AMPL：数理計画のためのモデリング言語

$A

付録A　AMPLリファレンスマニュアル

クイックリファレンス　—　字句規則から起動オプションまで、全23節の構文早見

23	14	全22章
節	表 (A-1〜A-14)	シリーズ最終回

Fourer, Gay, Kernighan 著　日本語版

目 付録Aの構成 —23節を5カテゴリーで整理

アジェンダ

本編の概念解説に対し、付録Aは逐条的な文法リファレンス。カテゴリー別に要点を凝縮する。

字 字句・式編 — 4節

字句規則 / 集合のメンバー / 添字式と下付き添字 / 式

宣 宣言編 — 7節

モデル実体の宣言 / 集合 / パラメータ / 変数 / 制約 / 目的関数 / 接尾辞記法

デ データ編 — 2節

標準データ形式 / データベースへのアクセスと表

コ コマンド編 — 7節

コマンド概観 / リダイレクト / 印字と表示 / データ読込 / モデル化コマンド / モデル検査 / 制御フロー

環 環境・拡張編 — 3節

計算機環境 / インポートされる関数 / AMPLの起動

字 字句規則

名前・数値定数・リテラル・コメント・予約語・演算子の同義語

A.1

- 名前: Unicode文字・数字・下線からなる英数字。大文字小文字を区別
- 数値定数: 標準の科学的記法。指数部は d/D/e/E のいずれでも可
- リテラル: ' または " で区切る文字列。区切り文字はそれを二重にしてエスケープ
- 文はセミコロンで終わる。空白(スペース・タブ・改行)は自由に置ける
- コメント: # から行末まで、または /* … */ (入れ子不可)
- 予約語(約44語)は他の用途に使えない。それ以外のキーワードは再定義可能

リテラル・コメント

```
1.23D-45            # 科学的記法(d/D/e/Eは同義)
'x''y'   ==  x'y    # 引用符の二重化でエスケープ
set prod;           # 事前定義語 prod を再定義
# コメントは行末まで
/* 複数行のコメント */
```

演算子の同義語 (左が推奨形)

`^  **`	べき乗
`=  ==`	等号
`<>  !=`	不等号
`and  &&`	論理積
`not  !`	論理否定
`or  ||`	論理和
`prod  product`	総乗の縮約演算子

予約語の例

`if`	`then`	`else`	`while`
`sum`	`forall`	`exists`	`in`
`within`	`union`	`integer`	`symbolic`

集合のメンバー / 添字式と下付き添字

A.2–A.3

集合はメンバーの重複しない順序付きリスト。添字式はダミーメンバーで反復範囲を指定する

集合リテラル

```
{"a","b","c"}                  # 1次元
{1,2,3,4,5,6,7,8,9}
{("a",2),("a",3),("b",5)}      # 多次元(タプル)
```

下付き添字 と 添字式の構文

```
cost[j, p[k]+1, "O+"]
indexing: { sexpr-list [ : lexpr ] }
sexpr-list: dummy-member in sexpr , ...
```

よく使う添字式パターン

パターン集

```
{A}                            # Aの全要素
{i in A, j in B}               # A×B の全ペア
{i in A, C[i]}                 # Aと C[i] の組
{i in A: p[i] > 0}             # such that (条件選択)
{i in A, (j,k) in D: i<=j}     # 多次元集合との組合せ

# ダミー添字の有効範囲は宣言/式の終わりまで
```

集合は 0 個以上のメンバーからなり、各メンバーは同じ成分数(=次元)をもつ。
1 と 0.01E2 のように浮動小数点として同じ値に丸まる数値メンバーは同一視される。

式 (1) — 演算子と優先順位

A.4

算術式 expr ・論理式 lexpr ・集合式 sexpr の3種類。表A-1は16段階の優先順位を規定

3種類の式

expr (A)	算術式。変数を含んでよい
cexpr	変数を含まない「定数式」
lexpr (L)	真偽を返す論理式
sexpr (S)	集合を返す式

式の例

```
sum {i in Prod} cost[i] * Make[i]
if x > 0 then 1 else 0
A union B     A diff B     A inter B
1 .. 10 by 2                # 等差数列の集合
setof {i in A} (i, i^2)     # 集合構成子
```

優先順位(抜粋・低い順)

