AMPL：数理計画のためのモデリング言語



第17章

区分的線形計画問題

折れ点と傾きで費用構造を表す ―― *AMPL の <<breakpoints; slopes>>* 記法

本章の内容

17.1 **費用項**
固定数／可変数の部分区間をもつ区分的線形コストの表現

17.2 **よくある 2・3 部分区間の項**
柔らかい制約・実行不可能性・可逆的活動のモデル化

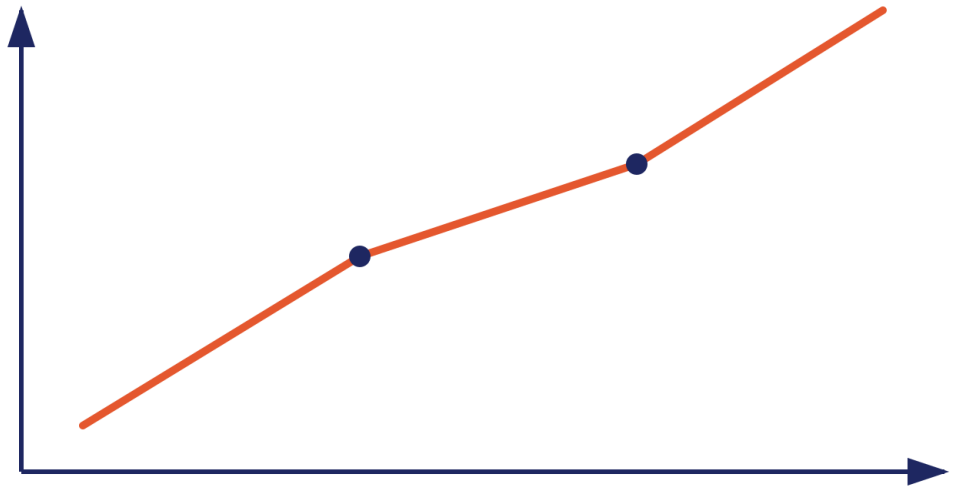
17.3 **他の区分的線形関数**
if-then-else、max、abs との等価性と記法の使い分け

17.4 **区分的線形最適化のための指針**
凸性・凹性と扱いやすさ、pl_linearize オプション

区分的線形とは

つながった線形の部分区間を継ぎ合わせてできる関数

- 線形計画問題の中には、厳密には線形でないが、直線の部分区間をつなぎ合わせてできる「区分的線形」の項を使うものがある。
- この形は思い描くのは容易だが、通常の代数記法で書くのは難しい。
- そこで AMPL は区分的線形項を記述する専用の簡潔な構文を提供する。
- 本章では、費用の記述・柔らかい制約・実行不可能性の分析・可逆的活動のモデル化という代表的な用途を、輸送モデルと生産モデルを例に見ていく。



図：傾きが変わる点（折れ点）をつないだ区分的線形関数のイメージ

固定数の部分区間 ——輸送コストの例

- 線形輸送モデルでは同じ単位費用が輸送量によらず一定。
- 現実にはより有利な料金は限られた単位数までしか適用されず、超過分にはより高い料金が課される。
- 各発地・着地の対に 3 つの費用水準（rate1, rate2, rate3）と 2 つの上限（limit1, limit2）を指定する。
- その結果、総輸送費用は輸送量とともに 3 つの部分区間をもつ区分的線形のやり方で増加する（図 17-1）。

