Z

AMPL：数理計画のためのモデリング言語

第20章　整数線形計画問題

整数変数と0-1（二値）変数で、分割できない実体や論理条件をモデル化する

目 本章の内容

20.1 整数変数

var 宣言に integer キーワードを加える。飲食モデルを例に、LP解と整数解の違いを見る

20.2 0-1変数と論理条件

binary 変数で固定費用・最小輸送量・カーディナリティ制限などの論理条件を表現する

20.3 整数計画法における実務上の考慮事項

整数計画問題の難しさ、変数を絞る工夫、緩和をタイトにする再定式化

整数計画問題とは

整数解が自然に得られる場合と、明示的に強制すべき場合

LPがたまたま整数解を返す場合

第15.5節のネットワーク線形計画問題では整数最適解が保証される。変数の数が制約の数を大きく上回る問題では、多くの変数が上下限にとどまる「極」解が得られ、たまたま整数値になりやすい（図16-4・16-5のスケジューリング問題）。

整数性を明示的に強制すべき場合

そうした保証がない状況は数多く残る。整数計画法ソルバーは線形計画法ソルバーよりはるかに難しい問題に直面し、より多くの計算時間・メモリと、定式化やオプション選択における利用者の助けを必要とする。

本章ではまず整数変数の宣言を説明し、次に論理条件をモデル化する*0-1変数の使い方を紹介する。*

整数変数の宣言

20.1　var 宣言に integer キーワードを加える

- var 宣言の限定句に integer を加えることで、宣言する変数を整数値に制限できる。
- 例：第2.3節の飲食モデルの元のLP最適解では、BEEF が 5.36061、SPG が9.30605という端数の購入量になった。
- 食品を整数量で購入したいなら、var 宣言（図2-1）に integer を加える。

var 宣言に integer を追加（dieti.mod）

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

元のLP最適解（display Buy 抜粋）

```
ampl: solve;
MINOS 5.5: optimal solution found.
13 iterations, objective 118.0594032
ampl: display Buy;
Buy [*] :=
BEEF   5.36061    MCH  10
  CHK  2          MTL  10
FISH   2          SPG   9.30605
  HAM 10          TUR   2
;
```

整数解を求める

*MINOS*は整数性を無視する → 整数対応のソルバーに切り替える

MINOSは整数制約を無視する

```
ampl: model dieti.mod;
ampl: data diet2a.dat;
ampl: solve;
MINOS 5.5: ignoring integrality
   of 8 variables.
13 iterations, objective 118.0594032
```

CPLEXに切り替えて解く

```
ampl: option solver cplex;
ampl: solve;
CPLEX 8.0.0: optimal integer
   solution; objective 119.3
11 MIP simplex iterations
1 branch-and-bound nodes
```

最小費用　118.06 ドル（LP）　→　119.30 ドル（整数）

整数性は追加の制約であるため、目的関数はより不利な方向にしか動かない。BEEFは5.36061→9、SPGは9.30605→7、HAMは10→8に変化。

非整数最適解を最も近い整数値に丸めるだけでは、整数最適解を常に導き出せるわけではない。

B 0-1変数と論理条件

20.2　マルチコモディティ輸送モデルに戻る

- 値として0と1だけをとる変数は整数変数の特別な場合。「0-1」変数、あるいは「二値」変数と呼ぶ。
- 目的関数や制約に巧みに組み込むことで、線形制約だけでは実用的な形で記述できないさまざまな論理条件を指定できる。
- 図4-1のマルチコモディティ輸送モデルに戻る。Trans[i,j,p] は発地 i から着地 j への製品 p の輸送トン数。
- **純粋なLPでは多くの発着対の間で少量の輸送を許してしまう → 0-1変数でこれを抑止する方法を導入する。**

LP最適解（線形計画）

199,500

総変動輸送費用（ドル）

41 iterations で最適解に到達。マルチコモディティ上限 625 に達する経路が6つ、100トン以下の少量輸送も残る。

固定費用の導入

経路ごとの固定費用を目的関数に加える

パラメータと0-1変数の宣言

```
param vcost {ORIG,DEST,PROD} >= 0;
   # variable cost on routes
param fcost {ORIG,DEST} > 0;
   # fixed cost on routes

var Use {ORIG,DEST} binary;
   # =1 only for routes used
```

目的関数（線形の式として書ける）

```
minimize Total_Cost:
   sum {i in ORIG, j in DEST,
        p in PROD}
      vcost[i,j,p] * Trans[i,j,p]
 + sum {i in ORIG, j in DEST}
      fcost[i,j] * Use[i,j];
```

- Use[i,j] は下限0・上限1の整数と言ってもよいし、同じことだがキーワード binary を使ってもよい。
- **連続（非整数）変数と整数変数を組み合わせるため、いわゆる「混合整数」計画問題になる。**

