

Xpress-MP メモ¹

Ver 1.1

筑波大学システム情報工学研究科
社会システム・マネジメント専攻
最適化研究グループ

2006. 2. 16 初稿 (Ver1.0)

2006. 3. 1 改訂 (Ver1.1)

¹この原稿は、2006年3月に筑波大学システム情報工学研究科社会システム・マネジメント専攻で学位を取得した善家大輔氏が、周囲の研究室の学生とともに、Xpress-MP 初心者のために作成したマニュアルです。これまで同専攻の最適化研究グループで活用してきましたが、このたび「Xpress-MP 学習広場」のオープンに伴い、作成者の了解のもと、公開させていただいています。2006年当時の同専攻の計算機環境をもとに書かれており、お使いのシステム、ソフトウェアによっては当てはまらない記述が含まれていることをご承知おきください。

まえがき

Xpress-MP は、線形計画問題などを解くソフトウェアです。Dash 社 [1] により提供されています。使用者は mose1 言語でプログラムを書くこととなりますが、初心者でも少し使えば、mose1 言語の簡単さに驚かされるでしょう。しかし、日本語のマニュアルがないこともあり、使う前に躊躇している学生が多いのではないのでしょうか？折角、shakosv 上でも Xpress-MP を使用できるので、最大限活用して欲しいと思います。

このメモの構成は以下ようになっておりますが、初心者は 2 節を眺めるだけで十分だと思います。なお Example フォルダにこのメモのために作成したソースプログラムを添付します。

- 1 節 Xpress-MP を使う環境について
- 2 節 mose1 言語のプログラム例でイメージアップ
- 3 節 mose1 言語の基礎的事項をおさえる
- 4 節 知っておくと便利な使い方の紹介

このメモを作っているときのバージョン情報ですが、shakosv 上の Xpress-MP については Hyper 版 (容量無制限)、Xpress-IVE については、図 0.1 です。

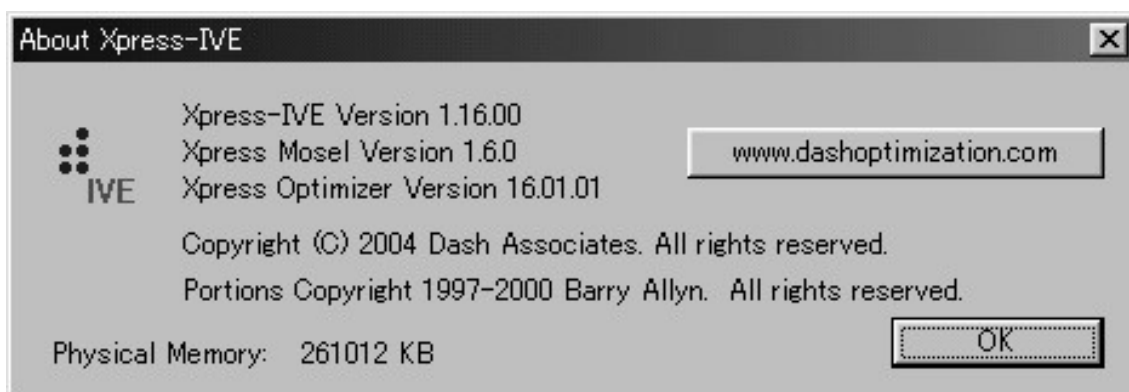


図 0.1: Xpress-MP のバージョン情報

困ったらヘルプを参照して下さい。このメモの作者は単なるユーザであり、使用して得た経験をまとめたに過ぎません。従って、このメモに起因する諸問題に関しては、作者は責任を負いかねます、ご了承下さい。

目次

1	Xpress-MP の使う環境	1
1.1	コンソール編 (shakosv にて)	1
1.2	Xpress-IVE 編	4
2	Xpress-MP では, こんな事ができます!	6
2.1	線形計画問題 (LP) を解くプログラム例 (LPsimple.mos)	6
2.2	整数計画問題 (IP) を解くプログラム例 (IPwithData.mos)	8
2.3	凸 2 次計画問題 (QP) を解くプログラム例 (QPwithData.mos)	11
2.4	0-1 ナップサック問題を解くプログラム例 (Knapsack.mos)	14
2.5	包絡分析法 (DEA) を解くプログラム例 (DEA.mos)	17
3	mosel 言語の基本的事項	22
3.1	主な予約語	22
3.2	model ブロックと uses 文	22
3.3	コメント文 !~ と標準出力 writeln()	23
3.4	宣言ブロック (declarations ~ end-declarations)	23
3.4.1	データ型 (real, integer, mpvar など)	23
3.4.2	定数の定義 (=)	24
3.4.3	集合 (set of, range)	24
3.4.4	配列 (array of)	24
3.5	二項演算と代入	25
3.5.1	二項演算 (+, -, *, / など)	25
3.5.2	代入演算 (:= など)	26
3.6	パラメータブロック (parameters ~ end-parameters)	27
3.7	条件式 (>=, <>, and, or など)	29
3.8	分岐とループ	29
3.8.1	分岐 (if, case 文)	29
3.8.2	ループ (forall, while, repeat 文など)	30
3.9	制約式に関して	31
3.9.1	非負制約を取り除く方法 (is_free)	31
3.9.2	整数制約と 0-1 制約 (is_integer, is_binary)	32
3.10	入力データをファイルから読み込む方法 (initializations from ~ end-initializations)	32
3.11	ユーザ関数の定義 (function, procedure)	33
3.12	関数リファレンス	33
3.12.1	数値計算関連 (sum, prod, round, abs など)	33
3.12.2	求解関連 (minimize, maximize)	34
3.12.3	ランダム関連 (random)	34

4	知っておくと便利かも!?	35
4.1	実行結果をファイルに出力する方法 (fopen など)	35
4.2	プログラムの実行時間を計測する方法 (gettime)	35
4.3	問題の有界性や実行不能性を表示する方法 (getprobstat)	35
4.4	分枝限定法の設定 (分枝方法, 子問題数の制限, 実行時間の制限など)	36
4.5	求解 (minimize, maximize) の途中で関数を呼び出す方法 (setcallback)	37
4.6	ループ中に制約式を取り除く方法 (sethidden)	37
4.7	変数を増やす方法 (create)	37
	謝辞	38
	参考文献	38

1 Xpress-MP の使う環境

Xpress-MP を使う環境には、主に

- (a) shakosv のコンソール上で Xpress-MP のコマンドを打ち込む。²
- (b) アプリケーションソフト Xpress-IVE を起動して実行ボタンをクリックする。
- (c) C 言語などで Xpress-MP をライブラリとして読み込む。

の3通りあるが、この節では (a), (b) について説明する。

1.1 コンソール編 (shakosv にて)

shakosv のコンソール上でコマンドを打ち込んで Xpress-MP を使う方法を説明します。注意事項として、shakosv にある Xpress-MP では、凸2次計画問題 (QP) は解けないなど若干の制限があります。しかし、単に線形計画問題を解くだけなら充分でしょう。整数計画問題も解けるので実習などでも活用できると思います。

手順 1: mosel 言語のプログラムをエディタで書く (図 1.1)。ただし、ファイルの拡張子は、‘‘mos’’ とする。例えば、以下のファイル ‘‘Ex01.mos’’ を作ってみましょう。

```
Ex01.mos
model Ex01
  uses "mmxprs"

  declarations
    a: mpvar
    b: mpvar
  end-declarations

  3*a + 2*b <= 400
  a + 3*b <= 200
  profit := a + 2*b

  maximize(profit)

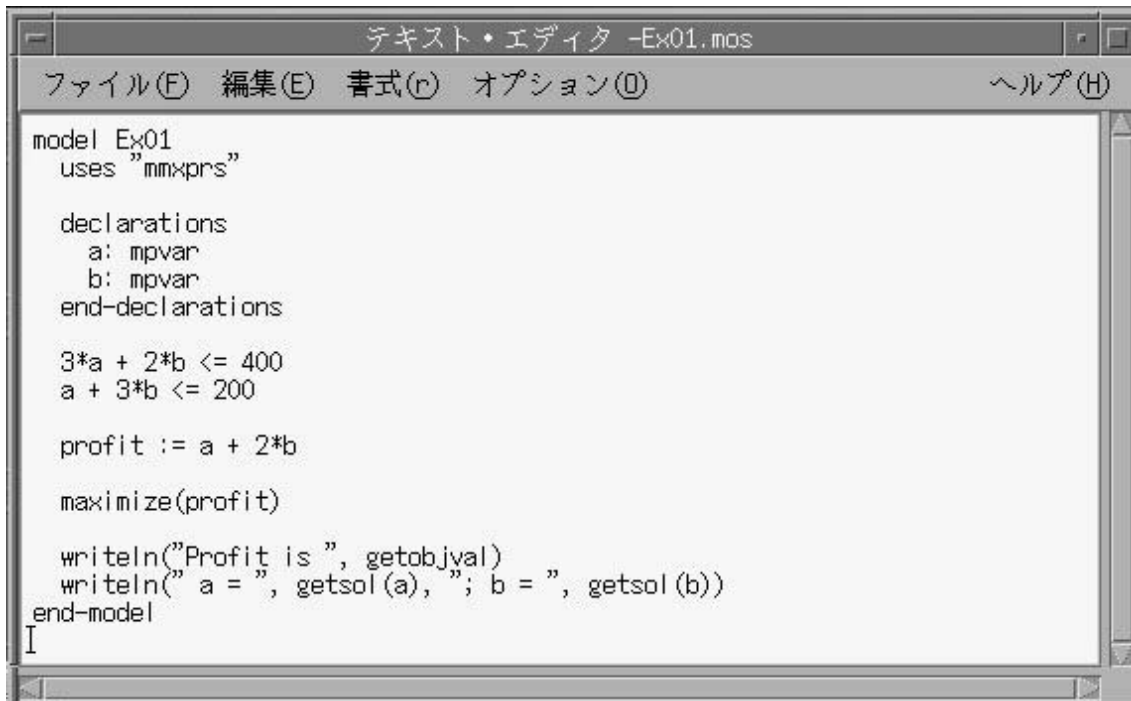
  writeln("Profit is ", getobjval)
  writeln(" a = ", getsol(a), "; b = ", getsol(b))
end-model
```

