08

第8章

非線形関数の区分的線形近似

非線形関数を区分的線形近似して混合整数最適化で解く

この章の内容

1. 凸関数の最小化
2. 凹関数の最小化
3. 一般の区分的線形関数の最小化
4. 経済発注量問題
5. 凹費用関数をもつ施設配置問題
6. 安全在庫配置問題

非線形な問題への2つの選択肢

非線形ソルバーを使う

- 凸関数の最小化は一般に易しい
- 凸二次・二次錐最適化は多くのソルバーで容易
- 一般の非線形関数では局所的最適解どまり

混合整数最適化に変形する

- 区分的線形関数で近似する
- 近似だが任意の精度で大域的最適解
- Big M を使わない「良い」定式化を使う
  - 非線形関数の構造を利用する

8.1 凸関数の最小化

凸関数の区分的線形近似は、整数変数なしの線形最適化で書ける

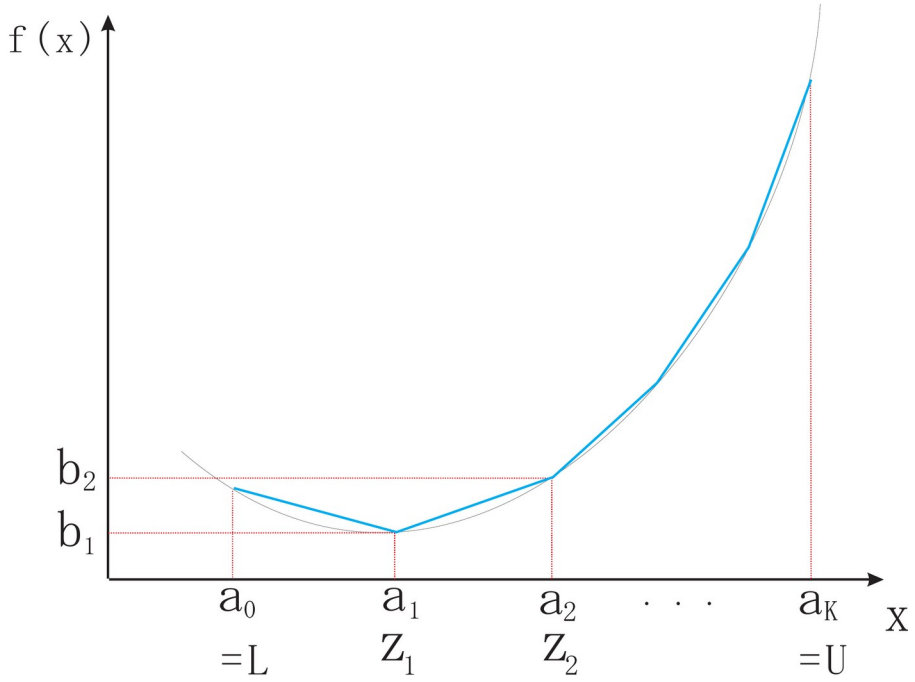
凸結合定式化

範囲 $[L, U]$ を $L = a_0 < a_1 < \cdots < a_K = U$ で K 個に分割し、$b_k = f(a_k)$ とする

$$
\begin{aligned}
f(x) &\approx \sum_{k=0}^{K} b_k z_k \\
x &= \sum_{k=0}^{K} a_k z_k \\
\sum_{k=0}^{K} z_k &= 1 \\
z_k &\geq 0 \qquad\qquad \forall k = 0, 1, \ldots, K
\end{aligned}
$$

- 凸関数なら、正になるのは隣り合う2つの z_k だけ
- 最小化では $\approx$ を $\geq$ に置き換えてよい
- 線分を凸結合で表すので「凸結合定式化」
- 多変数でも補助変数で分離可能な形に書ける

凸関数の区分的線形近似



凸関数の区分的線形関数による近似（凸結合定式化）

- 区分 $[a_k, a_{k+1}]$ 内の点は両端点の凸結合
  - $x = a_k z_k + a_{k+1} z_{k+1}$
  - $z_k + z_{k+1} = 1, z \geq 0$
- 関数値も同じ重みで近似
  - $f(x) \approx b_k z_k + b_{k+1} z_{k+1}$

