

最適化・シミュレーション演習 第3回 動的計画法

1. Knapsack Problem

次のナップサック問題 (KP) を考える。品物 1,2,3 の値段はそれぞれ 7,8,3(万円)、重さはそれぞれ 3,4,2(kg)である。重量制限 6 kgのナップサックに入る品物のうち、最も値段の和が大きくなるような組合せを求めよ。

決定変数は以下のように定義される。

$x_j=0$, 品物 j をナップサックに入れる

$x_j=1$, 品物 j をナップサックに入れない

(KP) 最大化 $z = 7x_1 + 8x_2 + 3x_3$

制約 $3x_1 + 4x_2 + 2x_3 \leq 6, \quad x_j \in \{0,1\}, \quad \forall j$

最適解は $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$ 最適目的関数値 $z = 11$

次のような3つのファイルを作業フォルダに作成する。

knapsack.mod ファイル: ナップサック問題の連続緩和問題を記述したファイル

knapsack.dat ファイル: ナップサック問題のデータを記述したファイル

knapsack.run ファイル: AMPL のコマンドを記述したファイル

以下に内容を記す。ただし「#」以降はコメントである。

ファイル knapsack.mod の内容

```
param Nitem;                #品物数を Nitem と定義
param weight{i in 1..Nitem}; #各品物に重さ (weight) を定義
param value{i in 1..Nitem};  #各品物に価値 (value) を定義
param l{i in 1..Nitem};      #0-1 制約の緩和制約における下限
param u{i in 1..Nitem};      #0-1 制約の緩和制約における上限
param capacity;              #ナップサックの容量
var x{i in 1..Nitem} binary; #各品物 (入れる/入れない) に対応する変数
maximize Profit: sum{i in 1..Nitem} value[i]*x[i]; #目的関数 Profit
subject to Capacity_Constraint: sum{i in 1..Nitem} weight[i]*x[i] <= capacity;
                                     #ナップサックの容量制約
subject to Upper_Lower_Bound {i in 1..Nitem}: l[i] <= x[i] <= u[i];
                                     #0-1 制約を緩和した上下限
```

ファイル knapsack-relax.dat の内容

```
param Nitem:= 3;
param value :=
1 10
2 6
3 2 ;
param weight :=
1 4
2 3
3 2 ;
param capacity:=
5;
param l :=
1 0
2 0
```

```

3 0;
param u :=
1 1
2 1
3 1
;

```

ファイル knapsack.run の内容

```

model knapsack.mod ;
data knapsack.dat ;
option display_round 6; #小数点以下 6 桁表示
option solver cplex;      #ソルバに CPLEX を指定、これがないと MINOS が動く
solve;                    #問題の求解
expand > knapsack.sol;    #ファイル(knapsack.sol)に定式化表示
display Profit > knapsack.sol; #ファイル(knapsack.sol)に目的関数値
display x > knapsack.sol;  #ファイル(knapsack.sol)に最適解

```

この問題を動的計画法により解く。

最適性の原理を表す再帰方程式： $f_k(\lambda) = \max \{ f_{k-1}(\lambda), c_k + f_{k-1}(\lambda - a_k) \}$

上の式を漸化式と考えて、最適解を求める。

ただし、 $f_k(\lambda)$ を $f_k(\lambda) = \max \{ \sum_{j=1}^k c_j x_j \mid \sum_{j=1}^k a_j x_j \leq \lambda, x_j \in \{0,1\}, j=1,\dots,k \}$ と定義する。
 また初期設定として次の値を用いる。： $f_0(\lambda) = 0$ ($\lambda \geq 0$ のとき), $f_0(\lambda) = -\infty$ ($\lambda < 0$ のとき)
 $f_k(\lambda) = 0$ ($\lambda = 0$ のとき), $f_k(\lambda) = -\infty$ ($\lambda < 0$ のとき)

以下に動的計画法のプログラムを示す。

```

#knapsack-dp.run
param Nitem;
param value{i in 1..Nitem};
param weight{i in 1..Nitem};
param capacity;
param l{i in 1..Nitem};
param u{i in 1..Nitem};
param f{k in 0..Nitem, la in -capacity..capacity}; #関数  $f_k(\lambda)$  を定義、 $\lambda$  をインデックス la
data knapsack.dat;
option display_transpose -20; #行列表示パラメータ
for{k in 0..Nitem, la in -capacity..-1}
{
    let f[k, la] := -1000;    #初期設定、実行不可能な場合を除外するため十分に小さく
}
for{la in 0..capacity}
{
    let f[0, la] := 0;        #初期設定、変数 k の個数が 0 の場合
}
for{k in 1..Nitem}
{
    let f[k, 0] := 0;         #初期設定、容量  $\lambda$  の値が 0 の場合
}

```

```

#以下は DP の本体部分
for {k in 1..Nitem} #変数個数 k=1, ..., Nitem の値まで反復
{
    for {la in 1..capacity} #λ=1, ..., capacity の値まで反復
    {
        if(
            ) then #  $f_{k-1}(\lambda) \geq c_k + f_{k-1}(\lambda - a_k)$ 
        {
            let
                #  $f_k(\lambda) = f_{k-1}(\lambda)$ 
        }
        if(
            ) then #  $f_{k-1}(\lambda) < c_k + f_{k-1}(\lambda - a_k)$ 
        {
            let
                #  $f_k(\lambda) = c_k + f_{k-1}(\lambda - a_k)$ 
        }
    }
}
display f > knapsack-dp.sol;

