AMPL：数理計画のためのモデリング言語

Σ

第5章

単純な集合と添字

集合はモデルの骨格――順序のない集合・数値の集合・集合演算・添字式・順序集合を学ぶ

本章の内容

第5章 単純な集合と添字 —アジェンダ

5-1	**順序のない集合**	文字列の集合を宣言・列挙する基本形
5-2	**数値の集合**	数値リテラルと ".." "by" による範囲指定
5-3	**集合演算**	union / inter / diff / symdiff で新しい集合を作る
5-4	**所属に関する演算と関数**	in / within と card() で所属・部分集合を調べる
5-5	**添字式**	{i in S} の形でモデル構成要素や総和を添字づける
5-6	**順序集合**	ordered / circular と prev・next・ord などの関数

導入

集合はAMPLモデルの最も基本的な構成要素

- パラメータ・変数・制約のほとんどすべては集合で添字づけられる
- 多くの式は集合上の演算（総和など）を含む
- 集合による添字づけこそ、簡潔なモデルで大規模な数理計画問題を記述可能にする機能
- AMPL は多種多様な集合の型と演算を提供する

集合の要素は…

- **文字列 or 数値**

  "coils" のような文字列、あるいは 1990 のような数値

- **順序あり or なし**

  既定は無順序。ordered / circular で順序を持たせられる

- **単独 or 組（タプル）**

  順序対・三つ組など、より長い組としても現れる（第6章）

集合の宣言と文字列リテラル

- 最も初歩的な集合＝文字列の順序のない集まり
- キーワード set と名前だけで宣言できる
- 名前は数字で始まらない文字・数字・下線の並び
- 大文字と小文字は区別される（"fish"≠"Fish"≠"FISH"）
- 文字列は単一引用符または二重引用符で区切る

```
set PROD;

# 予約語も再定義できる例
set prod;
```

引用の例

'A&P'
　単一引用符で区切る文字列
"Bell+Howell"
　二重引用符で区切る文字列

リテラルな集合とモデル／データの分離

```
set PROD = {"bands", "coils", "plate"};

{}                  # 空集合
{"bands"}           # 要素数 1 の集合
```

✓

データ分離が基本

要素は通常、モデルではなくデータ（第9章）で指定するのが大半の応用で推奨される

✓

リテラルが有効な場面

要素数が少ない／モデルの基本要素／頻繁に変わらない場合に限って直接列挙する

✓

"bands" と {"bands"} は別物

文字列そのものと、要素が1個の集合は AMPL 上で区別される

5-2　数値の集合

数値リテラルと ".." "by" による範囲指定

- 集合の要素は数値でもよい（文字列との混在も可能）
- {1,2,3,4,5,6} は 1..6 とより簡潔に書ける
- 既定の間隔は 1。by 句で任意の間隔を指定できる
- 数値要素は算術式でそのまま使える（例：Inv[p,t-1]）

```
set YEARS = 1990 .. 2020 by 5;

# 展開すると
{1990,1995,2000,2005,
 2010,2015,2020}
```

より良い書き方

```
param start integer;
param end > start integer;
param interval > 0 integer;
set YEARS = start..end by interval;
```

± 浮動小数点の丸め誤差に注意

集合の要素は、実数の場合は最も近い表現可能な浮動小数点数として格納される。
0 や 1 がちょうど現れると思うかもしれないが、丸め誤差のためそうはならない。

```
ampl: display -5/3 .. 5/3 by 1/3;
set -5/3 .. 5/3 by 1/3 :=
-1.6666666666666667   0.33333333333333326
-1.3333333333333335   0.6666666666666663
-1                    0.9999999999999998
-0.6666666666666667   1.3333333333333333
-0.3333333333333335   1.6666666666666663
-2.220446049250313e-16;
```

- 正確な値に依拠するモデルでは、集合に分数を使うのは避けるのが賢明
- 整数要素なら通常は問題ない（IEEE 標準では約 10^16 まで正確に表現可能）

5-3　集合演算

既存の集合から新しい集合を作る4つの演算子

`A union B`	**和**：A か B のいずれかに属する
`A inter B`	**共通部分**：A と B の両方に属する
`A diff B`	**差**：A に属し B に属さない
`A symdiff B`	**対称差**：A か B に属するが両方には属さない

union / diff / symdiff は同順位、*inter* *はそれより優先度が高い（左結合）*

```
ampl: set Y1 = 1990..2020 by 5;
ampl: set Y2 = 2000..2025 by 5;
ampl: display Y1 union Y2,
                Y1 inter Y2;
Y1 union Y2 := 1990 1995 2000
   2005 2010 2015 2020 2025;
Y1 inter Y2 := 2000 2005 2010
   2015 2020;
```

