NUOPT/SIMPLE 例題集

株式会社 数理システム
Phone: 03-3358-1701
Fax: 03-3358-1727
Email: nuopt-support@msi.co.jp

2010/11/15

目次

1．はじめに

本例題集では，様々な数理計画問題を紹介し，それらに対する SIMPLE での定式化の例を与えます．扱う問題の構成は以下の通りです．表における ○ は，それぞれの問題が，どのような種類の数理計画問題に属するかを表しています．例えば，ナップサック問題は混合線形整数計画問題です．

LP は線形計画問題， MIP(MILP) は混合線形整数計画問題， QP は二次計画問題， NLP は非線形計画問題，SDP は半正定値計画問題，WCSP は重み付き制約充足問題， RCPSP は資源制約付きスケジューリング問題を意味します．

問題	LP	MIP	QP	NLP	SDP	WCSP	RCPSP
配合問題	○						
輸送問題	○						
多期間計画問題	○						
包絡分析法（DEA）モデル	○						
ナップサック問題		○					
集合被覆問題		○					
最大流問題	○						
最小費用流問題	○						
多品種流問題	○						
p メディアン問題		○					
p センター問題		○					
巡回セールスマン問題		○					
割当問題		○				○	
二次割当問題						○	
設備計画問題						○	
最小二乗問題			○				
ポートフォリオ最適化問題			○				
ロジスティック回帰モデル				○			
イールドカーブ推定問題				○			
格付け推移行列推定問題				○			
相関行列取得問題					○		
ロバストポートフォリオ最適化問題					○		
隣接行列（最大カット問題）		○			○	○	
セミナー割当問題							○
ジョブショップスケジューリング問題							○
遷移確率行列	○						
ムーア・ペンローズ一般逆行列	○						

2．各種例題の紹介および SIMPLE での記述例

本章では， 1 章で紹介した各種例題に対する問題の説明および，SIMPLE での記述例を紹介します．

2.1 配合問題

配合問題の例として，ここでは特定の組成を持つ合金を生成する問題を扱います．この他にも薬剤の調合や必要な栄養素を含む献立を考えるダイエット問題など配合問題として扱えるものは多岐にわたります．

（例題）

鉛，亜鉛，スズの構成比率が，それぞれ 30%，30%，40%となるような合金を，市販の合金を混ぜ合わせ，できるだけ安いコストで生成することを考えます．現在手に入れることができる市販の合金は 9 種類で，それらの構成比率と単位量あたりのコストは以下の通りです．

市販の合金	1	2	3	4	5	6	7	8	9
鉛(%)	20	50	30	30	30	60	40	10	10
亜鉛(%)	30	40	20	40	30	30	50	30	10
スズ(%)	50	10	50	30	40	10	10	60	80
コスト($/lb)	7.3	6.9	7.3	7.5	7.6	6.0	5.8	4.3	4.1

所望の組成を持つ合金をコストを一番安く生成するには，市販の合金をどのように混ぜ合わせれば良いでしょうか．

この問題を NUOPT で解くために定式化を行います．本例題は文献[1]からの引用です．

まず，変数として市販の合金 1,2,3,…,9 の混合比率，つまり混ぜ合わせる割合を，それぞれ $x_1, x_2, x_3, \ldots, x_9$ としましょう．

次に，最小化するべき目的関数は，各市販の合金について「単位量当たりのコスト」と「混合比率」の積の総和として表現することができます．

最後に制約条件です．まず，混合比率は負の値をとれませんので，各変数に対して非負制約が必要です．混合比率の総和は 1 ですので，その制約も加えます．また，生成する合金の組成についての制約は，鉛，亜鉛，スズに対して，各市販の合金についての「構成比率」と「混合比率」の積の総和が，それぞれ 30%，30%，40%と等しい，という形になります．

以上のことから，次のように定式化することができます.

変数	x_1	市販の合金１の混合比率
	x_2	市販の合金２の混合比率
	x_3	市販の合金３の混合比率
	x_4	市販の合金４の混合比率
	x_5	市販の合金５の混合比率
	x_6	市販の合金６の混合比率
	x_7	市販の合金７の混合比率
	x_8	市販の合金８の混合比率
	x_9	市販の合金９の混合比率

目的関数（最小化） 総コスト

$$7.3x_1 + 6.9x_2 + 7.3x_3 + 7.5x_4 + 7.6x_5 + 6.0x_6 + 5.8x_7 + 4.3x_8 + 4.1x_9$$

非負制約

$$x_1 \geq 0$$
$$x_2 \geq 0$$
$$x_3 \geq 0$$
$$x_4 \geq 0$$
$$x_5 \geq 0$$
$$x_6 \geq 0$$
$$x_7 \geq 0$$
$$x_8 \geq 0$$
$$x_9 \geq 0$$

混合比率の制約

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 = 1$$
$$0.2x_1 + 0.5x_2 + 0.3x_3 + 0.3x_4 + 0.3x_5 + 0.6x_6 + 0.4x_7 + 0.1x_8 + 0.1x_9 = 0.3$$
$$0.3x_1 + 0.4x_2 + 0.2x_3 + 0.4x_4 + 0.3x_5 + 0.3x_6 + 0.5x_7 + 0.3x_8 + 0.1x_9 = 0.3$$
$$0.5x_1 + 0.1x_2 + 0.5x_3 + 0.3x_4 + 0.4x_5 + 0.1x_6 + 0.1x_7 + 0.6x_8 + 0.8x_9 = 0.4$$

この問題は，目的関数，制約式全て線形なので，線形計画問題となります.

