AMPL：数理計画のためのモデリング言語



第18章　非線形計画問題

滑らかな非線形モデルの記述と、局所最適解に潜む落とし穴

AGENDA

本章の内容

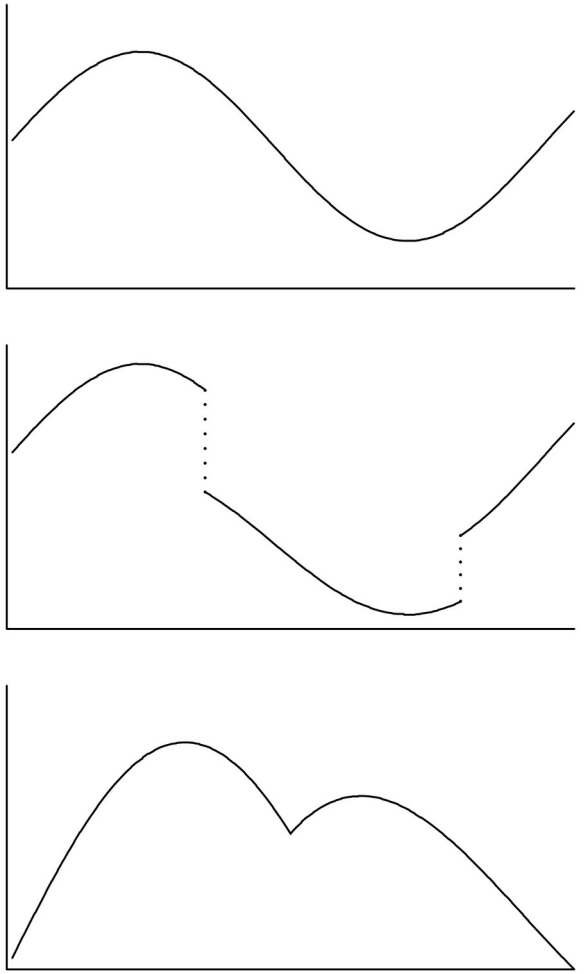
18.1	非線形性の発生源	仮定の緩和・望みの効果・物理過程という 3 つの経路
18.2	非線形変数	初期値の指定と、変数を自動的に代入で除去する機能
18.3	非線形の式	滑らかさを保つ書き方と AMPL 組み込み関数・外部関数
18.4	非線形計画法の落とし穴	値域違反、複数の局所最適解、その他の失敗の様式

非線形計画問題とは何か

- 連続変数（整数・離散ではない）を扱う点は線形計画問題と同じ
- 目的関数・制約の式は線形である必要はないが、「滑らか」でなければならない
- 滑らか＝すべての点で連続かつ勾配（1階微分）が一意に定まること
- **跳躍（不連続）や折れ目（微分不可能）をもつ関数は対象外**

なぜ特別扱いされるのか

他の「線形でない」問題と区別しやすく、応用が広く、確立されたアルゴリズムで解けるため。ただし同程度の線形計画問題より定式化・求解ははるかに難しい。



(a) 滑らかで連続　(b) 不連続　(c) 微分不可能



非線形性が持ち込まれる 3 つの経路

1



線形性の仮定を外す

輸送費用が数量に比例するといった仮定を緩め、区分的線形や滑らかな非線形の費用関数に置き換える

2



望みの効果を構成する

混雑による所要時間の急増など、線形関数では表せない振る舞いを狙って非線形関数を設計する

3



本質的に非線形な過程

パイプラインの圧力と流量の関係など、物理法則そのものが非線形な現象をそのまま定式化する



出発点：線形ネットワーク流モデル

- 都市 i から j への輸送費用は cost[i,j]、輸送量は Ship[i,j]
- LINKS は輸送経路が存在する都市の対の集合
- 各都市でフローが均衡するという制約をもつ
- この線形モデルを土台に、非線形の変種を組み立てていく