手順 2: コンソールを起動する。

手順 3: コンソール上で次のコマンドを実行してみましょう (図 1.2)。

```
shakosv> mp-model
> exec Ex01
> quit
```

² 社会システム・マネジメント専攻では平成 18 年度より Windows 上の Xpress-MP に移行しました。現在はこの方法のかわりに、次の (b) の方法を用います。



```
model Ex01
  uses "mxxprs"

  declarations
    a: mpvar
    b: mpvar
  end-declarations

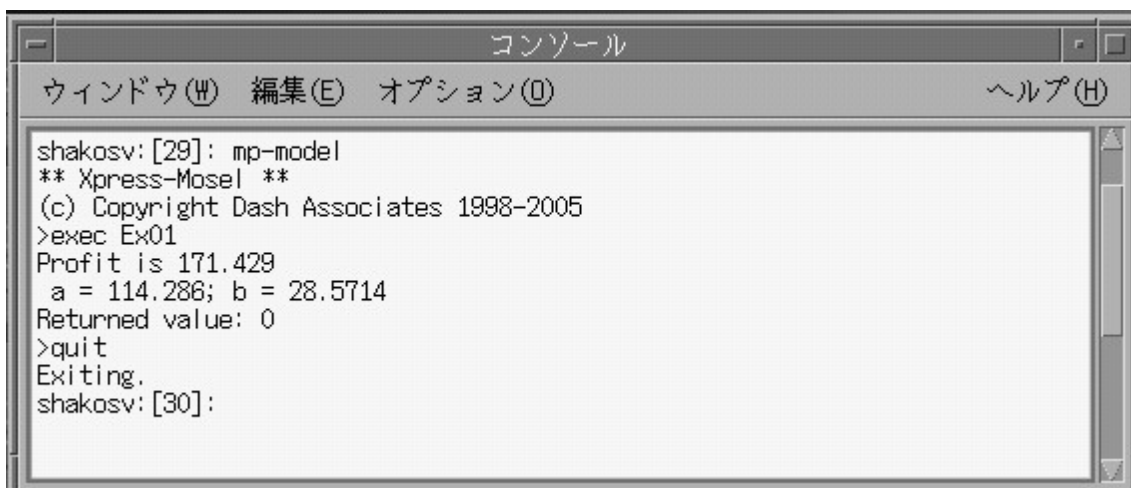
  3*a + 2*b <= 400
  a + 3*b <= 200

  profit := a + 2*b

  maximize(profit)

  writeln("Profit is ", getobjval)
  writeln(" a = ", getsol(a), "; b = ", getsol(b))
end-model
```

図 1.1: エディタでプログラムを作成する



```
shakosv: [29]: mp-model
** Xpress-Mosel **
(c) Copyright Dash Associates 1998-2005
>exec Ex01
Profit is 171.429
 a = 114.286; b = 28.5714
Returned value: 0
>quit
Exiting.
shakosv: [30]:
```

図 1.2: コンソール上で Xpress を実行

表 1.1: コンソール上で使う Xpress-MP の重要コマンド

コマンド	簡単な解説
mp-model	Xpress-MP を起動する. mp-mosel でもよい. shokosv 上 では mp- がつきます.
exec ファイル名 (.mos)	ファイルのプログラムをコンパイルして, 実行する.
quit	Xpress-MP を終了する.
help	ヘルプを表示する. コマンドの一覧が表示されます.
compile ファイル名 (.mos)	ファイルのプログラムをコンパイルする.
load ファイル名 (.mos)	ファイルのプログラムを読み込む.
cload ファイル名 (.mos)	ファイルのプログラムをコンパイルして, 読み込む.
run ファイル名 (.mos)	ファイルのプログラムを実行する. run とすると, 現在読み込んでいるプログラムを実行する.
list	現在, 読み込んでいるプログラムのリストを表示する.
exportprob -m ファイル名 (.mat)	読み込んでいる問題を MPS 形式で出力する.

他にも幾つかコマンドがあります.

1.2 Xpress-IVE 編

Xpress-IVE を起動して使う方法です。こちらの方が視覚的で簡単です。

手順 1: Xpress-IVE を起動する (図 1.3).