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 変数
Variable x1(name="市販の合金 1 の混合比率");
Variable x2(name="市販の合金 2 の混合比率");
Variable x3(name="市販の合金 3 の混合比率");
Variable x4(name="市販の合金 4 の混合比率");
Variable x5(name="市販の合金 5 の混合比率");
Variable x6(name="市販の合金 6 の混合比率");
Variable x7(name="市販の合金 7 の混合比率");
Variable x8(name="市販の合金 8 の混合比率");
Variable x9(name="市販の合金 9 の混合比率");
// 目的関数
Objective z(name="総コスト", type=minimize);
z=7.3*x1+6.9*x2+7.3*x3+7.5*x4+7.6*x5+6.0*x6+5.8*x7+4.3*x8+4.1*x9;
// 非負制約
x1 >= 0;
x2 >= 0;
x3 >= 0;
x4 >= 0;
x5 >= 0;
x6 >= 0;
x7 >= 0;
x8 >= 0;
x9 >= 0;
// 混合比率の制約
x1+x2+x3+x4+x5+x6+x7+x8+x9 == 1;
0.2*x1+0.5*x2+0.3*x3+0.3*x4+0.3*x5+0.6*x6+0.4*x7+0.1*x8+0.1*x9==0
.3;
0.3*x1+0.4*x2+0.2*x3+0.4*x4+0.3*x5+0.3*x6+0.5*x7+0.3*x8+0.1*x9==0
.3;
0.5*x1+0.1*x2+0.5*x3+0.3*x4+0.4*x5+0.1*x6+0.1*x7+0.6*x8+0.8*x9==0
.4;
// 求解
solve();
// 出力
z.val.print();
```

より汎用的に問題を定式化すると以下のようになります.

集合	$Alloy = \{1,2,3,4,5,6,7,8,9\}$ $Blend = \{Lead, Zinc, Tin\}$	市販の合金の種類の集合 構成金属の集合
定数	$r_{ij}, i \in Alloy, j \in Blend$	市販の合金iにおける構成金属jの比率
	$c_i, i \in Alloy$	市販の合金iの単位量あたりのコスト
	$b_j, j \in Blend$	構成金属jの目標比率
変数	$x_i, i \in Alloy$	市販の合金iの混合比率
目的関数（最小化）	$\sum_{i \in Alloy} c_i x_i$	総コスト
制約	$x_i \geq 0, \forall i \in Alloy$	非負制約
	$\sum_{i \in Alloy} x_i = 1$	比率の総和が１という制約
	$\sum_{i \in Alloy} r_{ij} x_i = b_j, \forall j \in Blend$	混合比の制約

次に，定数（構成比率，コスト，目標比率）をデータファイルから与える SIMPLE モデルを示します．このようにモデルとデータを分離することにより，市販の合金の数や構成金属の種類の数が変わったとしてもデータファイルを変更するだけで対応できるようになります．

```
// 集合と添字
Set Alloy(name="市販の合金集合");
Element i(set=Alloy);
Set Blend(name="構成金属集合");
Element j(set=Blend);

// パラメータ
Parameter r(name="構成比率", index=(i,j));
Parameter c(name="コスト", index=i);
Parameter b(name="目標比率", index=j);

// 変数
Variable x(name="混合比率", index=i);

// 目的関数
Objective z(name="総コスト", type=minimize);
z = sum(c[i]*x[i],i);

// 非負制約
x[i] >= 0;

// 混合比の制約
sum(x[i],i) == 1;
sum(r[i,j]*x[i],i) == b[j];

// 求解
solve();

// 出力
z.val.print();
x.val.print();
```

データファイル（.dat 形式）は以下のようになります.

```
構成比率 =
[1,Lead] 0.2 [1,Zinc] 0.3 [1, Tin] 0.5
[2,Lead] 0.5 [2,Zinc] 0.4 [2, Tin] 0.1
[3,Lead] 0.3 [3,Zinc] 0.2 [3, Tin] 0.5
[4,Lead] 0.3 [4,Zinc] 0.4 [4, Tin] 0.3
[5,Lead] 0.3 [5,Zinc] 0.3 [5, Tin] 0.4
[6,Lead] 0.6 [6,Zinc] 0.3 [6, Tin] 0.1
[7,Lead] 0.4 [7,Zinc] 0.5 [7, Tin] 0.1
[8,Lead] 0.1 [8,Zinc] 0.3 [8, Tin] 0.6
[9,Lead] 0.1 [9,Zinc] 0.1 [9, Tin] 0.8
;

コスト =
[1] 7.3
[2] 6.9
[3] 7.3
[4] 7.5
[5] 7.6
[6] 6.0
[7] 5.8
[8] 4.3
[9] 4.1
;

目標比率 =
[Lead] 0.3
[Zinc] 0.3
[ Tin] 0.4
;
```

このモデルを実行すると，市販の合金 6 を 40%，市販の合金 8 を 60%混ぜ合わせるのが最適で，そのときの総コストは 4.98 であることがわかります.

2.2 輸送問題

輸送問題は複数の供給地から複数の需要地への物の流れ方を決める問題ということができます．供給地と需要地をノード，物の流れる経路をアークとすれば，輸送問題はネットワークによって表現できる問題の一種と捉えることができます．輸送問題に対する定式化の方法は他のネットワークによって表現できる問題にも応用できます．

（例題）

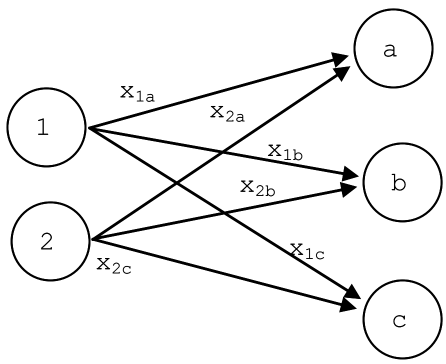
ある配送業者は二つの工場 1，2 から三つの店舗 a，b，c への製品の輸送を請け負っているとします．各工場，店舗について，それぞれ供給可能量と需要量が決められており，それらを満たしつつ，最もコストがかからない製品の運び方をどのように決定すればよいでしょうか．各工場の供給可能量，各店舗の需要量，単位量あたりの輸送コストは以下の通りです．

工場	供給可能量	店舗	需要量
1	250	a	200
2	450	b	200
		c	200

単位量あたりの輸送コスト

	a	b	c
1	3.4	2.2	2.9
2	3.4	2.4	2.5

この問題を NUOPT で解くために定式化を行います．本例題は文献[1]からの引用です．
まず，変数として各工場から各店舗への輸送量を以下の図のように設定します．



次に，目的関数は，工場から店舗への各経路について「単位量あたりの輸送コスト」と「輸送量」の積の総和として表現することができます.

最後に制約条件です．輸送量は負にはなり得ないので，各変数に対して非負制約が必要です．さらに，各工場について工場からの輸送量の和が供給可能量以下である，各店舗について店舗への輸送量の和が需要量と等しい，という制約が加わります.

以上のことから，次のように定式化することができます.

変数	x_{1a}	工場 1 から店舗 a への輸送量
	x_{1b}	工場 1 から店舗 b への輸送量
	x_{1c}	工場 1 から店舗 c への輸送量
	x_{2a}	工場 2 から店舗 a への輸送量
	x_{2b}	工場 2 から店舗 b への輸送量
	x_{2c}	工場 2 から店舗 c への輸送量

目的関数（最小化）　　総コスト

$$3.4x_{1a} + 2.2x_{1b} + 2.9x_{1c} + 3.4x_{2a} + 2.4x_{2b} + 2.5x_{2c}$$

非負制約	$x_{1a} \geq 0$	
	$x_{1b} \geq 0$	
	$x_{1c} \geq 0$	
	$x_{2a} \geq 0$	
	$x_{2b} \geq 0$	
	$x_{2c} \geq 0$	
工場の生産量の制約	$x_{1a} + x_{1b} + x_{1c} \leq 250$	工場 1 について
	$x_{2a} + x_{2b} + x_{2c} \leq 450$	工場 2 について
店舗の需要量の制約	$x_{1a} + x_{2a} = 200$	店舗 a について
	$x_{1b} + x_{2b} = 200$	店舗 b について
	$x_{1c} + x_{2c} = 200$	店舗 c について

ネットワークで表現できる問題の多くは，上で見てきたように各アークに対して変数を定義し，各ノードについて制約をたてることによって定式化されます.

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 変数
Variable x1a(name="工場 1 から店舗 a への輸送量");
Variable x1b(name="工場 1 から店舗 b への輸送量");
Variable x1c(name="工場 1 から店舗 c への輸送量");
Variable x2a(name="工場 2 から店舗 a への輸送量");
Variable x2b(name="工場 2 から店舗 b への輸送量");
Variable x2c(name="工場 2 から店舗 c への輸送量");

// 非負制約
x1a >= 0;
x1b >= 0;
x1c >= 0;
x2a >= 0;
x2b >= 0;
x2c >= 0;

// 工場の生産量の制約
x1a + x1b + x1c <= 250;
x2a + x2b + x2c <= 450;

// 店舗の需要量の制約
x1a + x2a == 200;
x1b + x2b == 200;
x1c + x2c == 200;

// 目的関数
Objective z(name="総コスト", type=minimize);
z = 3.4*x1a + 2.2*x1b + 2.9*x1c + 3.4*x2a + 2.4*x2b + 2.5*x2c;

// 求解
solve();

// 出力
z.val.print();
```

