

AMPL：数理計画のためのモデリング言語

第 7 章　パラメータと式

宣言・演算子・条件式でモデルの数値を組み立てる

本章の内容

7-1 パラメータの宣言

7-2 算術式

7-3 論理式と条件式

7-4 パラメータへの制限

7-5 計算されるパラメータ

7-6 ランダムに生成されるパラメータ

7-7 論理パラメータ

7-8 記号パラメータ

スカラーと添字づけられたパラメータ

- 最も単純な宣言はキーワード param と名前だけ：param T; と書けば以降どこでも T で数値を参照できる。
- 名前の後に添字式を続けると、集合の各要素につき一つのパラメータが定義される。
- 参照するときは名前に角括弧で「添字」を添える：avail[i]、demand[j,p] のように。
- 添字は、もとの添字集合の要素に評価される限り、任意の式でよい。

param 宣言の例

```
param T;

param avail {1..T};
param demand {DEST,PROD};
param revenue
   {p in PROD, AREA[p], 1..T};
```

添字を付けて参照する

```
avail[i]
demand[j,p]
revenue[p,a,t]
```

ベクトル・行列・疎行列としてのパラメータ

ベクトル

整数列で添字づけ

```
param avail {1..T};

avail[1], avail[2],
..., avail[T]
```

整数列にわたる添字づけは配列・ベクトルに対応するが、文字列の集合で添字づけてもよい。

行列

対の全組合せで添字づけ

```
set ORIG;  set DEST;
param cost {ORIG,DEST};

cost[i,j]  (すべての i,j)
```

ORIG の i と DEST の j のあらゆる組合せについて cost[i,j] が存在する。

疎な行列

対の部分集合で添字づけ

```
set LINKS within {ORIG,DEST};
param cost {LINKS};

cost[i,j]  ((i,j) in LINKS)
```

(i,j) が LINKS の要素であるときのみ cost[i,j] が存在する「疎な」行列となる。

7-2 算術式

演算子と優先順位（表 7-1）

通常の書き方	代替の書き方	型（被演算子 → 結果）
`if-then-else`		論理,算術 → 算術
`or`	`||`	論理 → 論理
`exists forall`		論理 → 論理
`and`	`&&`	論理 → 論理
`not`（単項）	`!`	論理 → 論理
`< <= = <> > >=`	`< <= == != > >=`	算術 → 論理
`in  not in`		対象,集合 → 論理
`+  -  less`		算術 → 算術
`sum prod min max`		算術 → 算術
`*  /  div  mod`		算術 → 算術
`+ -`（単項）		算術 → 算術
`^`	`**`	算術 → 算術

優先順位が下ほど高い（表は下に行くほど強く結合）。

べき乗と if-then-else は右結合、他は左結合。

if-then-else の論理被演算子は if の後、算術被演算子は then / else の後に置く。

Σ

7-2 算術式

組込み関数と反復演算子

表 7-2　組込み算術関数（抜粋）

abs(x) 絶対値	**sqrt(x)** 平方根	**exp(x)** 指数関数
log(x) 自然対数	**log10(x)** 常用対数	**min/max(...)** 最小・最大
sin cos tan 三角関数	**asin acos atan** 逆三角関数	**sinh cosh tanh** 双曲線関数

min・*max* 以外は再定義可能（元の意味は使えなくなる）。
card・*ord* も算術値を返す。

反復総和 sum

```
sum {i in ORIG} supply[i]

sum {i in ORIG, j in DEST}
   cost[i,j] * Trans[i,j]
```

sum の優先順位に注意

```
sum {p in PROD, t in 1..T}
  (sum {a in AREA[p]}
     revenue[p,a,t]*Sell[p,a,t]
   - prodcost[p]*Make[p,t] - ...);
```

- 他の反復演算子：prod（積）・min・max。sum は * より優先順位が低く、+ - より高い。

7-3 論理式と条件式

比較・論理演算子

比較演算子

=	等しい	<>	等しくない
<	より小さい	<=	以下
>	より大きい	>=	以上

- and：両方真のとき真 / or：少なくとも一方が真のとき真 / not：真偽を反転。
- 反復版：exists（反復 or）、forall（反復 and）。
- 優先順位：比較・in・算術 > not > and > or。

