対する主双対内点法です．目的関数・制約式に出現する項は線形である必要があります．内部でメリット関数の計算を行いません.

- csdp :凸半正定値計画問題に対する主双対内点法
  目的関数が凸非線形関数で，制約が線形な半正定値計画問題に対する主双対内点法です．この類の問題は，線形半正定値計画問題で記述することができますが，そのまま扱った方が高速に求解できます．内部でメリット関数の計算を行う点が lsdp と異なります.

- qnsdp :準ニュートン法を用いた非線形半正定値計画問題に対する主双対内点法
  目的関数・制約式に非線形項が出現する半正定値計画問題に対する主双対内点法です．メリット関数の降下を保証するために，準ニュートン法を利用しています.

◆ trsdp :信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法

目的関数・制約式に非線形項が出現する半正定値計画問題に対する主双対内点法です．メリット関数の降下を保証するために，信頼領域法を利用しています．準ニュートン法に基づく方法（qnsdp）より規模の大きな問題を取り扱う事ができます．

◆ wcsp :制約充足問題ソルバ(WCSP)

京都大学「問題解決エンジン」グループの開発による制約充足問題に対するアルゴリズムです．必ずしも厳密解が求まるわけではありませんが，大規模な整数計画問題に対し，非常に高速に実行可能解（近似解）を求めることができます．

整数変数のみを含み，かつすべての変数に上限と下限がある問題に対してのみ有効です．目的関数，制約式に重みを設定することができます．制約の重みには，ハード制約，セミハード制約，ソフト制約の三種類があります．

◆ rcpsp :資源制約付きスケジューリング問題ソルバ(RCPSP)

京都大学「問題解決エンジン」グループの開発による資源制約付きスケジューリング問題に対するアルゴリズムです．資源制約の下，決められた作業の開始・終了時刻を決定する問題の実行可能解を高速に求めることができます． rcpsp の記述にあたっては問題を SIMPLE の特殊なクラスを用いて記述する必要があります．完了時刻の最小化問題と，納期遅れ最小化問題を扱うことができます．前者を扱う際にはソフト制約，後者を扱う際にはハード制約のみが使用できます．

◆ global :大域的最適化アルゴリズム(GLOBAL)

一般の非線形計画問題または，整数変数を含む非線形計画問題に対し，大域的な最適解を求めることができます． NUOPT が備えている他の非線形計画法アルゴリズム(bfgs,trust)では，局所的な最適解を出力する可能性があるのに対し，この解法は大域的な最適解を与えます．ただし，凸緩和法に基づく分枝限定法を行いますので，計算機資源を多く消費します．数十変数程度の小規模でかつ複雑な問題に適しています．**本アルゴリズムの実行に際しては NLP モジュール 及び NUOPT/Global が必要です．NUOPT/Global は有償アドオンであり，別途購入が必要となります．**

◆ DFO ：目的関数の数式表現が困難な問題に対するアルゴリズム

目的関数について数式表現が困難あるいは微分に関する情報を用いることが出来ないような整数変数を含まない小規模な問題に有効な解法です. **本アルゴリズムの実行に際しては NLP モジュール 及び NUOPT/DFO が必要です．NUOPT/DFO は有償アドオンであり，別途購入が必要となります．また，NUOPT/DFO のご利用方法に関しては「NUOPT/DFO 利用ガイド」をご参照下さい.**

NUOPT のアルゴリズムは，SIMPLE で記述される目的関数，制約で四則演算および以下の関数を用いて記述されたものをサポートします.

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

erf 関数は バージョン 9 から導入された，ガウスの誤差関数です.

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

◆ iisDetect ：実行不可能性要因行の特定アルゴリズム

上記のアルゴリズムと異なり，本アルゴリズムは数理計画問題を解くためのものではありません．本アルゴリズムは NUOPT に与えられた問題が実行不可能と判定されたら，自動的に起動し，実行不可能性の原因となっている制約式の組（できるだけ式の少ないもの）を特定します. 線形計画問題(LP)，混合整数計画問題 (MILP) にのみ適用できます．内部では単体法を繰り返し呼び出しています． iis は初期設定では on になっています.

13.3 数理計画問題とアルゴリズムの対応

以下は， NUOPT で取り扱い可能な数理計画問題と， NUOPT が備えているアルゴリズムの対応一覧です.

表の内容は

○ 最も適している

△ 適用できるが非効率

-- 適用できない

を示します.

	LP	MILP	MIQP	MINLP	CQP	CP	NLP	SDP	NLSDP	RCPSP
simplex	○	○	--	--	--	--	--	--	--	△
asqp	○	--	○	--	○	--	--	--	--	--
higher	○	--	--	--	--	--	--	--	--	--
lipm/lepm/line	△	--	--	--	○	○	--	--	--	--
bfgs/lbfgs	△	--	--	--	--	△	○	--	--	--
tipm/tepm/trust	△	--	--	--	--	△	○	--	--	--
lsqp	△	--	--	--	△	○	○	--	--	--
tsqp	△	--	--	--	△	○	○	--	--	--
slpsqp	△	--	--	--	△	○	○	--	--	--
lsdp	△	--	--	--	△	△	--	○	--	--
csdp	△	--	--	--	△	△	--	○	--	--
qnsdp	△	--	--	--	△	△	△	△	○	--
trsdp	△	--	--	--	△	△	△	△	○	--
wcsp	--	○※	○※	○※	--	--	--	--	--	--
global	△	△	△	○	△	△	○	○	○	△
DFO	△	--	--	--	△	△	○	--	--	--
rcpsp	--	--	--	--	--	--	--	--	--	○
iisDetect	○	△	△	△	△	△	△	--	--	--

wcsp は○※とありますが， 0-1 整数変数と離散変数のみを含む問題に対して適用で

きます．連続変数を含む問題あるいは上限と下限を持たない整数変数を含む問題には適用できません．ご注意下さい．

13.4 アルゴリズムの設定方法

アルゴリズムを設定する方法には，

- モデルファイル内で指定する方法
- パラメータファイル nuopt.prm 内で指定する方法

の二通りがあります．指定可能なアルゴリズム名の一覧は，以下の通りです．これは 4.2 アルゴリズム一覧におけるアルゴリズムの見出し名と同一です．ただし，DFO に関しては実行方法が異なるためこの方法では指定できませんのでご注意下さい．

アルゴリズム名	日本語名
simplex	単体法（＋分枝限定法）
asqp	有効制約法
higher	線形計画専用内点法
lipm	（新版）直線探索内点法
lepm	直線探索外点法
line	（旧版）直線探索内点法
bfgs	準ニュートン法
tipm	（新版）信頼領域内点法
tepm	信頼領域外点法
trust	（旧版）信頼領域内点法
lsqp	直線探索を利用した逐次二次計画法
tsqp	信頼領域を利用した逐次二次計画法
slpsqp	逐次線形二次計画法
lsdp	線形半正定値計画問題に対する主双対内点法
csdp	凸半正定値計画問題に対する主双対内点法
qnsdp	準ニュートン法を用いた非線形半正定値計画問題に対する主双対内点法
trsdp	信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
wcsp	制約充足問題ソルバ
rcpsp	資源制約付きスケジューリング問題ソルバ
global	大域的最適化アルゴリズム

モデルファイル内で指定する方法：

```
options.method = “アルゴリズム名”;
```

パラメータファイル nuopt.prm 内で指定する方法：

```
method:アルゴリズム名
```

以下の例では，アルゴリズム simplex （単体法）をそれぞれモデルファイルから，あるいはパラメータファイルから設定しています.

モデルファイル内で指定する方法：

```
options.method = “simplex”;
```

パラメータファイル nuopt.prm 内で指定する方法：

```
method:simplex
```

実行不可能性検出アルゴリズム iisDetect の設定解除には，別の設定方法が用意されています．こちらも，

- モデルファイル内で指定する方法
- パラメータファイル nuopt.prm 内で指定する方法

の二通りがあります.

モデルファイル内で指定する方法：

```
options.iisDetect = “off”;
```

パラメータファイル nuopt.prm 内で指定する方法：

```
param:iis=off
```

13.5 アルゴリズムの自動選択

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

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

次のような場合，個別に設定するとよりよい結果が得られる可能性があります.

13.5.1 整数変数が含まれている非線形計画問題

整数変数が含まれている非線形計画問題の場合には，初期設定である asqp は二次計画問題でなければエラーを返します.

大域的最適化 global アドインをインストールされている場合には，大域的最適化アルゴリズム global が有効な場合があります.

また変数がすべて０－１整数変数であれば制約充足問題ソルバ wcsp も適用可能です.

13.5.2 整数変数が含まれない非線形計画問題

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

同じ信頼領域法でも外点法 tepm や旧版の内点法 trust ，逐次二次計画法 tsqp あるいは逐次線形二次計画法 slpsqp を用いることもできます．外点法 tepm は問題に対して比較的良い初期値が得られている場合に有効であることが示されています．旧版の内

点法 trust は，以前のバージョンとの整合を取る場合にご利用ください．逐次二次計画法 tsqp や逐次線形二次計画法 slpsqp は若干時間を所要するケースもございますが，最も精度良く非線形最適化を行う方法です．内点法 tipm/trust などで収束しないケースについて逐次線形二次計画法 slpsqp が有効な場合があります．

13.5.3 凸計画問題

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

13.6 クロスオーバー

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

バージョン 15 より，混合整数計画問題(MIP)に対しても，起動可能です．

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
cross:on
```

14. パラメータ設定

パラメータは， NUOPT の求解動作をより細かく制御するためのものです．パラメータを利用することにより，アルゴリズムの選択や，標準出力の制御，終了条件の調整などを行う事ができます． SIMPLE モデルに出現する定数クラス Parameter とは無関係です．パラメータを設定する方法には次の二通りの方法があります．

1. モデルファイル中に記述する
2. パラメータファイル nuopt.prm に記述する

コマンドラインから NUOPT を利用する場合，両方の方法を用いることができますが，GUI から NUOPT を利用する場合， 1 の方法のみ用いることができます．

パラメータの中には， 2 の方法でしか利用できない種類のものも存在します．

パラメータ指定が競合した場合，パラメータファイル nuopt.prm から指定する方法の方が優先されます．

14.1 パラメータファイル nuopt.prm

パラメータファイルは nuopt.prm という名前でなければなりません．

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

```
begin
end
```

begin と end の間に各種パラメータの設定を行うことができます．次の例ではアルゴリズムとして単体法 simplex を指定しています．

```
begin
method:simplex
end
```

パラメータは : ではなく， = で指定する場合もあります．次の例では，停止条件を 10^{-12} に設定しています．

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

パラメータファイル中には半角スペースを適宜入れる事ができます．次の例は，上記の

例と同じ意味です.

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

パラメータファイルは，複数のパラメータを指定する事もできます．以下の例では，停止条件を 10^{-4} に設定し，アルゴリズムに線形計画専用内点法 higher を設定しています.

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

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

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

パラメータファイルで設定する値には整数や小数のほか,

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

のような浮動小数点表記が許されています.

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

パラメータファイル

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

標準出力

```
<reading parameter file: nuopt.prm>
    begin
    maximize
    method:trust
    scaling:on
    crit:eps=1.0e-8
    end
<reading MPS file: myprob.mps >
        ...
```

14.2 共通パラメータ

本節では，求解アルゴリズムに依存しないパラメータについて解説します.

14.2.1 アルゴリズムの選択

パラメータを用いてアルゴリズムを設定する事ができます.

アルゴリズム名	日本語名
simplex	単体法（＋分枝限定法）
asqp	有効制約法
higher	線形計画専用内点法
lipm	（新版）直線探索内点法
lepm	直線探索外点法
line	（旧版）直線探索内点法
bfgs	準ニュートン法
tipm	（新版）信頼領域内点法
tepm	信頼領域外点法
trust	（旧版）信頼領域内点法
lsqp	直線探索を利用した逐次二次計画法
tsqp	信頼領域を利用した逐次二次計画法
slpsqp	逐次線形二次計画法
lsdp	線形半正定値計画問題に対する主双対内点法
csdp	凸半正定値計画問題に対する主双対内点法
qnsdp	準ニュートン法を用いた非線形半正定値計画問題に対する主双対内点法

trsdp	信頼領域法を用いた非線形半正定値計画問題に対する主双対内点法
wcsp	制約充足問題ソルバ
rcpsp	資源制約付きスケジューリング問題ソルバ
global	大域的最適化アルゴリズム

モデルファイル内で指定する方法

```
options.method = "アルゴリズム名";
```

パラメータファイル nuopt.prm 内で指定する方法

```
method:アルゴリズム名
```

以下の例では，アルゴリズム simplex （単体法）をそれぞれモデルファイルから，あるいはパラメータファイルから設定しています.

モデルファイル内で指定する方法

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

パラメータファイル nuopt.prm 内で指定する方法

```
method:simplex
```

実行不可能性検出アルゴリズム iisDetect の設定解除には，別の設定方法が用意されています.

モデルファイル内で指定する方法

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

パラメータファイル nuopt.prm 内で指定する方法

```
param:iis=off
```

14.2.2 標準出力制御

デフォルト設定の NUOPT は，標準出力に求解情報を表示し，解ファイル（モデル名.sol）を作成します．パラメータを用いる事で，これらを抑制できます.

標準出力による求解情報の表示を抑制するには，次のように記述します.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
output:mode=silent
```

14.2.3 解ファイル出力制御

パラメータを用いて解ファイルの出力を抑制するには，次のように記述します.
モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
output:name=_NULL_
```

パラメータを用いて，出力される解ファイルのファイル名を変更することもできます.
モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
output:name=ファイル名
```

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

14.3 アルゴリズム固有のパラメータ

本節では特定のアルゴリズムを選んだ際にのみ有効なパラメータ値と，その設定について解説します.

14.3.1 線形計画問題専用内点法（higher）/直線探索法（lipm/lepm/line）/逐次二次計画法（lsqp/tsqp/slpsqp）/半正定値計画専用内点法（lsdp/csdp/qnsdp/trsdp）に有効なパラメータ

以下，線形計画問題専用内点法のみ有効なパラメータ値と，その設定について解説します.

◆ スケーリング

アルゴリズムを効率よく安定に動作させるため，目的関数，制約式，変数に定数値を乗じる処置がスケーリングです. このパラメータはそれを行うか否かを指定するものです.