より汎用的に問題を定式化すると以下のようになります.

集合	$Cannery = \{1, 2\}$	工場
	$Warehouse = \{a, b, c\}$	店舗
定数	$upper_i, i \in Cannery$	工場 i の供給可能量
	$demand_j, j \in Warehouse$	店舗 j の需要量
	$c_{ij}, i \in Cannery, j \in Warehouse$	工場 i から店舗 j への単位量あたりの輸送コスト
変数	$x_{ij}, i \in Cannery, j \in Warehouse$	工場 i から店舗 j への輸送量
目的関数（最小化）	$\sum_{i \in Cannery} \sum_{j \in Warehouse} c_{ij} x_{ij}$	総コスト
制約	$x_{ij} \geq 0, \forall i \in Cannery, \forall j \in Warehouse$	非負制約
	$\sum_{j \in Warehouse} x_{ij} \leq upper_i, \forall i \in Cannery$	工場の生産量の制約
	$\sum_{i \in Cannery} x_{ij} = demand_j, \forall j \in Warehouse$	店舗の需要量の制約

次に，入力データとモデルを分離した SIMPLE モデルを示します.

```
// 集合と添字
Set Cannery(name="工場");
Element i(set=Cannery);
Set Warehouse(name="店舗");
Element j(set=Warehouse);

// パラメータ
Parameter upper(name="供給可能量", index=i);
Parameter demand(name="需要量", index=j);
Parameter cost(name="輸送コスト", index=(i,j));

// 変数
Variable x(name="輸送量", index=(i,j));

// 目的関数
Objective z(name="総コスト", type=minimize);
z = sum(cost[i,j]*x[i,j],(i,j));

// 非負制約
x[i,j] >= 0;

// 工場の生産量の制約
sum(x[i,j],j) <= upper[i];

// 店舗の需要量の制約
sum(x[i,j],i) == demand[j];

// 求解
solve();

// 出力
z.val.print();
x.val.print();
```

データファイル（.dat 形式）は以下のようになります.

```
供給可能量 =
[1] 250
[2] 450
;

需要量 =
[a] 200
[b] 200
[c] 200
;

輸送コスト =
[1,a] 3.4
[1,b] 2.2
[1,c] 2.9
[2,a] 3.4
[2,b] 2.4
[2,c] 2.5
;
```

このモデルを実行すると，以下のような結果が得られます.

```
総コスト=1620
輸送量[1,"a"]=26.4602
輸送量[1,"b"]=200
輸送量[1,"c"]=1.79049e-008
輸送量[2,"a"]=173.54
輸送量[2,"b"]=3.58099e-008
輸送量[2,"c"]=200
```

2.3 多期間計画問題

多期間計画問題とは，多期間にわたり期間ごとに意思決定をする問題のことをいいます．多期間計画問題を定式化する場合は，期間ごとに変数を定義するのが一般的です．

（例題）

2 種類の原料 A，B を加工して 2 種類の製品 I，II を生産している工場が，向こう３ヶ月間の生産計画を立てようとしています．各製品を 1 単位生産するために必要な原料の使用量，各製品の生産／在庫コスト，各製品の月ごとの出荷量，各原料の月ごとの利用可能量は以下のように与えられています．

原料使用量			生産／在庫コスト		
	I	II		I	II
A	2	7	生産	75	50
B	5	3	在庫	8	7

製品の出荷量			原料の利用可能量		
	I	II		A	B
1	30	20	1	920	790
2	60	50	2	750	600
3	80	90	3	500	480

各月に出荷する製品をその月中に全て生産できるとは限らないので，前の月に生産した製品を在庫として保管して来月に出荷することも考えられます．このような状況の下で，要求された製品の出荷量と与えられた原料の利用可能量の制約を満たしつつ総コストを最小にするには，各月における各製品の生産量と在庫量をどのように決定すればよいでしょうか．

この問題を NUOPT で解くために定式化を行います．本例題は文献[2]からの引用です．

まず，変数として各月における製品 I，II の生産量をそれぞれ $x_{\mathrm{I}1}, x_{\mathrm{II}1}, x_{\mathrm{I}2}, x_{\mathrm{II}2}, x_{\mathrm{I}3}, x_{\mathrm{II}3}$ とし，在庫量をそれぞれ $y_{\mathrm{I}1}, y_{\mathrm{II}1}, y_{\mathrm{I}2}, y_{\mathrm{II}2}$ としましょう．

次に，最小化するべき目的関数は，各生産量／在庫量とその単位量あたりのコストとの積の総和として表現することができます．

最後に制約条件です．まず，各変数に対して非負制約が必要です．次に，1 ヶ月に利用できる原料が決められているので，各月ごとに各原料に関する制約が必要です．さらに，各製品に対して在庫量は次の月に持ち越せますので，それを踏まえた出荷量の制約を加えます．

以上のことから，次のように定式化することができます．

変数	$x_{\mathrm{I}1}$	製品Ｉの１ヶ月目の生産量
	$x_{\mathrm{II}1}$	製品 II の１ヶ月目の生産量
	$x_{\mathrm{I}2}$	製品Ｉの２ヶ月目の生産量
	$x_{\mathrm{II}2}$	製品 II の２ヶ月目の生産量
	$x_{\mathrm{I}3}$	製品Ｉの３ヶ月目の生産量
	$x_{\mathrm{II}3}$	製品 II の３ヶ月目の生産量
	$y_{\mathrm{I}1}$	製品Ｉの１ヶ月目の在庫量
	$y_{\mathrm{II}1}$	製品 II の１ヶ月目の在庫量
	$y_{\mathrm{I}2}$	製品Ｉの２ヶ月目の在庫量
	$y_{\mathrm{II}2}$	製品 II の２ヶ月目の在庫量

