AMPL：数理計画のためのモデリング言語

第16章

列単位の定式化

行を追うか、列を追うか — 変数中心でモデルを組み立てる

投入産出モデル　|　スケジューリングモデル　|　obj / coeff 句の規則

AGENDA

本章の内容

16.1 投入産出モデル

行単位・列単位の両方で生産モデルを定式化し、obj句・coeff句を導入する

16.2 スケジューリングモデル

シフト勤務の割当問題を列単位で定式化し、集合被覆問題との関係を見る

16.3 列単位の定式化の規則

to_come・coeff句・obj句の一般規則と、多品種フローモデルへの応用

CONCEPT

行単位と列単位、二つの視点

AMPLは通常、`var` 宣言 → `maximize/minimize` → `subject to` の順に「行単位（制約指向）」で書く。しかし特定の線形計画問題では、一つの変数に関わるすべての係数をまとめて見るほうが単純になる。

行単位（制約指向）

- **var → maximize/minimize → subject to の順**
- **一つの制約について全変数の係数を並べる**
- **本書の大半の例で用いられる標準的な書き方**
- **特化した制約が多いモデルに向く**

列単位（変数指向）

- maximize/subject to → var の順
- 一つの変数について全制約の係数を並べる
- 第15章のnode/arc宣言もこの発想の一種
- obj句・coeff句をvar宣言に追加して実現

原料と活動：io[i,j] の意味

製鋼所や精製所のような複雑な操業では、ある生産単位の投入が他の単位の産出になる。原料集合 MAT と活動集合 ACT に対し、投入産出係数 io[i,j] を次のように定義する。



- io[i,j] > 0 ： 活動 j の1単位が生む原料 i の量（産出）
- io[i,j] < 0 ： 活動 j の1単位が消費する原料 i の量の負値（投入）
- io[i,j] = 0 ： 原料 i は活動 j に関与しない
- 変数 Run[j] は活動 j の運用水準（act_min[j] ≤ Run[j] ≤ act_max[j]）

まず行単位で定式化する

iorow.mod

```
set MAT;    # materials
set ACT;    # activities

param io {MAT,ACT};    # input-output coefficients
param revenue {ACT};
param act_min {ACT} >= 0;
param act_max {j in ACT} >= act_min[j];

var Run {j in ACT} >= act_min[j], <= act_max[j];

maximize Net_Profit:  sum {j in ACT} revenue[j] * Run[j];

subject to Balance {i in MAT}:
   sum {j in ACT} io[i,j] * Run[j] = 0;
```

図16-1　行単位の投入産出モデル

読み方

- Balance[i] は原料 i の産出量と消費量が均衡することを表す
- 資源の購入・製品の販売も「活動」として表現し、Run[i]の下限・上限で量を制御
- revenue[j] は活動ごとの単位収入（購入・生産は負＝費用）

obj句・coeff句で列単位に書き換える

iocol1.mod（抜粋）

```
maximize Net_Profit;

subject to Balance {i in MAT}: to_come = 0;

var Run {j in ACT} >= act_min[j], <= act_max[j],
   obj Net_Profit revenue[j],
   coeff {i in MAT} Balance[i] io[i,j];
```

図16-2　列単位の定式化

3つのポイント

- obj Net_Profit revenue[j] ： 目的 Net_Profit に項 revenue[j]*Run[j] を追加
- coeff {i in MAT} Balance[i] io[i,j] ： 各制約 Balance[i] に係数 io[i,j] を追加
- to_come は制約の「雛型」— 後続のvar宣言が係数を埋めていく

注意：obj/coeff句がNet_ProfitとBalanceを参照するため、`var` 宣言は `maximize/subject to` の後に置く必要がある。

販売活動を独立した変数に

完成原料 MATF を販売する変数 Sell[i] を新設。目的・制約の宣言は図16-2のまま変えず、変更はすべて var 宣言に反映される。

iocol2.mod（var宣言部）

```
var Run {j in ACT} >= act_min[j], <= act_max[j],
   obj Net_Profit -cost[j],
   coeff {i in MAT} Balance[i] io[i,j];

var Sell {i in MATF} >= sell_min[i], <= sell_max[i],
   obj Net_Profit revenue[i],
   coeff Balance[i] -1;
```

図16-3　販売活動を伴う列単位の定式化

利点

- Sell[i] の投入は原料iのみ→coeff は Balance[i] -1 の1項だけでよい
- 明示されない係数はすべて0とみなされる（零係数を書く必要がない）
- display Sell で販売量だけを容易に確認できる
- 新しい活動（例：購入変数Buy）の追加も定義を1つ足すだけ

シフトへの従業員割当

工場の生産はシフトに分かれる。各シフトに必要な人数が働くよう従業員を割り当てたいが、**5連勤は不可** など認められる週間スケジュールが限られる。そこで「従業員をスケジュールに割り当てる」問題として捉える。

```
set SHIFTS;              # shifts
param Nsched;            # number of schedules
set SCHEDS = 1..Nsched; # set of schedules
set SHIFT_LIST {SCHEDS} within SHIFTS;

param rate {SCHEDS} >= 0;      # 1人当たり賃金率
param required {SHIFTS} >= 0;  # シフトごとの必要人数
```

