01

第1章

数理最適化入門

簡単な例題を通して数理最適化とモデリング言語AMPLに入門する

この章の内容

1 数理最適化とは

2 線形最適化問題

3 整数最適化問題

4 輸送問題

5 双対問題

6 多品種輸送問題

7 混合問題

8 分数最適化

9 多制約0-1ナップサック問題

10 栄養問題

1.1

数理最適化とは

実際の問題を数式として書き下し、最適解やそれに近い解を得る方法論

数理最適化問題の種類

1つの目的関数を最小化（最大化）し、幾つかの制約式を満たす解を求める

1 線形最適化

目的関数とすべての制約式が線形式。解き易く、実際問題の最適解を極めて短時間で求められる

2 非線形最適化

線形でない式を含む問題。安定して解くことが難しい場合が多い

3 整数最適化

変数が整数に限定された問題。工夫次第で様々な実際問題を表現できる

4 混合整数最適化

一部の変数だけが整数に限定された問題。非線形関数の近似にも使える

1.2 線形最適化問題

簡単な例題で用語を確認し、AMPLとPythonで解いてみる

最初の例題と用語

線形の制約式の下で線形の目的関数を最大化する

$$
\begin{array}{lrrrcl}
\text{maximize} & 15x_1 & +18x_2 & +30x_3 & & \\
\text{subject to} & 2x_1 & +x_2 & +x_3 & \leq & 60 \\
& x_1 & +2x_2 & +x_3 & \leq & 60 \\
& & & x_3 & \leq & 30 \\
& & & x_1, x_2, x_3 & \geq & 0
\end{array}
$$

- 変数：色々変えて良い x1, x2, x3
- 目的関数：最大化したい関数
- 制約（制約式）：変数の範囲を制約
  - $x \geq 0$ は非負制約
- 実行可能解：すべての制約を満たす解
- 最適解：目的関数を最大にする実行可能解

AMPLの2ファイル構成

モデルファイル（.mod）

- 集合・パラメータ・変数の定義
- 目的関数と制約の定義「だけ」
- 数値データは書かない
- 数式をほぼそのまま書き写す
- 同じモデルを違うデータで使い回せる

Pythonファイル（amplpy）

- モデルの読み込み
- データの受け渡し
- ソルバーの設定と求解
- 結果の取得
- ソルバーは1行で取り替えられる

モデルファイル lo1.mod

lo1.mod

```
# lo1.mod
var x1 >= 0;
var x2 >= 0;
var x3 >= 0, <= 30;

maximize Profit:
   15*x1 + 18*x2 + 30*x3;

subject to Cap1:
   2*x1 + x2 + x3 <= 60;

subject to Cap2:
   x1 + 2*x2 + x3 <= 60;
```

- 変数は var で宣言し、下限・上限も宣言に書く
- x3 ≤ 30 は x3 の上限として書く
- セミコロンは宣言の終わりを表す
- 目的関数にも制約にも必ず名前を付ける

Pythonから解く

lo1.py

```
from amplpy import AMPL

ampl = AMPL()
ampl.read("lo1.mod")
ampl.option["solver"] = "highs"
ampl.solve()
assert ampl.solve_result == "solved"

print("Opt. Value=", ampl.get_objective("Profit").value())
for name in ["x1", "x2", "x3"]:
    print(name, ampl.get_variable(name).value())
```

- solver を書き換えるだけでソルバーを交換できる
- 求解後は必ず solve_result を確認する
- 結果：最適値 1230、x1=10, x2=10, x3=30

1.3 整数最適化問題

鶴と亀と蛸の頭の数が32、足の数が80。亀と蛸の数の和を最小にするには？

鶴亀蛸算の定式化

鶴が x 匹、亀が y 匹、蛸が z 匹いるとする

$$
\begin{aligned}
\text{minimize} \quad & y + z \\
\text{subject to} \quad & x + y + z = 32 \\
& 2x + 4y + 8z = 80 \\
& x, y, z \text{ は非負の整数}
\end{aligned}
$$

