AMPL：数理計画のためのモデリング言語



第8章

線形計画問題：変数・目的・制約

var 宣言・線形式・目的関数・制約の書き方を体系的に学ぶ

AGENDA

本章の内容

8.1	変数	var 宣言、上限・下限、定義された変数、整数・バイナリ変数
8.2	線形式	AMPL が線形と認識する式の形と、等価な書き換え規則
8.3	目的	minimize / maximize の宣言と、複数目的の切り替え
8.4	制約	subject to の宣言、添字づけ、二重不等式、線形化

INTRODUCTION

線形計画問題とは

変数は連続な範囲から値をとり、目的と制約は変数の線形関数のみを使う――これが線形計画問題（LP）の要件である。

- ここまでの全ての例（生産計画・配合・輸送・部屋割当）はすべて LP
- 線形関数の単純さが定式化・解釈・分析を容易にする
- 制約と変数が数千個に及んでも、最適解を正確かつ迅速に発見できる
- ネットワークなど特別な構造をもつ LP には専用の記法もある（第15〜17章）

本章で学ぶ4本柱

1. var（変数）
2. 線形式の規則
3. minimize / maximize
4. subject to（制約）

var 宣言の基本構文

- **var 宣言の構文は param 宣言と同一**
  - 違いはキーワードのみ： param → var
- **>= と <= の句は、変数の下限・上限（一種の制約）を課す**
  - 以前に定義した集合・パラメータと、現在のダミー添字による任意の算術式を指定できる
- **< 、 > 、 <> は変数宣言に使えない**
  - 線形計画では「< 3」に意味がない（2.99999…に望むだけ近づけてしまう）

```
# パラメータ宣言（第7章）
param market {PROD};

# 変数宣言：param → var のみ変更
var Make {p in PROD}
   >= 0, <= market[p];
```

- Make[p] は集合 PROD の各要素 p ごとに1つ作られる添字づけられた変数の集まり
- >= 0：すべての Make[p] は非負
- <= market[p]：各 p について Make[p] は market[p] を超えられない

上限・下限の指定パターン

直接指定

```
var Make {p in PROD}
   >= 0, <= market[p];
```

非負性を >= 0 で直接指定する、最も一般的な形。

間接指定

```
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];
```

図5-1 配合モデル：下限・上限をパラメータで間接的に指定する形。

>= と <= に続く値が、変数の下限（lower bound）と上限（upper bound）である。

var 宣言に上下限を置けばモデルは短く明快になるが、最適解自体は変わらない。ソルバーの表現によらず、AMPL はすべての上下限を自動的に識別する。

定義された変数（= 句）

- var 宣言の = 句はパラメータと同様に「定義」を生む
- 右辺には集合・パラメータに加えて既存の変数も使える
- = の右の線形式が、目的・制約中のその変数の出現箇所すべてに代入される
- 新しい制約が追加されるわけではない
- 目的の構造が明快になり、display で内訳を手軽に確認できる

```
var Total_Revenue =
   sum {p in PROD, t in 1..T}
      sum {a in AREA[p]}
         revenue[p,a,t]*Sell[p,a,t];

maximize Total_Profit:
   Total_Revenue - Total_Prod_Cost
      - Total_Inv_Cost;
```

```
ampl: display Total_Revenue, Total_Prod_Cost,
ampl?    Total_Inv_Cost;
Total_Revenue = 801385
Total_Prod_Cost = 285643
Total_Inv_Cost = 1221
```

その他の宣言オプション

1

:= と default（初期値）

ソルバー起動時の開始解を与える。LP のソルバーは開始解をうまく活かせないことが多く、非線形計画（第18章）でより有用。

2

integer（整数変数）

どの最適解でも整数値をとらねばならない。含むモデルは整数計画問題（第20章）となる。

3

binary（バイナリ変数）

0 と 1 の値しかとれない。整数計画問題の代表的な部品となる。

一時的に変数を固定したいだけなら宣言はそのままにして fix コマンド（11.4節）を使う。

恒久的にデータの値へ固定するなら、= の右辺に変数を含めず param 宣言を使うべきである。

線形式とは何か

変数が1単位増減するたびに式の値が一定量だけ増減する ―― これが「その変数について線形」の定義。すべての変数について線形な式を線形式と呼ぶ。

AMPL が線形と認識する項

```
constant-expr
variable-ref
(constant-expr) * variable-ref
```

この形の項の任意の和が線形式である。constant-expr は変数を含まない算術式、variable-ref は（添字付き可）変数への参照。

線形式の実例（図6-3）

```
avail[t]

Make[p,t] + Inv[p,t-1]

sum {p in PROD}
   (1/rate[p]) * Make[p,t]

sum {a in AREA[p]} Sell[p,a,t]
   + Inv[p,t]
```