モデルファイルに記述する方法

```
options.scaling = “on”;
```

パラメータファイル nuopt.prm に記述する方法

```
scaling:on
```

◆ 単体法へのクロスオーバー（線形計画専用内点法 higher のみ）

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

モデルファイルに記述する方法

```
options.crossover = “on”;
```

パラメータファイル nuopt.prm に記述する方法

```
cross:on
```

◆ 停止条件

停止条件として用いる最適性条件の残差です. 最適性条件の残差がこの値以下になったときに， 計算が収束したとみなして反復計算を終了します.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 反復回数上限

停止条件として用いる反復計算の回数の上限です. 反復回数のデフォルト値は 150 回となっています. 反復回数がこの回数を越えた場合には解が得られなかったとみ

なしてエラー(<<NUOPT 10>> IPM iteration limit exceeded.)を出力します. 逐次二次計画法(lsqp/tsqp/slpsqp)を用いた場合にはエラー(<<NUOPT 40>> SQP iteration limit exceeded.)を出力します.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
crit:maxitn=150
```

◆ 外点法の実行不可能性ペナルティー（外点法 lepm/tepm のみ）

問題が実行不可能に非常に近い場合，外点法は不等式制約を満たさない解を与える可能性があります(<<NUOPT 55>> exterior solution obtained.) . そのような場合には，パラメータ exrho をデフォルトよりも大きく設定すると解消する可能性があります．問題が実行可能な場合に，exrho が大きいと反復に時間がかかるケースがあります. 内点法に比べて外点法のパフォーマンスが悪い場合には exrho をあえて下げて実行してみるのも一つの方法です． exrho の初期設定値は 1.0e3 です.

モデルファイルに記述する方法

```
options.exrho = 1.0e3;
```

パラメータファイル nuopt.prm に記述する方法

```
param:exrho=1.0e3
```

14.3.2 単体法 (simplex)/有効制約法 (asqp)に有効なパラメータ

◆ スケーリング

アルゴリズムを効率よく安定に動作させるため，目的関数，制約式，変数に定数値を乗じる処置がスケーリングです. このパラメータはそれを行うか否かを指定するものです.

モデルファイルに記述する方法

```
options.scaling = “on”;
```

パラメータファイル nuopt.prm に記述する方法

```
scaling:on
```

◆ 双対変数の非零性判定に関するトレランス

計算の停止条件に用いる双対変数（シャドウプライス）の零判定値です（この値以下の解の改善可能性を無視します). told の初期値は 1.0e-6 です.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 解ベクトルの各要素の可能性判定に関するトレランス

計算の停止条件に用いる変数の上下限違反判定値です. この値以下の変数値に関する上下限違反を無視します. tolx の初期値は 1.0e-8 です.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

14.3.3 制約充足アルゴリズム(wcsp/rcpsp)に有効なパラメータ

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

◆ 反復回数上限

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 計算時間上限

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

します.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 制約式の重み設定

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

モデルファイル内

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

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

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

◆ 目的関数の重み設定

目的関数の重みはパラメータ defaultObjectiveWeight で指定します. defaultObjectiveWeight の値は，モデルファイル内でのみ設定できます. 次の例では，目的関数の重みに 5 を指定しています.

モデルファイル内

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

パラメータ defaultObjectiveWeight の初期設定値は 1 です.

◆ 目的関数の目標値設定（**wcsp のみ**）

目標値 target の値は，パラメータ defaultObjectiveTarget で指定することができます.

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

Objective の引数での target 値と，パラメータ defaultObjectiveTarget の値が競合した場合は， Objective の引数の値の方が優先されます．目標値 target の初期設定値は 0 です．このパラメータは rcpsp には無効です．

◆ 初期解生成時に使用する乱数発生の種

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

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

初期解生成時に使用する乱数発生の種の初期設定値は 1 です．

◆ 計算回数

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

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

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

◆ スレッド数の上限（**wcsp のみ**）

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

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

計算回数は wcspTryCount オプションで設定することができますが，バージョ 14 までは１つのスレッド（1 つのコア）しか使えなかったため，1 回の計算に 10 秒かかるような問題の場合，wcspTryCout×10(秒)の計算時間がかかります．
一方，本オプションに 2 を設定しますと，計算を 2 つのスレッドで同時に行いますので，wcspTryCount×10(秒) / 2(並列）と，約半分の時間になります．(ただし，実行 PC に指定スレッド数以上の CPU(コア) が搭載されている必要がございます)

本オプションのデフォルト値は 1 ですので，並列化を有効にする場合は 2 以上の値を設定してください．

◆ 制約充足フェーズにおける計算時間上限

ハードペナルティ及びセミハードペナルティが残っているフェーズの計算時間上限です．制約充足フェーズにおける計算時間上限の値を設定した場合，設定上限時間を越えてもハード・セミハードペナルティが残っている場合には計算が終了します．時間内にハード・セミハードペナルティを充足した場合には，時間をリセットしてから maxtim で設定した時間解の探索を行います．制約充足フェーズにおける計算時間上限は wcspPhaseOneMaxtime で指定することができます．

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

-1 を設定した場合，無制限と解釈されます．制約充足フェーズにおける計算時間上限の初期設定は -1 です．

◆ 解更新間隔計算時間上限

解が更新されてから指定した時間が経過すると計算を終了します．大規模問題では計算時間の設定は非常に難しいのですが，この機能を用いると簡単に有用な求解を行うことができます．解更新間隔計算時間上限は wcspPhaseTwoMaxInterval で指定することができます．

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

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

14.3.4 整数計画法(simplex/asqp)/ 大域的最適化（ global ）に有効なパラメータ

以下のパラメータは単体法 （ simplex ）/有効制約法 （ asqp ）/大域的最適化（ global ）を，整数変数を含む問題について適用した際，あるいは大域的最適化（ global ）を非線形計画問題に対して適用した場合にのみ有効です．

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

また，この分枝限定法というスキームの好ましからざる特徴として，最適解を発見して

最適性を証明するまでの時間が問題の規模のみならず，問題の性質に大きく左右されるということがあります．具体的には，数万変数の問題が数秒で解けてしまったり，一方では数百変数の問題を解くのに一時間以上を所要したりということがございます．

NUOPT は現在得られる最新の技法を組み込み，できるだけ多様な問題が安定にとけるようにパラメータをチューニングしておりますが，あらゆる性質の問題に対して常に良い設定を見出すことはできておりません．そのため，以下にご紹介するパラメータを適宜設定することによって，デフォルトの状態ではかなり時間を所要した問題をより効率よく解くことができることも十分にあり得ます．以下では効果的と思われる順にチューニングのためのパラメータをご紹介します．

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

- 切除平面法のパラメータ

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

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

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
branch:clevel=1
```

◆ ヒューリスティックサーチを調整するパラメータ

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

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 探索深さ

探索の深さを設定します. 1 とすると深さ優先探索を行います. 大きい値であるほど，発見的探索に近くなります.

p を大きめに設定すると実行可能解が見つかりにくく, 所要メモリが大きくなりがちです. そのような場合には p=1 (深さ優先探索) という設定が有効な場合があります.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

```
branch:p=10
```

◆ 足切り点設定用パラメータ

NUOPT は分枝限定法の探索中に，これまでで最良の目的関数値 f_{inc} を持つ実行可能解が求まると， f_{inc} を基に足切り点 f_{cut} を

$$f_{cut} = f_{inc} - \Delta_f \quad (\text{最小化問題})$$

$$f_{cut} = f_{inc} + \Delta_f \quad (\text{最大化問題})$$

と更新します. すなわち現在求まっている実行可能解から Δ_f 以上良い部分問題のみに探索を限ることになりますので， Δ_f をより大きく設定することにより，探索がより絞り込まれ，計算の手間を減らすことが期待できます.

ただし, その際には求まった解が最適であることは保証できません. その解から Δ_f 以上良い解がないことのみが保証されます.

Δf は，絶対値（addToCutoff）を設定する方法，下界値からの相対値（rel_addToCutoff）で設定する方法の二つがあります. デフォルト値は絶対値 1.0e-6，相対値 1.0e-4 です. 絶対値（addToCutoff）を設定する方法，LP 緩和解からの相対値（rel_addToCutoff）で設定する方法の二つがあります. デフォルト値は絶対値 1.0e-6 です. 相対値は 0 です.

絶対値 1.0e-6 という設定は,目的関数の 1.0e-6 程度のぶれは小さなものとして無視してよいとしたことに相当します.

相対値は例えば 1.0e-4 程度に設定するのが大多数の場合適切ですが，LP 緩和解が真の最適解と大きくへだたっている場合には不適切な動作を招くのでデフォルト値は 0 となっています．LP 緩和解が真の最適解に比べて同じ程度の大きさである場合には有用です.

非線形問題に対する大域的最適化（global）の場合には，絶対値で指定する場合のデフォルト値は 1.0e-5 となります．相対値の指定をすることはできません.

モデルファイルに記述する方法

```
options.addToCutoff = 1.0e-6;  // 絶対値で指定
options.rel_addToCutoff = 1.0e-4; // 相対値で指定（global では無効）
```

パラメータファイル nuopt.prm に記述する方法

```
branch:add=1.0e-6       * 絶対値
branch:reladd=1.0e-4    * 相対値（global では無効）
```

◆ 足切り点

足切り点設定用パラメータの箇所で説明した足切り点 f_{cut} の値そのものです．足きり点（cutoff）より悪い解しか与えないことが解った問題は探索の対象から外します．従って，適切に設定すると計算の無駄を省くことができます．この値を最小化問題の場合は小さく（最大化問題の場合は大きく）設定するほど，足切り条件は厳しく，探索の対象は狭くなり，計算の手間は減ります．厳しく設定しすぎると実行可能解がない(<<NUOPT16>> Infeasible MIP.)というエラーになりますので，注意が必要です．初期設定では何も設定されていません.

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 計算時間上限

計算時間の上限です．計算開始から秒単位の計算時間が計算時間上限 maxtim を

越えると，下記のエラーメッセージとともに現在までの最適解を出力して実行を終了します．(実行可能解が見つかっていない場合には LP(QP)　緩和解のみの出力となります．また，0 以下の値は設定していないのと同じ意味になります．)計算時間には，前処理や最初の緩和解を求める時間が含まれます．初期設定では計算時間上限なしを意味する -1 が設定されています

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

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 整数解個数上限

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

```
<<NUOPT 37>> B&B terminated with given # of feasible.sol.
```

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 最大メモリ量制限

NUOPT プロセスが使用することのできる最大メモリ量を制限するためのパラメータで，Mb 単位で指定します．例えば 1000 とすると 1GB を上限とすることを意味し，1GB を超えた場合に実行を停止します．

負の値を設定すると，システムで利用可能なメモリ量が残り -maxmem 以下になっ

たときに実行を停止します．メモリ上限によって実行が停止した場合には NUOPT43 エラーが，実行可能解が見つかっていない場合には NUOPT44 エラーが出力されます．この場合，現在までの最適解を出力して実行を終了します(実行可能解が見つかっていない場合には緩和解のみの出力となります)．

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

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

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

◆ 上下界値のギャップによる停止

上下界値のギャップが，指定した値を下回る場合に解の探索を停止します．
ただし，gap =(上界値 － 下界値)です． gap は目的関数の実際の値に依存することにご注意下さい．以下のエラーが出力されて止まります．
実行可能解が求まってはじめて上下界値のギャップは意味を持ちますので，このエラーで停止した場合には必ず実行可能解の出力が成されます．初期状態では，設定はなされていません．

```
<<NUOPT 45>> B&B gap reaches under the limit.
```

モデルファイルに記述する方法

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

パラメータファイル nuopt.prm に記述する方法

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

- スレッド数の上限

分枝限定法はマルチコア環境において並列処理を行うことにより，計算時間を短縮することができます．ユーザは分枝限定法が使用するスレッド数の上限を指定することができます．-1 を指定すると，NUOPT が内部で適切なスレッド数を設定します．0 あるいは 1 と設定した場合は，シングルスレッドの分枝限定法が動作します．分枝限定法の並列化を実行した場合，デフォルトの設定は 2 になります．

モデルファイルに記述する方法

```
        options.bbthreads = 2;
```

パラメータファイル nuopt.prm に記述する方法

```
        branch:threads=2
```

14.4 MPS ファイルに関する設定

この章では MPS ファイルの付加情報の指定方法を解説します. MPS ファイルに対する情報をパラメータで指定するには, パラメータファイル nuopt.prm を用いる方法のみが提供されています.

MPS ファイルから問題を入力する場合にのみ関係します.

- 最小化, 最大化の指定

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

```
        maximize
```

- 各種ラベル名

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

  デフォルトでは最初に現れたものとなります.

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

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

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

14.5 パラメータ一覧

NUOPT で設定可能なパラメータの一覧です.

名称	選択	Default	意味
outputMode	"silent", "normal",	"normal"	標準出力モード [output:mode = normal]
method	"auto, "lipm", "lepm", "line", "higher", "tipm", "tepm", "trust", "bfgs", "lbfgs", "simplex", "asqp", "lsqp", "tsqp" "slpsqp" "wcsp" "rcpsp"	"auto"	求解アルゴリズム [method:auto]
scaling	"off" "on"	"on"	スケーリングを行うか否か [scaling:on]
maxitn	int	150	内点法の反復回数の最大 [crit:maxitn = 150]
eps	double	自動設定	停止条件 [crit:eps = 1.0e-8]
clevel	0/1/2	1	導入される切除平面の数の目安（0は導入しない） (分枝限定法専用) [branch:clevel=1]
rounding	-1/0/1/2/3	-1	rouding戦略によるヒューリスティックサーチの頻度．-1 はシステムが適当に設定する． (分枝限定法専用) [branch:rounding=1]

feasPump	-1/0/1	-1	Feasibility Pump 戦略によるヒューリスティックサーチの頻度．-1 はシステムが適当に設定する． (分枝限定法専用) [branch:feas=0]
neighbourSearch	-1/0/1	-1	Neighbor search 戦略によヒューリスティックサーチの頻度．-1 はシステムが適当に設定する． (分枝限定法専用) [branch:neigh=1]
addToCutoff	double	1.0e-6 （global 以外） 1.0e-5 (global)	足切り点設定用パラメータ（絶対値） (分枝限定法専用) [branch:addtocutoff=1.0e-6]
rel_addToCutoff	double	-1 (0 と等価)	足切り点設定用パラメータ（緩和解からの相対値） (分枝限定法専用，global には無効) [branch:reladd=1.0e-4]
cutoff	double	未定義	足切り点 (分枝限定法専用) [branch:cutoff = 1.8]
p	int	10	探索深さ (分枝限定法専用) [branch:p = 10]
maxnod	int	-1（無制限）	探索問題数上限 (分枝限定法専用) [branch:maxnod=100000]
maxtim	int	-1（無制限）	計算時間上限(秒) (分枝限定法と内点法全般) [branch:maxtim=3600]
maxmem	int	-10(残り 10Mb)	分枝限定法のメモリ利用量上限(Mb)，残り利用可能メモリによる制限（負値の場合，Mb) [branch:maxmem=500]

bbthreads	int	2	並列化分枝限定法のスレッド数の上限
gaptol	double	-1（指定なし）	上下界ギャップの下限（この値を下回ったら停止）
tolx	double	1.0e-8	主問題の実行不可能性判定値（単体法のみ） [param:tolx=1.0e-8]
told	Double	1.0e-6	双対問題の実行不可能性判定値（単体法のみ） [param:told=1.0e-6]
maxintsol	int	-1（無制限）	整数解取得個数上限（秒） [branch:maxintsol=3]
iisDetect	"off" "on"	"on"（行う）	実行不可能な行集合（IIS）探索を行う/行わない
outfilename	char*	0(未定義)	NUOPT の解ファイル名 "_NULL_"とすると出力を行わない [output:name=myout]
noDefaultSolve	int	0	solve()を陽に呼ばないと求解を行わない
noDefaultSolout	int	0	Solout()を陽に呼ばないと解の出力を行わない
outputParameter outputSet outputElement outputExpression	int	0 0 0 1	Parameter,Set,Element,Expression の CSV ファイル出力を行うかどうか
multDataPolicy	int	0（許さない）	同一のデータについてデータを重複して与えることを許すかどうか（１に設定すると警告扱いとなる）
defaultConstraintWeight	double	-1	指定のない制約式の重み(デフォルトはハード制約)
defaultObjectiveWeight	double	1	目的関数を変形した制約式の重み(デフォルトは重み１のソフト制約)
defaultObjectiveTarget	double	0	目的関数の目標値(デフォルトは０)（wcsp のみ）

15. MPS ファイル

NUOPT は MPS ファイル形式で記述された数理計画問題を解くことができます. 具体的に, MPS ファイル foo.mps に記述された数理計画問題を解くには, 以下のようにします.

```
prompt% nuopt foo.mps
```

MPS ファイル形式の数理計画問題は, コマンドラインからのみ扱うことが可能です. GUI から MPS ファイルを扱うことはできません.