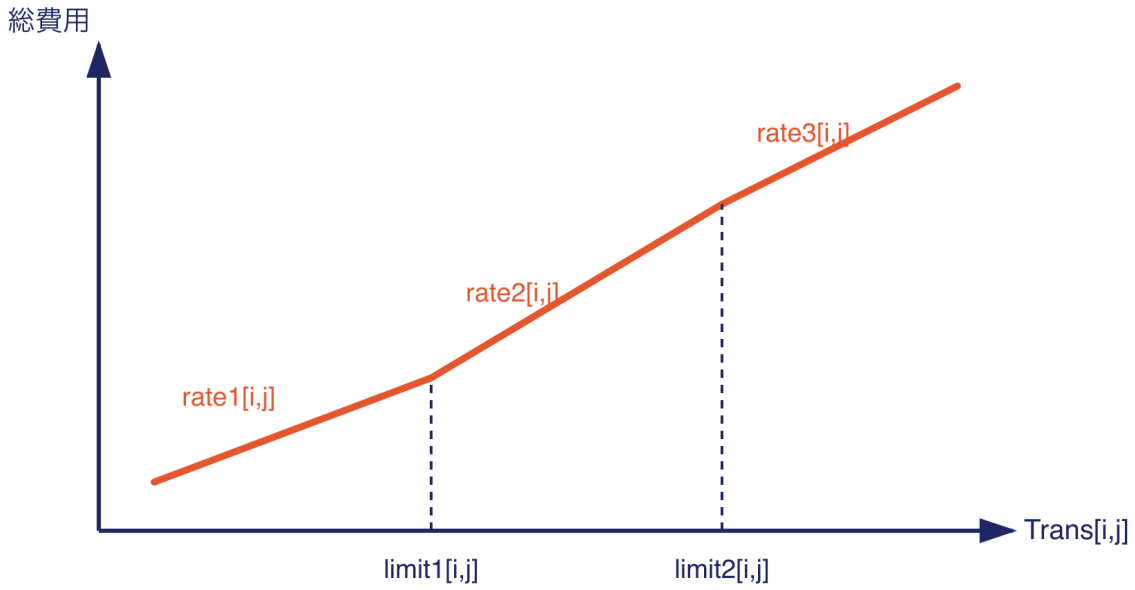


図17-1　3 つの傾きをもつ区分的線形関数

AMPL の区分的線形記法

```
<<limit1,limit2 ;   rate1,rate2,rate3 >>Trans
```

折れ点リスト　　傾きリスト

- « と » の間の式が区分的線形関数を記述し、続けて適用する変数名を書く。
- 折れ点リストと傾きリストはセミコロン（;）で区切り、各リストの要素はコンマ（,）で区切る。
- 傾きの数は折れ点の数よりちょうど 1 つ多くなければならない（最初の傾きは最初の折れ点より前、最後の傾きは最後の折れ点より後に適用）。
- 曖昧さを避けるため、AMPL は区分的線形関数が「零で零」の値をとると仮定する（既定の基準点）。

モデル例：transpl1.mod（3 部分区間）

```
set ORIG;   set DEST;
param supply {ORIG} >= 0;   param demand {DEST} >= 0;

param rate1 {i in ORIG, j in DEST} >= 0;
param rate2 {i in ORIG, j in DEST} >= rate1[i,j];
param rate3 {i in ORIG, j in DEST} >= rate2[i,j];
param limit1 {i in ORIG, j in DEST} > 0;
param limit2 {i in ORIG, j in DEST} > limit1[i,j];

var Trans {ORIG,DEST} >= 0;

minimize Total_Cost:
   sum {i in ORIG, j in DEST}
      <<limit1[i,j], limit2[i,j];
        rate1[i,j], rate2[i,j], rate3[i,j]>> Trans[i,j];
```

可変数の部分区間 —npiece によるデータ駆動

- 部分区間が 12 個などになると、rate1〜rate12 を個別に宣言するのは扱いにくい。
- 部分区間数そのものをパラメータ npiece[i,j] にするのがよい手法。
- 料金・上限をリンクと部分区間の組合せで添字づける：
- rate[i,j,p]、limit[i,j,p]。
- 折れ点・傾きのリストに添字式を使い、部分区間数をデータで変えられるようにする。

```
param npiece {ORIG,DEST} integer >= 1;

param rate {i in ORIG, j in DEST,
            p in 1..npiece[i,j]} >= ...;
param limit {i in ORIG, j in DEST,
             p in 1..npiece[i,j]-1} > ...;

minimize Total_Cost:
   sum {i in ORIG, j in DEST}
    <<{p in 1..npiece[i,j]-1} limit[i,j,p];
      {p in 1..npiece[i,j]} rate[i,j,p]>>
      Trans[i,j];
```