AMPLによる凸結合定式化

convex_comb.mod

```
param K > 0 integer;            # 区分数
param a {0..K};                 # 区分点
param b {0..K};                 # 区分点における関数値

var z {0..K} >= 0, <= 1;
var x >= a[0], <= a[K];
var f;

subject to XDef: x = sum {k in 0..K} a[k]*z[k];
subject to FDef: f = sum {k in 0..K} b[k]*z[k];
subject to Convexity: sum {k in 0..K} z[k] = 1;
```

- var f; は上下限なしの自由変数
  - 関数値は負にもなりうる
- >= 0 は既定値ではない
- 添え字 {0..K} が k=0,...,K に対応

8.2

凹関数の最小化

隣り合わない2つの z_k が正になる解をどう除くか？

特殊順序集合（SOS2）を用いた凸結合定式化

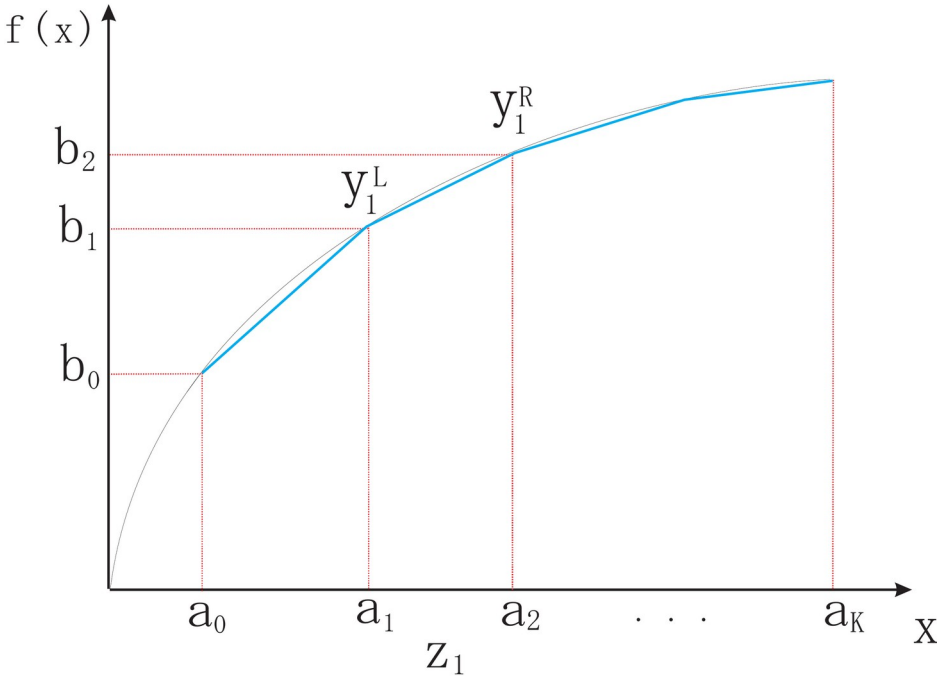
convex_comb.mod を変更

```
suffix sosno IN;
suffix ref IN;

var z {k in 0..K} >= 0, <= 1,
   suffix sosno -1, suffix ref k;
```

- SOS1：たかだか1つの変数が非0
- SOS2：順序集合で連続するたかだか2つが非0
- sosno：同じ値の変数が1つの SOS。負なら SOS2、正なら SOS1
- ref：集合の中での並び順
- HiGHS や AMPL 経由の SCIP は SOS2 非対応

整数変数を用いた凸結合定式化



凹関数の区分的線形関数による近似（凸結合定式化）

- 区分 k ごとに連続変数 yL[k], yR[k] と 0-1 変数 z[k]
- 区分 k を選ぶとき z[k] = 1
- そのときに限り yL[k] + yR[k] = 1
  - a[k]·yL[k] + a[k+1]·yR[k] は線分上の点

非集約型と集約型の凸結合定式化

【非集約型】

$$x = \sum_{k=0}^{K-1} \left(a_k y_k^L + a_{k+1} y_k^R \right)$$

$$f(x) \approx \sum_{k=0}^{K-1} \left(b_k y_k^L + b_{k+1} y_k^R \right)$$

$$y_k^L + y_k^R = z_k \quad k = 0, \ldots, K-1$$

$$\sum_{k=0}^{K-1} z_k = 1, \ z_k \in \{0, 1\}$$

【集約型】

$$x = \sum_{k=0}^{K} a_k y_k, \quad f(x) \approx \sum_{k=0}^{K} b_k y_k$$

$$\sum_{k=0}^{K} y_k = 1, \quad \sum_{k=0}^{K-1} z_k = 1$$

$$y_0 \le z_0, \quad y_K \le z_{K-1}$$

$$y_k \le z_{k-1} + z_k \quad k = 1, \ldots, K-1$$

- 非集約型：区分ごとに2つの連続変数
- 集約型：区分点ごとに1つの連続変数
- 集約型は変数が少ないが多少弱い
- y_k が正になれるのは区分 $k{-}1$ か k を選んだとき