15.1 MPS ファイルに対する標準出力

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

```
prompt% nuopt ex1.mps
NUOPT 6.0.0, Copyright (C) 1991-2001 Mathematical Systems Inc.
<reading MPS file: ex1.mps >
PROBLEM_NAME(TITLE)                              example1
ROWS                                                4
COLUMNS                                             3
NONZEROS                                           11
OBJECTIVE                                           f
RHS                                                 b
NUMBER_OF_VARIABLES                                    3
NUMBER_OF_FUNCTIONS                                    4
PROBLEM_TYPE                                MINIMIZATION
METHOD                                     HIGHER_ORDER
<preprocess begin>.....<preprocess end>
<iteration begin>
   res=2.4e+001 .... 2.7e-005  1.4e-008
<iteration end>
STATUS                                          OPTIMAL
VALUE_OF_OBJECTIVE                                -10.5
ITERATION_COUNT                                       6
FUNC_EVAL_COUNT                                       9
FACTORIZATION_COUNT                                   7
RESIDUAL                                1.354127444e-008
ELAPSED_TIME(sec.)                                 0.01
SOLUTION_FILE                                    ex1.sol
```

```
<reading MPS file: ex1.mps >
PROBLEM_NAME(TITLE)                                    example1
ROWS                                                          4
COLUMNS                                                       3
NONZEROS                                                     11
OBJECTIVE                                                     f
RHS                                                           b
```

はMPSファイルを読み込むインタフェース部分からのメッセージで，MPSファイルのNAMEセクションにあるタイトルexample1，ROWSセクションで指定された行の数(4)，COLUMNSセクションで指定された変数の数(3)と総非零要素数(11)，目的関数の行の名前(F)と右辺ラベル(B)を示しています．

以降の標準出力は SIMPLE モデルに対して NUOPT を適用した場合と同じです．詳しくは11 標準出力をご覧下さい．

15.2 MPS ファイルに対する解ファイル

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

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

解ファイルの冒頭部分には，標準出力と同じ内容が記載されます．

以下， MPS ファイルに特有の出力部分に関して説明します．

BOUNDS,RANGESという行はMPSファイル中にBOUNDS，RANGESセクションが存在しない場合には出力されません．存在する場合，計算に使用されたラベル名が例えば以下のように出力されます．

```
BOUNDS                                                 BND1 FR(2)
RANGE                                                  RNG1
```

BOUNDS ラベル名の後(上記下線部)には設定された BOUNDS レコードの種類の内分けと数が以下のように表示されます.

```
UP(21)/FX(3)                  #    上限 21 個/固定 3 個
UP(131)/FX(16)/FR(14)         #    上限 131 個/固定 16 個/自由 14 個
UP(65)/LO(64)/FX(18)/FR(1) #    上限 65 個/下限 64 個/固定 18 個/自由 1 個
```

NONZEROS_IN_HESSIAN, INIT_VALUE_GIVEN_COLUMNS も同様に，存在する場合にのみ以下のように表示されます.

```
NONZEROS_IN_HESSIAN                                          4
INIT_VALUE_GIVEN_COLUMNS                                     2
```

COLUMNS セクション中 MARKER 行によって，整数変数の指定を行った場合には，以下のように「整数変数である」と指定された数が表示されます.

```
   COLUMNS                                                2 INT(2)
```

解ファイルのこの行以降の書式は共通です．詳細は解ファイルの章をご覧下さい.

15.3 MPS ファイルに対するパラメータ設定

MPS ファイルに対しても，パラメータを用いて各種設定をすることができます． MPS ファイルに対する情報をパラメータで指定するには，パラメータファイル nuopt.prm を用いる方法のみが提供されています． MPS ファイルに特有のパラメータとして，以下が提供されています.

- 最小化，最大化の指定

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

```
      maximize
```

- 各種ラベル名

  これらは MPS ファイル中に複数の RHS/BOUNDS/RANGE/目的関数行があるとき，実際の計算で用いるものを指定します.
  デフォルトでは最初に現れたものとなります.

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

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

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

15.4 MPS ファイルの具体例

MPS ファイルは一般形の線形/二次計画問題を記述するためのものです.

一般の線形/二次計画問題は

最小／最大化　　$f(x)$

条　件　$c_{L_i} \le g_i(x) \le c_{U_i}, \quad i = 1, \cdots, m$

$$b_{L_j} \le x_j \le b_{U_j}, \quad j = 1, \cdots, n$$

初期値　　$x_j = x_j^{\,0} \qquad j = 1, \cdots, n$

ここで $f(x)$，$g_i(x)$ は二次関数で

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

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

と表されます.

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

目的関数，制約式の線形部分の係数　　c_i，a_{ij}

制約式の上下限　　c_{L_i}, c_{U_i}

目的関数，制約式の Hessian の要素　　H_0，H_i

変数の上下限　　b_{L_j}, b_{U_j}

変数の初期値 x_i^0

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

最小化	$4x_1 - x_3 + x_2^2$	
条件	$x_1 + x_4$	$= 4$
	$x_1 + 2x_2 - x_3 + x_1x_2$	≤ 10
	$x_2 + x_3$	≥ 2
	$x_1 \geq 3$	
	$4 \geq x_2 \geq 1$	
	$x_3, x_4 \geq 0$	
初期値	$x_1^0 = 4,\ \ x_2^0 = 2$	
	$x_3^0, x_4^0 = 0$	

次は上記の二次計画問題を記述した MPS ファイルの例です．

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

15.5 MPS ファイルへの変換

15.5.1 MPS ファイルへの変換方法

モデリング言語 SIMPLE を用いてモデルファイルを記述した後， mpsout 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[5]．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．

[5] この機能は Visual Studio 系のコンパイラを使用されている場合には，正常に動作しない可能性がございます．

```
mpsout(); // MPS 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
```

15.5.2 変換機能使用時の注意

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

- 変数名，関数名

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

```
VARIABLE NAME (MPSFILE - original)

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

FUNCTION NAME TABLE (MPSFILE - original)

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

- 最大化/最小化

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

- 問題規模の制限

  変数，関数のいずれかが 9999999 個以上の問題は出力できません．

特殊な数理計画問題

本章では，記述する際の難度が高い，特殊な数理計画問題の定式化を扱います.

16． 0-1 変数の高度な利用法

0-1 変数をうまく利用する事で，そのままでは記述することが難しい目的関数や制約式を表現できます．ここで紹介する例は

- 折れ線関数の表現
- 整数変数の同符号条件の表現

の二通りですが，その他にも様々な応用があります．このような用途で用いられる 0-1 変数を indicator 変数と呼びます.

16.1 折れ線関数の表現

折れ線関数 $y=\begin{cases}-x, x\le 0\\ x, x\ge 0\end{cases}$ を表現する事を考えます． ifelse 関数を用いて折れ線関数を表現することはできませんが， 0-1 変数 d を次のように導入する事で，折れ線関数を表現できます.

```
Variable x,y;
IntegerVariable d(type=binary);
Parameter M;
M = 10000; // 非常に大きな数
-M*(1-d) <= x <= M*d;
x - M*(1-d) <= y <= x + M*(1-d);
-x - M*d <= y <= -x + M*d;
```

x に関する上下限制約は $d=0\Rightarrow -M\le x\le 0$， $d=1\Rightarrow 0\le x\le M$ であることを意味しています． M として十分大きな数を取ると $d=0\Rightarrow x\le 0$，$d=1\Rightarrow 0\le x$ と等価です.

y に関する一つ目の上下限制約は $d=0\Rightarrow x-M\le y\le x+M$， $d=1\Rightarrow x\le y\le x$ を意味します．つまり，M として十分大きな数を取ると $d=1\Rightarrow y=x$ という意味になります．同様に考えると，二番目の上下限制約は $d=0\Rightarrow y=-x$ という意味になります.

16.2 整数変数の同符号条件の表現

二つの整数変数 x， y が同符号であるという条件は， $xy\ge 0$ という条件と同値ですが，整数変数の積を記述すると，式が線形ではなくなってしまいます．ここでは線形の範囲内で，この条件を取り扱う方法を示します.

```
IntegerVariable x,y;
IntegerVariable d(type=binary);
Parameter M;
M = 10000; // 非常に大きな数
- M*d <= x <= M*(1-d);
- M*d <= y <= M*(1-d);
```

M として十分大きな数を取ると $d=0 \Rightarrow 0 \leq x$, $d=1 \Rightarrow x \leq 0$ そして $d=0 \Rightarrow 0 \leq y$, $d=1 \Rightarrow y \leq 0$ となるので, 二つの整数変数 x, y は同符号となります.

17. NUOPT/SIMPLE FAQ

本章では，ある程度 NUOPT 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，「 NUOPT/SIMPLE チュートリアル」をご参照下さい．

17.1 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
```

17.2 整数の割り算

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;
```

17.3 添字付けに関するエラー

17.3.1 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;
```

17.3.2 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
```

17.3.3 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
```

付録 A　NUOPT/SIMPLE のエラーメッセージ

A.1 SIMPLE のエラーメッセージ

次表は SIMPLE の実行時のエラー一覧です．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 . <<SIMPLE 1>> 変数 XX について矛盾した上下限が与えられました.	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).
2	<<SIMPLE 2>>　関数 MINIMIZE/MAXIMIZE の中には、上下限を使用することができません.	
3	<<SIMPLE 3>> Infeasible bound for constraint XX . <<SIMPLE 3>>　制約 XX について矛盾した上下限が与えられました	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).
4	<<SIMPLE 4>> Warning: Length of a subscript exceeds 30. <<SIMPLE 4>> 警告: 添字の名前の長さが 30 文字を越えました.	警告: 添字として使われる文字列の文字数が 30 を越えています. (30 文字目以後の文字は無視して処理します.)
5	<<SIMPLE 5>> Subscript has the wrong dimension. <<SIMPLE 5>> 添字の次元が合致しません.	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下さい.)

6	<<SIMPLE 6>> Illegal characters ("[", "]", "..." and so on)included in subscript. <<SIMPLE 6>> 添字の中に、無効な文字 ("[", "]", "..." など) が含まれています.	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>> Subscript expected on rhs assignment. <<SIMPLE 10>> データファイル XX の YY 行目に記述エラーです. ZZ 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.
11	<<SIMPLE 11>> Data file: Value expected on rhs of assignment. <<SIMPLE 11>> データファイル XX の YY 行目に記述エラーです. オブジェクト ZZ の右辺で WW 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.
12	<<SIMPLE 12>> 〜<<SIMPLE 17>>Internal Error! <<SIMPLE 12>> 〜<<SIMPLE 17>> システム内部エラー!	SIMPLE の内部エラー. (nuopt-support@msi.co.jp へお知らせ下さい.)
19	<<SIMPLE 19>> Warning: No auto-assignment performed for constant set. <<SIMPLE 19>> 警告: 自動代入によって定義されない集合があります.	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. (警告) (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>> Data file: Can't match the pattern "from... to". <<SIMPLE 20>> データ読み込み: "from ... to" 形式の文法エラーです.	データファイルに from/to の省略記号 "..." が現れましたが, from/to がペアーになっていません.

22	<<SIMPLE 22>> Data file: Number of values not matched with the index dimension of set. <<SIMPLE 22>> データ読み込み: データ中の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 の数と合っていません.
23	<<SIMPLE 23>> Expand from-to: "..." or ".." appeared at the head or the tail of the stream. <<SIMPLE 23>> データ読み込み: "..." 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ました.	空集合から最大要素を求めようとしています.
26	<<SIMPLE 26>> Only one-dimension data can be specified as interval number. <<SIMPLE 26>> 1次元のデータに限って, interval の指定は有効です.	範囲 Interval の定義(宣言) 時に dim=1 以外を与えました.
30	<<SIMPLE 30>> Interval data not matched with the format: lower bound, upper bound. <<SIMPLE 30>> Interval の入力データ形式は " 下限, 上限" でなければなりません.	範囲 Interval の定義(宣言) 時に下限, 上限順を与えていません.
33	<<SIMPLE 33>> A call has been made to an invalid function on Interval. <<SIMPLE 33>> Interval に対して(要素取り出しなど) 無効な関数呼び出しが行なわれました.	Interval に対して無効な関数を実行しようとしました.

34	<<SIMPLE 34>> Warning: No auto-assignment performed for interval set. <<SIMPLE 34>> 警告: Interval に対しては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 Interval なので行われません. (警告)
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-dimension 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 データ形式は " 開始要素 .. 終了要素, 増量"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>> Sequence: Illegal operation. <<SIMPLE 40>> Sequence に対する無効な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>> 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ました.	空列から最大要素を求めようとしました.
47	<<SIMPLE 47>> Warning: No auto-assignment performed for sequence. <<SIMPLE 47>>警告: Sequence 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 Sequence なので行われません. (警告)

49	<<SIMPLE 49>> A call has been made to an invalid function. <<SIMPLE 49>> 無効な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 の間に演算が行われました.
58	<<SIMPLE 58>> Set difference: s1-s2: s1 doesn't include s2. <<SIMPLE 58>> 集合引算: 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>> Fiexed value (...) out of defined range (...). <<SIMPLE 59>>要素(...)に対して定義域外の値 (...) を代入しようとしました.	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>> Illegal characters included in subscript. <<SIMPLE 60>> 添字に無効な文字が含まれています.	添字に無効な文字が含まれています.
62	<<SIMPLE 62>> Comparison between elements of different dimemsion. <<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): 添字が合致しません.	制約式の添字エラーです.