Use と Trans を結びつける制約

Multi制約を修正してUse変数を組み込む

Multi制約の修正

```
subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];
```

Use[i,j] = 0	sum Trans[i,j,p] <= 0 → 輸送量は0でなければならない（非負変数の和のため）
Use[i,j] = 1	sum Trans[i,j,p] <= limit[i,j] → 以前と同じマルチコモディティ上限

fcost[i,j] が正である限り、最適解では Use[i,j] は輸送量が正であるとき、かつそのときに限り 1 になる。

完全なモデル：multmip1.mod

図20-1a　固定費用付きマルチコモディティモデル

```
set ORIG;  set DEST;  set PROD;
param supply {ORIG,PROD} >= 0;   param demand {DEST,PROD} >= 0;
param limit {ORIG,DEST} >= 0;    param vcost {ORIG,DEST,PROD} >= 0;
var Trans {ORIG,DEST,PROD} >= 0; # units to be shipped
param fcost {ORIG,DEST} >= 0;    # fixed cost for using a route
var Use {ORIG,DEST} binary;      # = 1 only for routes used

minimize Total_Cost:
   sum {i in ORIG, j in DEST, p in PROD} vcost[i,j,p] * Trans[i,j,p]
 + sum {i in ORIG, j in DEST} fcost[i,j] * Use[i,j];
subject to Supply {i in ORIG, p in PROD}:
   sum {j in DEST} Trans[i,j,p] = supply[i,p];
subject to Demand {j in DEST, p in PROD}:
   sum {i in ORIG} Trans[i,j,p] = demand[j,p];
subject to Multi {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] <= limit[i,j] * Use[i,j];
```

固定費用モデルの実行結果

図20-1bのデータ（multmip1.dat）を用いて解く

199,500	223,504	229,850
LP （固定費用なし）	整数性を無視 （MINOS、Useが端数）	真の整数最適解 （CPLEX）

CPLEXによる整数解の探索

```
ampl: option solver cplex; solve;
CPLEX 8.0.0: optimal integer solution; objective 229850
295 MIP simplex iterations   19 branch-and-bound nodes
```

整数制約を課すと総費用は223,504ドルから229,850ドルへ増加する一方、望んでいた通り使われない経路の数は0から7に増える。

≥ ゼロまたは最小量の制限

Min_Ship 制約 — 輸送するなら minload 以上

AMPLは制約の中で論理演算子 or を受け付けない

```
subject to Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] = 0 or
   sum {p in PROD} Trans[i,j,p]
      >= minload;          # NOT ALLOWED
```

Use[i,j] を使った正しい定式化

```
subject to Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p]
      >= minload * Use[i,j];
```

Use[i,j] が1なら最小輸送量制約になり、Use[i,j] が0なら 0 >= 0 に帰着して何の効果ももたない。

minload = 375 で解くと、目的関数は 233,150 ドル（+1.4%）

```
CPLEX 8.0.0: optimal integer solution; 279 MIP simplex iterations, 17 branch-and-bound nodes
```

使われない経路数は依然7つだが、同じ経路ではない。最小輸送量を満たすため解の実質的な組み換えが必要になった。

カーディナリティの制限

Max_Serve 制約 — 発地ごとの配送先数を制限

変数で定義される集合は制約の card に使えない

```
subject to Max_Serve {i in ORIG}:
   card {j in DEST:
      sum {p in PROD} Trans[i,j,p]
         > 0} <= maxserve;   # NOT ALLOWED
```

Use[i,j] を使った正しい定式化

```
subject to Max_Serve {i in ORIG}:
   sum {j in DEST} Use[i,j]
      <= maxserve;
```

sum {j in DEST} Use[i,j] は発地 i が配送する着地の数を表す。maxserve = 5 として解く。

固定費用のみ	229,850 ドル ／ 未使用経路 7
+ 最小輸送量（minload = 375）	233,150 ドル （+1.4%）
+ カーディナリティ（maxserve = 5）	235,625 ドル （+1.1%）

整数計画法の実務上の考慮事項

20.3　整数計画法における実務上の考慮事項

整数計画問題ははるかに解くのが難しい

同じ規模のLPよりずっと難しい。反復回数の目安：線形版の41回に対し、混合整数版は約280〜400回。計算時間も同様に増大する。

整数変数は少なく、上下限は厳しく

問題規模が増すと整数計画法の難しさは急速に増大する。連続な線形・ネットワーク定式化で十分でないか、まず検討すべきである。

緩和がタイトになるよう再定式化する

個々の Trans[i,j,p] を Use[i,j] に連動させる制約を追加すると、LP緩和での Use[i,j] が1に近づき、整数最適解を速く見つける助けになる。

再定式化の例：Avail制約（緩和をタイトにする）

```
subject to Avail {i in ORIG, j in DEST, p in PROD}:
   Trans[i,j,p] <= min(supply[i,p],demand[j,p]) * Use[i,j];
```

✓ まとめ

1. 整数変数は var 宣言に integer を、0-1（二値）変数は binary を加えて宣言する。パッケージや人など分割できない実体や、論理条件のモデル化に使われる。

2. LPソルバーがたまたま整数解を返すこともあるが（ネットワーク構造、極解など）、保証がない場合は整数計画法ソルバー（CPLEXなど）に明示的に切り替える必要がある。

3. 0-1変数 Use[i,j] を輸送量に連動させる制約（Trans <= limit*Use）により、固定費用・最小輸送量・カーディナリティ制限といった論理条件を線形制約として表現できる。

4. 整数制約は目的関数をより不利な方向にしか動かせず、非整数解を丸めるだけでは整数最適解を得られない。

5. 整数計画問題はLPよりはるかに解くのが難しい。少ない整数変数・厳しい上下限・緩和をタイトにする再定式化・実験的な試行錯誤が成功への鍵となる。

次章へ続く