AMPL — net1.mod

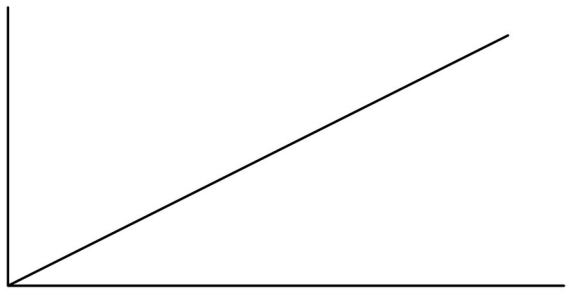
```
minimize Total_Cost:
   sum {(i,j) in LINKS} cost[i,j] * Ship[i,j];

subject to Balance {k in CITIES}:
   supply[k] + sum {(i,k) in LINKS} Ship[i,k]
      = demand[k] + sum {(k,j) in LINKS} Ship[k,j];
```

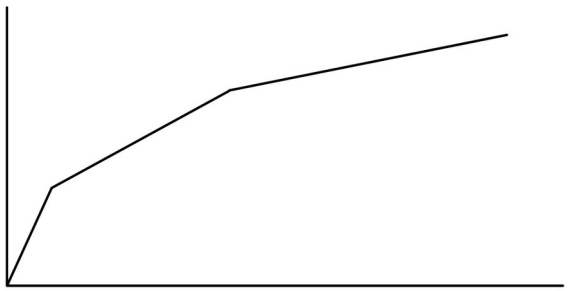
線形性の仮定

i から j への輸送は、どの単位も同じ単価 cost[i,j] を要すると仮定している。
この仮定を緩めることが、非線形化の第一歩になる。

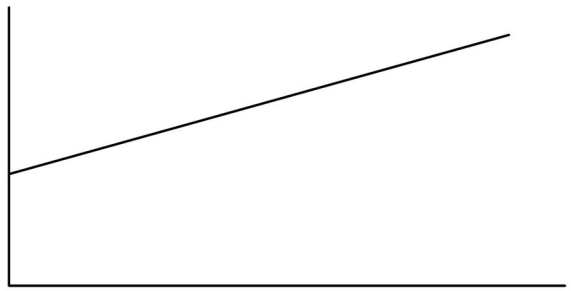
費用関数の形：線形から非線形へ



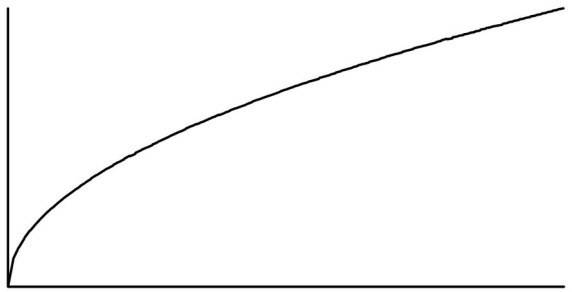
(a) 線形：一定の単価



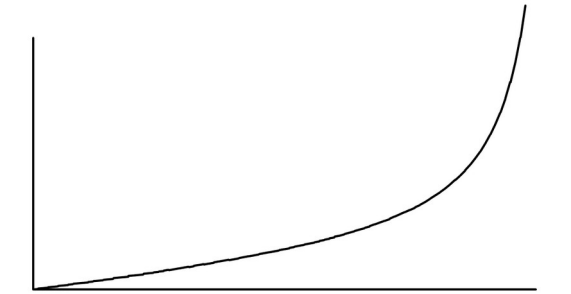
(b) 区分的線形：単価が段階変化



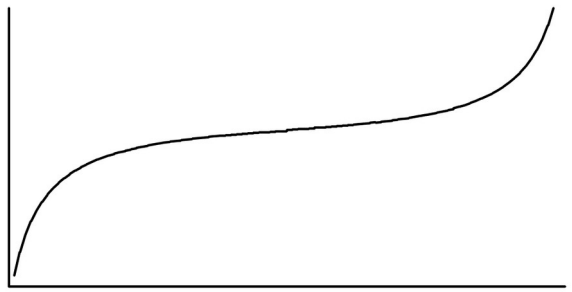
(c) 固定費用：跳躍あり



(d) 凹：数量割引に類似



(e) 凸：上限に近づき急増



(f) 凹凸混在

非線形費用関数の定式化

- cost[i,j]*Ship[i,j] を非線形の式に置き換えるだけでもよい
- 費用を変数 Cost[i,j] にすると、傾きが数量とともに連続変化する状況を表せる
- Ship がゼロ近くでは cost1 に近づき、大きくなると cost2 に落ち着く
- cost1 > cost2 なら図18-2d のような凹の費用曲線が得られる