67	<<SIMPLE 67>> Index error in reference of XX. <<SIMPLE 67>> 参照オブジェクト XX の添字付けに誤りがあります． ［引き続き，例えば次のようなメッセージが出力されます.］ スカラなのに，次元 XX 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．具体的にどのケースかこれに続いてメッセージが出力されます．
70	<<SIMPLE 70>> Unmatched or ambiguous element(s). <<SIMPLE 70>> 合致しないか， 不定の要素がありました．	添字の数が合っていません． （ 代入の左辺と右辺で現れる添字が違う場合等に生じます． ）
74	<<SIMPLE 74>> Improper use of a dependent subscript. <<SIMPLE 74>> 他の添字に依存する添字はその依存している添字と一緒に現われなければなりません．	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 Element j(set=S[i]); として宣言された Element が i を伴わず， 固定されずに使われた場合に生じます． ）
75	<<SIMPLE 75>> Index dimension error in assignment to object XX , scalar but with index. <<SIMPLE 75>> 代入の対象オブジェクト XX の添字付けに誤りがあります．スカラ(添字なしで宣言)ですが添字付けして値を設定しようとしています．	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>> Index dimension error in assignment to object XX, Defined on index set N but try to assign a value with index of dimension M. <<SIMPLE 76>> 代入の対象オブジェクト XX (定義集合の次元: N) の添字付けに誤りがあります.添字の次元が M です. ［あるいは］ 代入の対象オブジェクト XX の添字付けに誤りがあります. (Element の宣言の際に添字付きの集合の添字付けを忘れている可能性があります.)	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 dimension error in assignment to object XX, Defined on index set with N but try to assign a scalar value. <<SIMPLE 77>>代入の対象オブジェクト XX (定義集合の次元: N) の添字付けに誤りがあります.添字付けせずに値を設定しようとしています.	XX は N 個の添字を付けて定義されていますが, スカラ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>> Try to add condition (described below, ignored) to scalar expression [...]. (condition) <<SIMPLE 78>> 警告: 添字付けられていない式[...]に次の条件を付加していますが, 無視されます. (条件式)	条件式が, 添字付けられていない Expression の代入や定義に現れています.

79	<<SIMPLE 79>> The following condition never satisfied (Element dimension: n, Set dimension m). <<SIMPLE 79>警告: n 次元の添字と m 次元の要素の集合について以下の条件式が現れましたが，充足されることはありません.	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.
81	<<SIMPLE 81>> 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 ... out of range. <<SIMPLE 82>> ... の添字が定義域外である... 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ました.
91	<<SIMPLE 91>> Operation between set 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.
92	<<SIMPLE 92>> Warning: No auto-assignment performed for ... <<SIMPLE 92>> 警告: 演算結果から生成された以下の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，自動追加先が集合演算の結果(和集合) であるので行われません. (警告)
97	<<SIMPLE 97>> Warning: No auto-assignment performed for sets made by setOf. <<SIMPLE 97>>警告: "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. (警告)
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 of lhs. and rhs. conflicts. <<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ません.
103	<<SIMPLE 103>> Dimension not matched when using superSet. <<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ました. (Set T(dim=2); Set S(dim=1,superSet=T); とした場合等.)
104	<<SIMPLE 104>> ...(1) can not be used as a superset of ...(2) <<SIMPLE 104>> ...(1) は...(2) の superSet として使用することができません.	"...(1)" は"...(2)" の親集合になれません.
105	<<SIMPLE 105>> Argument error: ...(1) of ...(2) <<SIMPLE 105>>引数エラー: ...(1) の...(2)	オブジェクト定義時の引数エラーです.
106	<<SIMPLE 106>> Argument arcs must be a 2-dimensional set. <<SIMPLE 106>>引数 "arcs" は 2 次元の集合でなければなりません.	グラフ定義時に，属性 arcs が dim=2 の集合になっていません.
107	<<SIMPLE 107>> Argument nodes must be a 1-dimensional set. <<SIMPLE 107>>引数 "nodes" は 1 次元の集合でなければなりません.	グラフ定義時に， 属性 nodes が dim=1 の集合になっていません.
108	<<SIMPLE 108>> ERROR in check(): ... condition is not satisfied. <<SIMPLE 108>>関数 check() の中の条件 ... が満足されませんでした.	関数 check() の条件に違反しました.
110	<<SIMPLE 110>> Argument: invalid 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.
113	<<SIMPLE 113>> Invalid assignment. <<SIMPLE 113>> 無効な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: Invalid 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 set が無効に使われました.
116,117	<<SIMPLE 116>>,<<SIMPLE 117>> "set=" が誤った場所に使用されています.	
118	<<SIMPLE 118>> Set に対して無効な代入が行われました(マニュアルをご覧下さい) .	
119	<<SIMPLE 119>> Assignment: "[" or "]" occured 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, 文字列の中にも "[" 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>> Invalid 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.

122	<<SIMPLE 122>> Data file: = expected. <<SIMPLE 122>> データファイル...(1) の XX 行目に記述エラーです. オブジェクト名 ...(2) の右辺に等号 "=" がありません.	データファイル中, = が現れるはずのところにその他の文字が現れました. （ データファイルで a[1] = 3; a[2]= 5; を表現する場合には a= [1] 3 [2] 5 ; と記述します. 代入の左辺には[] 等の文字列は現れてはいけません. ）
123	<<SIMPLE 123>> Data file: Syntax error. <<SIMPLE 123>> データファイル...(1) の XX 行目に記述エラーです. オブジェクト ...(2) の定義の付近です. ［あるいは］モデル中の文字列の記述エラーです. ［引き続き，次のメッセージが出力されます.］文法エラーが起きました.	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";" を忘れたりした場合に生じます. ）
125	<<SIMPLE 125>> Remove/Restore Constraint: Constraint number out of range. <<SIMPLE 125>> 制約式に対する Remove/Restore 関数で: 指定された制約式の番号が存在しません.	制約式番号で削除/復帰させる制約式を指定する際にその番号が正しくありません.
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" がペアーになっていません.
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が含まれています.
130	<<SIMPLE 130>> Invalid 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 int 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>> Invalid cast from character to double. <<SIMPLE 131>> 文字から double へのキャストが行なわれました.	演算等で, 文字を数字 double へキャストする必要が生じましたが, 失敗しました.
135	<<SIMPLE 135>> Syntax error occured within [...] function. <<SIMPLE 135>> データ書式の[...] の中に文法エラーが発生しました.	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 occured within <...> function. <<SIMPLE 136>> データ書式の<...> の中に文法エラーが発生しました.	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.
140	<<SIMPLE 140>> Fixed value out of range for variable. <<SIMPLE 140>> Variable が上下限の範囲外に固定しようとしました.	変数の現在の値がその変数の上下限の範囲外にあるので, 変数を固定することができません.
142	<<SIMPLE 142>> Constraint object required in function call. <<SIMPLE 142>> 制約式がインスタンスでなければなりません.	制約式をシステムから削除／復帰する関数（deleteCo(), restoreCo()) の引数として宣言されている Constraint のインスタンス以外が渡されました. (deleteCo()/restoreCo() に渡せるのは Constraint と宣言されたオブジェクトのみです. 例えば, deleteCo(x[i]+y[i]>=6) などはエラーとなります.)
148	<<SIMPLE 148>> Argument: invalid 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.
150	<<SIMPLE 150>> Set Assignment: lhs of assignment must be an set instance. <<SIMPLE 150>> 演算結果などの代入できない集合が代入の左辺に現れています.	集合に対する代入の左辺として, 事前に定義した集合のインスタンス以外が使われました. (集合に対する代入の左側に来ることができるのは, 宣言されたオブジェクトのみです. 例えば, ST = "1 2 3"– などは違法となります.)

151	<<SIMPLE 151>> Set Auto-assignment failed. <<SIMPLE 151>> 集合に対する自動代入が失敗しました.	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. ）
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.
162	<<SIMPLE 162>> Invalid assignment to current, init or dual member. <<SIMPLE 162>> 値(.val)、 初期値(.init) または双対変数値(.dual) に対する代入が行なわれました(参照のみが可能です).	オブジェクトの参照可能な値（y[i].val, x.init, z.dual など）に対する代入を行おうとしました. （これらは通常の Parameter と同じに扱われますが，代入を行うことはできません. ）
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ました.
164	<<SIMPLE 164>> No init value for empty constraint. <<SIMPLE 164>>空の制約式... に関連する値は 0 として出力されます.	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ました.
165	<<SIMPLE 165>> Dual value of constraint XX is assumed zero Constraint is empty or model is not solved. <<SIMPLE 165>>制約式 XX 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.	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（警告です）.

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.	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]<=y[i]<=z[i] 等）を行った場合，その参照値の出力をすると最も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（ 8.3 オブジェクトに関連する値の参照を参照して下さい. ）
168	<<SIMPLE 168>> Objective can only be assigned for once. <<SIMPLE 168>>目的関数 (Objective)に対する代入は一度のみ可能です.	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)
169	<<SIMPLE 169>> Argument: type Error! <<SIMPLE 169>> "type=" が誤って使われました.	オブジェクト定義時の属性引数 type の指定が無効である.
171	<<SIMPLE 171>> Invalid assignment to Objective: <<SIMPLE 171>> 目的関数 (Objective) に対する無効な代入が行われました: "Objective = Expression" のみ有効です.	目的関数に対して変数を含まない式が代入された.
172	<<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.	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.
173	<<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ました.	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.

174	<<SIMPLE 174>> Set data can not be transformed to Parameter by .val . <<SIMPLE 174>>集合の値(.val) をパラメータ(Parameter)に変換しようとしました.	集合の現状値 val を定数 Parameter に変換しようとしました. (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. 表示したりダンプするのみです.)
177	<<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.	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ました.
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.
183	<<SIMPLE 183>> Argument: Either left or oleft is required in defining an Interval. <<SIMPLE 183>> Interval を定義する時には, "left=" か"oleft=" が必須です.	範囲 Interval の定義に属性引数 left または oleft の指定が現れていません.
184	<<SIMPLE 184>> Argument: Either right or oright is required in defining an Interval. <<SIMPLE 184>> Interval を定義する時には, "right=" か"oright=" が必須です.	範囲 Interval の定義に属性引数 right または oright の指定が現れていません.

185	<<SIMPLE 185>> Argument: name is ignored in the Interval/Sequence definition. <<SIMPLE 185>> Interval/Sequence を定義する時には，"name=" は無効です.	範囲 Interval または列 Sequence を定義するとき，属性引数 name が現れました．(警告です．無視して実行します．)
186	<<SIMPLE 186>> Argument: index is ignored in the Interval/Sequence definition. <<SIMPLE 186>> Interval/Sequence を定義する時に，"index=" は無効です.	範囲 Interval または列 Sequence の定義に属性引数 index が現れました．(警告です．無視して実行します．)
187	<<SIMPLE 187>> Argument: dim is ignored in the Interval/Sequence definition. <<SIMPLE 187>> Interval/Sequence を定義する時に，"dim=" は無効です.	範囲 Interval または列 Sequence の定義に属性引数 dim が現れました． (警告です． 無視して実行します．)
188	<<SIMPLE 188>> Argument: left, right, oleft, oright are ignored in <<SIMPLE 188>> Sequence を定義する時には、"left=, right=, oleft=, oright="は無効です.	列 Sequence を定義するとき， 属性引数 left などは無視されます.
189	<<SIMPLE 189>> Argument: from, to are ignored in the Interval definition. <<SIMPLE 189>> Interval を定義する時に，"from=, to=" は無効です.	範囲 Interval を定義するとき， 属性引数 from と to は無視されます.
190	<<SIMPLE 190>> Interval has no val member. <<SIMPLE 190>> Interval の".val" を参照しようとしました.	範囲 Interval に対して現状値 val を照会しました．(Interval に val を参照することはできません．)

191	<<SIMPLE 191>> No assignment is allowed to Sets having <<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>> Fatal Error in solve () before solution process <<SIMPLE 192>> NUOPT の重大エラー(求解処理の前)	NUOPT が前処理でエラーを起こしました．
193,194	<<SIMPLE 193>>, <<SIMPLE 194>> Error in solve (), Fatal Error in solve() NUOPT のエラー:XX，NUOPT の重大エラー：XX	NUOPT が計算途中にエラーを起こしました（193:解出力あり，194:解出力なし)．
195	<<SIMPLE 195>> Index with SuperSet error. <<SIMPLE 195>> Index と SuperSet の定義エラー．	Superset と添字つき集合の添え字付けに矛盾があります．
196	<<SIMPLE 196>>Warning: Objective function is constant. <<SIMPLE 196>> 警告: 目的関数がコンスタントとなっています．	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．
197	<<SIMPLE 197>> Set of fixed element can not be a superset. <<SIMPLE 197>>固定された要素によって定義される集合を superSet 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 superSet として使おうとしました．
198	<<SIMPLE 198>>Wrong argument for slice function. <<SIMPLE 198>> 関数 slice の引数が正しくありません．	Slice 関数の引数は slice 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 (..) appeared in constraint/objective definition. <<SIMPLE 199>> 文字列値(..)が制約式や目的関数の定義に現れています．	文字列 .. 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>> 警告: simple_fprintf () は引数並び中の Set オブジェクトを無視しました	simple_printf は Set の出力を行いません.
202	<<SIMPLE 202>> {...} appears invalid position. <<SIMPLE 202>> {...} が不正な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 <<SIMPL 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 unlock() しようとしました.	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 unlock() を呼びました.
205	<<SIMPLE 205>>Warning: Any Set without name is already locked. <<SIMPLE 205>> 警告: 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.	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ました.
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ました.	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>> 警告: XX の添字集合 (lock 済)への自動代入は抑制されました.	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（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警告出力）.
209	<<SIMPLE 209>> Note: Skip assignment to locked superSet XX <<SIMPLE 209>> 警告: XX の superSet (lock 済) への代入は抑制されました.	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ました（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警告出力）.
211	<<SIMPLE 211>> User Termination(at Data Input) <<SIMPLE 211>> ユーザによる中断(データ読み込み)	データの読み込みの際にユーザによる中止が命令されました.
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>> NUOPT より警告:XX	NUOPT 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).	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.
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 Error XX <<SIMPLE 217>> 内部エラー XX	内部エラー XX です
218	<<SIMPLE 218>> Error in CSV file XX. The header column have N fields but line M has P fields. <<SIMPLE 218>> = CSV ファイル XX のエラーです．ヘッダー行には N 個のフィールドがありますが、M 番目の行は p 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 XX 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>> Error in CSV file XX. Only the header row and no data row exist. <<SIMPLE 219>>CSV ファイル XX のエラーです．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>> Error in CSV file XX. No data row exist. <<SIMPLE 220>> CSV ファイル XX のエラーです．有効行がまったくありません．	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>> Error in CSV file XX. Doublicate name YY (field # N, and M) in header row. <<SIMPLE 221>> CSV ファイル XX のエラーです．名前 YY がヘッダー行で重複しています．(フィールド番号 N と M 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>> Error in CSV file XX. Invalid field string XX at line# N . <<SIMPLE 222>> CSV ファイル XX のエラーです．フィールドデータとして不適切な文字 XX が N 行目に現れています．	データファイルとして与えられた CSV ファイルに違法な文字が混ざっています．
223	<<SIMPLE 223>> Error in CSV file XX. Empty field at line# N, field# M. <<SIMPLE 223>> CSV ファイル XX のエラーです．N 行目の，フィールド M 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 N lines or 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，このファイルには無駄な行があります合計 (N)行です.	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ー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が，それ以上の行があります.
225	<<SIMPLE 225>> Try to read data YY (with N index), from CSV file XX but it has too few (M) preceding column(s). <<SIMPLE 225>> データ YY(添字の数 N) を CSV ファイル XX 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).
226	<<SIMPLE 226>> Try to read data YY (with N index, [1 or 2]D format) and ZZ (with M index, [1 or 2] D format) from the same CSV file XX (This file looks ambiguous) <<SIMPLE 226>> データ YY(添字の数 N, [1 or 2]D 書式)と ZZ(添字の数 M, [1 or 2]D 書式) が両方、同じ CSV file XX から読みとれてしまいます(記述が愛昧です).	データファイルとして与えられた CSV ファイルの記述が曖昧な場合のエラーです（1D 書式か 2D 書式か判然としません).
227	<<SIMPLE 227>> Multiple data entry XX found in data YY [and ZZ]. <<SIMPLE 227>> データ XX についての記述がファイル YY と ZZ に複数みつかりました.	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# NN of the first row (index) is empty. In reading data XX (with YY index, 2D format) from CSV file ZZ. <<SIMPLE 228>> データ XX （添え字の数 YY）を CSV file ZZ から読み込もうとしましたが，最初の行のフィールド NN （添え字として使われる）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が，フィールド NN が空になっています.
229	<<SIMPLE 229>> Error from asDouble(): this object is not scalar. <<SIMPLE 229>> asDouble() のコールを行ったオブジェクトはスカラではありません.	スカラーでないオブジェクトに asDouble()（doubl 値への変換）のコールを行いました.
230	<<SIMPLE 230>> Error in datafile FF at line NN In the RHS of xx, dimension of element [YY] should be same as others. <<SIMPLE 230>> データファイル FF の NN 行目の記述エラーです．オブジェクト XX の右辺に現れる添字 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 severe performance deterioration. <<SIMPLE 231>>同一のデータ定義が二つ以上あり，大規模データでは深刻なパフォーマンス下落を招く可能性があります.	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.