手順 2: mosel 言語のプログラムを書く (図 1.4)。ただし、拡張子は、‘mos’ とする。

手順 3: 実行ボタンを押す (図 1.4)。

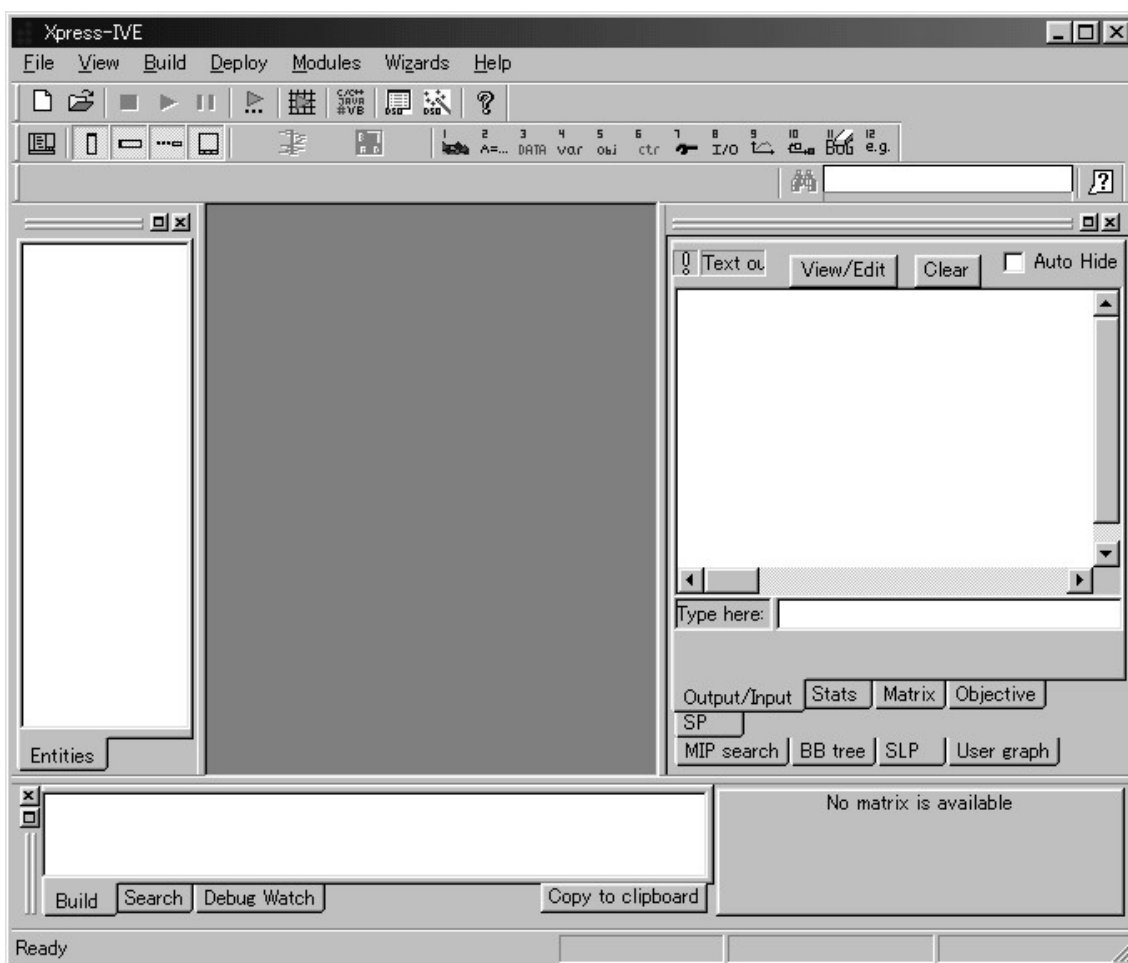


図 1.3: Xpress-IVE の起動画面

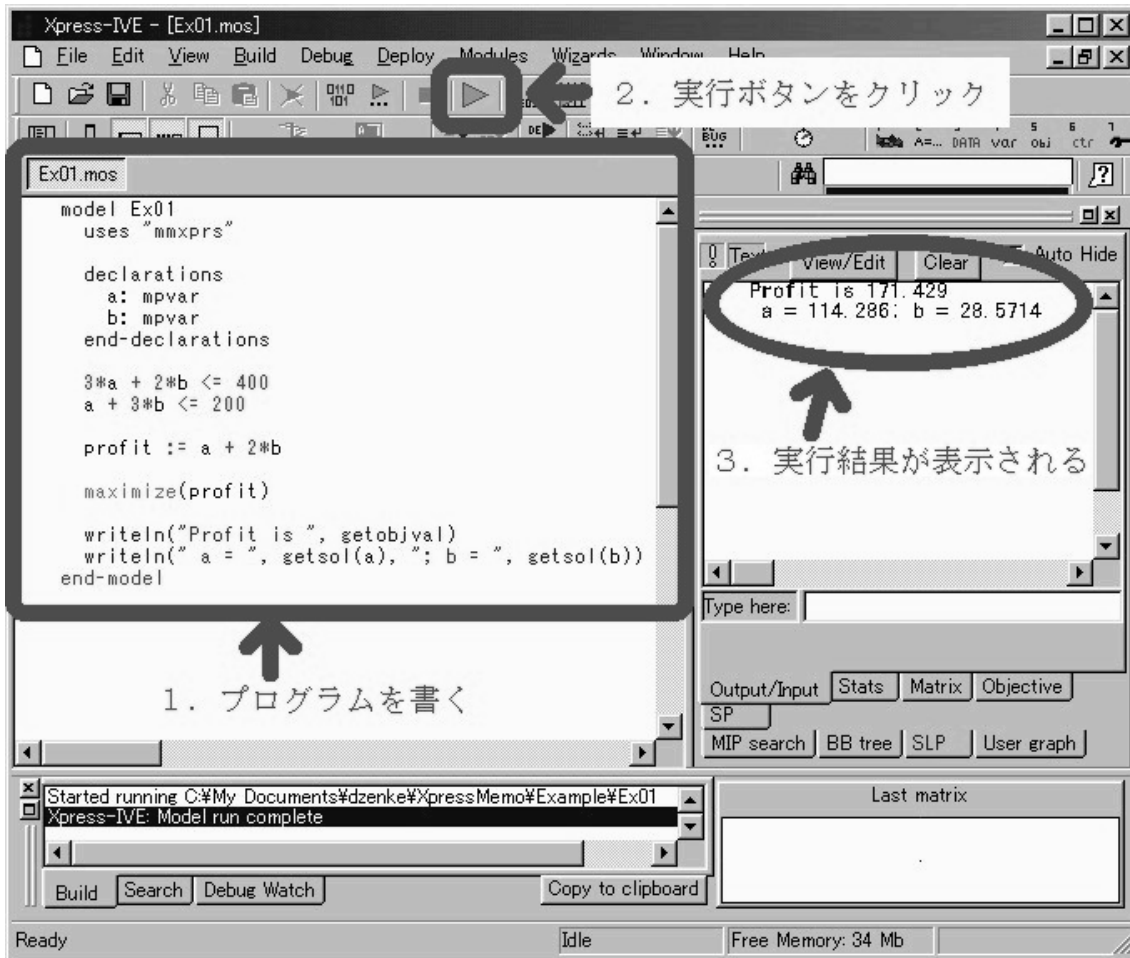


図 1.4: プログラムと実行ボタン

2 Xpress-MP では、こんな事ができます!

最初は、幾つかのプログラム例を見て、mose1 言語のイメージアップを図り、

「こんな事ができるんだ!」

と感じて頂きたい。取り敢えずの目標は、標準的な線形計画問題 (LP) と整数計画問題 (IP) が解けるようになることです。この節のプログラム例と 3 節 (22 ページ) を参照すれば、どんな (LP), (IP) も解けるようになるでしょう。

2.1 線形計画問題 (LP) を解くプログラム例 (LPSimple.mos)

最も基本的な線形計画問題:

$$(LP) \quad \left\{ \begin{array}{l} \min \quad cx \\ \text{s. t.} \quad Ax \geq b \\ \quad \quad x \geq 0, \end{array} \right.$$

を解いてみましょう。ここで、行列 A とベクトル b , c は、

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 0 & 2 \\ 2 & 1 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 4 \\ 5 \\ 7 \end{bmatrix}, \quad c = [3 \quad 2 \quad 4],$$

としておきます。(LP) を解くプログラムの例 ‘LPSimple.mos’ を次ページに書きます。

LPSimple.mos

```
1 model LPSimple
2   uses "mmxprs"
3
4   writeln("線形計画問題 (LP):")
5   writeln("| min  c x")
6   writeln("| s.t  Ax >= b, x >= 0")
7   writeln("を解くプログラム")
8
9   !!! A, b, c, x の宣言
10  declarations
11     I = 1..3                ! 制約の添字集合 I
12     J = 1..3                ! 変数の添字集合 J
13
14     A: array(I, J) of real   ! 行列 A
15     b: array(I) of real     ! ベクトル b
16     c: array(J) of real     ! ベクトル c
17
18     x: array(J) of mvar     ! 決定変数のベクトル x
19  end-declarations
20
21  !!! データの入力
22  writeln("> データを入力します.")
23  A := [1, 1, 2,
24        2, 0, 2,
25        2, 1, 3]
26  b := [4, 5, 7]
27  c := [3, 2, 4]
28  writeln(" A = ", A, "\n b = ", b, "\n c = ", c)
29
30  !!! 制約の定義 Ax >= b, x >= 0
31  writeln("> 制約式を定義する.")
32  forall (i in I) do
33    sum(j in J) A(i,j)*x(j) >= b(i)
34  end-do
35
36  !!! 目的関数の定義 ObjFunc := cx
37  writeln("> 目的関数を定義する.")
38  ObjFunc := sum(j in J) c(j)*x(j)
39
40  !!! 解を求める
41  writeln("> 求解")
42  minimize(ObjFunc)
43
44  !!! 最適値と最適解を表示
45  writeln(" 最適値 = ", getobjval)
46  forall (j in J) writeln(" x(", j, ") = ", getsol(x(j)))
47  end-model
```

実際に動かしてみると以下の実行結果が表示されます。

LPsimple.mos の実行結果

```

線形計画問題 (LP):
| min   c x
| s.t  Ax >= b, x >= 0
を解くプログラム
> データを入力します.
  A = [1,1,2,2,0,2,2,1,3]
  b = [4,5,7]
  c = [3,2,4]
> 制約式を定義する.
> 目的関数を定義する.
> 求解
最適値 = 9.5
x(1) = 0.5
x(2) = 0
x(3) = 2

```

プログラムの簡単な解説 (LPsimple.mos)

行	解説
1 ~ 47 行	model ブロックです。この間にプログラム本体を書きます。
2 行	mmxprs というライブラリを読み込みます。
4 行	文字列を表示するには、write か writeln 関数を使います。
9 行	! この記号から右は、コメント文です。
10 ~ 19 行	定数と変数を宣言します。mpvar が決定変数です。
23 ~ 27 行	行列 A とベクトル b, c の各データを入力しています。
32 ~ 34 行	制約式を定義しています。等式制約 $Ax = b$ の場合は、33 行目を $\text{sum}(j \text{ in } J) A(i,j)*x(j) = b(i)$ に変えます。 (注意) 変数の非負制約は書かなくとも自動的に設定されています。
38 行	目的関数となる式を定義しています。
42 行	minimize(式) か maximize(式) で線形計画問題を解く。
45 行	getobjval を用いて最適値を表示します。
46 行	最適解を取り出して表示します。 取り出すには getsol 関数を使います。

2.2 整数計画問題 (IP) を解くプログラム例 (IPwithData.mos)

次に、同じ A, b, c を用いて、整数計画問題 (IP) :

$$\begin{array}{l}
 \text{(IP)} \quad \left\{ \begin{array}{l}
 \min \quad cx \\
 \text{s.t.} \quad Ax \geq b \\
 \quad \quad x \geq 0 \\
 \quad \quad x \text{ は整数ベクトル,}
 \end{array} \right.
 \end{array}$$

を解いてみます。実際には, LPSimple.mos に

```
forall (j in J) x(j) is_integer
```

という制約式を一文追加するだけで終わりなのですが, それではつまらないので, 次のことを追加します.

- データ A, b, c をファイルから読み込む.
- parameters ブロックを使ってみる.
- 表示する procedure を作ってみる.