変数の積 → 非線形

```
var Cost {ORIG,DEST};    # 単価
var Ship {ORIG,DEST} >= 0; # 輸送量

minimize Total_Cost:
   sum {i in ORIG, j in DEST} Cost[i,j] * Ship[i,j];
```

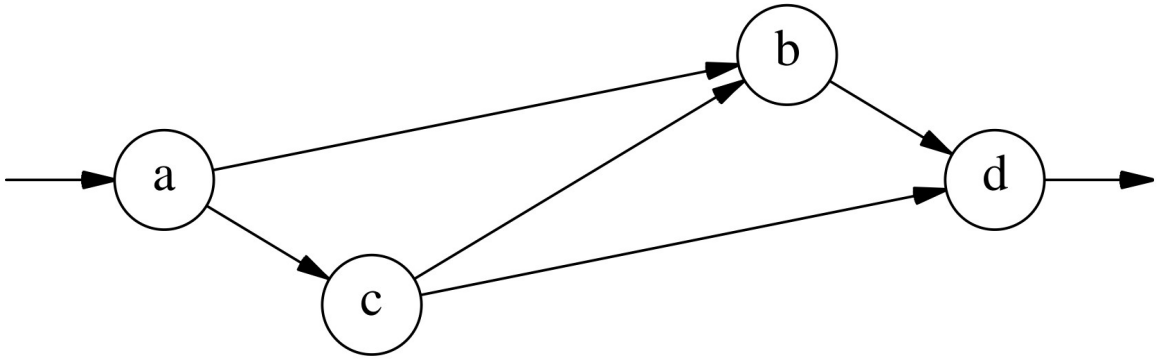
除算を含む方程式

```
subject to Cost_Relation {i in ORIG, j in DEST}:
   Cost[i,j] =
     (cost1[i,j] + cost2[i,j]*Ship[i,j])
     / (1 + Ship[i,j]);
```

ポイント：
積 Cost[i,j]*Ship[i,j] や比の式は AMPL の線形性の規則を満たさないため「非線形」として扱われる。式そのものは自由に書けるが、意味のある滑らかな関数になっているかは利用者の責任。

望みの効果を得る：交通混雑モデル

- 所要時間は交通量が少ないうちはほぼ一定
- リンクの容量 cap[i,j] に近づくにつれ無限大へ急増させたい
- この振る舞いは線形関数では表現できない
- sens[i,j] が両極端の間の形状を左右する



交差点 *a→b→d*、*a→c→d* を経由する交通流ネットワーク（演習*18-4*で詳しく扱う）

所要時間関数

```
time[i,j] +
  (sens[i,j]*Ship[i,j])
  / (1 - Ship[i,j]/cap[i,j])
```

常に凸な関数

Ship[i,j] が小さい値のときは time[i,j] にほぼ等しく、cap[i,j] に近づくにつれて無限大に向かう。図18-2e に似た凸のグラフをもつ。



望みの効果を得る：需要乖離への罰則

- 変数 Discrepancy[k] で実配送量と需要量の乖離を表す（正負どちらも可）
- 乖離が大きくなり過ぎないよう目的関数に罰則項を追加する

区分的線形の罰則

```
minimize Total_Cost:
  sum{(i,j) in LINKS} cost[i,j]*Ship[i,j] +
  sum{k in CITIES}
     pen * <<-1,1; 0>> Discrepancy[k];
```