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	モデルの展開，最適化計算の実行時に浮動小数点エラーが発生しました（発生箇所を伝える付加的なメッセージとともに表示されます).
234	<<SIMPLE 234>> 値 is not in the domain of DiscreteVariable. The domain: {DiscreteVariable の domain の集合} <<SIMPLE 234>> 値は DiscreteVariable の値として不適切です（以下のいずれかである必要があります)：{DiscreteVariable の domain の集合}	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. 例えば x の domain が｛a,b,c｝のとき, Boolean (x == "d") .. のように書いた場合.
235	<<SIMPLE 235>> DiscreteVariable 変数名 's domain is invalid (should contain only positive integer or string). domain:{ DiscreteVariable の domain の集合} <<SIMPLE235>> DiscreteVariable 変数名の domain が不適切です. （文字列または非負の整数を要素とする集合である必要があります). domain の内容：{ DiscreteVariable の domain の集合}	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 変数名 's domain has no element. <<SIMPLE 236>> DiscreteVariable 変数名の domain が空集合です.	

237	<<SIMPLE 237>> = is inappropriately used. to define equality constraint, use == instead of = . <<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.	x+y = z; 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.
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 変数名 has no doomain. <<SIMPLE 239>> DiscreteVariable 変数名 の domain が定義されていません.	
240	<<SIMPLE 240>> DiscreteVariable 変数名's domain should not be indexed. <<SIMPLE 240>> DiscreteVariable 変数名 の domain が添え字付けられています.	
241	<<SIMPLE 241>>Table/Parameter Table 名が 4 つ以上の DiscreteVariable で添字付けられています.	
242	<<SIMPLE 242>> ResourceRequire "名前" の引数 duration は整数の集合でなければなりません.	
243	<<SIMPLE 243>> 制約 " 名前" は rcpsp では扱う事は出来ません.	rcpsp では扱えない制約 alldiff, valgroup 等が定義された場合に出力されます.
244	<<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>> 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>> 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>> ResourceRequire で定義されている資源 ″名前″ が ResourceCapacity に定義されていません.	
248	<<SIMPLE 248>> rcpsp を適用するのに必要な Activity が定義されていません.	
249	<<SIMPLE 249>> 定義されていない資源 “名前” が直前先行制約で使われています.	
250	<<SIMPLE 250>> Boolean で指定しているモード″名前″は定義されていません.	
251	<<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>>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>>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>>一般の制約式に Activity は用いられません.startTime, endTime, processTime に対して記述して下さい	

258	<<SIMPLE 258>> ResourceRequire で設定されていないモード"名前”が Activity に与えられています.	
259	<<SIMPLE 259>> sourceActivity は Activity を定義した時のみ使用出来ます.	
260	<<SIMPLE 260>> sinkActivity は Activity を定義した時のみ使用出来ます	
261	<<SIMPLE 261>> Activity “名前" の index と引数 mode “名前” の index とが異なります.	
262	<<SIMPLE 262>> Activity "名前" の 引数に mode が設定されていません.	
263	<<SIMPLE 263>> Activity “名前”の引数 duedate “名前” の index が異なります.	
264	<<SIMPLE 264>>先行制約の 2 つの Activity 間で index が異なります.	
265	<<SIMPLE 265>>警告: 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	
266	<<SIMPLE 266>>警告: 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	
267	<<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>>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>>直前先行制約に資源が指定されていません	
270	<<SIMPLE 270>>先行制約の index と時間指定の index が異なります	
271	<<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>> 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>> ResourceCapacity "名前" の引数に resource が与えられていません	
274	<<SIMPLE 274>> ResourceCapacity "名前" の引数に timeStep が与えられていません	
275	<<SIMPLE 275>> ResourceCapacity "名前" の index と 引数 weight "名前" の index が異なります	
276	<<SIMPLE 276>> ResourceRequire "名前" の引数に mode が与えられていません	
277	<<SIMPLE 277>> ResourceRequire "名前" の引数に resource が与えられていません	
278	<<SIMPLE 278>> ResourceRequire "名前" の引数に duration が与えられていません	
279	<<SIMPLE 279>> ResourceRequire "名前" の 引数 duration に負の値が用いられています	

280	<<SIMPLE 280>> ResourceRequire にデータが設定されていません	
281	<<SIMPLE 281>>警告: モード"名前" が ResourceRequire に設定されていません	予期せぬ動作の原因になります.
282	<<SIMPLE 282>>警告: モード "名前" が ResourceRequire に定義されていますが使われていません.	予期せぬ動作の原因になります.
283	<<SIMPLE 283>>警告: 資源 "名前" 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	初期解の失敗の原因になります.
284	<<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)	
285	<<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>>警告: 納期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)	
288	<<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>>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>>警告: 納期最小化に対して重みを設定する事は出来ません	
291	<<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>>警告：納期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	
293	<<SIMPLE 293>>納期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>>警告：納期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	
295	<<SIMPLE 295>>納期最小化時は直前先行制約は定義出来ません	
296	<<SIMPLE 296>>納期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	
297	<<SIMPLE 297>> Gantt の dump に与えられたファイル〝ファイル名〞をオープン出来ません．	
298	<<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>>警告： ResourceCapacity で定義されている資源 "名前" が ResourceRequire で使われていません	
300	<<SIMPLE 300>> ResourceRequire "名前" 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 "名前" 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 "名前" 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 "名前" 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>>警告: ResourceCapacity "名前" 値が実数です. 整数に切り捨てます	
305	<<SIMPLE 305>>警告: ResourceCapacity "名前" 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	
306	<<SIMPLE 306>>警告: ResourceRequire "名前" 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	
307	<<SIMPLE 307>>警告: 一般の制約式"名前" 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	
308	<<SIMPLE 308>>警告: 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	
310	<<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>> 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>>警告: rcpps 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	
313	<<SIMPLE 313>> Activity "名前" 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>>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 は Activity を定義した時のみ使用出来ます	
316	<<SIMPLE 316>> ResourceRequire "名前" の引数 default に負の値が与えられています	
317	<<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>> ResourceRequrie "名前" に負の値が与えられています.	
319	<<SIMPLE 319>> ResourceCapacity "名前" に負の値が与えられています	
320	<<SIMPLE 320>> ResourceCapacity "名前" 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 "名前" のモード"名前" の処理時間が明確ではありません.(初期化して下さい)	
322	<<SIMPLE 322>> 警告: 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. 納期最小化は無効です.	
323	<<SIMPLE 323>> (直前)先行制約の先行関係に矛盾があります.	
324	<<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>> 納期最小化時には sourceActivity は用いる事は出来ません.	

326	<<SIMPLE 326>> 納期最小化時には sinkActivity は用いる事は出来ません.	
327	<<SIMPLE 327>> 納期最小化時には DummyMode は用いる事は出来ません.	
328	<<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>> 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>> 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>> 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>> 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>> 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 は Activity を定義した時のみ使用できます.	
335	<<SIMPLE 335>> ResourceRequire で定義されていない"名前"が Activity の初期値に与えられました	
336	<<SIMPLE 336>> ResourceRequire で定義されていない "名前" が Activity の初期値に与えられました	

337	<<SIMPLE 337>> 初期化されていない"名前" に対してモードの固定を行なおうとしました.	
338	<<SIMPLE 338>> 納期最小化時は fixActivity に重みを与える事は出来ません	
339	<<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>> Boolean で指定された "名前" はモード"名前" を取る事が出来ません	
341	<<SIMPLE 341>> 警告: mordeOrder 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	
342	<<SIMPLE 342>> 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.	
343	<<SIMPLE 343>> modeOrder 制約の 2 つの Activity 間で index が異なります.	
344	<<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>> 納期最小化時は modeOrder 制約を定義出来ません	
346	<<SIMPLE 346>> modeOrder 制約に与えられた 2 つの Activity "名前", "名前" 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.
347	<<SIMPLE 347>> 同一の Activity を modeOrder 制約に与える事は出来ません.	
348	<<SIMPLE 348>> 終了時刻を 0 に固定する事は出来ません("名前").	

349	<<SIMPLE 349>> 開始時刻を負の値で固定する事は出来ません("名前").	
350	<<SIMPLE 350>> 終了時刻を負の値で固定する事は出来ません("名前").	
351	<<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("名前").	
352	<<SIMPLE 352>> timeStep を越える時刻の固定は行なう事が出来ません("名前").	
353	<<SIMPLE 353>> 警告: 全ての Activity に対し初期値に作業リストが与えられていません. 初期値設定は無効です.	
354	<<SIMPLE 354>> 初期値に与えられた"名前"を "名前" は取り得る事が出来ません	
355	<<SIMPLE 355>> fixActivity に負の重みが与えられています	
356	<<SIMPLE 356>> fixActivity に負の重みが与えられています	
357	<<SIMPLE 357>> 警告: "名前"の時刻が実数の値で固定されています. 整数に切り捨てます.	
358	<<SIMPLE 358>> "名前" に初期値を代入する事は出来ません.	
359	<<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.	
360	<<SIMPLE 360>> setNuoptWatchFile() で設定したファイル "名前" を書き込みモードで開けません. Excel など別のアプリケーションでこのファイルを開いていないでしょうか.	

361	<<SIMPLE 361>> ResourceCapacity "名前" 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.
370	<<SIMPLE 370>> The index dim of the Matrix <<Name>> should be 2. <<SIMPLE 370>> 行列 "m" 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>> Invalid index dim of the Matrix <<Name>> or its Element. Now its index is dim <<Number>>, but should be <<Number>> (as Matrix) or <<Number>> (as Matrix Element). <<SIMPLE 371>> 行列 "m" (添字の次元:1) に，次元 0 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 1 に，要素を指すのなら，添字の次元は 3 でなければなりません.	
372	<<SIMPLE 372>> Warning: The Matrix <<Name>> is null. The SDP constraint on this matrix is ignored. <<SIMPLE 372>> 警告: 行列 "m" は空です．この行列に対する正定値制約は無視されます.	
373	<<SIMPLE 373>> Weight coefficients of the constraint <<Name>> are invalid. (a: Quadratic ,b: Linear) = (<<Number>>,<<Number>>). a and b both should be non-negative. <<SIMPLE 373>> 制約式 "p" のウエイト係数が不正です. (a:二次,b:一次) = (2,-0.5). a,b ともに零または正でなければなりません.	

374	<<SIMPLE 374>> entryVariable should not be called (DFO specific). <<SIMPLE 374>> entryVariable は DFO アドオン専用の機能です.	entryVariable は Derivative Free Optimziation アドオン専用の機能です.
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 invalid 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).
400	<<SIMPLE 400>> the dimension of reference object 行列名 1 and reference object 行列名 2 are not suitable. <<SIMPLE 400>> 参照オブジェクト 行列名 1 と参照オブジェクト 行列名 2 の次元が整合しません.	
401	<<SIMPLE 401>> reference object 行列名 1 cannot be converted into vector. <<SIMPLE 401>> 参照オブジェクト 行列名 1 はベクトルに変換できません.	

402	<<SIMPLE 402>> reference object 行列名 1 cannot be converted into scalar. <<SIMPLE 402>> 参照オブジェクト 行列名 1 はスカラーに変換できません.	
403	<<SIMPLE 403>> because the dimension of reference object 行列名 1 and reference object 行列名 2 are not suitable, inner product is not computable. <<SIMPLE 403>> 参照オブジェクト 行列名 1 と参照オブジェクト 行列名 2 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 行列名 1 and reference object 行列名 2 are not suitable, addition and subtraction are not computable. <<SIMPLE 404>> 参照オブジェクト 行列名 1 と参照オブジェクト 行列名 2 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 行列名 1 and reference object 行列名 2 are not suitable, multiplication is not computable. <<SIMPLE 405>> 参照オブジェクト 行列名 1 と参照オブジェクト 行列名 2 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 行列名 1 and scalar 行列名 2 are not compareable. <<SIMPLE 406>> 参照オブジェクト 行列名 1 とスカラー 行列名 2 は比較できません.	