1	**if-then-else**	A/S、else省略時は 0
2–4	**or/exists/and**	論理和・論理縮約・論理積
5	**比較演算子**	< <= = <> >= >
8–10	**union/inter/cross**	集合演算
12	**+ - less**	a less b = max(a-b,0)
13	**sum prod min max**	算術縮約演算子
16	**^ ****	べき乗(右結合)

関 式 (2) — 組込み関数と区分的線形項

A.4

算術・文字列・乱数の組込み関数(表A-2〜A-4)、および区分的線形項の構文

算術関数

abs	sqrt	exp	log	log10
ceil	floor	round(x,n)	trunc	min/max

文字列・正規表現関数

s & t	length	substr	sprintf	match
sub	gsub	num	num0	char

乱数生成関数

Uniform01()	Uniform(m,n)	Normal(μ,σ)	Poisson(μ)
Exponential()	Gamma(a)	Beta(a,b)	Irand224()

区分的線形項

```
<< breakpoints ; slopes >> var
<< 0,10 ; -1,0,1 >> x     # 折れ点;傾き
```

正規表現の主なメタ文字

.	任意の1文字
[a-z0-9]	文字クラス
r* r+ r?	0回以上・1回以上・0または1回
^r r$	先頭一致・末尾一致
r1|r2	選択(または)

集 モデル実体の宣言 / 集合の宣言

A.5–A.6

全実体は共通の宣言形式に従う。集合は *dimen / within / = / default の属性をもつ*

共通形式

```
entity name alias_opt indexing_opt body_opt ;
entity: set param var arc minimize maximize
        subject to  node
```

集合宣言の属性

dimen n	次元を明示
within sexpr	部分集合であることを要求(複数可)
= sexpr	値を確定(データ節で変更不可)
default sexpr	データ未指定時の既定値

順序付き集合 ordered / circular

```
set S ordered [ by [reversed] sexpr ];
set S circular [ by [reversed] sexpr ];

next(e,S,k)  prev(e,S,k)  first(S)  last(S)
member(j,S)  ord(e,S)  card(S)
```

推移閉包の例

```
set nodes;
set arcs within nodes cross nodes;
param n := card(nodes)-1;   # card(s)=要素数
set reach = ...;            # 再帰的な集合定義
```

P パラメータの宣言

A.7

param name alias_opt indexing_opt attributes_opt ; / check文とInfinity

属性

binary / integer	0,1に制限 / 整数に制限
symbolic	文字列も許容(比較演算子属性は不可)
relop expr	< <= = >= > による制約検査
in sexpr	所属する集合を検査
= expr / default expr	値を確定 / 既定値(互いに排他)

再帰的な定義 と Infinity

```
param comb {n in 0..N, k in 0..n} =
  if k=0 or k=n then 1
  else comb[n-1,k-1] + comb[n-1,k];
# 以前に計算された値だけを参照すれば再帰可
```

check文

```
check [ indexing_opt : ] lexpr ;
check {i in A}: p[i] >= 0;   # solve/write等で評価
```

Infinity

```
param Ub{A} default Infinity;
param Lb{A} default -Infinity;
var V {i in A} >= Lb[i], <= Ub[i];
```

V 変数の宣言

A.8

var name alias_opt indexing_opt attributes_opt ; / 定義変数と弧(arc)

主な属性

`binary / integer`	0,1に制限 / 整数に制限
`>= expr / <= expr`	下限 / 上限
`:= expr / default`	初期値 / 既定初期値(排他)
`= expr`	定義変数(他の属性と併用不可)
`coeff / obj / cover`	列方向の係数生成
`suffix sufname expr`	接尾辞の初期値

定義変数のポイント

- option substout=1 で、定義制約の左辺変数を共通部分式として除去
- 整数性・上下限などの制約を持つ変数は定義変数になれない
- option linelim=1(既定)で線形の定義変数を特別扱い

```
var x {J} >= 0, <= 100, integer;
var w = 2*x[1] + x[2];   # 定義変数(名前付き部分式)
```

arc宣言

```
arc F {(i,j) in ARCS} >= 0, <= cap[i,j],
    from node[i], to node[j];
# from/to は node制約に係数を寄与
```

制 制約の宣言 / 目的関数の宣言

A.9–A.10

[subject to] name : expr ; / minimize|maximize name : expr ; / 相補性制約

制約の形式

```
[subject to] name indexing_opt
  [:= initial_dual] [default d] [: expr] ;
expr: e<=e | e=e | e>=e | c<=e<=c | c>=e>=c
```