2次の罰則

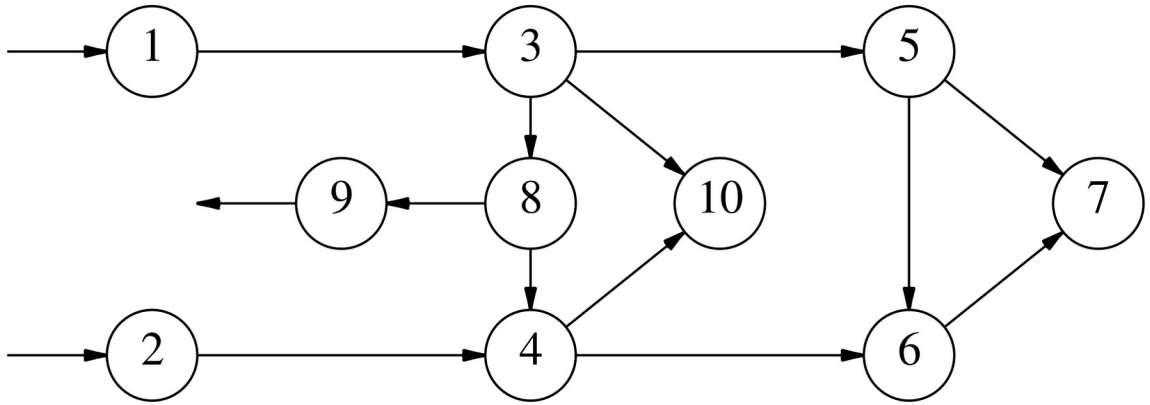
```
minimize Total_Cost:
  sum{(i,j) in LINKS} cost[i,j]*Ship[i,j] +
  sum{k in CITIES}
     pen * Discrepancy[k] ^ 2;
```

乖離が比例的に罰せられるだけで、単位ごとの罰則は一定。大きな乖離を十分には抑止できない。AMPL はこの目的関数を難なく線形に変換する。

乖離が悪化するほど単位当たりの罰則が着実に増大する。近似・データ当てはめでよく使われる「2乗和」の典型例。

本質的に非線形な過程：天然ガスパイプライン

- 流量 Ship[i,j] に加え、各都市 k の圧力 Press[k] も変数として定義
- 圧力が高い都市から低い都市へ流れが向かう
- c[i,j] はパイプの長さ・直径・効率・ガスの性質で決まる定数
- 圧縮機や弁はさらに別の非線形の流量関係を生む



流量・圧力の関係

```
Flow[i,j]^2 =
   c[i,j]^2 * (Press[i]^2 - Press[j]^2)
```

物理モデルの非線形性は経済ではなく物理法則・実測から生じるため形は特定しやすい一方、より複雑な関数形や変数間の相互作用を伴いやすい。

非線形変数：初期値の重要性

- solve すると初期値はソルバーへの出発点の推測として渡される
- 初期値を与えなければ AMPL は暫定的にゼロを設定する
- 良い初期値はソルバーの速度だけでなく、見つかる解そのものを左右し得る

宣言時に指定

```
var Ship {LINKS} >= 0, := 1;

var Ship {(i,j) in LINKS} >= 0,
   := cap[i,j] - 1;
```

data 文で指定

```
var Ship:   FRA   DET   LAN   WIN :=
  GARY      800   400   400   200
  CLEV      800   800   800   600
  PITT      800   800   800   200 ;
```

双対変数にも同様の機能

制約名への代入は、対応する双対変数（ラグランジュ乗数）の初期値として解釈される。MINOS などのソルバーはこれらを利用できる。

非線形変数：自動代入による変数の除去

- 定義変数（= の左辺に単一の変数をもつ制約）は代入で除去できる
- 除去しても変数名は結果の閲覧時に引き続き利用できる
- 式を複雑にする代わりに変数・制約の数を減らせる — 効果はモデル次第

方法1：オプション

```
option substout 1;
solve;

#  「定義」制約の条件：
#  ・左辺に変数がちょうど1つ
#  ・その変数に上下限等の制限なし
#  ・既に代入済みの制約に現れない
```

方法2：= 句で明示

```
var Cost {(i,j) in LINKS}
   = (cost1[i,j] +
      cost2[i,j]*Ship[i,j])
     / (1 + Ship[i,j]);
```

非線形部分は1つだけ保持し代入先リストを添える（オプション linelim=0 で無効化可）ため、線形項のみが実際に複製される。



非線形の式：滑らかさを壊す落とし穴

- AMPL の式構文は自由度が高いため、書けてしまうことと滑らかであることは別問題
- abs・min・max を変数に適用すると一般に滑らかでなくなる
- if_then_else も、変数を含む条件を伴うと不連続になりやすい