407	<<SIMPLE 407>> because reference object 行列名 1 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 行列名 1 のトレースは計算できません.	
408	<<SIMPLE 408>> not-square matrix object 行列名 1 was given into the argument of logdet. <<SIMPLE 408>> logdet の引数に正方行列でないオブジェクト 行列名 1 が与えられました.	
409	<<SIMPLE 409>> indexed matrix object 行列名 1 was given into the argument of logdet. <<SIMPLE 409>> logdet の引数に添字付きの行列オブジェクト 行列名 1 が与えられました.	
410	<<SIMPLE 410>> function sum of reference object 行列名 1 is not computable. <<SIMPLE 410>> 参照オブジェクト 行列名 1 の sum は計算できません.	
411	<<SIMPLE 411>> the multiplication of reference object 行列名 1 and indexed parameter is not computable. <<SIMPLE 411>> 参照オブジェクト 行列名 1 と添字付き Parameter の乗算は計算できません.	
412	<<SIMPLE 412>> indexed reference object 行列名 1 has mistake of indexing. <<SIMPLE 412>> 添字付きの参照オブジェクト 行列名 1 の添字付けに誤りがあります.	

413	<<SIMPLE 413>> substitution for indexed reference object 行列名 1 is prohibited. <<SIMPLE 413>> 添字付きの参照オブジェクト 行列名 1 には代入が禁止されています.	
449	<<SIMPLE 449>> error occurred by matrix or vector arithmetic operation. <<SIMPLE 449>> 行列 または ベクトル の演算でエラーが発生しました.	
550	<<SIMPLE 550>> Warning: Elements of a empty Set are expanded. <<SIMPLE 550>> 警告: 空集合に対して添字の展開をしようとしました.	
551	<<SIMPLE 551>> Invalid semidefinite constraint on "Matrix name".RHS must not have movable index. <<SIMPLE 551>> 行列 "行列名" 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 "Matrix name" has index "index" out of its dimSet. <<SIMPLE 552>> 対称行列 "行列名" のインデクス "添字" 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 "<<object-name>>" is circular. <<SIMPLE 553>> 漸化式において、オブジェクト"<<オブジェクト名>>" 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>> Invalid evaluation in a recurrence relation. <<SIMPLE 554>> 漸化式定義中に不正な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.
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.

A.2 NUOPT のエラー/警告メッセージ

NUOPT 本体の出力するエラー/警告メッセージは,

- NUOPT 本体部の計算で検出されたエラー/警告

```
<<NUOPT 番号>>      エラー/警告メッセージ
```

- パラメータに関するエラー/警告

```
<<PARAMETER 番号>>   エラーメッセージ
```

- MPS ファイル解釈時に検出されたエラー/警告

```
<<MPS FILE 番号>>    エラーメッセージ
```

の３種類です．エラーメッセージは，標準出力（WindowsGUI では結果表示ウインドウとメッセージウインドウ）に出力されます．SIMPLE ロードモジュールの場合には

```
ex2.smp の 5 行目の実行中にエラーが起きました.
<<SIMPLE 193>> NUOPT のエラー: <<NUOPT 10>> IPM iteration limit exceeded.
```

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

A.2.1 NUOPT のエラー/警告

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

エラー番号	エラーメッセージ	説明
1	<<NUOPT 1>> memory error in preprocessing.	前処理部で所要メモリが，使用可能な量をオーバーしました．[解出力なし]
2	<<NUOPT 2>> infeasible (linear constraints and variable bounds).	線形制約や変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました．[解出力なし]
3	<<NUOPT 3>> neither valid objective nor constraint in this model.	モデル(問題)に目的関数および制約式がありません．[解出力なし]

5	<<NUOPT 5>> infeasible integer variable bounds.	変数の上下限制約のために問題が infeasible となっていることが前処理部で検出されました. (整数変数が整数性に違反する様な上下限を課されている場合等に発生します.) [解出力なし]
6	<<NUOPT 6>> unbounded (linear constraints and variable bounds).	線形制約と変数の上下限から問題の最適解は有界とはならないことが前処理部で判定されました. [解出力なし]
7	<<NUOPT 7>> internal error. [内部ルーチン名]	内部エラーが発生しました. (nuopt-support@msi.co.jp にご連絡下さい). [解出力なし]
8	<<NUOPT 8>> memory error in optimization phase.	計算部において所要メモリが使用可能な量をオーバーしました. [解出力なし]
9	<<NUOPT 9>> step reduction limit exceeded.	直線探索アルゴリズムにおいて step reduction の失敗が起き, 最適化実行が止まってしまいました (凸でない問題に直線探索アルゴリズムを適用した場合や, 問題が infeasible である場合に起きます). [解出力あり]
10	<<NUOPT 10>> IPM iteration limit exceeded.	内点法の反復回数が上限を越えました (上限は特に指定しない場合には 150 回です. パラメータファイルからは criteria: maxitn =300 として設定することができます). [解出力あり]
11	<<NUOPT 11>> infeasible.	問題が実行不可能であると判定されました. [解出力あり]
13	<<NUOPT 13>> unbounded.	問題の最適解が有界でないことが判定されました. [解出力あり]
14	<<NUOPT 14>> integrality is violated.	整数変数を含む問題に単体法以外を適用したため, 整数変数が整数となっていない解が出力されています. [解出力あり]
15	<<NUOPT 15>> simplex misapplied to NLP.	非線形計画問題に単体法が適用されようとしています. [解出力なし]
16	<<NUOPT 16>> Infeasible MIP.	混合整数計画問題に整数解が存在しないことがわかりました (LP/QP 緩和解が出力されます). [解出力あり]

17	<<NUOPT 17>> B & B node limit reached (with feas.sol.).	分枝限定法において生成する部分問題数が上限を越えました．最適解である保証はありませんが，整数解は得られています．(branch:maxnod を設定した場合)．[解出力あり]
18	<<NUOPT 18>> MIP iteration failed (with feas.sol.).	分枝限定法のループが途中で数値的問題等によって失敗しました．最適解である保証はありませんが，整数解は得られています．[解出力あり]
19	<<NUOPT 19>> B & B node limit reached (no feas.sol.).	17 番と同じですが，整数解が得られていません．LP（QP）緩和解が出力されます．[解出力あり]
20	<<NUOPT 20>> MIP iter. failed (no feas.sol.).	18 番と同じですが，整数解が得られていません．LP（QP）緩和解が出力されます．[解出力あり]
21	<<NUOPT 21>> B & B itr. timeout (with feas.sol.).	整数計画法 を解いている場合の NUOPT の起動時間が上限を越えました．最適である保証はありませんが，整数解は得られています (branch:maxnod を設定した場合)．[解出力あり]
22	<<NUOPT 22>> B & B itr. timeout (no feas.sol.).	21 番と同じですが，整数解が得られていません．LP/QP 緩和解が出力されます．[解出力あり]
25	<<NUOPT 25>> Can't open file in current directory [no .sol made]	解ファイル (.sol ファイル)のオープンに失敗したので解ファイルが出力されていません．(警告です．計算は行われます)．[解出力なし]
26	<<NUOPT 26>> LP/IP module cannot handle NLP.	LP/IP モジュールで非線形計画問題を解こうとしました．[解出力なし]
27	<<NUOPT 27>> SIMPLEX iteration limit exceeded.	単体法の反復回数の上限をオーバーしました．[解出力あり]
28	<<NUOPT 28>> higher-order method is only for LP.	非線形計画問題に線形計画法専用の内点法が適用されようとしています．[解出力なし]
29	<<NUOPT 29>> iteration diverged.	線形計画法専用の内点法が発散しました．[解出力あり]
30	<<NUOPT 30>> terminated by user.	ユーザの指示により計算の中断を行いました．[解出力あり]

31	<<NUOPT 31>> B&B terminated by user (with feas.sol.).	分枝限定法の演算中ユーザの指示により計算の中断を行いました．最適解である保証はありませんが，整数解は得られています．[解出力あり]
32	<<NUOPT 32>> B&B terminated by user (no feas.sol.).	31 番と同じですが，整数解が得られていません．LP/QP 緩和解が出力されます． [解出力あり]
33	<<NUOPT 33>> Bound violated.	変数の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
34	<<NUOPT 34>> Bound and Constraint violated.	変数および制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
35,36	<<NUOPT 35>> Constraint violated. <<NUOPT 36>> Equality Constraint violated.	制約式の上下限を有意に違反している解が出力されています．違反している個所は，解ファイル（*.sol）で"INFS"と表示されています．スケールの悪い（制約式や変数の値のオーダーの差が激しい）問題と思われます．内点法で解いている場合には停止条件を小さくして実行する必要があります．[解出力あり]
37	<<NUOPT 37>> B&B terminated with given # of feas.sol.	options.maxintsol で指定した数だけの整数解が発見されたので分枝限定法を停止しました．[解出力あり]
38	<<NUOPT 38>> dual infeasible.	リスタート時に起動される双対単体法のプロセスで実行不可能性が検出されました．[解出力あり]

39	<<NUOPT 39>> IPM iteration timeout	options.maxtim で設定した時間に対して, 内点法の反復がタイムアウトしました. [解出力あり]
40	<<NUOPT 40>> SQP iteration limit exceeded	SQP (lsqp,tsqp,slpsqp) の反復が上限を超えました. 他のアルゴリズムをお試しください. [解出力あり]
41	<<NUOPT 41>> SQP internal error	SQP (lsqp,tsqp,slpsqp) の実行中に内部的なエラーが発生しました. 他のアルゴリズムをお試しください. [解出力なし]
43	<<NUOPT 43>> B&B memory error (with feas.sol.)	分枝限定法の実行中に, options.maxmem で設定したメモリオーバーが起こり, 実行を停止しましたが, 実行可能解は得られています. [解出力あり]
44	<<NUOPT 44>> B&B memory error (no feas.sol.)	分枝限定法の実行中に, options.maxmem で設定したメモリオーバーが起こり, 実行を停止しました. 実行可能解は得られていません [解出なし]
45	<<NUOPT 45>> B&B gap reaches under the limit.	上下界の差が options.gaptol で与えられたよりも小さくなりましたので, 分枝限定法を停止します. [解出力あり]
46	<<NUOPT 46>> Continuous Variable 変数名 cannot be included in model for wcsp.	連続変数を含む問題を wcsp で解こうとしました. wcsp は連続変数を扱うことができません. [解出力なし]
47	<<NUOPT 47>> IntegerVariable 変数名 should be declared as binary.	0-1 整数変数以外の整数変数が表れています. wcsp は一般の整数変数を扱うことができません. [解出力なし]
48	<<NUOPT 48>> Variable 変数名 appear in two selection ().	二つ以上の selection にまたがって現れている 0-1 整数変数が表れました. [解出力なし]
49	<<NUOPT 49>> Variable 変数名 is fixed to infeasible value.	0-1 整数変数が 0, 1 以外の値に固定されました. [解出力なし]
50	<<NUOPT 50>> Both of two variables 変数名 1 and 変数名 2 cannot be 1.	変数名 1 と変数名 2 の両方は (単一の selection に現れているので) 両方 1 になることができません. [解出力なし]

51	<<NUOPT 51>> wcsp is not available without SIMPLE.	wcsp は SIMPLE によって定義された問題に対してのみ有効です. [解出力なし]
52	<<NUOPT 52>> 数字 th selection () statement has no movable binary variables.	対応する変数がすべて固定されているか存在しない selection ()文が「数字」番目に現れました. [解出力なし]
53	<<NUOPT 53>> Constraint 制約式名's weight is 値 should be -1 or non-negative value.	重みとしては-1 もしくは非負の数を与えねばなりません. [解出力なし]
54	<<NUOPT 54>> Constraint 制約式名's weight is 値 should be -1 or non-negative value.	重みとしては-1 もしくは非負の数を与えねばなりません. [解出力なし]
55	<<NUOPT 55>> exterior solution obtained.	外点法（lepm/tepm）が不等式制約を違反する解を出力しました. パラメータ exrho（デフォルトは 1.0e4）を大きく設定すると消える可能性がありますが，そうしてもこのエラーが出る場合には，問題が実行不可能である可能性があります. [解出力あり]
56	<<NUOPT 56>> アドオン情報 not installed.	有償のアドオン（global,DFO）がインストールされていない状態で，有償のアドオンの機能を利用しようとしました.
57	<<NUOPT 57>> You cannot use any method but "rcpsp" for model with Activity. "	Activity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました. [解出力なし]
58	<<NUOPT 58>> You cannot use any method but "rcpsp" for model with ResourceRequire.	ResourceRequire の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました. [解出力なし]
59	<<NUOPT 59>> You cannot use any method but "rcpsp" for model with ResourceCapacity.	ResourceCapacity の定義があるにも関わらず, rcpsp 以外のメソッドを適用しようとしました. [解出力なし]
60	<<NUOPT 60>> You cannot use any method but "rcpsp" for model with tardiness and completionTime.	目的関数が tardiness/completionTime に設定されているのにも関わらず，rcpsp 以外のメソッドを適用しようとしました. [解出力なし]

61	<<NUOPT 61>> You cannot use any method but "rcpsp" for model with sourceActivity	sourceActivity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
62	<<NUOPT 62>> You cannot use any method but "rcpsp" for model with sinkActivity	sinkActivity の定義があるにも関わらず，rcpsp 以外のメソッドを適用しようとしました．[解出力なし]
63	<<NUOPT 63>> You cannot use Variable for "rcpsp"	rcpsp は Variable は用いる事が出来ません．[解出力なし]
64	<<NUOPT 64>> You cannot use IntegerVariable for "rcpsp"	rcpsp は IntegerVariable は用いる事が出来ません.[解出力なし]
65	<<NUOPT 65>> failed to generate an initial solution	rcpsp において初期解生成に失敗しました．[解出力なし]
66	<<NUOPT 66>> can't find feasible solution.	rcpsp において実行可能解を得る事が出来ませんでした.[解出力なし]
67	<<NUOPT 67>> DiscreteVariable （変数名） 's bound: [a,b] and domain: {..} conflicts. (Regard Bound as a definition).	DiscreteVariable の定義された範囲では満たすことのできない上下限があたえられました．[解出力なし]
68	<<NUOPT 68>> can't open saveFile for restart	rcpsp においてリスタート情報を保存する為のファイルをオープン出来ませんでした.[解出力なし]
69	<<NUOPT 69>> can't open loadFile for restart	rcpsp においてリスタート情報を読み込む為のファイルをオープン出来ませんでした.[解出力なし]
70	<<NUOPT 70>> initial order is invalid between "名前" and "名前" . activities related imprecedene have to continue.	rcpsp において初期解として不正な順番が与えられました.直前先行制約に関係する Activity の順番は連続していなければなりません．[解出力なし]