目的関数（最小化）　　総コスト

$$75x_{\mathrm{I}1}+50x_{\mathrm{II}1}+8y_{\mathrm{I}1}+7y_{\mathrm{II}1}+75x_{\mathrm{I}2}+50x_{\mathrm{II}2}+8y_{\mathrm{I}2}+7y_{\mathrm{II}2}+75x_{\mathrm{I}3}+50x_{\mathrm{II}3}$$

非負制約	$x_{\mathrm{I}1}\ge 0$，$x_{\mathrm{II}1}\ge 0$	
	$x_{\mathrm{I}2}\ge 0$，$x_{\mathrm{II}2}\ge 0$	
	$x_{\mathrm{I}3}\ge 0$，$x_{\mathrm{II}3}\ge 0$	
	$y_{\mathrm{I}1}\ge 0$，$y_{\mathrm{II}1}\ge 0$	
	$y_{\mathrm{I}2}\ge 0$，$y_{\mathrm{II}2}\ge 0$	
原料の制約	$2x_{\mathrm{I}1}+7x_{\mathrm{II}1}\le 920$	原料Ａについて（1ヶ月目）
	$5x_{\mathrm{I}1}+3x_{\mathrm{II}1}\le 790$	原料Ｂについて（1ヶ月目）
	$2x_{\mathrm{I}2}+7x_{\mathrm{II}2}\le 750$	原料Ａについて（2ヶ月目）
	$5x_{\mathrm{I}2}+3x_{\mathrm{II}2}\le 600$	原料Ｂについて（2ヶ月目）
	$2x_{\mathrm{I}3}+7x_{\mathrm{II}3}\le 500$	原料Ａについて（3ヶ月目）
	$5x_{\mathrm{I}3}+3x_{\mathrm{II}3}\le 480$	原料Ｂについて（3ヶ月目）
出荷量の制約	$x_{\mathrm{I}1}-y_{\mathrm{I}1}=30$	製品Ｉについて（1ヶ月目）
	$x_{\mathrm{II}1}-y_{\mathrm{II}1}=20$	製品 II について（1ヶ月目）
	$x_{\mathrm{I}2}+y_{\mathrm{I}1}-y_{\mathrm{I}2}=60$	製品Ｉについて（2ヶ月目）
	$x_{\mathrm{II}2}+y_{\mathrm{II}1}-y_{\mathrm{II}2}=50$	製品 II について（2ヶ月目）
	$x_{\mathrm{I}3}+y_{\mathrm{I}2}=80$	製品Ｉについて（3ヶ月目）
	$x_{\mathrm{II}3}+y_{\mathrm{II}2}=90$	製品 II について（3ヶ月目）

問題を SIMPLE で記述すると以下のようになります.

```
// 変数
Variable x11(name="製品Ｉの１ヶ月目の生産量");
Variable x21(name="製品 II の１ヶ月目の生産量");
Variable x12(name="製品Ｉの２ヶ月目の生産量");
Variable x22(name="製品 II の２ヶ月目の生産量");
Variable x13(name="製品Ｉの３ヶ月目の生産量");
Variable x23(name="製品 II の３ヶ月目の生産量");
Variable y11(name="製品Ｉの１ヶ月目の在庫量");
Variable y21(name="製品 II の１ヶ月目の在庫量");
Variable y12(name="製品Ｉの２ヶ月目の在庫量");
Variable y22(name="製品 II の２ヶ月目の在庫量");
// 目的関数
Objective z(name="総コスト", type=minimize);
z=75*x11+50*x21+8*y11+7*y21+75*x12+50*x22+8*y12+7*y22+75*x13+50*x23;
// 非負制約
x11 >= 0; x21 >= 0;
x12 >= 0; x22 >= 0;
x13 >= 0; x23 >= 0;
y11 >= 0; y21 >= 0;
y12 >= 0; y22 >= 0;
// 原料の制約
2*x11 + 7*x21 <= 920;
5*x11 + 3*x21 <= 790;
2*x12 + 7*x22 <= 750;
5*x12 + 3*x22 <= 600;
2*x13 + 7*x23 <= 500;
5*x13 + 3*x23 <= 480;
// 出荷量の制約
x11       - y11 == 30;
x21       - y21 == 20;
x12 + y11 - y12 == 60;
x22 + y21 - y22 == 50;
x13 + y12       == 80;
x23 + y22       == 90;
// 求解
solve();
// 出力
z.val.print();
```

より汎用的に問題を定式化すると以下のようになります.

集合	$Product = \{\mathrm{I}, \mathrm{II}\}$	製品
	$Material = \{A, B\}$	原料
	$Period = \{1, 2, 3\}$	期間
定数	$use_{ij}, i \in Product, j \in Material$	製品 i を生産するのに必要な原料 j の使用量
	$out_{it}, i \in Product, t \in Period$	t ヶ月目の製品 i の出荷量
	$upper_{jt}, j \in Material, t \in Period$	t ヶ月目の原料 j の利用可能量
	$costp_i, i \in Product$	製品 i の生産コスト
	$costi_i, i \in Product$	製品 i の在庫コスト
変数	$x_{it}, i \in Product, t \in Period$	製品 i の t ヶ月目の生産量
	$y_{it}, i \in Product, t \in Period$	製品 i の t ヶ月目の在庫量
目的関数（最小化）	$\sum_{t \in Period} \sum_{i \in Product} (costp_{it} x_{it} + costi_{it} y_{it})$	総コスト
制約	$x_{it} \geq 0, \forall i \in Product, \forall t \in Period$	生産量の非負制約
	$y_{it} \geq 0, \forall i \in Product, \forall t \in Period$	在庫量の非負制約
	$\sum_{i \in Product} use_{ij} x_{it} \leq upper_{jt}$ $, \forall j \in Material, \forall t \in Period$	原料の使用量の制約
	$x_{i1} - y_{i1} = out_{i1}$	1 ヶ月目の出荷量について
	$x_{i2} + y_{i1} - y_{i2} = out_{i2}$	2 ヶ月目の出荷量について
	$x_{i3} + y_{i2} = out_{i3}$	3 ヶ月目の出荷量について