AMPLによる非集約型凸結合定式化

convex_comb_dis.mod

```
param K > 0 integer;
param a {0..K};
param b {0..K};

var yL {0..K-1} >= 0, <= 1;
var yR {0..K-1} >= 0, <= 1;
var z {0..K-1} binary;
var x >= a[0], <= a[K];
var f;

subject to XDef:
   x = sum {k in 0..K-1} (a[k]*yL[k] + a[k+1]*yR[k]);
subject to FDef:
   f = sum {k in 0..K-1} (b[k]*yL[k] + b[k+1]*yR[k]);
subject to Pick {k in 0..K-1}:
   yL[k] + yR[k] = z[k];
subject to One:
   sum {k in 0..K-1} z[k] = 1;
```

- 0-1 変数だけで SOS2 は不要
- 例：$f(x) = -x^2+10x$ を $[0,10]$ で10等分、$x \geq 3$ で最小化
  - 凸結合定式化：$x=3, f=0$（誤り。近似値は 21）
  - 本定式化：$x=10, f=0$（正しい）

対数個の 0-1 変数を用いた定式化

区分 k を2進数で表し、ビット j ごとに 0-1 変数 g_j を用意する（Vielma ら）

$$G = \lceil \log_2 K \rceil$$

$$\sum_{k=0}^{K-1} \left(y_k^L + y_k^R \right) = 1$$

$$\sum_{k:\ \text{第 } j \text{ ビットが } 1} \left(y_k^L + y_k^R \right) \le g_j \qquad \forall j = 0, \ldots, G-1$$

$$\sum_{k:\ \text{第 } j \text{ ビットが } 0} \left(y_k^L + y_k^R \right) \le 1 - g_j \quad \forall j = 0, \ldots, G-1$$

- K = 6 なら G = 3 ビットで区分を表せる
- 区分 5 = (1,0,1) は g = (1,0,1) のときだけ選べる
- 集約型では Gray コード（隣が1ビット違い）を使う
  - 例：000, 001, 011, 010, 110, 111, 101, 100

AMPLによる対数個の変数を用いた定式化

convex_comb_dis_log.mod

```
param K > 0 integer;
param a {0..K};  param b {0..K};
param G = ceil(log(K)/log(2));    # 区分を表すのに必要なビット数
param N = 2**G;                   # G ビットで表現可能な区分数
var yL {0..N-1} >= 0, <= 1;
var yR {0..N-1} >= 0, <= 1;
var g {0..G-1} binary;            # 区分番号の 2 進表現
var x >= a[0], <= a[K];
var f;
subject to XDef:
   x = sum {k in 0..K-1} (a[k]*yL[k] + a[k+1]*yR[k]);
subject to FDef:
   f = sum {k in 0..K-1} (b[k]*yL[k] + b[k+1]*yR[k]);
subject to One:
   sum {k in 0..K-1} (yL[k] + yR[k]) = 1;
subject to Unused {k in K..N-1}:
   yL[k] + yR[k] = 0;
subject to Bit1 {k in 0..K-1, j in 0..G-1: (k div 2**j) mod 2 = 1}:
   yL[k] + yR[k] <= g[j];
subject to Bit0 {k in 0..K-1, j in 0..G-1: (k div 2**j) mod 2 = 0}:
   yL[k] + yR[k] <= 1 - g[j];
```

多重選択定式化

区分 k の直線：傾き $c_k = (b_{k+1} - b_k)/(a_{k+1} - a_k)$、切片 $d_k = b_k - c_k\, a_k$

$$a_k z_k \le y_k \le a_{k+1} z_k \qquad \forall k = 0, 1, \ldots, K-1$$

$$\sum_{k=0}^{K-1} z_k = 1, \quad z_k \in \{0, 1\}$$

$$x = \sum_{k=0}^{K-1} y_k$$

$$f(x) \approx \sum_{k=0}^{K-1} (d_k z_k + c_k y_k)$$

- 区分を選んだら、その区分の直線で関数値を表す
- y_k は $z_k = 1$ のときだけ区分 k 内の値をとる
- 凸関数なら直線の最大値で表せるが、凹関数では 0-1 変数で選ぶ