論理式の例

```
T >= 0 and T <= 10

i in MAXREQ or n_min[i] > 0

exists {i in ORIG} demand[i] > 10
forall {i in ORIG} demand[i] > 10
```

not の優先順位（表 7-1 参照）

```
not i in MAXREQ or
   n_min[i] > 0 and n_min[i] <= 10
# は次のように解釈される：
(not (i in MAXREQ)) or
  ((n_min[i]>0) and (n_min[i]<=10))
```

if-then-else 条件演算子

`if a then b else c`

- a は論理式。真なら値 b、偽なら値 c をとる。
- c が 0 のとき else c は省略できる。
- b と c は同じ種類の式であれば算術式・文字列式・集合式でもよい。
- then / else は他のどの演算子より優先順位が低いので、文末以外では括弧が必要。

二つの制約を一つにまとめる例

```
subject to Balance0 {p in PROD}:
  Make[p,first(WEEKS)] + inv0[p]
  = Sell[p,first(WEEKS)]
     + Inv[p,first(WEEKS)];

subject to Balance
   {p in PROD, t in WEEKS: ord(t)>1}:
  Make[p,t] + Inv[p,prev(t)]
     = Sell[p,t] + Inv[p,t];

# ↓ if-then-else でひとつに統合
subject to Balance {p in PROD, t in WEEKS}:
  Make[p,t] +
   (if t=first(WEEKS) then inv0[p]
    else Inv[p,prev(t)])
   = Sell[p,t] + Inv[p,t];
```

>1

7-4 パラメータへの制限

宣言時に制限句でデータを検査する

- integer・binary、または比較演算子＋算術式を宣言に付けられる。
- 添字づけられた宣言では制限が各要素に適用される。ダミー添字を使えば要素ごとに異なる制限も可能。
- 同じ宣言中のパラメータ自身を参照する自己参照には if-then-else が必要（t-1 が未定義になる場合を避ける）。
- 一つの宣言に複数の制限句を（任意でコンマ区切りで）置ける。

整数かつ 1 より大きい

```
param T > 1 integer;

param f_max {j in FOOD} >= f_min[j];
```

対話セッションでのエラー例

```
error processing param T:
    failed check: param T = 1
        is not > 1;
```

自己参照には if-then-else

```
param cumulative_market {p in PROD, t in 1..T}
  >= if t>1 then cumulative_market[p,t-1];
```

7-4 パラメータへの制限

check 文によるデータ検証

- check 文は「データのみに対する制約」。制限句より複雑な条件を課せる。
- 制限節は論理式で、既定の集合・パラメータ・添字式のダミー添字を使える。
- データ読込み後に真でなければならず、失敗するとデータをすべて拒絶する。
- 制限句・check 文の利用は強く推奨：誤ったデータによる誤ったLPの生成を防げる。

単純な check（供給＝需要）

```
check: sum {i in ORIG} supply[i]
   = sum {j in DEST} demand[j];
```

添字づけられた check

```
check {p in PROD}:
  sum {i in ORIG} supply[i,p]
  = sum {j in DEST} demand[j,p];
```

早期発見が鍵：捕らえられなかった誤りは、最悪の場合もっともらしい誤った解を生み、検出されないまま残る。

7-5 計算されるパラメータ

= による代入句で新しいパラメータを定義

- 代入句は = 演算子を使い、パラメータをある式に等しく設定する（制限句は不等式だけ）。
- 複雑な式を目的・制約に直接書くより、計算されるパラメータを介した方がモデルは読みやすい。
- = で定義したパラメータには、データ文で値を指定できない（エラーになる）。
- セッション中、= による定義はすべて最新に保たれる（依存元を変えれば自動的に再計算）。

需要シェアの正規化

```
param share {DEST} >= 0, <= 100;
param tot_sh =
   sum {j in DEST} share[j];

param tot_dem {PROD} >= 0;
param demand {j in DEST, p in PROD}
   = share[j]*tot_dem[p]/tot_sh;
```

最小在庫を計算する

```
param frac > 0;
param market {PROD,1..T+1} >= 0;
param mininv {p in PROD, t in 0..T}
   = frac * market[p,t+1];
var Inv {p in PROD, t in 0..T}
   >= mininv[p,t];
```

D

7-5 計算されるパラメータ

default 演算子と再帰的定義

=（代入句）

パラメータを定義する。データ文で値を指定できない。依存元が変われば常に再計算される。

default（既定句）

パラメータを初期化するだけ。データ文や代入文で後から値を上書きできる。

default の例

```
param mininv {p in PROD, t in 0..T}
   default frac * market[p,t+1];

# データ文で例外的な値を
# 上書きできる
param mininv :=
   bands 2 3000
   coils 2 2000 ;
```

再帰的定義（移動平均）

```
param mininv {p in PROD, t in 0..T} =
  if t = 0 then inv0[p]
  else 0.5 * (mininv[p,t-1]
       + frac * market[p,t+1]);

# 循環参照はAMPLがエラーで検出
```