プログラムを作る際に, 以下の内容のテキストファイル ‘‘Abc.dat’’ を作っておきます.

```
Abc.dat  
!!! A, b, c  
A: [  
  1 1 2  
  2 0 2  
  2 1 3  
]  
  
b: [  
  4 5 7  
]  
  
c: [  
  3 2 4  
]
```

次のページに, プログラム例 ‘‘IPwithData.mos’’ を載せます.

```

IPwithData.mos
1  model IPwithData
2    uses "mmxprs"
3
4    !!! パラメータブロック
5    parameters
6      FName = "Abc.dat"
7    end-parameters
8
9    writeln("整数計画問題 (IP):")
10   writeln("| min  c x")
11   writeln("| s.t  Ax >= b, x >= 0, x is integer")
12   writeln("| を解くプログラム")
13
14   !!! A, b, c, x の宣言と入力
15   declarations
16     I = 1..3                ! 制約の添字集合 I
17     J = 1..3                ! 変数の添字集合 J
18     A: array(I, J) of real  ! 行列 A
19     b: array(I) of real     ! ベクトル b
20     c: array(J) of real     ! ベクトル c
21
22     x: array(J) of mvar     ! 決定変数のベクトル
23   end-declarations
24   initializations from FName
25     A b c
26   end-initializations
27   writeln(" A = ", A, "\n b = ", b, "\n c = ", c)
28
29   !!! プロシージャ定義
30   procedure PrintSol
31     writeln(" 最適値 = ", getobjval)
32     forall (j in J) writeln("  x(", j, ") = ", getsol(x(j)))
33   end-procedure
34
35   !!! 制約の定義 Ax >= b, x >= 0
36   writeln("> 制約式を定義する.")
37   forall (i in I) do
38     sum(j in J) A(i,j)*x(j) >= b(i)
39   end-do
40   forall (j in J) x(j) is_integer
41
42   !!! 目的関数の定義 ObjFunc := cx
43   writeln("> 目的関数を定義する.")
44   ObjFunc := sum(j in J) c(j)*x(j)
45
46   !!! 解を求める
47   writeln("> 求解")
48   minimize(ObjFunc)
49
50   !!! 最適値と最適解を表示 (関数呼び出し)
51   PrintSol
52 end-model

```

実際に動かしてみると以下の実行結果が表示されます。

IPwithData.mos の実行結果

```

整数計画問題 (IP):
| min   c x
| s.t. Ax >= b, x >= 0, x is integer
を解くプログラム
  A = [1,1,2,2,0,2,2,1,3]
  b = [4,5,7]
  c = [3,2,4]
> 制約式を定義する.
> 目的関数を定義する.
> 求解
最適値 = 10
x(1) = 2
x(2) = 0
x(3) = 1

```

プログラムの簡単な解説 (IPwithData.mos)

LPSimple.mos と異なる点のみ解説します。

行	解説
5 ~ 7 行	parameters ブロックです。文字列 FName を設定します。
24 ~ 26 行	行列 A とベクトル b, c の各データを入力しています。 FName で設定されているファイルから読み込みます。
30 ~ 33 行	プロシージャ PrintSol を定義しています。
40 行	整数制約を加えます。
51 行	定義したプロシージャ PrintSol を呼び出しています。

2.3 凸 2 次計画問題 (QP) を解くプログラム例 (QPwithData.mos)

凸 2 次計画問題:

$$(QP) \quad \begin{cases} \min & \frac{1}{2}x^T Qx + cx \\ \text{s.t.} & Ax \geq b \\ & x \geq 0, \end{cases}$$

も解いてみます。ここで、 A, b, c は前述のものとし、新たに追加された行列 Q を

$$Q = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

とします。まず、以下の入力データファイル ‘‘ABcQ.dat’’ を作ります。

AbcQ.dat

```
!!! A, b, c, Q  
A: [  
  1 1 2  
  2 0 2  
  2 1 3  
  ]  
b: [  
  4 5 7  
  ]  
c: [  
  3 2 4  
  ]  
Q: [  
  3 1 0  
  0 2 1  
  1 0 2  
  ]
```

次のページに、プログラム例 ‘QPwithData.mos’ を載せます。

QPwithData.mos

```
1 model QPwithData
2   uses "mmxprs", "mmquad"
3
4   parameters
5     FName = "AbcQ.dat"
6   end-parameters
7
8   writeln("凸 2 次計画問題 (QP):")
9   writeln("| min 1/2 x^T Q x + cx")
10  writeln("| s.t Ax >= b, x >= 0")
11  writeln("を解くプログラム")
12
13  !!! A, b, c, Q, x の宣言と入力
14  declarations
15    I = 1..3                ! 制約の添字集合 I
16    J = 1..3                ! 変数の添字集合 J
17    A: array(I, J) of real  ! 行列 A
18    b: array(I) of real     ! ベクトル b
19    c: array(J) of real     ! ベクトル c
20    Q: array(J,J) of real   ! 行列 Q
21
22    x: array(J) of mvar     ! 決定変数のベクトル x
23  end-declarations
24  initializations from FName
25    A b c Q
26  end-initializations
27  writeln(" A = ", A, "\n b = ", b, "\n c = ", c, "\n Q = ", Q)
28
29  !!! 制約の定義 Ax >= b, x >= 0
30  writeln("> 制約式を定義する.")
31  forall (i in I) do
32    sum(j in J) A(i,j)*x(j) >= b(i)
33  end-do
34
35  !!! 目的関数の定義 1/2 (x^T) Qx + cx
36  writeln("> 目的関数を定義する.")
37  ObjFunc := 1/2*sum(j1 in J, j2 in J) x(j1)*Q(j1,j2)*x(j2) + sum(j in J) c(j)*x(j)
38
39  !!! 解を求める
40  writeln("> 求解")
41  minimize(ObjFunc)
42
43  !!! 最適値と最適解を表示
44  writeln(" 最適値 = ", getobjval)
45  forall (j in J) writeln(" x(", j, ") = ", getsol(x(j)))
46 end-model
```

実際に動かしてみると以下の実行結果が表示されます。

QPwithData.mos の実行結果

```

凸 2 次計画問題 (QP):
| min 1/2 x^T Q x + cx
| s.t Ax >= b, x >= 0
を解くプログラム
A = [1,1,2,2,0,2,2,1,3]
b = [4,5,7]
c = [3,2,4]
Q = [3,1,0,0,2,1,1,0,2]
> 制約式を定義する.
> 目的関数を定義する.
> 求解
最適値 = 14.375
x(1) = 0.500587
x(2) = 3.59439e-008
x(3) = 1.99961

```

プログラムの簡単な解説 (QPwithData.mos)

行	解説
2 行	mmquad を読み込みます.
37 行	目的関数が 2 次関数です.

2.4 0-1 ナップサック問題を解くプログラム例 (Knapsack.mos)

ここからは、典型的な問題を用いて、少し高度なプログラム例を示します。完全に理解することは難しいかも知れませんが、飛ばして進めても構いませんが、目で追っかけるだけでもなんとなく理解できます。まずは、次に説明するナップサック問題を取り上げます。

重さと価値をもつ品物が幾つかあり、それをナップサックに入れる状況を考えます。ナップサックには(重さに関する)容量があるので、その容量以下で価値の総和を最大にする品物を選びたい。この問題をナップサック問題といいます。今、ナップサックの容量 (*Capacity*) を 100 とし、次のような品物が与えられたとします。

表 2.1: 品物一覧

品物名	時計	バック	テレビ	指輪	掛け軸	壺	銅像
重さ	20	30	50	10	25	40	70
価値	10	15	30	5	15	25	40

ここで、品物の集合を $Item$ 、品物 i に対する重さを $Weight(i)$ 、価値を $Value(i)$ とします。また、決定変数 $x(i)$ 、($i \in Item$) を

$$x(i) = \begin{cases} 1 & \text{(品物 } i \text{ をナップサックに入れるとき)} \\ 0 & \text{(その他),} \end{cases}$$

とします。このとき、ナップサック問題 (KP) は次のように定式化されます。

$$(KP) \quad \begin{cases} \max & \sum_{i \in Item} Value(i) \cdot x(i) \\ \text{s.t.} & \sum_{i \in Item} Weight(i) \cdot x(i) \leq Capacity \\ & x(i) \in \{0, 1\} \end{cases} \quad (\forall i \in Item).$$

(KP) を Xpress-MP で解くにあたって、次の入力データファイル ‘Knapsack.dat’ を作ります。

```
Knapsack.dat
!!! 品物 Item, 重さ Weight, 価値 Value
Item: [
  "時計" "バック" "テレビ" "指輪" "掛け軸" "壺" "銅像"
]
Weight: [
  20      30      50      10      25      40      70
]
Value: [
  10      15      30      5      15      25      40
]
]
```