次に，定数をデータファイルから与える場合のモデルを示します．

```
// 集合と添字
Set Product(name="製品");
Element i(set=Product);
Set Material(name="原料");
Element j(set=Material);
OrderedSet Period(name="期間");
Element t(set=Period);
// パラメータ
Parameter use(name="原料使用量", index=(i,j));
Parameter out(name="出荷量", index=(i,t));
Parameter upper(name="取扱可能量", index=(j,t));
Parameter costp(name="生産コスト", index=i);
Parameter costi(name="在庫コスト", index=i);
// 変数
Variable x(name="生産量", index=(i,t));
Variable y(name="在庫量", index=(i,t));
// 目的関数
Objective z(name="総コスト", type=minimize);
z = sum(costp[i]*x[i,t]+ costi[i]*y[i,t], (i, t));
// 非負制約
x[i,t] >= 0;
y[i,t] >= 0;
// 原料の制約
sum(use[i,j]*x[i,t],i) <= upper[j,t];
// 在庫量の制約
x[i,t]          - y[i,t] == out[i,t], t == 1;
x[i,t] + y[i,t-1] - y[i,t] == out[i,t], t == 2;
x[i,t] + y[i,t-1]         == out[i,t], t == 3;
// 求解
solve();
// 出力
z.val.print();
x.val.print();
y.val.print();
```

データファイル（.dat 形式）は以下のようになります.

```
原料使用量 =
["I", "A"] 2
["I", "B"] 5
["II", "A"] 7
["II", "B"] 3
;

出荷量 =
["I", 1] 30
["I", 2] 60
["I", 3] 80
["II", 1] 20
["II", 2] 50
["II", 3] 90
;

取扱可能量 =
["A", 1] 920
["A", 2] 750
["A", 3] 500
["B", 1] 790
["B", 2] 600
["B", 3] 480
;

生産コスト =
["I"] 75
["II"] 50
;

在庫コスト =
["I"] 8
["II"] 7
;
```

このモデルを実行すると，以下のような結果が得られます.

```
総コスト=21199.2
生産量["I",1]=38
生産量["I",2]=67.8621
生産量["I",3]=64.1379
生産量["II",1]=20
生産量["II",2]=86.8966
生産量["II",3]=53.1034
在庫量["I",1]=8
在庫量["I",2]=15.8621
在庫量["I",3]=6.15667e-07
在庫量["II",1]=2.24177e-06
在庫量["II",2]=36.8966
在庫量["II",3]=7.03619e-07
```

なお，一つ注意点があります.

上記の求解実行において，在庫量に関する添字付きの変数を 3 ヶ月目についても用意しています（在庫量["I",3]，在庫量["II",3]）．これらは，この章のはじめに定式化した際には，用意していなかった変数です.

しかし，これらの変数について非負制約以外の制約がないこと，および目的関数（総コスト）は最小化についてのものであること，この両者から，在庫量["I",3]，在庫量["II",3] の求解結果は上記のようにほぼ 0 と等しい値になります.

従って在庫量["I",3]，在庫量["II",3] を定義しても，一般性は失われません.

2.4 包絡分析法（DEA）モデル

包絡分析法（DEA；Data Envelopment Analysis）とは，複数の企業体の相対的な効率性を測定する方法で，評価対象の具体例としては，銀行・病院があげられます．

以下の例題では，チェーン展開している 6 店舗に対して，DEA に基づいた効率性の評価を行います．なお，本節では[3]を参考文献としています．

（例題）

ある会社は，以下の 6 店舗を抱えている．

店舗番号	1	2	3	4	5	6
店員数	5	10	20	20	30	50
稼働時間	24	12	12	24	12	12
売上	2	3	5	12	12	20

各店舗が，全店舗に対して相対的に効率的であるかどうかを判定せよ．

この問題の定式化は次のようになります．

集合	I	入力項目集合
	J	出力項目集合
	K	店舗集合
	$\overline{K}$	対象店舗（単一要素からなる集合）
定数	$inD_{k,i}, \quad k \in K, i \in I$	入力データ
	$outD_{k,j}, \quad k \in K, j \in J$	出力データ
変数	$inW_i, \quad i \in I$	入力項目に対する重み
	$outW_j, \quad j \in J$	出力項目に対する重み
目的関数（最大化）	$\sum_{\overline{k},j} outD_{\overline{k},j} outW_j$	対象店舗に対する出力
制約式	$\sum_i inD_{\overline{k},i} inW_i = 1, \quad \forall \overline{k} \in \overline{K}$	対象店舗の入力条件

$\sum_j outD_{k,j} outW_j \leq \sum_i inD_{k,i} inW_i,$ $\forall k \in K$	全店舗に対する入出力条件
$0 \leq inW_i, \quad \forall i \in I$ $0 \leq outW_j, \quad \forall j \in J$	重みの非負条件

以下で，本問題の定式化の説明を行います．

DEA では，本例題の店舗のような分析対象を DMU（Decision Making Unit）と呼びます．全ての DMU を一度に評価することは困難なため，店舗集合 K に属するある店舗 $\bar{k} \in \bar{K} \subset K$ に着目した問題を考えます．

対象とする問題は，効率を最大化するような問題です．この効率 $f_{\bar{k}}$ を以下で定義することとします．

$$f_{\bar{k}} = \frac{\sum_j outD_{\bar{k},j} outW_j}{\sum_i inD_{\bar{k},i} inW_i}$$

即ち，対象店舗 $\bar{k}$ に対する，（総出力/総入力）を効率と定め，これを最大化するような問題を扱います．なお，この問題の変数となる各入出力項目に対する重みは，各店舗共通で与えられるものとします．制約条件としては，任意の店舗に関して，（総出力/総入力）が 1 以下になるという条件が必要です．即ち，

$$\frac{\sum_j outD_{k,j} outW_j}{\sum_i inD_{k,i} inW_i} \leq 1, \quad \forall k \in K$$

となります．また，各入出力項目に対する重みの非負制約も考慮する必要があります．なお，この問題は目的関数・制約式ともに分数式で表わされる分数計画問題となります．これらの分数関数の分母は正であるため，制約式は両辺に分母部分をかけても一般性は失われなく，また目的関数 $f_{\bar{k}}$ についても，分母部分を 1 と等価であるという制約式で表わして，分子部分のみを目的関数としても一般性は失われません．

このようにして，等価な線形計画問題に置き換えた問題の定式化がはじめに示したものとなります．

以上が，本問題に対する定式化の説明です．

この問題を SIMPLE で記述すると，以下のようになります．

```
// 入出力項目集合
Set I(name="入力項目集合"); Element i(set=I);
Set J(name="出力項目集合"); Element j(set=J);
// 店舗集合
Set K(name="店舗集合"); Element k(set=K);
// 対象店舗
Set Kbar(name="対象店舗", superSet=K); Element kbar(set=Kbar);
// 入力データ
Parameter inD(name="入力データ", index=(k,i));
// 出力データ
Parameter outD(name="出力データ", index=(k,j));
// 変数
// 入力項目に対する重み
Variable inW(name="inW", index=i);
// 出力
Variable outW(name="outW", index=j);
// 目的関数
Objective f(name="f", type=maximize);
f = sum(outD[kbar,j]*outW[j], (kbar,j));
// 制約式
sum(inD[kbar,i]*inW[i], i) == 1;
sum(outD[k,j]*outW[j], j) <= sum(inD[k,i]*inW[i], i);
// 非負条件
0 <= inW[i];
0 <= outW[j];
options.method = "simplex"; // 単体法の利用
// 求解
solve();
// 結果出力
f.val.print();
inW.val.print();
outW.val.print();
```

入力データとしては以下の 3 種類（csv ファイル：2 種類，dat ファイル：1 種類）を用意する必要があります.