「柔らかい」制約への罰則項

第 4 章の複数週生産モデルを拡張

- 硬い制約「使用時間 ≤ avail[t]」を、望ましい水準を超えると罰則がかかる柔らかい制約に置き換える。
- avail_min[t]：罰則なしで使える時間
- avail_max[t]：使える上限時間
- time_penalty[t]：avail_min を超えた 1 時間ごとの利益減少
- 新変数 Use[t] （0 ≤ Use[t] ≤ avail_max[t]）を導入し、Time 制約を等式に変更する。

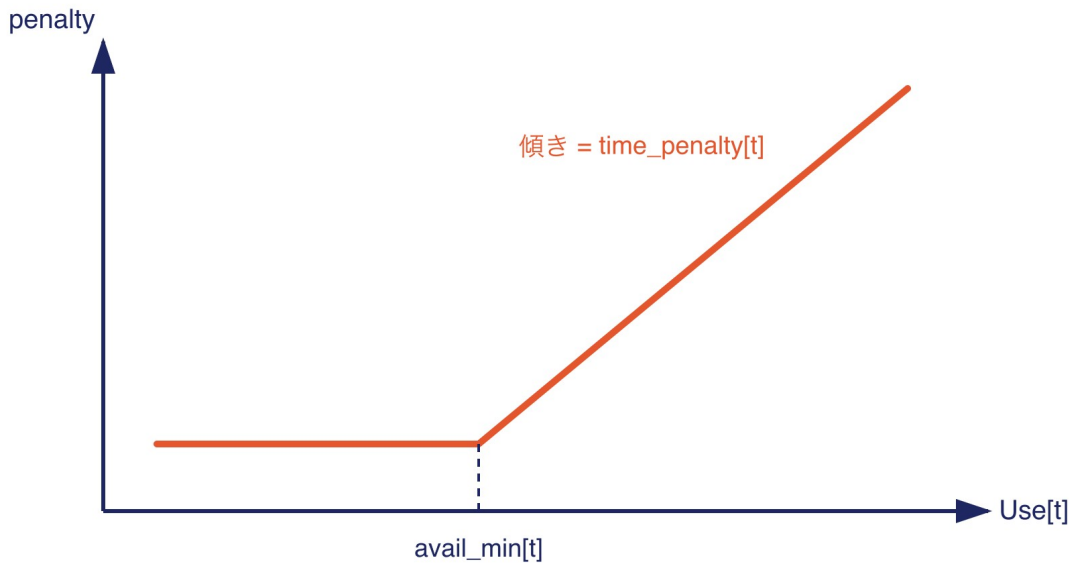


図17-4　利用時間に対する区分的線形の罰則関数（1 折れ点）

罰則項を組み込んだ目的関数と解

```
maximize Net_Profit:
   sum {p in PROD, t in 1..T}
      (revenue[p,t]*Sell[p,t]
       - prodcost[p]*Make[p,t]
       - invcost[p]*Inv[p,t])
  - sum {t in 1..T}
      <<avail_min[t]; 0,time_penalty[t]>>
      Use[t];
```

AMPL 対話セッション

```
ampl: model steelpl1.mod; data steelpl1.dat;
ampl: solve;
MINOS 5.5: optimal solution found.

ampl: display avail_min,Use,avail_max;
: avail_min Use avail_max :=
1     35      35     42
2     35      42     42
3     30      30     40
4     35      42     42 ;
```

第 1・3 週は罰則のない時間だけを使うが、第 2・4 週では罰則のある時間も使う。区分的線形計画問題の解では、多くの変数がいずれかの折れ点に「くっつく」性質がしばしば見られる。

実行不可能性への対処 ①：時間不足として見る

- 確約量（commit）を引き上げると時間が足りず、ソルバーが infeasible を返すことがある。
- 返る「実行不可能な解」は在庫が負になったり最小量を割り込んだりと、あちこちに散らばり原因がつかみにくい。
- 対策：avail_max[t] を折れ点にし、その先に非常に大きな傾き（例：100,000）を追加する。
- これにより Use[t] は avail_max を超えられるが、超過には極めて重い罰則がかかるため、必要最小限しか超えない。