配合モデルでの活用例（図5-1 dietu.mod）

- 下限のある栄養素 MINREQ、上限のある栄養素 MAXREQ を宣言
- 全栄養素 NUTR を毎回別に指定させると、整合性の保証がモデル利用者任せになる
- 代わりに NUTR を union で自動的に定義すれば、冗長性や誤りを防げる
- 原則：データで与えるのは必要最小限の集合、他は式で定義する

```
set MINREQ;  # 下限がある栄養素
set MAXREQ;  # 上限がある栄養素

set NUTR = MINREQ union MAXREQ;

param n_min {MINREQ} >= 0;
param n_max {MAXREQ} >= 0;
param amt {NUTR,FOOD} >= 0;
```

in / within / not と card()

in	`"B2" in NUTR`	要素が集合に属するとき真（代数記法の ∈ に対応）
within	`MINREQ within NUTR`	左の集合がすべて右の集合に属するとき真（⊆ に対応）
not in / not within	`j not in FOOD`	結果の真偽を反転する
card( )	`card(NUTR)`	集合の要素数（濃度）を返す組込み関数

宣言に直接 within を使うと部分集合であることをAMPLに強制できる： **`set MAXREQ within NUTR;`**

波括弧 { } で書く、AMPLの添字づけの基本形

- 代数記法の「すべての $i \in P$ について」に対応するのが添字式 {...}
- 最も単純な形は、波括弧に囲まれた集合名または集合式
- t や j のような名前は「ダミー添字」と呼ばれる
- パラメータ・変数の宣言は param・var に続けて全く同じ書き方ができる

```
param rate {PROD} > 0;
param avail {1..T} >= 0;

param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];

var Buy {j in FOOD}
   >= f_min[j], <= f_max[j];
```

ダミー添字の有効範囲と条件付き添字式

- ダミー添字が有効なのは、それを定義する添字式の有効範囲の内側だけ
- 有効範囲が終われば未定義に戻るので、同じ添字名を繰り返し使ってよい
- 集合名の後に「: 条件」を続けると、条件を満たす要素だけの部分集合になる
- 空集合で添字づけても誤りではない（該当する構成要素は生成が飛ばされる）

```
{j in FOOD: f_max[j] - f_min[j] < 1}

set NUTREQ =
   MAXREQ union {i in MINREQ: n_min[i] > 0};
```

```
subject to Food_Ratio {j in FOOD}:
   Buy[j] >= min_frac[j] *
      sum {jj in FOOD} Buy[jj];

# 外側の j と区別するため jj を使用
```

display 文での添字式の活用（図5-1・5-2に基づく）

```
ampl: display MAXREQ union {i in MINREQ:
                              n_min[i] > 0};
set ... := A NA CAL C;

ampl: display {j in FOOD: Buy[j] > f_min[j]};
set ... := CHK MTL SPG;

ampl: display {i in MINREQ:
                Diet_Min[i].slack = 0};
set ... := C CAL;
```

対話コマンドの添字式では変数・制約も条件に参照できるが、モデル内で参照できるのは集合・パラメータ・ダミー添字だけ。

ordered / circular と前後を扱う関数

- ordered / circular を加えると、要素の並び順をAMPLが保持する
- ordered：最初の要素に前者なし、最後の要素に後者なし
- circular：最後の次は最初、最初の前は最後とみなす

- **first(S) / last(S)**　絶対位置の要素
- **prev(t) / next(t)**　相対位置の要素
- **ord(t) / ord0(t)**　集合内での位置（番号）

```
set WEEKS ordered;

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];

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

5-6　順序集合

区間式と定義済みの無限集合

`interval [a,b]`	a ≤ x ≤ b の実数区間
`interval (a,b]`	a < x ≤ b の実数区間
`integer [a,b]`	a ≤ x ≤ b の整数区間

param rate {PROD} in (0,maxrate); は *param rate {PROD} > 0, <= maxrate;* と同じ意味

```
定義済みの無限集合
Reals        # すべての浮動小数点数
Integers     # すべての整数
ASCII        # ASCII 照合順序
EBCDIC       # EBCDIC 照合順序
Display      # display の並び順

set PROD ordered by ASCII;
```

✓ 第5章のまとめ

- **集合はAMPLモデルの基本構成要素**
  パラメータ・変数・制約のほとんどは集合で添字づけられる
- **順序のない集合と数値の集合**
  set宣言、文字列・数値のリテラル、".." "by" 記法
- **集合演算子は4種類**
  union（和）・inter（共通部分）・diff（差）・symdiff（対称差）
- **所属演算と添字式が核心**
  in / within / card() と {i in S: 条件} がモデル全体を貫く
- **順序集合で「前後」を扱う**
  ordered / circular、first・last・prev・next・ord 関数

次章（第6章）：対や三つ組の集合、集合の添字づけられた集まりを含む複合集合へ