- 整数条件なしで解くと x=29.3333, y=0, z=2.66667 という変な答え
- 生きている匹数は分割できないので整数条件を付加する
- 整数条件が付加された線形最適化問題が整数最適化問題

AMPLによるモデル puzzle.mod

puzzle.mod

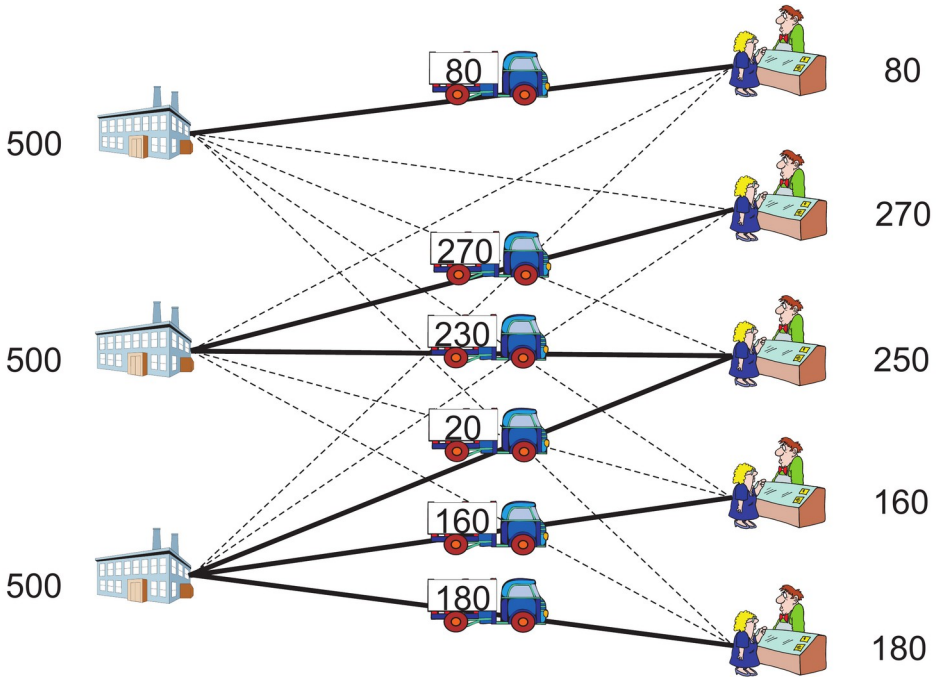
```
# puzzle.mod
var x integer >= 0;
var y integer >= 0;
var z integer >= 0;

minimize TurtleOctopus:
   y + z;

subject to Heads:
   x + y + z = 32;

subject to Legs:
   2*x + 4*y + 8*z = 80;
```

- integer が整数条件を表す
  - 0-1変数なら binary と書く
- 等号は == ではなく = を1つ書く
- 結果：x=28, y=2, z=2（最適値 4）

1.4 輸送問題

3つの工場から5つの顧客へ、総輸送費用が最小になるように製品を運ぶには？

輸送問題の例題



輸送問題の例題と最適輸送量

- 顧客 i の需要量 d[i]
- 工場 j の容量 M[j]（各500）
- 工場 j から顧客 i への単位輸送費用 c[i,j]
- 需要を満たし容量を超えずに総費用を最小化

輸送問題の定式化

x[i,j]：工場 j から顧客 i に輸送される量（連続変数）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in J} x_{ij} = d_i \qquad \forall i \in I \\
& \sum_{i \in I} x_{ij} \leq M_j \qquad \forall j \in J \\
& x_{ij} \geq 0 \qquad \forall i \in I; j \in J
\end{aligned}
$$