```

結果は以下の通りである。

```

#knapsack-dp.sol
f [*,*]
:   -6    -5    -4    -3    -2    -1    0    1    2    3    4    5    6 :=
0  -1000  -1000  -1000  -1000  -1000  -1000  0    0    0    0    0    0
1  -1000  -1000  -1000  -1000  -1000  -1000  0    0    0    7    7    7    7
2  -1000  -1000  -1000  -1000  -1000  -1000  0    0    0    7    8    8    8
3  -1000  -1000  -1000  -1000  -1000  -1000  0    0    3    7    8    10   11
;
#下線部以外は初期設定の値による。

```

2. ロットサイズ決定問題

T 期間のロットサイズ決定問題を考える。決定変数として t 期における生産量と t 期末の在庫量をそれぞれ、 x_t, I_t と定義する。また、 t 期における需要、製造固定費、単位製品あたりの製造変動費、期末在庫維持費をそれぞれ d_t, a_t, v_t, h_t とする。次の表で与えられる数値を用いて、3 期間のロットサイズ決定問題を考える。

期 t	1 月	2 月	3 月
需要 d_t	10	10	10
固定費 a_t	100	120	100
変動費 v_t	20	30	10
期末在庫費 h_t	8	2	(3) 3 月期末在庫は 0 なので考えない

以下に定式化を示す。

$$\text{最小化 } z = \sum_{t=1, \dots, T} (a_t y_t + v_t x_t + h_t I_t)$$

$$\text{制約 } I_{t-1} + x_t - d_t = I_t, \quad t=1, \dots, T \quad (\text{流量保存})$$

$$x_t \leq M y_t \text{ ただし } M \text{ は十分大きい正数、} M = \sum d$$

$$I_t \geq 0 \text{ (品切れ不許可)}, \quad x_t \geq 0, \quad \forall t$$

$$I_0 = 0, \quad I_T = 0$$

AMPL のモデルファイルは以下の通りである。

```
#ww.mod
```

```
param term; #計画期間数 T はインデックスで用いることの多い t と混同しないようにした。
```

```
param a{i in 1..term};
```

```
param v{i in 1..term};
```

```
param h{i in 1..term};
```

```
param d{i in 1..term};
```

```
var y{i in 1..term} binary;
```

```
var x{i in 1..term} >=0;
```

```
var inv{i in 0..term} >=0; #在庫量 I はインデックスの i と混同しないよう inv とした。
```

```
minimize cost: sum {i in 1..term} (a[i]*y[i]+v[i]*x[i]+h[i]*inv[i]);
```

```
s.t. c1 {i in 1..term} :inv[i-1]+x[i]-d[i]=inv[i];
```

```
s.t. c2 :inv[0]=0;
```

```
s.t. c3 {i in 1..term} :x[i]<=30*y[i]; #大きい正数 M=Σd
```

```
#ww.dat
param term := 3;
param a:=
1 100  2 120  3 100;
param d:=
1 10  2 10  3 10;
param v:=
1 20  2 30  3 10;
param h:=
1 8  2 2  3 3;
```

この問題を解くための動的計画法の再帰方程式は次のように表される。

$$\text{再帰方程式: } f_t = \min_{k=t+1, \dots, T+1} \{ c_{t,k-1} + f_k \}$$

ただし、 f_t と $c_{t,k-1}$ を次のように定義する。

$f_t = t-1$ 期の期末在庫量 $I_{t-1} = 0$ のときに、 t 期以降の最小費用

$c_{t,k-1} = t$ 期の生産量を $= d_t + d_{t+1} + \dots + d_{k-1}$ としたときの t 期から $k-1$ 期までの総費用

解答:

$f_4 = 0$ と定める。

$$f_3 = 100 + 10 \times 10 + f_4 = 200^* \text{ (3 月に 3 月分生産)}$$

$$\begin{aligned} f_2 &= \min\{2 \text{ 月分のみ生産, } 2-3 \text{ 月分同時生産}\} \\ &= \min\{120 + 30 \times 10 + f_3, 120 + 30 \times 20 + 2 \times 10 + f_4\} \\ &= \min\{620^*, 740\} = 620 \text{ (2 月分のみ生産)} \end{aligned}$$

$$\begin{aligned} f_1 &= \min\{1 \text{ 月分のみ生産, } 1-2 \text{ 月分同時生産, } 1-3 \text{ 月分同時生産}\} \\ &= \min\{100 + 20 \times 10 + f_2, 100 + 20 \times 20 + 8 \times 10 + f_3, 100 + 20 \times 30 + 8 \times 20 + 2 \times 10 + f_4\} \\ &= \min\{920, 780^*, 880\} = 780 \text{ (1 月に 1-2 月分生産)} \end{aligned}$$

これより、最適解は次のようになる。総費用は 780 である。

$$x_1, = 20, I_1 = 10, x_3, = 10$$

```

option solver cplex;
model ww.mod;
data ww.dat;
solve;
display cost;
display x, y, inv;

param f{i in 1..term+1};
param t;
param c{i in 1..term, j in 1..term};

let t:=term;
let f[term+1]:=0;
for{i in 1..term}
{
    let f[i]:=10000;
}
repeat while (t>0) #t 期から後の最適費用が f[t]なので計算は t>0 で行う
{
    display t >ww.sol;
    for{k in t+1..term+1}
    {
        let c[t,k-1]:=0;
        let c[t,k-1]:=c[t,k-1]+a[t]+v[t]*sum{j in t..k-1}d[j]; #固定費と変動費
        for{l in t..k-2}
        {
            let c[t,k-1]:=c[t,k-1]+h[l]*sum{j in l+1..k-1}d[j]; #在庫費用
        }

        print 'k=',k, ' c[t,k-1]=' , c[t,k-1], ' f[k]=' , f[k] >ww.sol;
        if (
            )then
        {
            let
                ; #f[t]の更新
        }
    }
    printf:"***** ¥n" >ww.sol;
    let t:=t-1;
}
display f >ww.sol;
display f;

quit;

```