AMPLによる多重選択定式化

mult_selection.mod

```
param K > 0 integer;
param a {0..K};
param b {0..K};

# 各区分の直線の傾きと切片
param c {k in 0..K-1} = (b[k+1]-b[k]) / (a[k+1]-a[k]);
param dd {k in 0..K-1} = b[k] - c[k]*a[k];

var y {0..K-1};
var z {0..K-1} binary;
var x >= a[0], <= a[K];
var f;

subject to Lo {k in 0..K-1}: y[k] >= a[k]*z[k];
subject to Hi {k in 0..K-1}: y[k] <= a[k+1]*z[k];
subject to One: sum {k in 0..K-1} z[k] = 1;
subject to XDef: x = sum {k in 0..K-1} y[k];
subject to FDef:
   f = sum {k in 0..K-1} (dd[k]*z[k] + c[k]*y[k]);
```

- 傾き c と切片 dd を計算されるパラメータで定義
- 区分点のデータ a, b から決まる値は、モデルファイルの中で定義するのが自然

8.3 一般の区分的線形関数の最小化

不連続な区分的線形関数も扱えるか？

下半連続な区分的線形関数の例

$$f(x) = \begin{cases} 2x & x \in [1, 2) \\ 4 & x \in [2, 3) \\ -3x + 12 & x \in [3, 4] \\ 2x - 6 & x \in (4, 5] \end{cases}$$

- 下半連続：不連続点の近くの値が f(x) に近いか大きい
- 非集約型・対数版・多重選択定式化で最小化できる
  - ただし f(x) ≈ ... を f(x) ≥ ... に置き換える
- 集約型は区分点で値を1つしかもてず適用できない

例の多重選択定式化

$$
\begin{aligned}
&x = y_0 + y_1 + y_2 + y_3 \\
&f(x) \geq (2y_0 + 0z_0) + (0y_1 + 4z_1) + (-3y_2 + 12z_2) + (2y_3 - 6z_3) \\
&1z_0 \leq y_0 \leq 2z_0, \quad 2z_1 \leq y_1 \leq 3z_1 \\
&3z_2 \leq y_2 \leq 4z_2, \quad 4z_3 \leq y_3 \leq 5z_3 \\
&z_0 + z_1 + z_2 + z_3 = 1, \quad z_k \in \{0, 1\}
\end{aligned}
$$

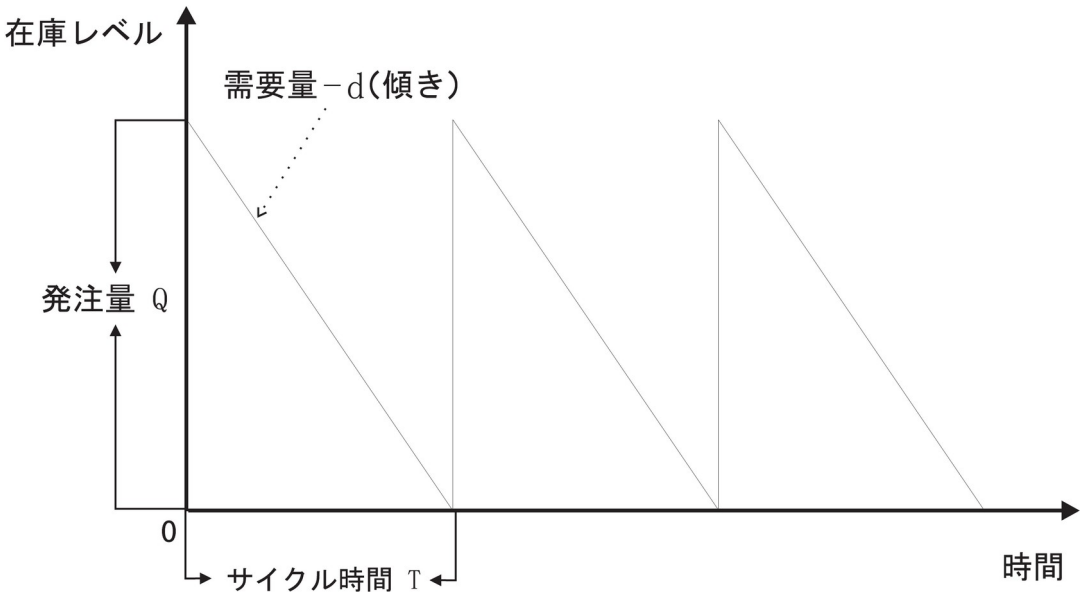
- 各区分の傾きと切片がそのまま係数に現れる
- 区分ごとに範囲を別々に指定できる