- 目的関数：輸送費用の和の最小化
- 1番目の制約：需要を満たす条件
- 2番目の制約：工場の容量制約

集合とパラメータで書く transp.mod

transp.mod

```
set I;                  # 顧客の集合
set J;                  # 工場の集合

param d {I} >= 0;       # 需要量
param M {J} >= 0;       # 容量
param c {I,J} >= 0;     # 輸送費用

var x {I,J} >= 0;       # 輸送量

minimize Cost:
   sum {i in I, j in J} c[i,j] * x[i,j];

subject to Demand {i in I}:
   sum {j in J} x[i,j] = d[i];

subject to Capacity {j in J}:
   sum {i in I} x[i,j] <= M[j];
```

- 数値が1つも現れない「問題」の記述
- {i in I} で ∀i∈I の制約を一度に生成
- データはPythonから set_data などで渡す
- 結果：最適値 3370

1.5 双対問題

どの工場を拡張すれば費用が下がるか？追加注文にはいくら請求すればよいか？

余裕変数と最適双対変数

接尾語 .slack と .dual を get_data で取り出す

制約	添え字	.slack	.dual
Demand（需要）	顧客 1〜5	0, 0, 0, 0, 0	4, 5, 4, 3, 4
Capacity（容量）	工場 1	400	0
	工場 2	0	−1
	工場 3	160	0

容量いっぱいの工場2を拡張すると1単位あたり1の費用削減。追加注文は1単位あたり4,5,4,3,4の費用増

双対問題の導出

1.2節の例題（主問題）の制約に $\pi \geq 0$ を乗じて上界を作る

$$
\begin{array}{lrrrcl}
\text{minimize} & 60\pi_1 & +60\pi_2 & +30\pi_3 & & \\
\text{subject to} & 2\pi_1 & +\pi_2 & & \geq & 15 \\
& \pi_1 & +2\pi_2 & & \geq & 18 \\
& \pi_1 & +\pi_2 & +\pi_3 & \geq & 30 \\
& & & \pi_1, \pi_2, \pi_3 & \geq & 0
\end{array}
$$

- π は双対変数（双対価格、潜在価格）
- x の係数が0以下でないと上界が∞になる
- 強双対定理：一方が最適解をもてば他方ももち、最適値が一致
- 一方が非有界なら他方は実行不可能

1.6 多品種輸送問題

複数の製品を運ぶ。工場ごとに製造できる製品が決まっているときは？

多品種輸送問題の定式化

x[i,j,k]：工場 j から顧客 i に製品 k が輸送される量

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} c_{ijk} x_{ijk} & \\
\text{subject to} \quad & \sum_{j \in J} x_{ijk} = d_{ik} & \forall i \in I; k \in K \\
& \sum_{i \in I} \sum_{k \in K} x_{ijk} \leq M_j & \forall j \in J \\
& x_{ijk} \geq 0 & \forall i \in I; j \in J; k \in K
\end{aligned}
$$

- 製品ごとに需要を満たす
- 全製品の合計量が工場の容量を超えない
- 製品の重量で輸送費用が変わる

疎な添え字集合 ARCS を使う

mctransp.mod

```
set I; set J; set K;
set ARCS within {I,J,K};  # 輸送しうる3つ組

param d {I,K} >= 0;
param M {J} >= 0;
param c {ARCS} >= 0;

var x {ARCS} >= 0;

minimize Cost:
   sum {(i,j,k) in ARCS} c[i,j,k] * x[i,j,k];

subject to Demand {i in I, k in K}:
   sum {j in J: (i,j,k) in ARCS} x[i,j,k]
      = d[i,k];

subject to Capacity {j in J}:
   sum {i in I, k in K: (i,j,k) in ARCS}
      x[i,j,k] <= M[j];
```

- 必要な経路だけに変数を作る
- within は部分集合であることの検査
- コロンの後ろは添え字の条件
- 結果：最適値 43 536

1.7

混合問題

4種類の原料を混ぜて、成分の条件を満たす製品1トンを最小費用で作るには？

混合問題のデータ

各原料に含まれる成分の比率（%）と原料の単価

	成分1	成分2	成分3	単価（万円/t）
原料1	25	15	30	5
原料2	30	30	10	6
原料3	15	65	5	8
原料4	10	5	85	20
製品の条件	10〜20%	35%以下	45%以上	