次ページに (KP) を解くプログラム例 ‘Knapsack.mos’ を載せます。

Knapsack.mos

```
1 model Knapsack
2   uses "mxxprs"
3
4   parameters
5     FName    = "Knapsack.dat"    ! 入力ファイル名
6     Capacity = 100                ! ナップサックの容量
7   end-parameters
8
9   writeln("0-1 ナップサック問題を解くプログラム")
10
11  !!! Capacity, Weight, Value, x の宣言と読み込み
12  declarations
13    Item:    set of string          ! 品物の文字列集合 Item
14    Value:   array(Item) of integer ! 価値 Value
15    Weight:  array(Item) of integer ! 重さ Weight
16
17    x:      array(Item) of mpvar    ! 決定変数のベクトル x
18  end-declarations
19  initializations from FName
20    Item Weight Value
21  end-initializations
22
23  writeln("-----")
24  writeln("   商品 重さ 価値")
25  writeln("=====")
26  forall (i in Item) do
27    writeln(strfmt(i,8), strfmt(Weight(i),5), strfmt(Value(i),5))
28  end-do
29  writeln("袋の容量  ", Capacity)
30  writeln("-----")
31
32  !!! 変数を作る
33  forall (i in Item) create(x(i))
34
35  !!! 制約の定義 Weight * x <= Capacity, x is binary
36  writeln("> 制約式を定義する.")
37  sum(i in Item) Weight(i)*x(i) <= Capacity
38  forall (i in Item) x(i) is_binary
39
40  !!! 目的関数の定義 ObjFunc := Value * x
41  writeln("> 目的関数を定義する.")
42  ObjFunc := sum(i in Item) Value(i)*x(i)
43
44  !!! 解を求める
45  writeln("> 求解")
46  maximize(ObjFunc)
47
48  !!! 総価値と品物名を表示
49  writeln("   総価値 : ", getobjval)
50  write("   品物名 : ")
51  forall (i in Item | round(getsol(x(i))) = 1) write(i, " ")
52 end-model
```

実際に動かしてみると以下の実行結果が表示されます。

```

Knapsack.mos の実行結果
-----
0-1 ナップサック問題を解くプログラム
-----
商品 重さ 価値
=====
時計 20 10
バック 30 15
テレビ 50 30
指輪 10 5
掛け軸 25 15
壺 40 25
銅像 70 40
袋の容量 100
-----
> 制約式を定義する.
> 目的関数を定義する.
> 求解
総価値 : 60
品物名 : テレビ 指輪 壺

```

プログラムの簡単な解説 (Knapsack.mos)

行	解説
12 ~ 18 行	Item が文字列集合 set of string として定義されています。 その他の定数や変数は、Item を添え字とする配列ですが、 Item が入力されて初めて要素数が決定します。
27 行	strfmt("文字列", サイズ) は、表の縦軸を揃えるために使います。
33 行	変数は Item の入力によって自動的に作られないので、変数を作ります。
38 行	0-1 制約を加えます。
51 行	x(i) が 1 の品物の名前を表示しています。

2.5 包絡分析法 (DEA) を解くプログラム例 (DEA.mos)

ここでは、包絡分析法を題材にループ中で何度も線形計画問題を解く例を示します。制約式の格納 (linctr 型の使い方) や制約式の解除 (sethidden 関数) が鍵となります。

包絡分析法 (Data Envelopment Analysis, 以降 DEA と略す) は、1978 年に Charnes-Cooper らによって開発された多入力多出力システムの効率性を測るためのモデルである。今、 n 個の事業体 (Decision Making Unit, 以降 DMU と略す) が、 p 個のデータを入力し、 q 個のデータを出力するシステムを考える。ここで、第 k 事業体を DMU_k ($k = 1, \dots, n$) とし、第 k 事業体の入力データを X_{ik} ($i = 1, \dots, p$)、出力データを Y_{jk} ($j = 1, \dots, q$) とする。

DEA ではまず、分析対象となる DMU を指定する。入力データに指定した DMU にとって最も有利な重みを掛けて足し合わせ、入力量という一次元値にし、出力についても同様な重みを掛けて足し合わせ、出力量という一次元値にする。ここで、第 i 入力データに対する重みを v_i とし、第 j 出力データに対す

る重みを u_j とする。これらの重みは後に定義される問題の決定変数となる。その後、出力量を入力量で割ることで、指定した DMU の効率性を測る。

実際には、 DMU_k ($k = 1, \dots, n$) ごとに分数計画問題:

$$(DEA(k)) \quad \left\{ \begin{array}{l} \max \quad \theta_k = \frac{\sum_{j=1}^q Y_{jk} u_j}{\sum_{i=1}^p X_{ik} v_i} \\ \text{s. t.} \quad \frac{\sum_{j=1}^q Y_{jd} u_j}{\sum_{i=1}^p X_{id} v_i} \leq 1 \quad (d = 1, \dots, n) \\ u_j \geq 0 \quad (j = 1, \dots, q) \\ v_i \geq 0 \quad (i = 1, \dots, p). \end{array} \right.$$

を解くことになるが、この問題は同値な線形計画問題:

$$(DEA(k)) \quad \left\{ \begin{array}{l} \max \quad \theta_k = \sum_{j=1}^q Y_{jk} u_j \\ \text{s. t.} \quad \sum_{i=1}^p X_{ik} v_i = 1 \\ \sum_{j=1}^q Y_{jd} u_j \leq \sum_{i=1}^p X_{id} v_i \quad (d = 1, \dots, n) \\ u_j \geq 0 \quad (j = 1, \dots, q) \\ v_i \geq 0 \quad (i = 1, \dots, p). \end{array} \right.$$

に書き直せることが知られている。問題 $(DEA(k))$ の最適値 θ_k^* は DMU_k の効率値と呼ばれ、 $\theta_k^* = 1$ であるとき、 DMU_k は効率的であるといい、 $\theta_k^* < 1$ であるとき、 DMU_k は非効率的であるという。

例として日本におけるスポーツ振興くじの運営の効率性を計算し、海外との比較を行ってみる。使用したデータは、

表 2.2: 入力データと出力データ

DMU	オーストリア	フランス	ドイツ	イタリア	オランダ	スペイン	スウェーデン	スイス	日本
投資額 (入力 1)	8189	60469	82689	58093	16299	43064	9041	7252	128085
維持費 (入力 2)	318	2216	2906	1836	629	1120	383	384	4799
売上額 (出力)	16	70	487	577	11	387	308	19	112

であり、ループを用いて各 $k \in \{\text{オーストリア}, \dots, \text{日本}\}$ について $(DEA(k))$ を解きます。Xpress-MP で解くにあたって、次の入力データファイル ‘DEA.dat’ を作ります。

```

DEA.dat
!!! 入力名
In: [
  "投資費" "維持費"
]

!!! 出力名
Out: [
  "売上額"
]

!!! 分析対象
DMU: [
  "オーストリア" "フランス" "ドイツ" "イタリア" "オランダ" "スペイン" "スウェーデン" "スイス" "日本"
]

!!! 入力データ (投資額, 維持費) × 各国
X: [
  8189 60469 82689 58093 16299 43064 9041 7252 128085
  318 2216 2906 1836 629 1120 383 384 4799
]

!!! 出力データ (売上額, 単位: 百万ユーロ)
Y: [16 70 487 577 11 387 308 19 112]

```

次ページに $(DEA(k))$ を解くプログラム例 ‘DEA.mos’ を載せます。

DEA.mos

```
1 model DEA
2   uses "mmsxprs"
3
4   writeln("-----")
5   writeln(" 包絡分析法 (DEA) \n coded at Feb.5.2006 by T. 奥田 featuring D.Z ")
6   writeln("=====")
7
8   !!!! 定義 1
9   declarations
10    In:   set of string           ! 入力データ集合
11    Out:  set of string           ! 出力データ集合
12    DMU:  set of string           ! 分析対象 (DEA) の集合
13    X:    array (In, DMU) of real ! 入力データ行列
14    Y:    array (Out, DMU) of real ! 出力データ行列
15    v:    array (In)      of mpvar ! 入力の重みベクトル
16    u:    array (Out)     of mpvar ! 出力の重みベクトル
17    Const: array(DMU)     of lincpr ! 制約式
18  end-declarations
19
20  !!!! データ入力 1
21  initializations from "DEA.dat"
22    In Out DMU X Y
23  end-initializations
24
25  !!! 変数を作る.
26  forall (i in In) create(v(i))
27  forall (j in Out) create(u(j))
28
29  !!! 基本制約  $Y_u \leq X_v, u \geq 0, v \geq 0$ 
30  forall (d in DMU) do
31    sum (j in Out) Y(j,d)*u(j) <= sum(i in In) X(i,d)*v(i)
32  end-do
33
34  !!! ループ
35  forall (k in DMU) do
36    !!! 毎回変わる制約
37    Const(k) := sum (i in In) X(i,k)*v(i) = 1
38    !!! 目的関数
39    ObjFunc := sum(j in Out) Y(j,k)*u(j)
40    !!! 求解
41    maximize(ObjFunc)
42    !!! 表示
43    writeln(k, "\n 効率値 : ", getobjval)
44    forall (i in In) writeln(" ", i, "重み = ", getsol(v(i)))
45    forall (j in Out) writeln(" ", j, "重み = ", getsol(u(j)))
46    writeln("-----")
47    !!! 制約を忘れる
48    sethidden(Const(k), true)
49  end-do
50 end-model
```