（csv 形式ファイル）

```
入力データ,定員数,稼働時間
1,5,24
2,10,12
3,20,12
4,20,24
5,30,12
6,50,12
```

```
出力データ,売上
1,2
2,6
3,10
4,12
5,12
6,20
```

（dat 形式ファイル）

```
対象店舗 = 1;
```

対象店舗（店舗番号：1～6）に関して逐次変化させて解いた結果を以下の表にまとめます.
なお，一部結果は小数点以下第四位を四捨五入した値で表わしています.

店舗番号		1	2	3	4	5	6
目的関数値		0.667	1	1	1	0.9	1
入力項目	重み（店員数）	0.2	0.067	0.04	0.033	0.025	0.017
	重み（稼働時間）	0	0.028	0.017	0.014	0.021	0.014
出力項目	重み（売上）	0.333	0.167	0.1	0.083	0.075	0.05

目的関数値が 1 となる店舗（店舗番号：2，3，4，6）は効率的であるといえます．一方，1 より小さな値となる店舗（店舗番号：1，5）は，各種制約を満たすどのような重みを選択しても効率が 1 になりえないということで，非効率であるということになります.

非効率な店舗に関しては,非効率となっている原因を形成している店舗集合を参照集合として導きだすことも可能です．詳細については，[4] を参照ください.

2.5 ナップサック問題

ナップサック問題は, ナップサックの中にいくつかの品物を詰め込み入れた品物の総価値を最大にするという問題です．ただし，ナップサックと品物にはそれぞれ容量が与えられていて，入れた品物の総容量がナップサックの容量を超えてはいけないという条件があります. この問題は, 組合せ最適化問題の代表的な例の一つとしてよく知られていて, プロジェクトの選択や物資の購入などの問題に応用されています．以下は，整数ナップサック問題と呼ばれるものです．なお，0－1 ナックサック問題につきましては，本節の最後で紹介します．

（例題）

容量 65 のナップサックに次の表にある品物を詰め込むことにします．この時，詰め込んだ品物の総価値を最大にするためには何をいくつ詰め込むと良いでしょうか．ただし，同じ品物を何個詰め込んでも良いものとします．

品物	1 個あたりの価値	1 個あたりの容量
缶コーヒー	１２０	１０
水入りペットボトル	１３０	１２
バナナ	８０	７
りんご	１００	９
おにぎり	２５０	２１
パン	１８５	１６

まず，変数は各品物を詰め込む個数です．よって，この変数は整数値しか取らないということになります. ただし,「-1 個詰め込む」というようなありえない答えを排除する必要があります. このため，各変数は 0 以上の値しか取らないということを制約条件として明示しておく必要があります．なお，「0 個詰め込む」は「その品物を詰め込まない」と解釈します．

次に，最大化することになる目的関数は詰め込んだものの総価値です．これは，各品物について「1 個あたりの価値」と「その品物を詰め込んだ個数」の積を求め，その総和を取ることで表現できます．

制約条件は，先ほど述べた変数に関するものの他に，詰め込んだ品物の総容量がナップサックの容量を超えないというものがあります．目的関数の時と同様に考えると，各品物に関する「1 個あたりの容量」と「その品物を詰め込んだ個数」の積の総和をとると詰め込んだ品物の総容量が得られます．よって，この総和がナップサックの容量である 65 を超えないということを式で表せばよいことになります．

以上のことから，この例題は次のように定式化することが出来ました．

整数変数	$coffee$	缶コーヒーの個数
	$water$	水入りペットボトルの個数
	$banana$	バナナの個数
	$apple$	りんごの個数
	$rice\_ball$	おにぎりの個数
	$bread$	パンの個数
目的関数（最大化）	$120coffee+130water+80banana+100apple+250rice\_ball+185bread$	総価値を最大化する
制約条件	$10coffee+12water+7banana+9apple+21rice\_ball+16bread\le 65$	容量に関する制約
	$coffee\ge 0$	缶コーヒーは 0 個以上詰め込む
	$water\ge 0$	水入りペットボトルは 0 個以上詰め込む
	$banana\ge 0$	バナナは 0 個以上詰め込む
	$apple\ge 0$	りんごは 0 個以上詰め込む
	$rice\_ball\ge 0$	おにぎりは 0 個以上詰め込む
	$bread\ge 0$	パンは 0 個以上詰め込む

定式化した結果を SIMPLE で記述すると次のようになります.

```
//整数変数を宣言する
IntegerVariable coffee(name= "缶コーヒーの個数");
IntegerVariable water(name="水入りペットボトルの個数");
IntegerVariable banana(name="バナナの個数");
IntegerVariable apple(name="りんごの個数");
IntegerVariable rice_ball(name="おにぎりの個数");
IntegerVariable bread(name="パンの個数");

//総価値を最大化する
Objective total_value(name="総価値",type=maximize);
total_value=120*coffee+130*water+80*banana+100*apple+
            250*rice_ball+185*bread;

//容量に関する制約
10*coffee+12*water+7*banana+9*apple+21*rice_ball+16*bread<=65;

//各品物は 0 個以上詰め込む
coffee>=0;
water>=0;
banana>=0;
apple>=0;
rice_ball>=0;
bread>=0;

//求解し結果を出力する
solve();
coffee.val.print();
water.val.print();
banana.val.print();
apple.val.print();
rice_ball.val.print();
bread.val.print();
total_value.val.print();
```

このモデルを NUOPT で実行すると，最後に

```
缶コーヒーの個数=3
水入りペットボトルの個数=0
バナナの個数=2
りんごの個数=0
おにぎりの個数=1
パンの個数=0
総価値=770
```

という表示がされます．そして，この表示から「缶コーヒーを 3 個，バナナを 2 個，そしておにぎりを 1 個詰め込むと良い」というこの例題の答えを確認できます．

ところで，このモデルについて品物の種類などを変更したい場合 SIMPLE での記述を修正する箇所が多く大変な手間がかかってしまいます．この対策として，ナップサックの容量，品物の価値および品物の容量を別に用意した dat ファイルから与えることにします．このようにすることで，SIMPLE での記述が汎用的なものになり，品物の種類が変わったとしても dat ファイルの変更のみで対応できるようになります．そのために，ここでは「品物の集合」という概念を導入します．すると定式化は次のように書き直すことができます．