目的関数の形式

```
minimize Cost: sum {j in J} cost[j]*x[j];
maximize Profit: to_come;   # 列方向の係数生成
```

ノード制約 (net_in / net_out)

```
node Bal {i in NODES}: net_in = net_out;
```

相補性制約

```
name : constr1 complements constr2 ;

expr4 complements lb <= var <= ub
```

- constr1・constr2 は合計2個の不等式演算子を含む(=は2個と数える)
- 両方が成り立ち、少なくとも一方が等式で満たされるとき充足
- 均衡制約付き数理計画(MPEC)の表現に使う。他の制約・目的関数と共存可能
- 同義語 _scvar[i] がどの変数がどの制約に相補するかを示す

尾 補助的な値のための接尾辞記法

A.11

name.suffix で変数・制約・目的関数の付随値にアクセス(表A-6〜A-8)

変数の接尾辞(抜粋)

`.lb .ub`	下限・上限
`.val`	現在の値
`.rc`	換算費用
`.status`	ステータス

制約の接尾辞(抜粋)

`.body`	制約本体の値
`.dual`	双対変数
`.slack`	スラック
`.lb .ub`	下限・上限

目的関数の接尾辞

`.val`	目的関数の現在値

```
display x.lb, x.ub, x.rc;
display Balance.dual;
```

suffix宣言(方向)

```
suffix name alias_opt attributes_opt ;
suffix priority IN, integer;
direction: IN | OUT | INOUT | LOCAL   (既定 INOUT)
```

ステータス接尾辞

- .sstatus: ソルバーが返す基底情報 (次のsolveで再利用)
- .astatus: AMPLのdrop/restore・fix/unfixの状態を反映
- .status: .astatus と .sstatus から導かれる総合ステータス

デ 標準データ形式 (1) — 集合のデータ

*data*節はキーワード・リテラル・数・区切り文字からなる書式自由なトークン列

1次元・タプル集合

```
data;
set A := a b c;                 # 1次元(引用符省略可)
set B := "x y" z;               # 空白を含む文字列は引用符
set PAIR := (a,1) (a,2) (b,3);
```

2次元集合(表形式)

```
set PAIR : b1 b2 b3 :=
   a1     +   -   +
   a2     -   +   -  ;
# 行=第1添字, 列=第2添字, + は所属, - は非所属
```

集合データ文の一般形

```
set set-name := set-spec set-spec ... ;
set-template: ( templ-item, ... )
templ-item: object | * | :
```

- テンプレートの * にオブジェクトが左から代入される
- テンプレート省略時は全て * (集合の次元数分)が仮定される
- (tr) を付けると表が転置され、行と列の役割が入れ替わる
- 添字付き集合は成分ごとに別々のデータ文で与える
- データの書式(改行・空白)は人間の可読性のためで、AMPL自体は無視する

標準データ形式 (2) — パラメータのデータ

param param-name := value-list | value-table ; / 複数パラメータの同時定義

1次元(リスト形式)

```
param cost :=
  a1  10
  a2  20 ;
```

複数同時定義

```
param: DEMAND: d cost :=
  chi   200    1.5
  den   150    2.1 ;
# 集合DEMANDと複数パラメータを同時定義
```

```
param p {A} default 0 := a1 5, a2 . ;   # . は欠損値→既定値
```

param表の要点

- テンプレートは丸括弧でなく角括弧 [* , *] で与える
- 2次元表は +/- でなく数値(記号パラメータはリテラル)を格納
- default r 句で、値未指定のパラメータに既定値を一括付与
- 「.」は欠損値。default指定があればその値に解決される
- var name はデータ節では param の同義語(変数の初期値指定に使用)

```
var x {J} := 0;       # 初期値もparamと同じ書式で指定可
```

表 データベースへのアクセスと表

table宣言は外部リレーショナルテーブルの列とAMPLの集合・パラメータを接続する

table宣言の形式

```
table table-name indexing_opt in-out_opt
   string-list_opt :
   key-spec, data-spec, data-spec, ... ;
in-out: IN | OUT | INOUT   (既定 INOUT)
```