実際に動かしてみると以下の実行結果が表示されます。

DEA.mos の実行結果

包絡分析法 (DEA)

coded at Feb.5.2006 by T. 奥田 featuring D.Z

オーストリア

効率値 : 0.0625664

投資費重み = 0

維持費重み = 0.00314465

売上額重み = 0.0039104

フランス

効率値 : 0.0392804

投資費重み = 0

維持費重み = 0.000451264

売上額重み = 0.000561149

(長いので、この間は省略します)

スイス

効率値 : 0.0769062

投資費重み = 0.000137893

維持費重み = 0

売上額重み = 0.0040477

日本

効率値 : 0.0290212

投資費重み = 0

維持費重み = 0.000208377

売上額重み = 0.000259118

プログラムの簡単な解説 (DEA.mos)

行	解説
17 行	linctr 型の制約式の配列 Const を定義します。
35 ~ 39 行	ループ中に繰り返して線形計画問題を解きます。
37, 38 行	k ごとの制約や目的関数を定義します。
48 行	sethidden を用いて次のループに入る前に k に関する制約を解除します。

3 mosel 言語の基本的事項

この節では, mosel 言語を書く上で最低限知っておいた方がよい事項を説明します.

3.1 主な予約語

次の語は予約語なので, 定数名としては使わないようにしましょう.

and, array, as, boolean, break, case, declarations, div, do, mpvar, dynamic, elif, else, end, false, forall, forward, from, function, if, in, include, initialisations, initializations, integer, inter, is_binary, is_continuous, is_free, is_partint, is_semcont, is_semint, is_sos1, is_sos2, linctr, max, min, mod, model, next, not, of, options, or, parameters, procedure, public, prod, range, real, repeat, set, string, sum, then, to, true, union, until, uses, while.

3.2 model ブロックと uses 文

model モデルの名前

プログラム本体

end-model

(説明) model ブロックを作成します. この間にプログラムを書きます.

uses "ライブラリ名"

(説明) ライブラリを読み込みます.

表 3.1: 主なライブラリー覧

ライブラリ名	解説
mmxprs	線形式を表現したり, 線形計画問題を解くために必須です. 整数計画問題も解けます.
mmquad	2次計画問題を解く際に必要になります.
mmsystem	時間を計測したり, ファイル生成や削除などをする際に必要です.
mmodbc	Excel とリンクしたり, SQL コマンドが使えるらしい.
mmive	グラフが書けるっぽい.

3.3 コメント文 !~ と標準出力 writeln()

```
! ~, または, (! ~ !)  
(説明) コメント文を書きます.  
!   この記号より右側はコメント文です  
(!  
   ちなみに, コメントが複数行にわたるときはこのようにします.  
!)
```

```
write(), または, writeln()  
(説明) write("文字列"), または, writeln("文字列") で文字列を表示します.
```

write() は改行されませんが, writeln() は表示後改行されます. また, 「,」 で区切るにより, 続けて表示できます. 例えば, writeln(" A = ", A) と書けば, A = に続いて A の要素が表示された後改行されます. ただし, LPSimple.mos の実行結果からも分かるように, A が行列であっても一次元配列と同様に一行で表示されてしまいます. ちなみに, write("\n"), または, writeln と書くと改行します.

3.4 宣言ブロック (declarations ~ end-declarations)

```
declarations  
  記号 : 型名  
  記号 1, 記号 2, ... : 型名      ! 複数の場合 「,」 で区切る.  
  記号 = 定数  
end-declarations  
(説明) 定数と変数を宣言します.
```

3.4.1 データ型 (real, integer, mpvar など)

mosel 言語では, 次表の型が使用できます.

表 3.2: 型名の一覧

型名	解説
integer	-214783648 ~ 214783648 の整数値
real	-1.7e+308 ~ 1.7e+308 の実数値
string	文字列
boolean	ブール値, true か false
mpvar	決定変数
linctr	線形制約式

3.4.2 定数の定義 (=)

定数を作るには以下のように書きます。

記号 = 定数

(説明) 定数を定義します。

(例)

```
STR    =  "My Const String"    !  文字列定数
MyVal  =  5.0                   !  実数定数
S      =  {2, 0.1, "a", "e"}   !  集合定数
Hanni  =  1..4                  !  Hanni = {1, 2, 3, 4} と同意
```

(注意)

定数は、declarations ブロック中、または、後述する parameters ブロック中で設定します。それらのブロック外では制約式扱いになります。従って、ブロック外で STR = "String" と書いた場合、「STR が定義されていません」というエラーメッセージが出ます。

3.4.3 集合 (set of, range)

{a, b, c} のように中括弧で囲むと集合になります。また、2..7 は {2, 3, 4, 5, 6, 7} と同じ意味になります。要素は、3.4.1 節の型から取ることができ、例えば、{1, 2, 3} は要素が整数の集合ですが、{"1", "2", "3"} は要素が文字列の集合となります。これらは、要素数が定義されると同時に静的に決まっていますが、要素数が動的に決定される集合を作るには、以下の set of または range を用います。

set of 型名

(説明) 型の要素を持つ集合を用意します。要素数は動的に決定されます。

(例)

```
S1:  set of integer
S2:  set of string など
```

range

(説明) 範囲を表す型です。要素数は動的に決定されます。

(例) R1: range

3.4.4 配列 (array of)

配列を作るには、次の array を用います。

array (集合, 集合, ...) of 型名

(説明) 集合分だけ型の配列を用意します.

`dynamic` を `array` の前に書くと動的に確保します.

(例)

A1: `array (1..2, 1..5) of integer`

A2: `array ({1.5, 2.3}, 1..3, {"上", "中", "下"}) of real`

A1 は整数を要素とする 2×5 の 2 次元配列で, 参照するときは, `A1(2,3)` などと書く. A2 は実数を要素とする $2 \times 3 \times 3$ の 3 次元配列で, 参照は, `A2(1.5, 2, "下")` などと書く.

(注意 1)

3.4.3 節では, `S = {1, 2, 3}` と書けば, 要素数が 3 の集合が作られました. しかし, 配列の場合, いきなり `T = [1, 2, 3]`, または, `T := [1, 2, 3]` と書くとエラーになります. 配列 T を作るには, まず, `declarations` ブロックにて

```
T: array (1..3) of integer
```

と要素が 3 つの配列を作っておいて, `T := [1, 2, 3]` のように値を代入します (代入については後述).

(注意 2)

集合に含まれていない添え字の要素を参照しようとするエラーとなる. 例えば, 前述の T に対して, `T(5)` や `T("上")` を参照しようとする `5` や `"上"` が集合 `1..3` に含まれていないのでエラー.

3.5 二項演算と代入

3.5.1 二項演算 (+, -, *, / など)

Xpress-MP では, 以下の二項演算が可能です.

表 3.3: 二項演算の一覧

演算	解説	A, B の型
<code>A + B</code>	A と B の和	整数, 実数, 集合, 文字列
<code>A - B</code>	A から B への差	整数, 実数, 集合, 文字列
<code>A * B</code>	A と B の積	整数, 実数, 集合
<code>A / B</code>	A から B への商	整数, 実数
<code>A ^ B</code>	A の B 乗	整数, 実数
<code>A mod B</code>	A から B を割った余り	整数, 実数

(例) `{1, 2, 3} + {3, 4} = {1, 2, 3, 4}`, `{1, 2, 3} - {3, 4} = {1, 2}`, `{1, 2, 3} * {3, 4} = {3}` となる. また, `"今日は-"は"+"は暑い" = "今日は暑い"` となる.