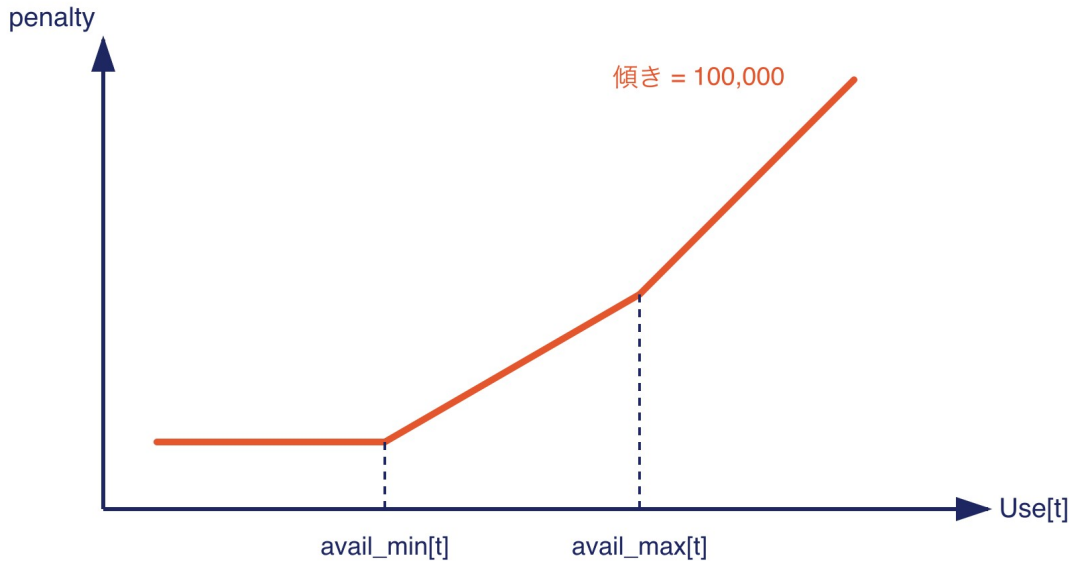


図17-6　2 つの折れ点をもつ利用時間の罰則関数

実行不可能性への対処 ②：確約超過として見る

- Sell[p,t] が commit[p,t] を下回る単位ごとに大きな罰則を引く（図 17-7）。
- 折れ点は零で零になるという既定の仮定があるため、単純な <<commit[p,t]; -100000,0>> だけでは値がずれる。

```
<<commit[p,t]; -100000,0>>
   (Sell[p,t], commit[p,t]);

# 第2引数 zero-expr で
# 「関数が commit[p,t] で零」と
# 明示できる
```