71	<<NUOPT 71>> initial order is invalid. "名前"'s order have to be previous to "名前" by way of precedence.	rcpsp において初期解として不正な順番が与えられました.先行関係がある Activity はその順番が守られなければなりません. [解出力なし]
80	<<NUOPT 80>> # of variables(数字) reached size limitation for LP,NLP,LP/QP	現在お使いの NUOPT モジュールにある変数の数の制限のため, この問題を解くことはできません. [解出力なし]
81	<<NUOPT 81>> You cannot use any method but "wcsp" for model with DiscreteVariable (s).	DiscreteVariable を含む問題に wcsp 以外の方法は適用できません. [解出力なし]
82	<<NUOPT 82>> Trust region too small	関数の二次近似に失敗しつづけて信頼領域が小さくなりすぎましたので実行を停止します. [解出力あり]
83	<<NUOPT 83>> Feas.sol found (numerical error in B&B).	分枝限定法の途中で数値的問題が発生, 実行可能解はもとまりましたが, 最適性の保証はありません. [解出力あり]
110	<<NUOPT 110>> CF failed at getcky	正定値で無い行列が与えられ, コレスキー分解に失敗しました. [解出力なし]
111	<<NUOPT 111>> CF failed at logdet	メリット関数計算時において, 正定値でない行列が与えられ, コレスキー分解に失敗しました. [解出力なし]
112	<<NUOPT 112>> InvMat cannot obtained at calxx1	逆行列を求める計算に失敗しました. [解出力なし]
113	<<NUOPT 113>> GSEP failed at minevl	一般化固有値問題を標準固有値問題に変換する事ができませんでした. [解出力なし]
114	<<NUOPT 114>> trianglization failed at minevl	三重対角化に失敗しました. [解出力なし]
115	<<NUOPT 115>> Minimum eigenValue cannot obtained	最小固有値の導出に失敗しました. [解出力なし]
120	<<NUOPT 120>> PDgap is too large[PDfeasible]	主双対ギャップが十分小さくならない内に, エラーが発生し計算を終了致しました. 主問題・双対問題の実行可能解は満たされています. [解出力あり]

121	<<NUOPT 121>> PDgap is too large[Pfeasible]	主双対ギャップが十分小さくならない内に，エラーが発生し計算を終了致しました．主問題の実行可能解は満たされています．[解出力あり]
122	<<NUOPT 122>> minus stepsize detected	正であるべきステップサイズに負の値が検出されました．解法を trust に変更してお試し下さい．[解出力なし]
123	<<NUOPT 123>> The SDP constraint cannot be treated by specified algorithm.	半正定値制約が存在しますが，半正定値制約を解釈できるアルゴリズムが選択されていません．[解出力なし]
124	<<NUOPT 124>> No SDP constraint is detected.	半正定値計画法用のアルゴリズムが起動されましたが，半正定値制約が存在しません．[解出力なし]
126	<<NUOPT 126>> type semicont/partial is obsolete.	準連続変数と部分連続変数は NUOPT V15 より廃止されました．[解出力なし]

A.2.2 パラメータのエラー

nuopt.prm または options を用いた NUOPT のパラメータ設定のエラーです.

エラー番号	エラーメッセージ	説明
1	<<PARAMETER 1>> Syntax error in parameter file.	パラメータファイルの記述にエラーが検出されました.
2	<<PARAMETER 2>> Parameter file is empty.	パラメータファイルが全く空になっています.
3	<<PARAMETER 3>> Invalid 型名 value 値 for parameter パラメータ名.	パラメータ名に対して不適切な値が設定されています(パラメータファイル以外からパラメータ値を与える場合のみに発生します).
4	<<PARAMETER 4>> Internal error [内部ルーチン名]	パラメータの解釈を行うルーチンにて内部エラーが発生しました(nuopt-support@msi.co.jp にご連絡下さい).
5	<<PARAMETER 5>> Invalid option in nuopt options.	AMPL 対応ロードモジュールがパラメータを読み込む環境変数 nuopt_options に無効なパラメータ名が含まれていました.

パラメータファイルの記述にエラーが発生した場合(エラー番号 1) にはパラメータファイル読み込み部から標準出力（Windows では GUI のメッセージウインドウ）に補助的なメッセージが表示されますので, 併せて参考にして下さい.

エラーのあるパラメータファイル :

```
begin
      maximize
      method:trust
      criteria:eps=1.0e-8
```

標準出力（WindowsGUI のメッセージボックス）:

```
<reading parameter file: nuopt.prm >
    begin
    maximize
    method:trust
    criteria:eps=1.0e-8

 +error+ undefined command. check the command name.
         criteria           ( 行名が違う)
 +error+ last command must be end-command
         end                ( end で終わっていない)
 ---- input data error occurred ----
<< PARAMETER 1 >> Syntax error in parameter file.
```

A.2.3 MPS ファイルのエラー

エラー番号	エラーメッセージ	説明
1	<<MPS FILE 1>> Failed to open mps file: ファイル名.	MPS ファイルのオープンに失敗しました.
2	<<MPS FILE 2>> Undefined row name: 行名.	COLUMNS/RHS/RANGES セクションに未定義の行が現れました.
3	<<MPS FILE 3>> Internal error. [内部ルーチン名]	[内部ルーチン] で内部エラーが発生しました. (nuopt-support@msi.co.jp にご連絡下さい.)
4	<<MPS FILE 4>> Syntax error in セクション名 section.	「セクション名」の示すセクションで文法エラーが発生しました.

5	<<MPS FILE 5>> Too many 'INTORG' marker.	'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません. ('INTORG' マーカー行が多すぎます.)
6	<<MPS FILE 6>> Too many 'INTEND' marker.	'INTORG' マーカー行と'INTEND' マーカー行の対応が取れていません. ('INTEND' マーカー行が多すぎます.)
7	<<MPS FILE 7>> Unknown marker: フィールド３の内容	'INTORG','INTEND' 以外のマーカー行が現れました. (NUOPT の MPS ファイル解釈部は'INTORG','INTEND' 以外のマーカー行を解釈しません.)
8	<<MPS FILE 8>> Undefined row: 行名 in HESSIAN section.	HESSIAN セクションの二次の項を付加する行名として, 定義されていないものが現れました.
9	<<MPS FILE 9>> Undefined column: 変数名 in HESSIAN section.	HESSIAN セクションの非零要素の場所を示す際の変数名として, 定義されていないものが現れました.
10	<<MPS FILE 10>> row: 行名 appeared more than once.	ROWS セクションに同一の行名が二度以上現れました.
11	<<MPS FILE 11>> Specified bound: BOUND データラベル not found.	パラメータファイルで指定された (mpsfile: bound = BOUND データラベル) ラベルを持つ BOUNDS データが MPS ファイルには存在しません.
12	<<MPS FILE 12>> Specified objective: 目的関数行名 not found.	パラメータファイルで指定された (mpsfile: objective = 目的関数行名) 名前を持つ目的関数行が MPS ファイルには存在しません.
13	<<MPS FILE 13>> Specified rhs: RHS データラベル not found.	パラメータファイルで指定された (mpsfile: rhs = RHS データラベル) ラベルを持つ RHS データが MPS ファイルには存在しません.
14	<<MPS FILE 14>> Range data: RANGE データラベル contains unsuitable row.	「RANGE データラベル」を持つ RANGE データが目的関数行に対しての指定を行っています.

15	<<MPS FILE 15>> Specified range data: RANGE データラベル not found.	パラメータファイルで指定された(mpsfile: range = RANGE データラベル)ラベルを持つ RANGE データが MPS ファイルには存在しません.
17	<<MPS FILE 17>> Undefined column name: 変数名 in INITIAL section.	INITIAL セクションに定義されていない変数名が現れました.
18	<<MPS FILE 18>> Bound spec. on column : 変数名 should appear earlier.	「変数名」に関する上下限設定の現れるのはより前でなければなりません. (警告です. 最初に発見されたものに対してのみこのメッセージが出力されます.)
19	<<MPS FILE 19>> 数字 column(s) appeared disorderly in BOUNDS section	BOUNDS section において「数字」個の変数の現れる順番が違法です. (警告です. エラー18 と組になって出力されます.)
20	<<MPS FILE 20>> 2-pass required for reading from stdin. Please try again.	MPS ファイルを標準入力から入力している際に, MPS ファイルの読み直しが必要となりました. (このメッセージが表示された際には読み込みに必要なメモリ所要量を設定し, ファイルに書き落としていますので, もう一度同じ MPS ファイルを入力すれば正常に動作します.)
21	<<MPS FILE 21>> Undefined column name: 変数名 in BOUNDS section.	BOUNDS section において定義されていない変数名が現れました.
22	<<MPS FILE 22>>Hessian is implicitly bound for nonexistent objective.	HESSIAN セクションでは暗黙のうちに(行名を指定せず) 目的関数に対して二次の項が設定されていますが, MPS ファイルには全く目的関数が存在しません.
23	<<MPS FILE 23>> Same bound spec. on column: 変数名 appeared more than once.	BOUNDS section で「変数名」に関して同一の制約指定が二度以上行われました.

24	<<MPS FILE 24>> Column : 変数名 has bound specification FX and other.	BOUNDS section で「変数名」に関して FX 制約と他の制約指定が同時に行われました.
25	<<MPS FILE 25>> Column : 変数名 has bound specification FR and other.	BOUNDS section で「変数名」に関して FR 制約と他の制約指定が同時に行われました.
26	<<MPS FILE 26>> Column : 変数名 has bound specification LO and MI.	BOUNDS section で「変数名」に関して LO 制約と MI 制約指定が同時に行われました.
27	<<MPS FILE 27>> Column : 変数名 has bound specification UP and PL.	BOUNDS section で「変数名」に関して UP 制約と PL 制約指定が同時に行われました.

付録 B　NUOPT アルゴリズム概説

B.1 内点法

B.1.1 問題

次の最適化問題

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

を考えます[6]. ここで, 変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで, 関数 f は目的関数です. また, $g : R^n \to R^m$ は m 次元のベクトル値関数です. この問題のラグランジュ関数を

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

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

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

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

NUOPT ではバリヤペナルティ関数：

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

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

非線形最適化問題に対する内点法では,「修正 KKT 条件を満たす点を求めて，修正 KKT 条件を更新する」という作業を逐次行います. この際に，メリット関数 $F(x, z)$ の一次近似 $F_l(x, z)$ あるいは二次近似 $F_q(x, z)$ の変化量が重要な手がかりとなります[7]. 次項以降で示すように，修正 KKT 条件を求める方法は複数存在します（直線探索を利用する方法・信頼領域を利用する方法）.

この作業を通して，最終的に元来の KKT 条件を満たす点を求めます.

---

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

[7]詳細は論文[14][15][16]を参照

B.1.2 直線探索を利用する方法

修正 KKT 条件に対するニュートン法は次のようになります.

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

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

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

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

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

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

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

次に,

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

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

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

をみたす最小の正整数として定義されます[8]. $\varepsilon_0 \in (0,1)$ です.

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

ラグランジュ関数のヘッセ行列が非負定値となるのは

- 線形計画問題 (ヘッセ 行列は０)
- 一般の凸計画問題

です. また, 一般の非線形計画問題ではヘッセ行列を準ニュートン法で近似することによって

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

行列Gを常に正定値に保つことができます．従って，このような場合に直線探索を利用した手法を使用することができます．

B.1.3 信頼領域を利用する方法

行列Gが非負定値でないとき直線探索法を利用することは困難です．そこで，Gが不定であるときも利用できる方法として主双対変数に対する信頼領域法を採用します．このとき，探索方向ベクトル w ，サイズ $\delta>0$ ，ステップ幅αは次のような関係を満たします．

$$\|w\| \le \delta,$$

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

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

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

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

$$\Delta z_{SD} \ = \ -X^{-1}Z\Delta x_{SD} - X^{-1}r_C,$$

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

$$\alpha^* = \arg\min\left\{\Delta F_q\left(x,z,\alpha\Delta x_{SD},\alpha\Delta z_{SD}\right) \middle| \left\|\alpha\left(\Delta x_{SD}+\Delta z_{SD}\right)\right\| \le \delta, \alpha\in\left[0,\bar{\alpha}\right]\right\}$$

$$\bar{\alpha} = \min\left\{1, \gamma\alpha_{\max}\right\}, \gamma\in(0,1),$$

信頼領域のステップ幅 α は以下の条件をみたすように設定します．

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

$$\left\|\Delta x_{Nk}\right\| \le M\left\|\Delta x_{SDk}\right\|,$$

$$\left\|\Delta z_{Nk}\right\| \le M\left\|\Delta z_{SDk}\right\|$$

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

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

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

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

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

B.1.4 線形計画問題専用内点法

問題が線形の場合，KKT 条件

$$\begin{aligned}\nabla_x L(x,y,z) &= 0,\\ g(x) &= 0,\\ XZe &= 0 \qquad (1)\end{aligned}$$

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

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

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

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

B.1.5 半正定値計画問題専用内点法

次の最適化問題

$$\begin{aligned}&\text{最小化} \quad f(x) \qquad\qquad x \in R^n, X \in S^p\\ &\text{条件} \quad g(x) = 0, X_0(x) = X, x \ge 0, X \succeq 0\end{aligned}$$

を考えます. ここで, 変数 $x = (x_1, \cdots, x_n)^t$ は n 次元ベクトルで, 関数 f は目的関数です. また, $g : R^n \to R^m$ は m 次元のベクトル値関数です. 変数 X は p 次元対称正方行列で， $X \succeq 0$ は

行列 X の半正定値制約を意味するものとします．$X_0 : R^n \to S^p$ は n 次元ベクトルを対称正方行列空間に写す写像です．問題が線形の場合， X_0 は $X_0(x) = \sum_{i=1}^{n} A_i x - B$ と表現しても構いません．

この問題のラグランジュ関数を

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

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

$$\begin{aligned}
&\nabla_x L(w) = \nabla f(x) - \nabla g(x)^T \Delta y - A^*(x) Y = 0 \\
&\nabla_X L(w) = Y - Z = 0 \\
&g(x) = 0 \\
&X_0(x) - X = 0 \\
&X \circ Z = 0, X \succeq 0, Z \succeq 0
\end{aligned}$$

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

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

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

非線形半正定値計画問題を扱う場合，バリヤペナルティ関数：

$$\begin{aligned}
&F(x, X, Z) = F_{BP}(x, X) + \nu F_{PD}(x, X, Z) \\
&F_{BP}(x, X) = f(x) - \mu \log(\det X) + \rho \| g(x) \|_1 + \rho' \| X_0(x) - X \|_1 \\
&F_{PD}(X, Z) = \langle X, Z \rangle - \mu \log(\det X \det Z)
\end{aligned}$$

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

探索方向を求める Newton 方程式は以下のように記述されます．

$$\begin{pmatrix} G + H & -\nabla g(x)^T \\ -\nabla g(x) & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = - \begin{pmatrix} \nabla f(x) - \nabla g(x)^T y - \mu A^*(x) X^{-1} + A^*(x)\left(Z X_0 X^{-1} - Z\right) \\ g(x) \end{pmatrix}$$

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

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

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

法は異なります.

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

問題が非凸な場合，大域的収束性を確保するための工夫が必要になります． NUOPT では準ニュートン法に基づいて G の正定値性が確保され，大域的収束性が保証されます．信頼領域法に基づく方法を利用する場合，大域的収束を保証する方向と，局所的収束に有利な方向を混ぜ合わせて探索方向を決定します.