3.5.2 代入演算 (:= など)

代入演算は以下のように書きます。

記号 := 式

(説明) 記号に式を代入する。または、記号を定義する。

(例)

```
B      := 5           ! 整数代入
STR    := "Hello"    ! 文字列代入
V      := [1, 2, 3]  ! 配列代入
Func   := 5*x + 3    ! 式代入, 目的関数などの定義
```

(注意 1)

Xpress-MP では、等式や不等式も代入することができます。例えば、

```
Const1 := 5*x + 3 = 0 ! 等式
Const2 := x >= 30     ! 不等式
```

と書けば、制約式を格納できます。

(注意 2)

declarations ブロック中で用意されている記号以外でも、代入操作ができます。例えば、B という記号が宣言されていない場合でも B := 2 と書けば、整数型の B が作られます。ただし、3.4.4 節で述べた通り、配列に関しては既に用意されていなければなりません。

(注意 3)

左辺と右辺の型が一致しなければなりません。また、declarations ブロック中で宣言されていない B と C に関して、B := 2 と C := 2.0 と書くと、意味が違ってきます。前者は整数型として設定されますが、後者は実数型です。従って、この後 B := C という代入演算を行おうとするとエラーになります。

(注意 3)

定数の定義 (=) とは、本質的に違います。declarations ブロック、または、parameters ブロック中で := を用いるとエラーになります。

他に、次のような代入演算が可能です。

表 3.4: 代入演算の一覧

演算	解説
A += B	A := A + B と同意
A -= B	A := A - B と同意

3.6 パラメータブロック (parameters ~ end-parameters)

以下を用いれば、プログラムを実行する前に定数を設定できます。

```
parameters
  記号 = 定数
end-parameters
(説明) 定数を設定します。
また、実行オプションでプログラムを実行する前に、パラメータを変更することができます。
```

(例)

```
parameters
  M      = 15          ! 整数
  PI     = 3.14        ! 実数
  FName  = "Abc.dat"  ! 文字列
end-parameters
```

とする。前述の IPwithData.mos を解くとき、コンソール上で

```
> exec IPwithData FName="Abc2.dat"
```

とすると、実行の際、入力データファイルが "Abc2.dat" に自動的に置き換えられます。Xpress-IVE の場合は、実行オプションで設定します (図 3.1, 図 3.2 参照)。

(注意 1)

declarations ブロックと異なり、集合を定義することはできません。つまり、parameters ブロック中に $S = \{1, 2, 3\}$ と書くとエラーになります。

(注意 2)

declarations ブロックは、model ブロック中に幾つも書けますが、parameters ブロックは、model ブロック中にひとつのみで、また uses との間に他の命令があってははいけません。つまり、

```
model DEMO
  uses "mmxprs"
  writeln("表示") ! parameters ブロックの前に他の命令があるとエラー
  ...
  parameters
    記号1 = 定数
  end-parameters
  ...
  parameters ! paramters ブロックが 2 つあるとエラー
    記号2 = 定数
  end-parameters
```

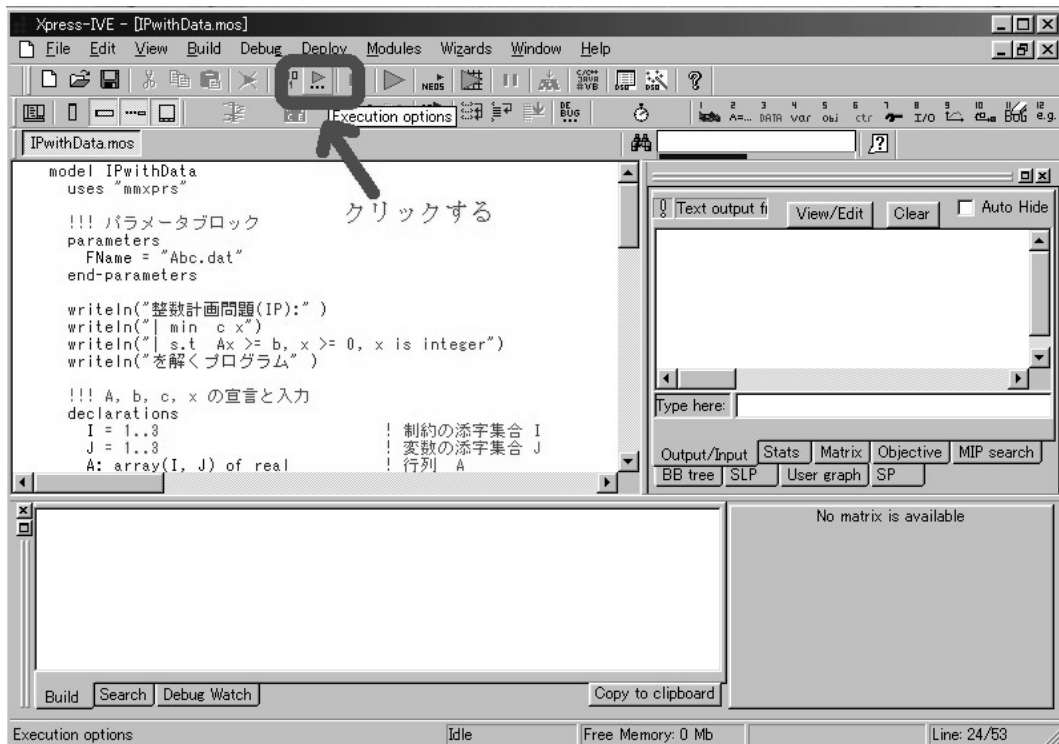


図 3.1: 実行オプション



図 3.2: パラメータ変更

3.7 条件式 (\geq , $<$, $>$, and, or など)

以下は Xpress-MP の条件式の一覧です。ここで, true と false は, boolean 型です。

表 3.5: 条件式の一覧

条件式	解説
$A < B$	$A < B$ なら true, それ以外は false
$A > B$	$A > B$ なら true, それ以外は false
$A \geq B$	$A \geq B$ なら true, それ以外は false
$A \leq B$	$A \leq B$ なら true, それ以外は false
$A = B$	$A = B$ なら true, それ以外は false
$A <> B$	$A \neq B$ なら true, それ以外は false
命題 1 and 命題 2	命題 1 と命題 2 が同時に true なら true, それ以外は false
命題 1 or 命題 2	命題 1 か命題 2 の一方が true なら true, それ以外は false

3.8 分岐とループ

Xpress-MP の分岐とループについて紹介します。

3.8.1 分岐 (if, case 文)

```
if (条件式) then
  命令文
elif (条件式) then
  命令文
else
  命令文
end-if
```

(注意) if は then とセットであり改行も意味を持つので, 次の書き方はエラーになります。

```
if (条件式) 命令文
```

または,

```
if (条件式) then 命令文 end-if
```

```
case 記号 of
  式: 命令文
  式: do 命令文 end-do
  式: do 命令文 end-do
  else 命令文
end-case
```

(例)

```
case B of
  1 : writeln("処理 1")           ! B が 1 のとき
  2..5 : writeln("処理 2")        ! B が 2~5 の整数のとき
  6, 8 : do writeln("処理 3") end-do ! B が 6 か 8 のとき
  else writeln("処理 4")         ! その他
end-case
```

3.8.2 ループ (forall, while, repeat 文など)

Xpress-MP のループ表現は次のように幾つかありますが、細かい説明は不要でしょう。

```
forall (要素 in 集合) 命令文, または, 命令文が複数行の場合は,
forall (要素 in 集合) do
  命令文
end-do
(説明) 集合に含まれる全ての要素について, 命令文を繰り返します.
また forall (要素 in 集合 | 条件式) で条件を満たす要素に限定できます.
```

```
while (条件式) 命令文, または, 命令文が複数行の場合は,
while (条件式) do
  命令文
end-do
(説明) 条件が満たされている間, 命令文を繰り返します. 条件の判定が最初.
```

```
repeat
  命令文
until (条件式)
(説明) 条件が満たされている間, 命令文を繰り返します. 条件の判定が最後.
```

```
break [n]
(説明) [ n 回上の ] ループを抜けます.
```

```
next [n]
```

(説明) [n 回上の] ループの先頭まで戻ります.

(注意) forall を使うときは、ループの途中で要素を変更できない.

例えば、次の書き方はエラーになります.

```
forall (i in 1..9) do
  if (i = 3) then
    i := 7      !    これが許されない
  end-if
end-do
```

ループの途中で要素を変えたいならば、

```
i := 1
while (i < 10) do
  if (i = 3) then
    i := 7
  end-if
  i += 1
end-do
```

のように while を用いて書きます.

3.9 制約式に関して

変数を含んだ等式か不等式を書きますと、制約式となります.

3.9.1 非負制約を取り除く方法 (is_free)

前述した通り、決定変数は自動的に非負に制限されています. これを取り除くには、例えば、

$$x(1) \geq -1000$$

のような制約を書けば、変数 $x(1)$ に関する非負制約は解除されます. その他、下記のような制約を追加しても解除されます.

```
変数 is_free
```

(説明) 変数の非負制約を解除し、 $-\infty$ まで取れるようにする.