線形式の書き換え規則

除算 ＝ 逆数の乗算

```
Make[p,t] / rate[p]
```



```
(1/rate[p]) * Make[p,t]
```

乗算の順序は自由

```
revenue[p,a,t] * Sell[p,a,t]
```



```
Sell[p,a,t] * revenue[p,a,t]
```

乗除算は sum に分配できる

```
mt_t * sum{...}( ... )
```



```
sum{...}( ... ) / t_mt
```

AMPL はこれらの等価な式を自動的に「線形の項の単純な和」へ変換する。演算子の優先順位（表A-1）で結果が同じになるなら括弧は省略できる。

if-then-else と非線形の扱い

線形になる場合

then と else の式がともに線形で、if と else の間の論理式に変数が現れなければ結果は線形。

```
Make[j,t] + (if t = first(WEEKS)
   then inv0[j]
   else Inv[j,prev(t)])
```

線形にならない場合

線形式中の変数は、他のいかなる演算子の被演算子にも、いかなる関数の引数にもなれない。

- 反復演算子： max, min, abs, forall, exists
- 数値関数： ^（べき乗）, sqrt, log, cos など

線形式の一般形： constant-expr ／ var-ref ／ (constant-expr)*(linear-expr) ／ (linear-expr)/(constant-expr) ／ if logical-expr then linear-expr else linear-expr の和。この処理は AMPL が自動で行う。



8.3 目的

minimize / maximize の宣言

- キーワード minimize または maximize、名前、コロン、線形式で構成
- 目的の名前は代数モデル中でそれ以上の役割をもたない（列単位の宣言を除く）
- コマンド環境では目的の名前がその値を指す
- 1つの LP は目的関数を1つもたねばならないが、モデルには複数の objective 宣言があってよい
- 添字式を付ければ、添字づけられた目的の集まりも定義できる

```
minimize Total_Cost:
   sum {j in FOOD} cost[j]*Buy[j];
```

```
maximize Total_Profit:
   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]
      - invcost[p]*Inv[p,t]);
```

8.3 目的

objective コマンドで目的を切り替える

図2-1 配合モデル：総費用の最小化とは別に、総ナトリウム最小化の目的を追加して比較する。

```
minimize Total_NA:
   sum {j in FOOD} amt["NA",j]*Buy[j];
```

```
ampl: solve;   # 既定：先頭の目的
Objective = Total_Cost

ampl: objective Total_NA;
ampl: solve;
ampl: display Total_Cost;
```

目的を切り替えた結果

Total_Cost 最小化

```
Total_Cost = 118.06    Total_NA = 50000
```

Total_NA 最小化

```
Total_NA = 48186    Total_Cost = 123.63
```

ナトリウムを約*1800*減らすと費用は約 *$5.50* 押し上げられる

subject to の宣言

- キーワード subject to、名前、コロンで始まる（subject to 自体は省略可）
- コロンの後には制約の代数的記述（線形式 と 線形式 を等号・不等号で結んだもの）が続く
- 制約名の後に添字式を与えると、添字づけられた制約の集まりになる
- 制約名はコマンド環境で双対値の指定などに使われる（第14章）

```
subject to Time:
   sum {p in PROD} (1/rate[p])
      * Make[p] <= avail;
```

```
subject to Time {t in 1..T}:
   sum {p in PROD} (1/rate[p])
     * Make[p,t] <= avail[t];
```

```
subject to Balance {p in PROD,
                    t in 1..T}:
   Make[p,t] + Inv[p,t-1] =
      Sell[p,t] + Inv[p,t];
```

制約の応用パターン

条件付き添字

```
subject to Time {t in 1..T:
                 avail[t] > 0}: ...
```

二重不等式

```
subject to Diet {i in NUTR}:
  n_min[i] <= sum{j in FOOD}
   amt[i,j]*Buy[j] <= n_max[i];
```

擬似添字 *{if avail > 0}* は単一の制約を条件付きで生成／非生成にする。

非線形 → 線形への書き換え

```
# 線形でない（変数の和で除算）
Make[p] / sum{q in PROD} Make[q]
   <= max_frac;

# 書き換えれば線形になる
Make[p] <= max_frac
   * sum {q in PROD} Make[q];
```

前処理（presolve）

ソルバーに渡す前に AMPL が制約を自動簡約：固定変数の除去、単一変数制約の上下限化、冗長な制約の除去など。

SUMMARY

第8章のまとめ

1

var 宣言は param と同じ構文

>= と <= が変数の下限・上限を課す。= 句は変数を「定義」し、目的・制約中に代入される。

2

線形式は決まった形の項の和

constant-expr、variable-ref、その積の和のみ。max/min/abs 等の引数に変数を置くと非線形になる。

3

目的は minimize / maximize で宣言

1つの LP に目的は1つ必須だが宣言は複数可。objective コマンドで最適化対象を切り替えられる。

4

制約は subject to、添字づけ自在

二重不等式や条件付き添字も使える。除算や比の形は乗算に書き換えて線形化する。

次章：第9章へ続く