二重不等式で書く mix.mod

mix.mod

```
set I;                      # 原料の集合
set K;                      # 成分の集合

param p {I} >= 0;           # 単価
param a {I,K} >= 0;         # 成分比率
param LB {K} >= 0;
param UB {k in K} >= LB[k];

var x {I} >= 0;             # 混合比率

minimize Cost:
   sum {i in I} p[i] * x[i];

subject to Total:
   sum {i in I} x[i] = 1;

subject to Content {k in K}:
   LB[k] <= sum {i in I} a[i,k] * x[i] <= UB[k];
```

- 上下限制約を1本の二重不等式で書ける
- $UB \geq LB$ をパラメータの検査として書く
- 結果：原料1 約65%、3 約5%、4 約30%
  - 最適値 9.6216 万円

1.8 分数最適化

鶴と亀の「頭の数と足の数の比」を最小にするには？

変数変換による線形最適化への帰着

$t = 1/(2x+4y)$ とおき、$x' = tx$, $y' = ty$, $z' = tz$ と置き換える

$$
\begin{aligned}
\text{minimize} \quad & \frac{x+y}{2x+4y} \\
\text{subject to} \quad & x+y+z = 32 \\
& 2x+4y+8z = 80 \\
& x, y, z \geq 0
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\text{minimize} \quad & x'+y' \\
\text{subject to} \quad & 2x'+4y' = 1 \\
& x'+y'+z' = 32t \\
& 2x'+4y'+8z' = 80t \\
& x', y', z' \geq 0,\ t > 0
\end{aligned}
$$

- 元の解は x'/t, y'/t, z'/t で得られる
- 結果：最適値 0.4、鶴24匹、亀8匹、蛸0匹（偶然ほぼ整数）

整数の場合：二分探索

fractional2.py

```
ampl = AMPL()
ampl.read("fractional2.mod")
ampl.option["solver"] = "highs"

LB, UB, EPS = 0.0, 1.0, 0.01
while True:
    theta = (UB + LB) / 2
    ampl.param["theta"] = theta
    ampl.solve(verbose=False)
    if ampl.solve_result == "solved":
        UB = theta
        best = (ampl.get_variable("x").value(),
                ampl.get_variable("y").value(),
                ampl.get_variable("z").value())
        if UB - LB <= EPS:
            break
    else:
        LB = theta
```

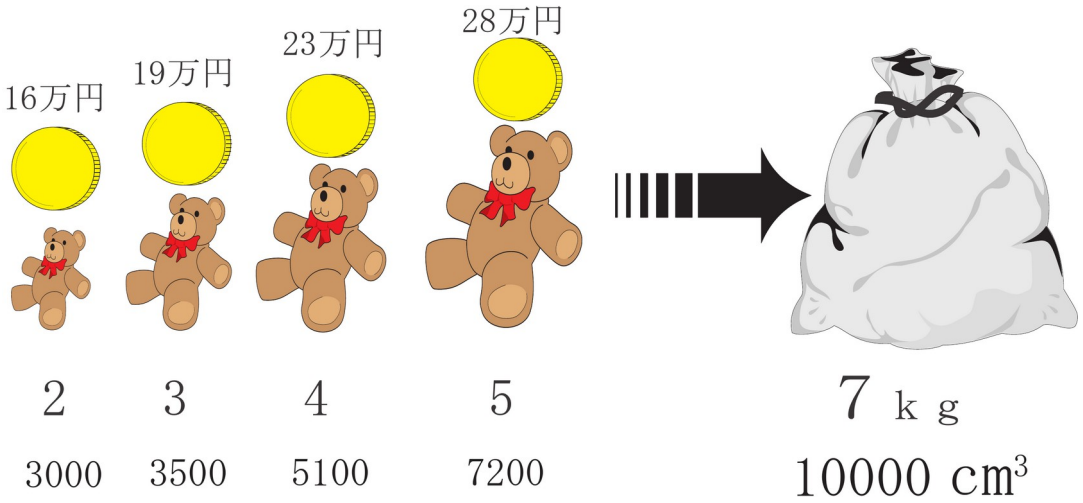
- 比 $\leq \theta$ を線形制約に直す
  - $(2\theta-1)x + (4\theta-1)y \geq 0$
- θ をパラメータにして値だけ差し替える
- 結果：θ=0.40234375、(24, 8, 0)