B.2 単体法・有効制約法

単体法は線形最適化問題

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

に対して，有効制約法は二次計画問題

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

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

B.2.1 改訂単体法

NUOPT に実装されている単体法は大規模問題用の改訂単体法と呼ばれる手法です. 単体法自身に関する解説は例えば[3]を参照して下さい.

B.2.2 有効制約法

NUOPT に実装されている有効制約法は改訂単体法に基づいています. 有効制約法に関する解説は例えば[13] を参照して下さい．この手法は 5 千変数以上の大規模問題では，一般に内点法（直線探索法(Line Search Method)）に劣りますが，

変数に比べて制約式の数が非常に少ない（1/10 以下）場合

目的関数のヘッセ行列が密行列である場合

には内点法よりも高速かつ高精度です.

B.2.3 分枝限定法

NUOPT は線形な整数・二次計画問題に対して，単体法・有効制約法にもとづく分枝限定法を用います．分枝限定法とは，組み合わせ的に膨大な数にのぼる可能解の中から最適解を見つけ出す(探索する)一般的な手法です．分枝限定法にて，整数計画問題を解く場合には整数性の条件

を一部緩めた問題(緩和問題)を繰り返し解いて，その解から最適性を調べる範囲を限定しながら探索を行います．その際，単体法・有効制約法の上記 2.の特徴を生かすと，線形な整数計画法に対する分枝限定法を効率的に実現することができます．この手法自体については例えば[9]を参照して下さい．

B.3 逐次二次計画（SQP）法

逐次二次計画法では，次のような等式・不等式制約つきの非線形最適化問題の解を求めることができます．[9]

目的関数 $f(x)$ → 最小化

制約条件
$$g_j(x) = 0, j \in J_E$$
$$g_j(x) \geq 0, j \in J_I$$

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

B.3.1 準ニュートン法を用いる方法

本手法では，元の問題を次のような二次計画問題で近似します．

目的関数 $\frac{1}{2}\Delta x^T B_k \Delta x + \nabla f(x_k)^T \Delta x$ → 最小化

制約条件
$$g_j(x_k) + \nabla g_j(x_k)^T \Delta x = 0, j \in J_E$$
$$g_j(x_k) + \nabla g_j(x_k)^T \Delta x \geq 0, j \in J_I$$

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

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

B.3.2 信頼領域法を用いる方法

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

目的関数　$\frac{1}{2}(\Delta x_k^{SD})^T D_k \Delta x_k^{SD} + \nabla f(x_k)^T \Delta x_k^{SD}$　→　最小化

制約条件
$$g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} = 0, j \in J_E$$
$$g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^{SD} \geq 0, j \in J_I$$

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

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

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

となります．このとき，もう一つの二次計画問題は次のように定められます．

目的関数　$\frac{1}{2}(\Delta x_k^N)^T G_k \Delta x_k^N + \nabla f(x_k)^T \Delta x_k^N$　→　最小化

制約条件　$g_j(x_k) + \nabla g_j(x_k)^T \Delta x_k^N = 0, j \in J_A^k$

ここでG_kは元の問題のラグランジュ関数のヘッセ行列です．この問題の解Δx_Nは，反復点が元の問題の解に近づいたときに速い収束をするための方向になっています．また，この問題のKKT条件は，次のように線形方程式系として表すことができます．

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

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

本手法では，Δx_{SD}とΔx_Nの凸結合

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

を探索方向とし，定められた信頼領域の中を探索し，次の反復点を生成します．

B.3.3 逐次線形二次計画（SLP-SQP）法

逐次線形二次計画法(SLP-SQP)は制約充足のためのフェーズと目的関数値最小化のフェーズという二つのフェーズから構成され，これらを交互に行う事でペナルティ関数無しに大域的収束性を保証します．各フェーズでは元の問題を線形計画問題，あるいは二次計画問題で近似し，それらの解を用いて探索を行います．

ここでは簡単のため，不等式制約は各変数の下限制約のみとします．制約充足のフェーズでは次の問題を繰り返し解きます．

$$
\begin{aligned}
\text{最小化}\quad & \nabla f(x_k)^T s + \frac{1}{2} s^T G_k s \\
& g_j(x_k) + \nabla g_j(x_k)^T s = 0,\ j \in J_E \\
\text{条　件}\quad & x_k + s \geq 0,\ j \in J_I \\
& \|s\|_\infty \leq \Delta_k
\end{aligned}
$$

G_k は零行列あるいはラグランジュ関数のヘッセ行列，または，その近似行列です．条件式が与えられた上下限制約のもとでの $g_j(x) = 0,\ j \in J_E$ の一次近似になっているため，上記問題の解方向に直線探索をすると，制約の充足度を向上させる事が出来ます．また，G_k をラグランジュ関数のヘッセ行列，あるいは近似行列とすれば，反復点 x_k が解に近い時には通常の信頼領域二次計画法と同等の反復点を生成する事になり，超一次収束が保障されます．

制約充足のフェーズが，実行可能領域への接近を目標としている事に対し，目的関数値最小化フェーズでは等式制約が定める領域の接方向へと移動し目的関数値の最小化を目指します．

具体的には，実行可能領域へ近付くステップ s と，目的関数の減少を意図した接方向へのステップ s_T との組合せ s_ρ を探索方向の候補とします．s_T は次の問題の解として取得します．

$$\begin{aligned} \text{最小化} \quad & \nabla f(x_k)^T s_T + \frac{1}{2} {s_T}^T G_k s_T \\ & \nabla g_j(x_k)^T s_T = 0, j \in J_E \\ \text{条　件} \quad & x_k + s_T \geq 0, j \in J_I \\ & \|s_T\|_\infty \leq \Delta_T \end{aligned}$$

ここでは制約充足フェーズの場合と同様に G_k は零行列あるいはラグランジュ関数のヘッセ行列，または，その近似行列とします．この問題は $\Delta_T = \Delta_{Tk}$ から始め s_ρ が一定の制約充足度を満たすようになるまで Δ_T を半減させつつ繰り返し解きます．この内部反復により最終的に得られた s_ρ を s_{ρ_k} として，s_{ρ_k} による目的関数の減少量が，その二次近似関数の減少量と比べて十分だった場合は，Δ_T を拡大し $\Delta_{Tk+1} = 2\Delta_T$ とします．逆に一定比率以上で二次近似関数の減少量の方が上回った時は $\Delta_{Tk+1} = \frac{1}{2}\Delta_T$ として，どちらでもない場合は $\Delta_{Tk+1} = \Delta_T$ とします．そして次の反復点候補 $x_k + s_{\rho_k}$ が現在点での目的関数値 $f(x_k)$ をより減少させるのならば $x_{k+1} = x_k + s_{\rho_k}$，そうでなければ $x_{k+1} = x_k$ として次の反復へと進みます．

制約充足フェーズ，目的関数値最小化フェーズにおいて G_k がラグランジュ関数のヘッセ行列，あるいは近似行列であった場合，各反復において二計画問題を解くことになります．問題によっては，二次計画問題の求解が高コストになる事もあります．そのために，NUOPT では，この二次計画問題を直接解くのではなくて，線形計画問題と，その結果得られる等式のみの二次計画問題の解を組合せ，近似的な解を構成するという工夫をしています．

B.4 制約充足問題ソルバ wcsp

このアルゴリズムは離散変数，0-1 整数変数を変数とした制約充足問題：

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

をメタヒューリスティクスの解法の一つであるタブー・サーチを用いて解くものです．ここで，

X_j は各変数の定義域に対応する有限集合です.

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

この際,重みの大きな制約式は優先的に満たされるように解の探索が行われます.

重要度の高い制約式の重みを大きくすることで,より有用な解が期待できます.

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

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

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

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

制約充足問題ソルバは,以下のいずれかの条件を満たせば停止します..

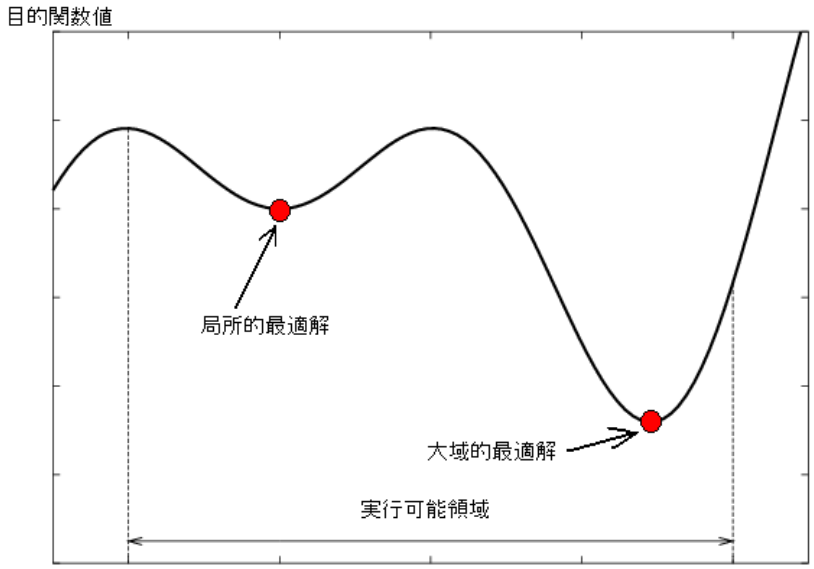
1. すべてのハード制約およびソフト制約を満たす解が求まった
   (目的関数に関しては目標値を満たす解が求まった)
2. 反復回数が指定の上限を超えた
3. 計算時間が指定の上限を超えた
4. ユーザが停止を命じた

本アルゴリズムは,京都大学「問題解決エンジン」グループの開発によるものです.詳細については [10], [11] をご参照下さい.

B.5 凸緩和法に基づく大域的最適化アルゴリズム

一般に,非線形最適化問題は,実行可能領域(制約条件を満たす領域)の中に複数の局所的最適解を持ちます.「局所的最適解」とは,「その近くでは最も良い解だが,実行可能領域全体では最も良いとは限らない解」のことです.これに対し,「大域的最適解」とは「実行可能領域全体で最も良いことが保証された解」となります[10].次は 1 変数の非線形な最小化問題(制約は変数の上下限のみ)における局所的最適解が複数存在する例です.

[10] 大域的最適解も局所的最適解の一つと言えます.



非線形最適化問題を解く際に適用する通常のアルゴリズムでは，「局所的最適解を一つ見つける」ことが目的とされています．これは，非線形最適化問題において大域的最適解を見つけることが一般に困難であるからに他なりません．

しかし，大域的最適解を求めるための研究は比較的古くから行われており，そのための手法の一つとして，組み合わせ最適化問題でよく用いられる「分枝限定法」の考え方を用いた手法が構成できます．

本手法は以下のようなステップを繰り返します．

1. （大域的最適解が存在する可能性のある）実行可能領域を分割し，分割された領域で「子問題」を生成して解く
2. 元の問題の実行可能解を何らかの方法で求める
3. １．２．の結果を用いて分割した各領域に大域的最適解が存在する可能性を判断し，存在しないと判断された領域を考慮から除外する

このアルゴリズムが機能するための前提条件として，各区間において生成された「子問題」が

『子問題の目的関数は，その実行可能領域において元の問題の目的関数を下から抑える』(*)

なる性質を持たなければなりません．したがって，このアルゴリズムを実行するには，制約式や目的関数の解析的な性質を用いながら上記の性質を持つように，子問題を生成する必要があります．

非線形関数として表された制約式や目的関数の形は非常に多様ですので,この子問題生成過程を自動化することは一般に困難です．しかし，定式の情報がモデリング言語を通じて，初等的な関数の合成（計算グラフ）として表現されていることを利用すれば，その過程を自動化，ユーザが特に意識することなく，大域的最適解の探索を行うことができます.

本アルゴリズムを用いる際に注意すべき点は,多数の子問題を繰り返し解くことによるメモリ消費です．本アルゴリズムで子問題を生成する仕組みは，組み合わせ最適化問題でよく用いられる分枝限定法に基づくものですが,組み合わせ問題の性質によっては分枝限定法が長い時間を要するのと同じように，本問題でも長い計算時間を要する可能性があります.

B.6 タブー・サーチによる資源制約スケジューリング問題解法

このアルゴリズムは以下の資源制約付きスケジューリング問題:

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

をタブー・サーチを用いたリストの探索を用いて解くものです.

アルゴリズムは wcsp と同じ京都大学「問題解決エンジン」グループによるものです．詳細については [18] をご覧下さい.

B.7 外点法

内点法は変数が正の領域（内点）に反復点を取る方法ですが，B.1.1 で取り上げた問題を

$$f(x) + \rho\sum_i |x_i|_-, \quad x \in \mathbf{R}^n,$$

$$g(x) = 0$$

のように変形して（実際にはこの問題の滑らかな近似を考えます)，非負制約を除去，この KKT 条件を解くと考えると，やはり直線探索・信頼領域法に基づく最適化アルゴリズムを構成することができ，大域的収束性を保証することができます． NUOPT に組み込まれている lepm は直線探索法に基づく外点法， tepm は信頼領域法に基づく外点法です.

内点法とほぼ同等のパフォーマンスを示しますが， ρ の大きさが相対的に小さいと非負制約を満たさない実行不可能な解に収束することはあり得ます．逆に実行不可能な問題でも，等式制約を満たし，なるべく最適解に近い点を返すことが期待されます． ρ の値はパラメータ exhro ですることができます.

内点法に比べてのメリットは，内点以外の点からも出発できるという点です．内点法では変数値を，非負制約を満たす内点に移動してから反復計算をスタートさせなければならないので，良い初期値が得られている場合でも，その操作により情報を失ってしまいます．外点法なら初期値を加工することなく，そのまま用いることができますので，良い初期値が得られている場合やリスタート時に比較的良好な性能が得られます.

付録ｃ　参考文献

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

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

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

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

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

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

[7] 逸見宣博，山下浩，計算グラフと分枝限定法を利用した大域的最適化－２，2004 年日本オペレーションズ・リサーチ学会秋季研究発表会アブストラクト集，42-43.

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

[9] 茨木俊秀・福島雅夫著，FORTRAN77 最適化プログラミング，岩波書店，1991.

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

[11] K. Nonobe and T. Ibaraki, An improved tabu search method for the

weighted constraint satisfaction problem, INFOR 39, 131-151, 2001.

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

[13] 矢部博，八巻直一，非線形計画法，朝倉書店，1999.

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

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

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

[17] 山下浩，逸見宣博，計算グラフと分枝限定法を利用した大域的最適化，2003 年日本オペレーションズ・リサーチ学会秋季研究発表会アブストラクト集，284-285.

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

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

索引

.

<

0

1

2

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z

あ

い

う

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

の

は

ひ

ふ

へ

ま

め

も

ゆ

ら

り