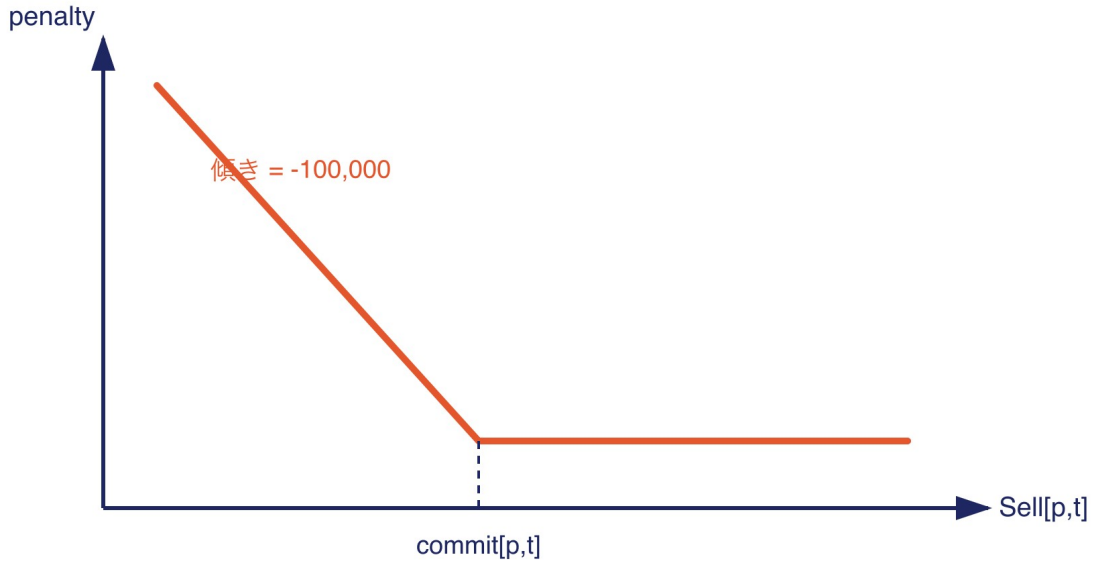


図17-7　販売量に対する罰則関数（commit[p,t] に折れ点）

可逆的な活動 ——在庫と受注残

- Inv[p,t] の宣言から >= 0 を外すと、負の値を「受注残」として解釈できる。
- しかし単純な線形費用 invcost[p]*Inv[p,t] のままでは、負の Inv が利益を見かけ上増やしてしまい、不自然な解になる。
- 適切な費用関数は零を折れ点とし、正負どちらの方向に離れても増加する（図 17-8）。
- 傾きは -backcost[p]（負側）と invcost[p]（正側）：<<0; -backcost[p], invcost[p]>> Inv[p,t]。

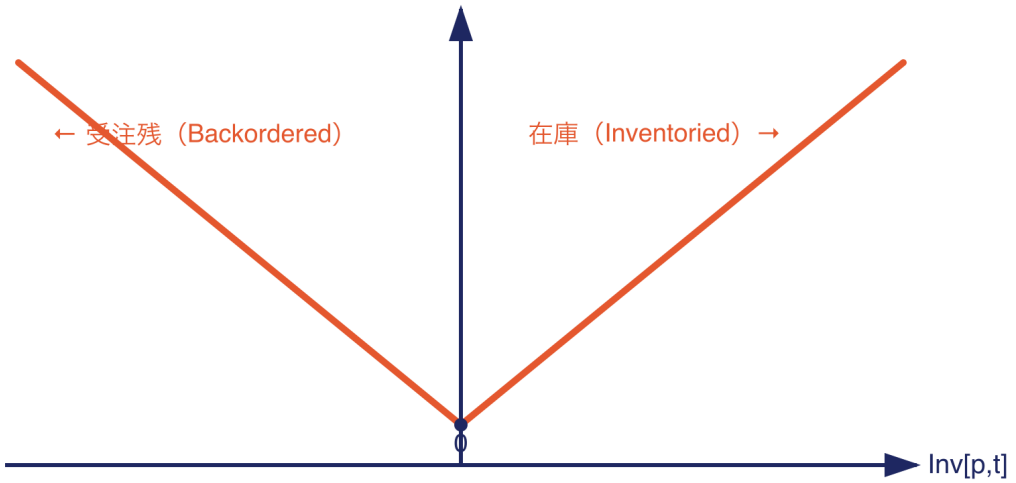


図17-8　在庫費用関数（零に折れ点をもつ V 字形）

他の書き方との等価性 ——if・max・abs

同じ関数でも解けるかどうかは記法次第

```
# 同じ関数の別表現
if Use[t] > avail_min[t]
then time_penalty[t]*
     (Use[t]-avail_min[t])
else 0

max(0, time_penalty[t]*
  (Use[t]-avail_min[t]))
```

AMPL はこれらの式が区分的線形だと自動検出しない –– 満足のいく解は得にくい

```
# 明示的な区分的線形記法
<<avail_min[t]; 0,time_penalty[t]>>
  Use[t]

# abs も同様
time_penalty[t] *
  <<avail_min[t]; -1,1>> Use[t]
```

構造がソルバーに直接伝わり、*LP として効率よく解ける*

分離不可能な例 ——最小最大の割当

- 目的 max{i} sum{j} cost[i,j]*Assign[i,j] は区分的線形だが、個々の変数について分離可能ではないため <<…>> 記法では書けない。
- 解決策：最大値を表す新変数 M を導入し、線形計画問題に書き直す。
- M はすべての i について右辺以上という制約 M_def を課し、目的は minimize M とする。
- 同じ考え方は「最大最小」目的にも適用でき、その場合は maximize と M <= … を使う。

```
var M;
minimize Max_Cost: M;

subject to M_def {i in PEOPLE}:
   M >= sum {j in PROJECTS}
        cost[i,j]*Assign[i,j];
```