8.4

経済発注量問題

凸費用関数をもつ問題の例：容量制約付き複数品目の EOQ

Harris のモデル



在庫レベルの時間的変化

- 需要は一定速度 d、品切れなし、リード時間 0
- 発注費用 F、在庫保管費用 h（1個1日あたり）
- 在庫ゼロで発注 → ノコギリの歯型
- 1日あたり費用 $F/T + hdT/2$
  - $T^* = \sqrt{(2F/(hd))}$、$Q^* = \sqrt{(2Fd/h)}$

容量制約付き複数品目モデル

品目 i の発注費用 F_i、在庫費用 h_i、需要量 d_i、大きさ w_i、倉庫容量 W

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} \left(\frac{F_i}{T_i} + \frac{h_i d_i T_i}{2} \right) \\
\text{subject to} \quad & \sum_{i \in I} w_i d_i T_i \leq W \\
& T_i > 0 \qquad\qquad \forall i \in I
\end{aligned}
$$

- 目的関数は各 T_i の凸関数
- 各品目の費用を凸結合定式化で区分的線形近似する
- 整数変数なしの線形最適化として解ける

AMPLによる区分的線形近似

eoq.mod

```
set I;                          # 品目の集合
param K > 1 integer;            # 区分点の数（区分数は K-1）
param F {I} > 0;  param h {I} > 0;   # 発注費用・在庫保管費用
param d {I} > 0;  param w {I} > 0;   # 需要量・大きさ
param W > 0;                        # 倉庫の容量
param a0 > 0;  param aK > a0;        # 発注間隔の下限・上限
param delta = (aK - a0) / (K-1);
param a {i in I, k in 0..K-1} = a0 + delta*k;
param b {i in I, k in 0..K-1} = F[i]/a[i,k] + h[i]*d[i]*a[i,k]/2;
var y {I, 0..K-1} >= 0, <= 1;
var x {i in I} >= a[i,0], <= a[i,K-1];   # 品目 i の発注間隔
var c {I};                               # 品目 i の総費用
minimize Cost: sum {i in I} c[i];
subject to Convexity {i in I}:
   sum {k in 0..K-1} y[i,k] = 1;
subject to XDef {i in I}:
   sum {k in 0..K-1} a[i,k]*y[i,k] = x[i];
subject to FDef {i in I}:
   sum {k in 0..K-1} b[i,k]*y[i,k] = c[i];
subject to Capacity:
   sum {i in I} w[i]*d[i]*x[i] <= W;
```

区分的線形関数の演算子

eoq.mod の目的関数

```
param slope {i in I, k in 1..K-1}
   = (b[i,k] - b[i,k-1]) / (a[i,k] - a[i,k-1]);
param const {i in I} = b[i,0] - slope[i,1]*a[i,0];

minimize Cost:
   sum {i in I}
      (<<{k in 1..K-2} a[i,k]; {k in 1..K-1} slope[i,k]>> x[i]
       + const[i]);
```

- << 区分点 ; 傾き >> 変数 と書く
- 変数 y も凸結合の制約も不要
- pl_linearize = 0 でソルバーにそのまま渡す
- eoq.mod と同じ最適値 1390.3575
- x は単純な変数に限る

8.5 凹費用関数をもつ施設配置問題

倉庫の費用が出荷量の平方根に比例するとき

凹費用施設配置問題の定式化

施設 j の費用は出荷量の合計 X_j の凹関数 $f_j\sqrt{X_j}$（在庫費用と輸送の固定費用の和を表す）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{m} f_j \sqrt{\sum_{i=1}^{n} x_{ij}} + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} & \\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = d_i & \forall i = 1,\ldots,n \\
& \sum_{i=1}^{n} x_{ij} \le M_j & \forall j = 1,\ldots,m \\
& x_{ij} \ge 0 & \forall i,j
\end{aligned}
$$