1.9

多制約0-1ナップサック問題

重さ7kg、容積10リットルまでのナップサックで、価値の合計を最大にするには？

クマさん人形を選ぶ



クマさん人形のラインアップと愛用のナップサック

- 4体のクマさん人形の値段・重さ・容積
- 重さ 7kg、容積 10 000cm³ を超えられない
- 転売価格の合計を最大にしたい
  - 制約1本でも NP-困難

多制約0-1ナップサック問題の定式化

x[j]：アイテム j をナップサックに詰めるとき 1、それ以外のとき 0

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j \in J} v_j x_j \\
\text{subject to} \quad & \sum_{j \in J} a_{ij} x_j \leq b_i \quad \forall i \in I \\
& x_j \in \{0, 1\} \qquad \forall j \in J
\end{aligned}
$$

- v[j]：アイテム j の価値
- a[i,j]：アイテム j の制約 i に対する重み
- b[i]：制約 i の上限値

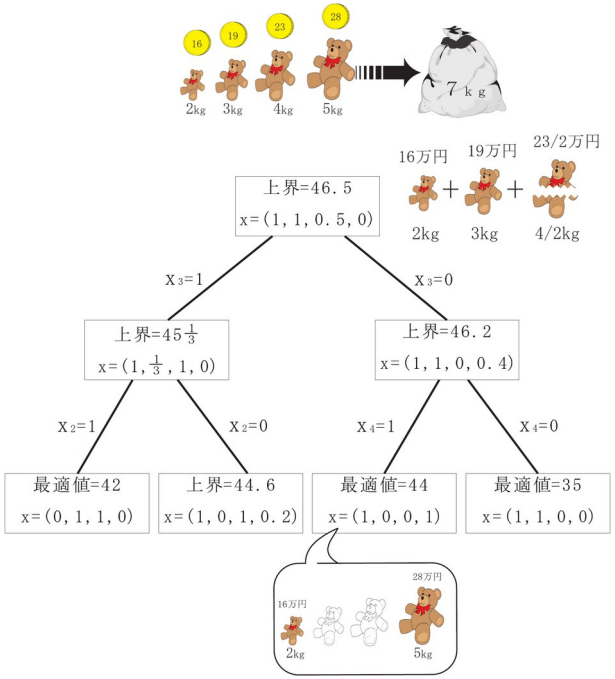
expand で定式化を確認する

mkp.py

```
I, J, v, a, b = example()
ampl = mkp(I, J, v, a, b)
ampl.eval("expand;")

# --- 出力 ---
maximize Value:
   16*x[1] + 19*x[2] + 23*x[3] + 28*x[4];

subject to Knapsack[1]:
   2*x[1] + 3*x[2] + 4*x[3] + 5*x[4] <= 7;

subject to Knapsack[2]:
   3000*x[1] + 3500*x[2] + 5100*x[3]
      + 7200*x[4] <= 10000;
```