区分的線形最適化のための指針

```
<<breakpoint-list; slope-list>> pl-argument
```

凸関数（傾きが非減少）

分離可能な凸区分的線形関数の最小化は LP として効率よく解ける（本章の例はすべてこの型）。

凹関数（傾きが非増加）

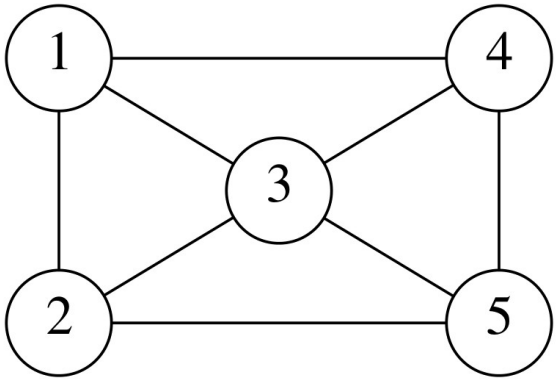
分離可能な凹区分的線形関数の最大化も同様に扱いやすい。

- この 2 つを外れる場合、分離可能な区分的線形の最適化は整数計画（第20章）の応用として扱われ、AMPL が等価な整数計画に自動変換する。
- 制約中の区分的線形は「凸で ≤ 制約左辺」または「凹で ≥ 制約左辺」の場合にかぎり LP で扱える。
- 直接扱えるソルバーがあれば option pl_linearize 0; で変換を止められる。
- 部分区間の総数が増えるほど難しさは増す —— 10 数個を超えるなら第18章の非線形関数も検討する。

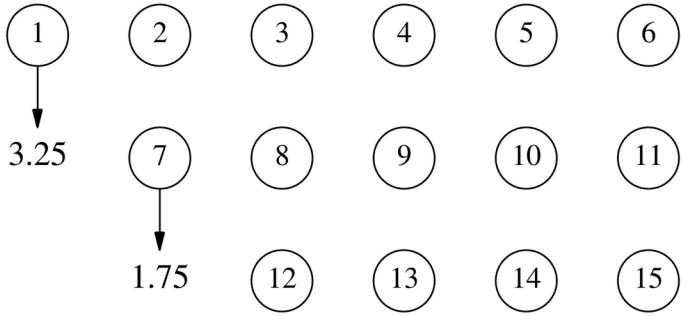
応用：構造設計への拡張（練習問題 17-7）

abs() の区分的線形記法は構造の最小重量設計にも使える

- 継手（joint）を棒（bar）でつないだ平面構造の最小重量設計問題。
- 棒の応力 F(i,j) は張力（正）・圧縮（負）のどちらもあり得る。
- 重量は w(i,j) = l(i,j) × |F(i,j)| ―― 絶対値が登場する。
- 均衡制約のもとで総重量を最小化する線形計画問題として、abs 項を区分的線形記法 <<0; -1,1>> で表現できる。



継手1〜5と棒の例



15 継手のトラス構造と外力の例

区分的線形記法は輸送や生産計画だけでなく、絶対値・逸脱量が現れるあらゆる線形計画（構造設計、回帰、割当など）に応用できる。

まとめ

第17章　区分的線形計画問題

1. **<<breakpoint-list; slope-list>> pl-argument**

   折れ点と傾きをセミコロンで区切って書く AMPL 独自の構文で、区分的線形の費用・罰則を簡潔に表現できる。

2. **データ駆動の部分区間数**

   npiece のような添字パラメータと {p in 1..npiece[i,j]} の添字式で、部分区間数を可変にできる。

3. **柔らかい制約・実行不可能性・可逆的活動**

   少数の折れ点をもつ罰則関数として、超過ペナルティや在庫／受注残の費用を自然にモデル化できる。

4. **if/max/abs より <<...>> を**

   同じ式でも明示的な区分的線形記法にしないと、AMPL は構造を検出できず効率よく解けない。

5. **扱いやすさは凸性・凹性で決まる**

   凸最小化／凹最大化は LP で自動的に解ける。それ以外は整数計画に変換され、部分区間が増えるほど難しくなる。

次章：*第18章 非線形計画問題*