集合	$Object=\{$缶コーヒー, 水入りペットボトル, バナナ, りんご, おにぎり, パン$\}$	品物の集合
整数変数	$count_i \quad ,i \in Object$	品物 i を詰め込む個数
定数	$capacity$	ナップサックの容量
	$value_i \quad ,i \in Object$	品物 i の 1 個あたりの価値
	$weight_i \quad ,i \in Object$	品物 i の 1 個あたりの容量
目的関数（最大化）	$\sum_{i \in Object} value_i \cdot count_i$	総価値を最大化する
制約条件	$\sum_{i \in Object} weight_i \cdot count_i \le capacity$	容量に関する制約
	$count_i \ge 0, \forall i \in Object$	各品物は 0 個以上詰め込む

この定式化を SIMPLE で記述すると，以下のような簡潔なものになります．なお，品物の集

合の具体的な要素については NUOPT では dat ファイルから自動的に認識します.

```
//品物の集合を宣言する
Set Object;
Element i(set=Object);

//データファイルから与えるパラメータを宣言する
Parameter capacity(name="ナップサックの容量");
Parameter value(name="品物の価値",index= i);
Parameter weight(name="品物の容量",index= i);

//整数変数を宣言する
IntegerVariable count(name="詰め込む個数",index= i);

//総価値の最大化
Objective total_value(name="総価値",type=maximize);
total_value=sum(value[i]*count[i], i);

//容量に関する制約
sum(weight[i]*count[i], i)<=capacity;

//各品物は 0 個以上詰め込む
count[i]>=0;

//求解し結果を出力する
solve();
count.val.print();
total_value.val.print();
```

なお，今回の例題についての dat ファイルは次のようになります.

```
ナップサックの容量=65;
品物の価値=
[缶コーヒー]120
[水入りペットボトル]130
[バナナ]80
[りんご]100
[おにぎり]250
[パン]185;
品物の容量=
[缶コーヒー]10
[水入りペットボトル]12
[バナナ]7
[りんご]9
[おにぎり]21
[パン]16;
```

最後に，この例題では「缶コーヒーが 3 個」というように容量が許す限り同じ品物を何個でも詰め込むことができました．それでは，各品物についてナップサックに詰め込むことができるのは 1 個だけということにするとどのようになるでしょうか．なお，この制限を加えると 0-1 ナップサック問題という典型的な 0-1 整数計画問題になります．SIMPLE では，0-1 変数であるという宣言を簡単に行うことができます．具体的には，先ほど汎用化させた SIMPLE モデル中で変数を宣言している部分を

```
IntegerVariable count(name="詰め込む個数",index=i, type=binary);
```

とするだけです．さらに，この宣言から各変数がとりうる値は 0 か 1 しかないということが明らかなため，各品物は 0 個以上詰め込むという制約「count[i]>=0;」を記述する必要がなくなります．以上の点についてモデルファイルを書き換えた上で， NUOPT で実行させると，

```
詰め込む個数["おにぎり"]=1
詰め込む個数["りんご"]=1
詰め込む個数["バナナ"]=1
詰め込む個数["パン"]=1
詰め込む個数["缶コーヒー"]=0
詰め込む個数["水入りペットボトル"]=1
総価値=745
```

という結果が得られ，缶コーヒー以外の品物を詰め込むと良いという事がわかります．

2.6 集合被覆問題

集合 U とその部分集合の族および各部分集合に対応するコストが与えられているものとします．この時，全ての U の要素をカバーするように部分集合の族から部分集合を選び，その際にかかるコストを最小にするという問題が集合被覆問題です．応用例には乗務員スケジューリング問題などがあります．なお集合被覆問題の場合， U の各要素について複数個の部分集合でカバーすることを許しています．このことに関連して，複数個の部分集合でカバーすることを許さない場合，集合分割問題と呼ばれ，選挙区の設定問題などに応用されています．

（例題）

ある企業は A，B，C，D，E，F，G の 7 つのエリアがある都市で宅配便の配達事業を始めるため，配達員を採用することになりました．この都市には配達員の候補は 10 人いて，各人が配達できるエリアおよび配達を依頼した際にかかるコストは次の表のようになっています．

候補者	配達可能エリア	コスト
佐藤	A，B，C	200
鈴木	A，D，F	280
高橋	B，E	175
田中	C，D，E，F，G	560
渡辺	A，F	205
伊藤	B，D，F	245
山本	D	80
中村	C，G	195
小林	C，F，G	265
斉藤	B，E，G	190

この時，最も少ないコストで 7 つのエリアすべてに配達するためには誰を採用すると良いでしょうか．ただし，配達可能エリアの一部のみを依頼する（例：伊藤に B と D のみ依頼する）ことはできないものとします．

この例題の変数は，各候補者について「採用する」もしくは「採用しない」という状態を表現できるものとなります．よって，ここでは各候補者に対し採用する場合は 1，採用しない場合は 0 を取るような変数 $x_{(\text{候補者})}$ を導入します．

この時，目的関数はどのように表すことができるでしょうか．まず，佐藤を例に実際にかかる

コストを表現します．佐藤を採用した場合（$x_{佐藤}=1$の場合）には 200 のコストがかかります．

一方，採用しなかった場合（$x_{佐藤}=0$の場合）は佐藤については何もコストがかかりません．

言い換えると，かかったコストが 0 であるということになります．よって，先ほど導入した変数を用い 2 つの場合をまとめると，佐藤については実際には$200x_{佐藤}$のコストがかかったと表現できます．他の候補者に対しても同様に実際のコストを表し，その総和を取ると実際にかかった総コストとなります．この総コストが，最小化することになる目的関数です．

制約条件については，「各エリアに 1 人以上配置されている」という条件を式で表現することになります．例えば，エリア A の場合佐藤・鈴木・渡辺の中から 1 人以上採用することで条件を満たすことができます．このことを式で表すと$x_{佐藤}+x_{鈴木}+x_{渡辺}\geq 1$となります．他のエリアについても同様に考えることで制約条件を表現できます．

以上のことから，この例題は 0-1 整数計画問題として次のように定式化できます．

0-1 変数	$x_{佐藤}$, $x_{鈴木}$, $x_{高橋}$, $x_{田中}$, $x_{渡辺}$, $x_{伊藤}$, $x_{山本}$, $x_{中村}$, $x_{小林}$, $x_{斉藤}$	各候補者について採用する時 1，採用しない時 0 を取る変数
目的関数（最小化）	$200x_{佐藤}+280x_{鈴木}+175x_{高橋}+560x_{田中}+205x_{渡辺}+245x_{伊藤}+80x_{山本}+195x_{中村}+265x_{小林}+190x_{斉藤}$	総コストの最小化
制約	$x_{佐藤}+x_{鈴木}+x_{渡辺}\geq 1$	エリア A で配達可能にする
	$x_{佐藤}+x_{高橋}+x_{伊藤}+x_{斉藤}\geq 1$	エリア B で配達可能にする
	$x_{佐藤}+x_{田中}+x_{中村}+x_{小林}\geq 1$	エリア C で配達可能にする
	$x_{鈴木}+x_{田中}+x_{伊藤}+x_{山本}\geq 1$	エリア D で配達可能にする
	$x_{高橋}+x_{田中}+x_{斉藤}\geq 1$	エリア E で配達可能にする
	$x_{鈴木}+x_{田中}+x_{渡辺}+x_{伊藤}+x_{小林}\geq 1$	エリア F で配達可能にする
	$x_{田中}+x_{中村}+x_{小林}+x_{斉藤}\geq 1$	エリア G で配達可能にする