滑らかでない例 ①

```
cost[i,j] * abs(Flow[i,j])
```

ゼロで傾きが急変（折れ目）

滑らかでない例 ②

```
if Flow[i,j] = 0 then 0
else base[i,j]+cost[i,j]*Flow[i,j]
```

ゼロで値が跳躍（不連続）

AMPL は文句を言わない

こうした式を使ってもエラーにはならず、ソルバーが「最適解」を返すことさえある。だが結果は誤っている可能性がある。滑らかさの保証はモデラーの責任。

非線形の式：滑らかさを保つ工夫

- 流量の符号で向きを表せば、1対の方程式を1つの if 式にまとめられる
- Flow=0 で2つの2次関数が滑らかに「出会う」ため結果は滑らか

対称的な if の活用

```
(if Flow[i,j] >= 0 then
    Flow[i,j]^2
 else -Flow[i,j]^2)
  = c[i,j]^2*(Press[i]^2-Press[j]^2)
```

特異点の回避

```
if abs(Flow[i,j]-1) > 0.00001 then
   log(Flow[i,j])/(Flow[i,j]-1)
else
   1.5 - Flow[i,j]/2
```

Flow[i,j]の符号で i→j／j→i の向きを判定。定数 c[i,j]=c[j,i] を利用し1本の制約に統合できる。

log(x)/(x-1) は x=1 で 0/0 になりエラー。厳密には滑らかでなくても、数値的には十分滑らかな線形近似に置き換える。

落とし穴①：関数の値域違反

- rate*Trans/(1-Trans/limit) を最小化する非線形輸送モデルで発生
- ソルバーが Trans[i,j] = limit[i,j] を許し、ゼロ除算でエラー
- 出発点を変えても Trans が limit を超え、目的関数が -10^{18} 規模に発散

amplsession

```
ampl: solve;
MINOS 5.5 Error evaluating objective Total_Cost
can't compute 8000/0
MINOS 5.5: solution aborted.
```

修正：上限を課す

```
var Trans {i in ORIG, j in DEST}
   >= 0, <= .9999 * limit[i,j];

# 修正後 → optimal solution found
```

教訓

非線形関数は変数の意図した範囲の外では非常に悪い振る舞いをし得る。特異点（limit[i,j]）から十分に遠ざける上限（.9999倍）を明示的に与える必要がある。



落とし穴②：複数の局所最適解

- Trans^0.8 を導入すると凹凸混在の費用曲線になる
- 出発点 alpha=0.5 → 目的値 355,438　／　alpha=0 → 目的値 427,568
- どちらも局所最適だが、大域最適とは限らない
- 目的関数の凸性＋制約の線形性が揃えば局所最適＝大域最適が保証される

355,438
alpha = 0.5 の最良解

427,568
alpha = 0 の局所解

alpha を変えたときの Total_Cost（一部抜粋）

α=0.0	α=0.2	α=0.5	α=0.6	α=0.9	α=1.0
427568.1	366791.2	355438.2	356531.5	402795.3	365827.2

複数の出発点を系統的に試し、最良の解を採用する — 大域最適化の単純だが実用的な方法。より高度な手法は探索の順序をより体系立てて行う。

その他の落とし穴と実務のコツ

非線形性は制約より目的関数へ

自動代入オプションを使い、可能なら非線形性を目的関数側に集める

変数のスケールを揃える

値がせいぜい2〜3桁しか違わないよう単位を選ぶ。桁違いはソルバーを誤誘導する

現実的な上下限を課す

定常点・符号の取り違え・偽の実行不可能宣言を防ぎ、成功の見込みを高める

解けたら上下限を確認

「安全のための」上下限に解が達していないか確認し、達していれば緩めて解き直す

まとめ

1. **滑らかさが鍵**
   非線形計画問題の対象は連続かつ勾配が一意に定まる「滑らか」な関数に限られる
2. **3つの発生源**
   仮定を外す・望みの効果を構成する・本質的な物理過程という3経路で非線形性が生じる
3. **初期値と自動代入**
   良い初期値はソルバーの成否を左右し、定義変数は代入で除去してモデルを整理できる
4. **滑らかさを壊さない書き方**
   abs・if_then_else の乱用に注意し、対称的な if や近似で滑らかさを保つ
5. **局所最適 ≠ 大域最適**
   値域違反や複数の局所最適解に注意し、複数の出発点や適切な上下限で対処する

次章：第19章へ続く