■ = 句や default 句の後には、制限句と同様の追加チェック（例：, >= 0）も付けられる。

乱数関数で試験用データを作る

- 計算されるパラメータの一種。式に組込みの乱数生成関数（Normal・Uniform 等）を使う。
- モデル開発の初期段階や、異なる定式化・ソルバーの比較実験に便利。
- option randseed n; で種を変更できる。0 を指定すると毎回異なる系列になる。
- reset data コマンドで新しい乱数の標本を取り直せる。

正規分布に従う利用可能時間

```
param avail_mean > 0;
param avail_variance
   > 0, < avail_mean/2;
param avail =
  max(Normal(avail_mean,
             avail_variance), 0);
```

一様分布に従う需要シェア／randseed

```
param share {DEST} = Uniform(0,100);
param tot_sh =
   sum {j in DEST} share[j];

option randseed n;
```

0/1

7-7 論理パラメータ ／ 7-8 記号パラメータ

binary と symbolic

```
param promote {PROD,1..T+1} binary;
```

- AMPL に専用の論理型はないが、binary パラメータ（0/1）で代用できる。
- 算術式が論理式の場所に現れると、0 以外は真、0 は偽と解釈される（暗黙変換に注意）。

```
param entr symbolic in INTER;
```

- symbolic を付けると文字列値を表せる（数値でも文字列でもよいが、文字列は算術に参加できない）。
- 用途：特別な集合要素の指定、集合要素への説明文字列の付与など。

promote を条件式で使う

```
param mininv {p in PROD, t in 0..T} =
  (if promote[p,t] then fr_pro
   else fr_reg) * market[p,t+1];
```

発地の集合に説明文字列を添える

```
set ORIG;
param orig_name {ORIG} symbolic;
param supply {ORIG} >= 0;
```

木構造を集合とパラメータで表す

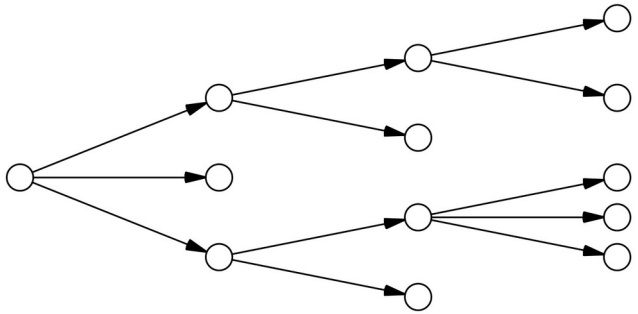

図 7-1　根から伸びる木

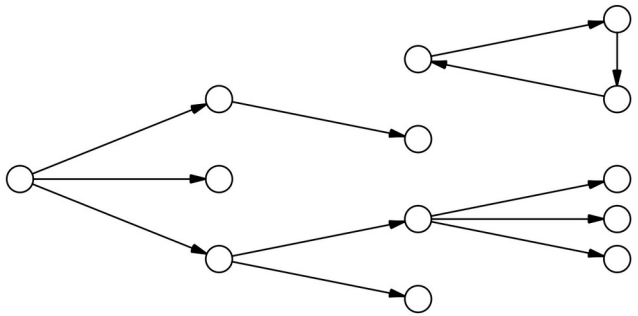

図 7-2　データ誤りで生じた切り離された閉路

木構造の宣言

```
set NODES;
param Root symbolic in NODES;
param pred {i in NODES diff {Root}}
   symbolic in NODES diff {i};
```

- Root 以外のすべての節点 i は、先行節点 pred[i] を一つもつ。
- 節点の「深さ」＝根に到達するまでに遡る先行節点の数（根の深さは 0）。
- depth[i] は pred を使った再帰的な param 定義で計算できる。
- 閉路を含む誤ったデータでは、depth の再帰的定義が循環参照となり計算できない。

まとめ —第 7 章　パラメータと式

1	**param 宣言**	スカラーから多次元・疎な集まりまで、添字式で自在に定義できる。
2	**演算子と反復式**	算術・論理演算子に加え、sum / prod / exists / forall で代数記法の Σ・Π を一般化。
3	**if-then-else**	論理式の真偽で値を選ぶ条件式。制約や計算されるパラメータの記述を簡潔にする。
4	**制限句と check 文**	宣言時の制限句とデータ全体を検査する check 文で、誤りを早期に発見する。
5	**=／default／binary／symbolic**	計算・初期化・真偽・文字列など、パラメータの多様な役割を使い分ける。

次章：第 8 章「変数」へ続く