(例) $x(1)$ が $-\infty$ まで取れるようにするためには、以下のような制約を書きます.

```
x(1) is_free
```

3.9.2 整数制約と 0-1 制約 (is_integer, is_binary)

他にも幾つかありそうですが、以下だけ覚えておけば十分でしょう。

変数 is_integer

(説明) 変数の整数制約を加えます。

変数 is_binary

(説明) 変数の 0-1 制約を加えます。

3.10 入力データをファイルから読み込む方法 (initializations from ~ end-initializations)

入力データをファイルから読み込むには、以下を用います。

initializations from "ファイル名"

記号 記号 ...

end-initializations

(説明) ファイルからデータを取ってきて、記号にセットする。

(データファイルにおけるデータの書き方)

```
r : 1.5          ! 実数値
z : 2            ! 整数値
M : [           ! 行列
    1 0 0 0
    0 0 1 2
    0 0 0 3
]
```

(ゼロが多い行列の場合の効率的な書き方) 上記の行列 M は次のように書くと効率的です。

```
M : [           ! 行列
    (1 1) 1      !   1 行 1 列目が 1
    (2 3) 1 2    !   2 行 3 列目から 1, 2 と続く
    (3 4) 3
]
```

(3 次元以上のデータの場合) 基本的に上記の M の書き方を応用します。

```
D : [           ! 2 x 2 x 3 の 3 次元データの格納
    (1 1 1) 1 2 3 !   (1,1,1) から (1,1,3) の要素までを順に書く。
    1 1 1         !   改行すると (1,2,1) から (1,2,3) の要素になる。
    (2 1 1) 3 3 1 !   (2,1,1) から (2,1,3) まで
    5 2 7         !   (2,2,1) から (2,2,3) まで
]
```

3.11 ユーザ関数の定義 (function, procedure)

ユーザ関数を用いれば、プログラムが見やすくなることがあります。 Xpress-MP では、戻り値がある function と戻り値がない procedure が用意されています。

```
procedure プロシージャ名 [(引数:型, ...)]
```

プロシージャの定義

```
end-procedure
```

(説明) プロシージャを定義します。

```
function 関数名 [(引数:型, ...)] : 戻り値の型
```

関数の定義

```
returned := 式 ! これが戻り値となる.
```

```
end-function
```

(説明) 関数を定義します。

これらは基本的に、関数呼び出しの記述がある前に定義されていなければなりません。関数を呼び出す記述の後で定義するためには、事前に以下のように宣言しておきます。

```
forward procedure プロシージャ名 [(引数:型, ...)]
```

```
forward function 関数名 [(引数:型, ...)] : 戻り値の型
```

(説明) 呼び出す記述の後でプロシージャや関数を定義するために宣言する。

3.12 関数リファレンス

ここでは、重要だと思われる関数だけ簡単に説明します (逐次追加予定)。詳細は、ヘルプを参照して下さい。

3.12.1 数値計算関連 (sum, prod, round, abs など)

```
sum (要素 in 集合) 式
```

(説明) 集合に含まれる全ての要素を式に代入して足し合わせます。

```
sum (要素 in 集合 | 条件) とすれば要素に条件をつけることもできます。
```

```
prod (要素 in 集合) 式
```

(説明) 集合に含まれる全ての要素を式に代入して掛け合わせます。

```
prod (要素 in 集合 | 条件) とすれば要素に条件をつけることもできます。
```

```
round (値)
```

(説明) 四捨五入します。

```
abs (値)
```

(説明) 絶対値を返します。

3.12.2 求解関連 (minimize, maximize)

`minimize(式)`, または, `maximize(式)`

(説明) 式を目的関数とする問題を解きます. `getobjval` に最適値が格納されます.

`getsol(変数)`

(説明) 解を取り出す.

(注意)

決定変数 x がベクトルの場合, `getsol(x)` とはできません. `forall (j in J) getsol(x(j))` のように要素をひとつずつ取り出さなければなりません.

3.12.3 ランダム関連 (random)

`random`

(説明) $[0, 1)$ の乱数を生成します.

(例) $[1, \text{Num}]$ の整数を乱数で生成する方法

`round(random*Num + 0.5)`

`setrandseed (値)`

(説明) 乱数の種を設定します.

4 知っておくと便利かも!?

4.1 実行結果をファイルに出力する方法 (fopen など)

実行結果をファイルに保存するには、コンソール上では、script コマンドなどで typescript を作れば良く、Xpress-IVE では、標準出力をコピーしてテキストエディタなどに貼り付ければ良いわけですが、この操作は若干面倒です。そこで、実行中にテキストファイルを作り、自動的に保存する方法を以下に説明します。

```
fopen("ファイル名", F_OUTPUT)    ! ファイルを開く。  
fclose(F_OUTPUT)                ! ファイルを閉じる。  
(説明) F_OUTPUT を開いている間に表示した文字はファイルに保存されます。
```

(例)

```
fopen("Log.txt", F_OUTPUT)  
writeln("最適値 = ", getobjval)  
fclose(F_OUTPUT)
```

4.2 プログラムの実行時間を計測する方法 (gettime)

数値実験などで実行時間を計測したいときがあります。その場合は、gettime を用います。

```
gettime  
(説明) 現在の時間を返します。
```

(注意) uses "mmxprs", "mmsystem" のように、mmsystem を読む必要があります。

(例) 時間の計測方法

```
時間計測  
StartTime := gettime                計測開始  
何らかの処理  
writeln("実行時間 = ", gettime - StartTime, "秒")  計測終了
```

4.3 問題の有界性や実行不能性を表示する方法 (getprobstat)

Xpress-MP では、問題が非有界や実行不能であっても、標準出力にはそのことが表示されないのが、問題が解けたように勘違いしてしまうことがあります。問題のステータスを知るためには getprobstat を用います

getprobstat

(説明) 問題を解いた後に、ステータスを返します。

getprobstat の値とステータスの関係は以下の通りです。

表 4.1: 問題のステータスと getprobstat の値

getprobstat の値	ステータス
2	最適解を得ました。
4	途中で停止しました。
6	問題が実行不能です。
8	問題が非有界です。

4.4 分枝限定法の設定 (分枝方法, 子問題数の制限, 実行時間の制限など)

setparam を使えば、分枝限定法の分枝方法を変更したり、子問題数や実行時間の制限をすることができます。

setparam("予約定数", 値)

(説明) 予約されている定数, パラメータに値を代入します。

予約定数はたくさんありますので、ヘルプを見てください。

(例) 分枝方法の変更

setparam("XPRS_NODESELECTION", 値) ! 分枝方法を変更します。値は下表 4.2。

表 4.2: 分枝方法

値	名称	説明
1	Local first	子孫と姉妹のノードがあれば、その中から最良な下界値をもつノードを分枝し、無ければ全てのノードから最良な下界値をもつノードを分枝します。
2	Best first	全てのノードの中から最良な下界値をもつノードを分枝します。
3	Local depth first	子孫と姉妹のノードがあれば、その中から最良な下界値をもつノードを分枝し、無ければ最も深いノードの中から最良な下界値をもつノードを分枝します。
4	Best first, then local first	最初は, Best first で分枝し, 子問題数が "XPRS_BREADTHFIRST" を越えると, Local first で分枝します。 子問題数の設定は, 例えば, setparam("XPRS_BREADTHFIRST", 10)。
5	Pure depth first	最も深いノードの中から最良な下界値をもつノードを分枝します。

(例) 子問題数や実行時間の制限方法


```

setparam("XPRS_MAXNODE", 5000)      ! 子問題数を 5000 に制限します。
setparam("XPRS_MAXTIME", 100)      ! 実行時間を 100 秒に制限します。
setparam("XPRS_MIPABSSTOP", 0.99)  ! |上界値 - 下界値| <= 0.99 なら停止する。

```

4.5 求解 (minimize, maximize) の途中で関数を呼び出す方法 (setcallback)

(IP) に対して minimize(式) または, maximize(式) を実行すると, 分枝限定法が始まりますが, 整数変数の数が増えると求解に時間がかかることがあり, 下手をするとメモリ不足で強制停止されてしまいます。そのため, 整数解を見つける毎に解を表示したいという人もいます。その場合は次の setcallback 関数を用います。

```

setcallback(予約定数, "関数名")
(説明) ある条件を満たしたら, 関数を呼び出す。

```

(例) setcallback(XPRS_CB_INTSOL, "MyFunc") ! 整数解が得られたら MyFunc を呼び出す。
(注意) 実は, XPRS_CB_INTSOL 以外にも沢山あります, 詳しくは Xpress-Mosel Language Reference Manual を参照して下さい。

4.6 ループ中に制約式を取り除く方法 (sethidden)

ループ中に制約を変えて, 何度も問題を解く場合を考えます。制約は一旦設定されると解除しない限り, 残っているので, ループが進むと制約だらけになってしまいます。制約を解除するには, sethidden 関数をつかいます。

```

sethidden(制約式, 0)      ! 制約式を解除します。
sethidden(制約式, 1)      ! 解除した制約式を復活させます。
(説明) 制約式を解除/設定します。

```

(例)

```

Const := 3*a + b <= 3
forall (itr in 1..2) do
  minimize(ObjFunc)      ! 1 回目は Const あり, 2 回目はなし
  sethidden(Const, 0)
end-do

```

4.7 変数を増やす方法 (create)

動的に変数を確保したり, ループ中に変数を増やしたいときがあります。そのときは, create 関数を使います。

```

create(変数)
(説明) 変数を作ります。

```

謝辞

このメモを作成するに当たり，協力してくれた研究室の皆さん，特に，有用なコメントを下された先生方，プログラムソースを提供してくれた友人に感謝します．

そして，Dash 社と Xpress-MP を勉強する機会を与えてくれた卒業生達に感謝します．

参考文献

[1] Dash Optimization Corp. URL: <http://www.dashoptimization.com>.