- 本では5通りの解き方を示す
- 区分的線形近似：SOS2・整数変数・多重選択
- AMPL の演算子 << >> を使う
- sqrt のまま大域最適化ソルバーで解く

SOS2 による近似と強化式

$0 = a_{j0} < \cdots < a_{jK} = M_j$、$b_{jk} = f_j\sqrt{a_{jk}}$、$[z_{j0}, \ldots, z_{jK}]$ を SOS2 と宣言

$$\text{minimize} \sum_{j=1}^{m}\sum_{k=0}^{K} b_{jk} z_{jk} + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = \sum_{k=0}^{K} a_{jk} z_{jk}, \quad \sum_{k=0}^{K} z_{jk} = 1 \qquad \forall j$$

$$x_{ij} \le \sum_{k=1}^{K} \min\{\hat{d}_{ij}, a_{jk}\} z_{jk} \qquad \forall i, j$$

$$\hat{d}_{ij} = \min\{a_{jk} \mid a_{jk} \ge d_i\}$$

- x[i,j] ≤ d[i]·y[j] に相当する強化式を z で作る
- $\min\{d_i, \cdot\}$ は凹なので、区分点の値の単純な凸結合は上限を過小評価する
- d_i 以上で最小の区分点 dhat[i,j] を使うと上から押さえられる
- 下界は強化されるが計算時間が短縮されるとは限らない

演算子版と sqrt 版のモデル

flp_nonlinear_pl.mod / _sqrt.mod

```
# flp_nonlinear_pl.mod（抜粋）
param U = max {j in J} M[j];
param width = U / (K-1);
param a {j in J, k in 0..K-1} = k * width;
param b {j in J, k in 0..K-1} = f[j] * sqrt(a[j,k]);
param slope {j in J, k in 0..K-2}
   = (b[j,k+1]-b[j,k]) / (a[j,k+1]-a[j,k]);
minimize Cost:
   sum {j in J}
      <<{k in 1..K-2} a[j,k]; {k in 0..K-2} slope[j,k]>> X[j]
   + sum {i in I, j in J} c[i,j] * x[i,j];

# flp_nonlinear_sqrt.mod（抜粋）
minimize Cost:
   sum {j in J} f[j] * sqrt(X[j])
   + sum {i in I, j in J} c[i,j] * x[i,j];

# 共通：x {I,J} >= 0;  X {j in J} >= 0, <= M[j];
subject to Demand {i in I}: sum {j in J} x[i,j] = d[i];
subject to Total {j in J}: sum {i in I} x[i,j] = X[j];
```

凹費用施設配置問題の計算結果

$n = 5$、$m = 3$、$d = (80, 270, 250, 160, 180)$、$M_j = 500$、$f_j = 100$

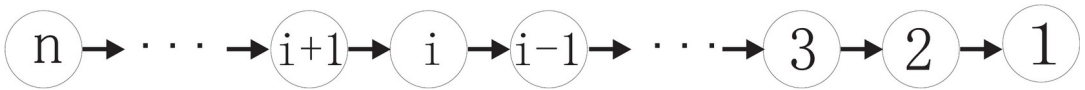
モデル	ソルバー	最適値
flp_nonlinear.mod（多重選択、$K = 20$）	HiGHS	7943.494
flp_nonlinear_pl.mod（<<...>>、$K = 20$）	HiGHS	7943.494
flp_nonlinear_sqrt.mod（sqrt）	SCIP	7943.686

いずれも施設*1*を使わず、施設*2*から *500*、施設*3*から *440* を出荷する。区分的線形近似は凹関数を下から近似するので値がわずかに小さい

8.6 安全在庫配置問題

直列多段階システムで、どこに安全在庫を置くか？

直列多段階モデル



直列多段階モデル

- n 個の在庫点が直列に並ぶ
  - 第 n 段階が調達、第1段階が最終需要
- T_i：第 i 段階の生産時間
- L_i：保証リード時間（$L_1 = 0$）
- $x_i = L_{i+1} + T_i - L_i$：正味補充時間