使用例

```
table Orders IN "ODBC" (dsn="mydb"):
   [ORD] ORD ~ id, Qty ~ qty;
read table Orders;    # 外部→AMPL
write table Orders;   # AMPL→外部
```

キー列とデータ列

`key-name`	ダミー添字と列名を兼用
`key ~ col`	ダミー添字keyを列colに対応付け
`param ~ col`	AMPL名を外部列名に対応付け
`<- -> <->`	読込/書出/双方向の向き

- 組込みハンドラは .tab / .bit ファイルの読み書きに対応
- DB・スプレッドシート接続には別途ハンドラの導入が必要
- 組込み集合 _HANDLERS が利用可能なハンドラ名を列挙

コマンド言語の概観

A.14

モデルモード / データモードの2モード。option コマンドでオプションを管理(表A-9)

基本フロー

```
model diet.mod;     data diet.dat;
option solver cplex;
option *col*;    # 名前パターンで一括表示
```

- data文でデータモードへ。モデルモードに戻すのは他のコマンド
- include filename でファイルを挿入(model/data命令の省略形あり)
- $opname=現在値、$$opname=既定値(環境から継承 or AMPLの既定)
- option opname evalue; で設定、option opname; で現在値を表示

代表的なコマンド(表A-9より抜粋・全44種)

solve	display	print	printf
let	fix	unfix	drop
restore	expand	show	xref
reset	write	read	option
problem	check	cd	shell

印 入出力のリダイレクト / 印字と表示のコマンド

A.15–A.16

> >> < でファイル入出力を切替。display / print / printf で結果を整形出力

リダイレクト

```
> filename      # 上書き(最初にオープン時)
>> filename     # 追記
< filename      # read コマンド用の入力
close filename;   remove filename;
```

3つの出力コマンド

```
display [indexing:] disparglist redirection ;
print   [indexing:] arglist redirection ;
printf  [indexing:] fmt, arglist redirection ;
```

```
printf "%s: %8.2f\n", j, cost[j] > "out.txt";
```

printf 変換指定(表A-10・抜粋)

%d %i	符号付き10進
%s	文字列
%f %e %g	浮動小数点(固定/指数/自動)
%q %Q	必要時/常に引用符付き文字列
%x %o	16進 / 8進

- display: 数値を既定で有効数字6桁に丸める(display_precision等で変更)
- print: 完全精度。区切り文字は print_separator で変更可
- display_1col, display_width, gutter_width 等が表の体裁を制御

解 データの読み込み / solve・write コマンド

A.17–A.18

read で書式なしデータを取込み。solve でソルバー起動、write で問題を書き出す

read コマンド

```
read [indexing:] arglist redirection ;
read {j in DEST} demand[j] < "demand.txt";
```

solve コマンド

```
solve redirection_opt ;
  1) 一時ファイルへ問題を書出し
  2) option solver の値でソルバーを起動
  3) stub.sol から主変数・双対変数を読込む
```

```
write bstub;    # スタイル文字+スタブでファイル生成
```

solveに関わる主なオプション

`solver`	起動するソルバー名
`relax_integrality`	integer/binary属性を無視
`reset_initial_guesses`	初期推定値を送るか制御
`send_suffixes`	接尾辞値の送受信を制御
`solve_result_num`	ソルバーの戻りコード

更 モデル化コマンド — 変更・固定・問題・データ更新

delete / drop / fix / problem / reset / let で実体やデータを操作する

実体の変更・除去

delete / purge	実体を削除 / 依存物ごと削除
redeclare	実体の宣言を再定義
drop / restore	制約・目的関数をソルバーから除外/復帰
fix / unfix	変数を現在値に固定/解除

```
fix x[1] := 5;
drop Balance[1];
let demand[j] := demand[j]*1.1;
```

問題(problem)とデータ更新

```
problem name indexing_opt : itemlist ;
problem name;      # 切替
problem SubProb: x, Balance;
```

reset data	読込済みデータを破棄
update data	後続データ節での上書きを許可
let name := expr	集合・パラメータ・変数の値を変更

検 モデルの検査

A.19

show / xref / expand / solexpand と組込みの総称的な名前(表A-13)

show namelist	実体の名前・種類を列挙
xref itemlist	依存する実体を表示
expand itemlist	生成された制約・目的関数を展開表示
solexpand	ソルバーから見た展開(前処理後)を表示
check;	全ての check 文を評価