- 展開された制約を数式の形で表示
- 誤りの大部分は expand 1回で見つかる
- 結果：クマ2と3を選び 42万円

分枝限定法



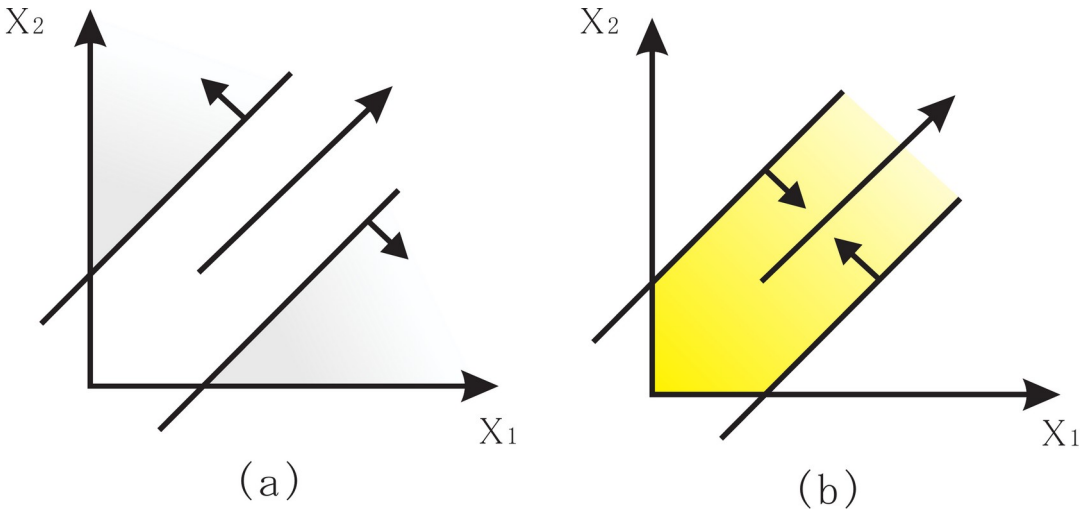
ナップサック問題に対する分枝限定法の適用例

- 線形緩和問題で上界を得る
- 小数の変数で2つに分枝する
- 上界 ≤ 下界なら探索を打ち切る（限定）
- 重量だけの問題の最適値は44

1.10 栄養問題

最適化問題が最適解をもたないときは、どうすればよいか？

実行不可能と非有界



(a) 実行不可能、*(b)* 非有界

- 実行不可能：実行可能解が存在しない
- 非有界：目的関数値が無限に良くなる
- solve_result が infeasible / unbounded で区別
- 非有界と報告されたら目的関数を 0 にして解き直す

弾力化したモデル diet_relax.mod

diet_relax.mod

```
param max_penalty default 1e6;

var x {F} integer >= 0;
var under {N} >= 0;   # 下限を下回った量
var over {N} >= 0;    # 上限を上回った量

minimize Penalty:
   sum {i in N} (under[i]/a[i] + over[i]/b[i]);

minimize Cost:
   sum {j in F} c[j] * x[j];

subject to NutrLB {i in N}:
   sum {j in F} d[j,i] * x[j] + under[i] >= a[i];

subject to NutrUB {i in N}:
   sum {j in F} d[j,i] * x[j] - over[i] <= b[i];

subject to PenaltyLimit:
   sum {i in N} (under[i]/a[i] + over[i]/b[i])
      <= max_penalty;
```

- 元の問題は実行不可能（推奨値がきつい）
- 逸脱量 under, over を追加して弾力化
- 第1段階：Penalty を最小化
- 第2段階：逸脱を固定し Cost を最小化

弾力化したモデルの結果

2510円でまずまず健康な食事をとれる

区分	項目	結果
購入する商品	Fries	10 個
	Milk	3 本
	VegJuice	1 本
制約の逸脱	VitA	上限を 240 超過
	VitC	下限を 27 不足
	その他の栄養素	すべて基準を満たす

どの制約をどれだけ破らざるを得なかったかが一目で分かる

まとめ

- ✓ 数理最適化は目的関数と制約式で問題を記述する
- ✓ AMPLはモデル、Pythonはデータと求解を担う
- ✓ 集合とパラメータで問題の大きさに依存しないモデル
- ✓ 接尾語 .dual で双対変数、添え字集合で疎性を表す
- ✓ 分数最適化は変数変換や二分探索で線形に帰着
- ✓ expand で定式化を確認し、弾力化で実行不可能に対処