安全在庫配置問題の定式化

安全在庫量 $f_i(x_i) = D(x_i) - x_i\,\mu$、たとえば $D(t) = \mu\,t + z\,\sigma\,\sqrt{t}$（凹関数）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} h_i f_i(x_i) \\
\text{subject to} \quad & x_i + L_i - L_{i+1} = T_i \quad && \forall i = 1, \ldots, n \\
& L_1 = 0,\ L_{n+1} = 0 \\
& L_i \geq 0 && \forall i = 2, \ldots, n \\
& x_i \geq 0 && \forall i = 1, \ldots, n
\end{aligned}
$$

- D(t)：t 日間の需要の最大値
- 正味補充時間内の最大需要から平均需要を引いた量を保持
- h_i：第 i 段階の在庫費用（1個1日あたり）
- 各段階に SOS2 の凸結合定式化を適用する

AMPLによる安全在庫配置モデル（SOS2）

ssa.mod

```
param n > 0 integer;          # 段階数
set S = 1..n;
param K > 0 integer;          # 区分点の数
param h {S} >= 0;             # 段階 i の在庫費用
param T {S} >= 0;             # 段階 i のリード時間
param b {S, 0..K-1};          # 区分点 k における関数値
param a {i in S, k in 0..K-1} = k;
suffix sosno IN;
suffix ref IN;
var z {i in S, k in 0..K-1} >= 0, <= 1,
   suffix sosno -i, suffix ref k;
var x {S} >= 0;               # 段階 i の正味補充時間
var y {S};                    # 段階 i の安全在庫量
var L {1..n+1} >= 0;          # 段階 i の保証リード時間
minimize Cost: sum {i in S} h[i] * y[i];
subject to XDef {i in S}: x[i] = sum {k in 0..K-1} a[i,k]*z[i,k];
subject to YDef {i in S}: y[i] = sum {k in 0..K-1} b[i,k]*z[i,k];
subject to Convexity {i in S}: sum {k in 0..K-1} z[i,k] = 1;
subject to Balance {i in S}: x[i] + L[i] = T[i] + L[i+1];
subject to Start: L[1] = 0;
subject to End: L[n+1] = 0;
```

- sosno -i で段階ごとに別の SOS2
- 接尾語の値も添え字の式で書ける
- SOS2 対応ソルバー（Gurobi 等）が必要
- HiGHS・SCIP では not implemented エラー

演算子版と sqrt 版の安全在庫配置モデル

ssa_pl.mod / ssa_sqrt.mod

```
# ssa_pl.mod（抜粋）
param slope {i in S, k in 0..K-2}
   = (b[i,k+1]-b[i,k]) / (a[i,k+1]-a[i,k]);
var x {i in S} >= 0, <= a[i,K-1];   # 段階 i の正味補充時間
var L {1..n+1} >= 0;                # 段階 i の保証リード時間
minimize Cost:
   sum {i in S} h[i] *
      (<<{k in 1..K-2} a[i,k]; {k in 0..K-2} slope[i,k]>> x[i]
       + b[i,0]);

# ssa_sqrt.mod（抜粋）
var x {S} >= 0;              # 段階 i の正味補充時間
var L {1..n+1} >= 0;         # 段階 i の保証リード時間
minimize Cost:
   sum {i in S} h[i] * sqrt(x[i]);

# 共通の制約
subject to Balance {i in S}: x[i] + L[i] = T[i] + L[i+1];
subject to Start: L[1] = 0;
subject to End: L[n+1] = 0;
```

安全在庫配置問題の計算結果

$n = 4$、$h_i = i$、$T_i = 2$、区分点 $a_{ik} = k$（$k = 0, \ldots, 9$）、$b_{ik} = \sqrt{k}$

モデル	ソルバー	最適値
ssa.mod（特殊順序集合）	Gurobi	2.828
ssa_pl.mod（<<...>>）	HiGHS	2.828
ssa_sqrt.mod（sqrt）	SCIP	2.828

いずれも $x = (8, 0, 0, 0)$：在庫費用の最も小さい第1段階だけで安全在庫を保持する。最適値は $\sqrt{8} \approx 2.828$

まとめ

- ✓ 凸関数の区分的線形近似は線形最適化（凸結合定式化）で解ける
- ✓ 凹関数には SOS2 か 0-1 変数（非集約型・対数版・多重選択）が要る
- ✓ 不連続な関数には非集約型や多重選択定式化を使う
- ✓ AMPL の << >> 演算子なら定式化を書かずに済む
- ✓ sqrt のまま書いて SCIP などで大域最適化する方法もある
- ✓ HiGHS・AMPL経由の SCIP は一般の SOS2 に非対応