組込みの総称的な名前(抜粋)

_nvars	**_ncons**	**_nobjs**
_VARS	**_CONS**	**_OBJS**
_varname{}	**_conname{}**	**_PARS**

```
show;                  # 全モデル実体を列挙
expand Balance[1];     # 特定制約の展開
```

```
display _nvars, _ncons, _nobjs;
```

制 スクリプトと制御フロー文

if-then-else / for / repeat による条件分岐とループでスクリプトを自動化

for + if-then-else

```
for {t in TIME} {
  if t <= 6 then
    let cmin[t] := 3;
  else
    let cmin[t] := 4;
}
```

repeat / break / continue

```
repeat loopname_opt while lexpr { cmds } ;
break loopname_opt ;   continue loopname_opt ;
```

構文まとめ

`if lexpr then cmd`	[else cmd] 条件分岐
`for indexing cmd`	添字集合にわたる反復
`repeat {cmds}`	while/until条件付き反復
`break / continue`	ループの中断 / 次の反復へ

- cmd はセミコロン終端の単文、または{ }で囲んだ文の並び
- option single_step n; で1コマンドずつ実行するデバッグモード

環 計算機環境

shell / cd コマンド、組込みの時間計測パラメータ、ログ記録オプション(表A-14)

外部コマンド・終了

```
shell 'command-line' redirection_opt ;
cd new-directory ;
quit;   exit expr_opt;   end;
```

- quit: $outoptが暗示するファイルを書き出さず終了
- exit: quitの同義語。ステータス値を環境に返せる
- log_file が空でなければ標準入力の内容をログに複写
- log_model / log_data で他ファイルのコマンド・データも記録

時間計測パラメータ(抜粋)

`_solve_time`	直近solveのCPU時間
`_total_solve_time`	全solveの累積CPU時間
`_shell_elapsed_time`	直近shellの経過時間
`_ampl_elapsed_time`	AMPLプロセス開始からの経過秒数

```
display _total_solve_time;
```

関 インポートされる関数

function宣言で外部関数を取込む。パイプ関数は別プロセスとして評価される

function宣言

```
function name alias_opt (domain-spec)_opt
  type_opt [pipe litseq_opt [format fmt]] ;
```

使用例

```
function mean2 pipe "awk '{print ($1+$2)/2}'";
display mean2(1,2) + 1;
call myfunc(x, y);   # 戻り値を使わない呼出し
```

ドメイン方向とロード

`IN / OUT / INOUT`	情報の流れる向き(既定IN)
`symbolic / random`	文字列を返す/毎回異なる値を返す
`load / unload / reload`	共有ライブラリの制御
`reset function name`	パイプ関数を閉じて再起動可能に

- domain-spec が集合を指定する引数は、その集合への所属を検査
- 単独の * はドメイン検査を省略する印

起 AMPL の起動

コマンドライン引数とセッション間で設定を記憶するオプション(OPTIONS_IN 等)

コマンドライン例

```
ampl '-?'           # 起動オプション一覧を表示
ampl -s1 model.mod  # 乱数シードを1に指定
ampl -R             # 制限付きサーバーモード
```

セッションを跨ぐオプション記憶

OPTIONS_IN	起動時に読み込むオプションファイル
OPTIONS_INOUT	起動時に読込み、終了時に書出す
OPTIONS_OUT	終了時に現在の設定を書出す

- -v: AMPLのバージョンを印字(option version と同等)
- -ilibs: 最初にロードする関数/tableハンドラのライブラリを指定
- -R: cd/shell禁止、ソルバー名やリダイレクト名を英数字に制限

ア まとめ —本編と付録Aの使い分け

本編(第1〜21章)

- モデリングの考え方・ワークフローを、実例を通じて学ぶ章立て
- 生産・輸送・スケジューリングなど分野別の応用例が中心

付録A(本スライド)

- 正確な文法・キーワード・オプション名を引くための逐条リファレンス
- コーディング中に「あの構文は？」を素早く確認する用途

よく使うキーワード早見表

`set`	`param`	`var`	`subject to`
`minimize`	`maximize`	`sum`	`forall`
`if-then-else`	`in / within`	`display`	`solve`
`let`	`option`	`table`	`function`

全22章シリーズ、完結。

詳細な最新情報は www.ampl.com を参照