モデル化の要点

- **SHIFT_LIST[j] ： スケジュールjに含まれるシフトの集合**
- **変数 Work[j] ： スケジュールjに割り当てる人数**
- **各シフトiは、iを含むスケジュールに割り当てられた人数の合計でカバーされる**

行単位で書くと制約が扱いにくい

```
var Work {SCHEDS} >= 0;
minimize Total_Cost: sum {j in SCHEDS} rate[j] * Work[j];

subject to Shift_Needs {i in SHIFTS}:
   sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j]
      >= required[i];
```

左辺は「`i in SHIFT_LIST[j]` を満たすすべての j」にわたる和 — この`sum ... : 条件` という書き方の扱いにくさが、列単位への動機になる。

発想の転換

- データはスケジュール別（j）に整理されている
- 各スケジュールに変数も1つずつ対応する
- → 係数を「変数ごと」に指定する方が自然で、大規模例ではより効率的

coeff句 と cover句

sched.mod（図16-4）

```
minimize Total_Cost;
subject to Shift_Needs {i in SHIFTS}:
   to_come >= required[i];

var Work {j in SCHEDS} >= 0,
   obj Total_Cost rate[j],
   coeff {i in SHIFT_LIST[j]} Shift_Needs[i] 1;
```

図16-4　列単位のスケジューリングモデル

cover句（係数1を簡潔に指定）

```
var Work {j in SCHEDS} >= 0,
   obj Total_Cost rate[j],
   cover {i in SHIFT_LIST[j]} Shift_Needs[i];
```

係数の意味

- SHIFT_LIST[j]の各シフトiについて、Shift_Needs[i]で係数1をもつ
- cover句はfrom/toと同様、係数1を簡潔に指定するための専用句
- 航空会社の乗務員スケジューリングなど「集合被覆問題」の典型形

実例：126通りのスケジュール

月〜金1日3シフト・土2シフト、各日100/78/52人が必要という設定で `sched.mod` を解く。

AMPLセッション

```
ampl: model sched.mod; data sched.dat; solve;
MINOS 5.5: optimal solution found.
19 iterations, objective 265.6
ampl: display Work;
Work [*] :=
  10 28.8   30 14.4   71 35.6   106 23.2   123 35.6
  18 7.6    35 6.8    73 28     109 14.4
  24 6.8    66 35.6   87 14.4   113 14.4
;
```

265.6
最適目的値（延べ勤務量）

13 / 126
実際に使われたスケジュール数

271人
各解を切り上げた実行可能整数解

分数解を切り上げた271人が最良の整数解かどうかは、第20章の**整数計画**の技法で判定する。

to_come と coeff / obj 句の文法

雛型制約の3つの形

```
to_come
to_come + arith-expr
arith-expr + to_come
```

- const-expr 関係演算子 to_come の形が最も一般的（例：to_come >= required[i]）
- 雛型制約が変数を含む場合、それは行単位・列単位の混成モデルになる

var 宣言に追加できる句

- coeff {添字式} 制約名 arith-expr ： 制約係数を指定（複数可）
- obj 目的名 arith-expr ： 目的関数の係数を指定
- 添字式は {if logical-expr} の形も可（条件付きで係数を生成）
- arc宣言のfrom/to句に加えcoeff句も使用可（側面制約の列単位記述）

省略のルール：目的が後続の`var`宣言だけで指定される線形項の和なら、目的式は `to_come` のみとなり省略できる。

多品種フローモデルを完全に列単位で

第15章の多品種フローモデル（図15-13）を、`arc` 宣言の `coeff` 句で完全に列単位に書き直す。フロー収支は `from/to` 句、リンク総量の側面制約は `coeff` 句で表す。

netmcol.mod（図16-6）

```
minimize Total_Cost;

node Balance {k in CITIES, p in PRODS}:
   net_in = demand[k,p] - supply[k,p];

subject to Multi {(i,j) in LINKS}:
   to_come <= cap_joint[i,j];

arc Ship {(i,j) in LINKS, p in PRODS} >= 0, <= capacity[i,j,p],
   from Balance[i,p], to Balance[j,p],
   coeff Multi[i,j] 1.0,
   obj Total_Cost cost[i,j,p];
```

ポイント

- **from/to でフロー収支制約の係数を、coeffで側面制約の係数を同時に指定**
- **Multi[i,j] はリンク合計流量の上限（cap_joint）を表す雛型制約**
- **node/arc句とcoeff句は併用できる**

図16-6　図15-13の列単位の定式化

✓ まとめ

1. 行単位（変数→目的→制約）に対し、列単位は「先に目的と制約、後から変数」の順で書く発想
2. var 宣言に obj句・coeff句を加えることで、変数ごとに目的係数・制約係数をまとめて指定できる
3. 制約・目的の雛型は to_come で示し、後続のvar宣言が係数を埋めていく
4. 投入産出モデルやスケジューリングモデルなど、1変数が複数制約に現れる問題（集合被覆問題を含む）と相性が良い
5. arc宣言のfrom/to/coeff句との併用で、行単位・列単位を柔軟に混成できる