この定式化した結果についてそのまま SIMPLE で記述すると次のようになります.

```
//変数の宣言
IntegerVariable x_sato(name="佐藤",type=binary);
IntegerVariable x_suzuki(name="鈴木",type=binary);
IntegerVariable x_takahashi(name="高橋",type=binary);
IntegerVariable x_tanaka(name="田中",type=binary);
IntegerVariable x_watanabe(name="渡辺",type=binary);
IntegerVariable x_ito(name="伊藤",type=binary);
IntegerVariable x_yamamoto(name="山本",type=binary);
IntegerVariable x_nakamura(name="中村",type=binary);
IntegerVariable x_kobayashi(name="小林",type=binary);
IntegerVariable x_saito(name="斉藤",type=binary);
//目的関数の設定
Objective total_cost(type=minimize);
total_cost=200*x_sato+280*x_suzuki+175*x_takahashi+560*x_tanaka+
205*x_watanabe+245*x_ito+80*x_yamamoto+195*x_nakamura+
265*x_kobayashi+190*x_saito;
//各エリアに 1 人以上配置する
x_sato+x_suzuki+x_watanabe>=1;
x_sato+x_takahashi+x_ito+x_saito>=1;
x_sato+x_tanaka+x_nakamura+x_kobayashi>=1;
x_suzuki+x_tanaka+x_ito+x_yamamoto>=1;
x_takahashi+x_tanaka+x_saito>=1;
x_suzuki+x_tanaka+x_watanabe+x_ito+x_kobayashi>=1;
x_tanaka+x_nakamura+x_kobayashi+x_saito>=1;
//求解し解を出力する
solve();
x_sato.val.print();
x_suzuki.val.print();
x_takahashi.val.print();
x_tanaka.val.print();
x_watanabe.val.print();
x_ito.val.print();
x_yamamoto.val.print();
x_nakamura.val.print();
x_kobayashi.val.print();
x_saito.val.print();
total_cost.val.print();
```

この記述を見て分かるように，このままですと修正が大変ですし汎用的でもありません．このため，汎用的になるように定式化した結果を見直すことにします．

前節のナップサック問題の時と同様に「候補者の集合」および「エリアの集合」という概念を導入します．また，各候補者のコストは定数として外部から与えることにします．

さらに，制約条件についても見直します．例えば，エリア A に 1 人以上配置されているという制約条件 $x_{佐藤}+x_{鈴木}+x_{渡辺}\geq 1$については，ほかの候補者も考慮すると

$$1x_{佐藤}+1x_{鈴木}+0x_{高橋}+0x_{田中}+1x_{渡辺}+0x_{伊藤}+0x_{山本}+0x_{中村}+0x_{小林}+0x_{斉藤}\geq 1$$

と記述できます．ここで，この式の各変数についている係数 0 または 1 を外部から定数として与えることにします．また，他の地域についても同様に定数を導入します．すると，各エリアに対する制約条件は全て同じ枠組みで定式化できます．その結果，SIMPLE では一般的な形での記述で対応でき，よりわかりやすいものになります．

以上のことを反映し，汎用化させた結果は以下のようになります．

集合	$Man=\{佐藤,鈴木,高橋,田中,渡辺,伊藤,山本,中村,小林,斉藤\}$	候補者集合
	$Area=\{A,B,C,D,E,F,G\}$	エリア集合
0-1 変数	$x_i,\ i\in Man$	候補者 i を採用する時 1， 採用しない時 0 を取る変数
定数	$deliver_{ij}, i\in Man,\ j\in Area$	候補者 i がエリア j に配達可能な時 1， 不可能な時 0 を取る
	$cost_i,\ i\in Man$	候補者 i を採用した際のコスト
目的関数 （最小化）	$\sum_{i\in Man} cost_i x_i$	総コストの最小化
制約	$\sum_{i\in Man} deliver_{ij} x_i \geq 1, \forall j\in Area$	すべてのエリアで配達可能になる ようにする

これを SIMPLE で記述すると，次のようになります．なお，制約条件についてはデータファイルの内容をもとに各エリアに対して自動的に生成されます．

```
//候補者集合およびエリア集合の宣言
Set Man;
Element i(set=Man);
Set Area;
Element j(set=Area);
//パラメータおよび変数の宣言
Parameter deliver(name="配達可能エリア",index=(i,j));
Parameter cost(name="コスト",index=i);
IntegerVariable x(name="採用",index=i,type=binary);
//総コストを最小化する
Objective total_cost(name="総コスト",type=minimize);
total_cost=sum(cost[i]*x[i],i);
//各エリアに配置する
sum(deliver[i,j]*x[i],i)>=1;
//求解し結果を表示する
solve();
x.val.print();
total_cost.val.print();
```

実行させる際には，次の配達可能エリアを表す csv ファイル（左）とコストを表す dat ファイル（右）を与えます．

```
配達可能エリア,A,B,C,D,E,F,G
佐藤,1,1,1,0,0,0,0
鈴木,1,0,0,1,0,1,0
高橋,0,1,0,0,1,0,0
田中,0,0,1,1,1,1,1
渡辺,1,0,0,0,0,1,0
伊藤,0,1,0,1,0,1,0
山本,0,0,0,1,0,0,0
中村,0,0,1,0,0,0,1
小林,0,0,1,0,0,1,1
斉藤,0,1,0,0,1,0,1
```

```
コスト=
[佐藤]200
[鈴木]280
[高橋]175
[田中]560
[渡辺]205
[伊藤]245
[山本]80
[中村]195
[小林]265
[斉藤]190;
```

このモデルファイルを実行させることにより，佐藤・伊藤・斉藤の 3 人を採用すると良く，

この場合の総コストは 635 であるということが分かります.

ここで，NUOPT の解法について少し述べておきます. 今回の例題では厳密解法を用い求解をしました. 一方で，0-1 整数計画問題の場合には近似解法である wcsp を用い近似解を求めることもできます. wcsp を用いた際の近似解を求めたい場合には，SIMPLE モデル中の solve(); より前に

```
options.method="wcsp";
```

と記述します. なお，wcsp を用いる場合には終了条件に関する定数をいくつか設定することができます. 例えば，wcsp を最大で 1 秒間実行し終了したい場合には，モデルファイルに

```
options.maxtim=1;
```

と記述します. 詳しいことにつきましては NUOPT/SIMPLE マニュアルを参考にしてください.

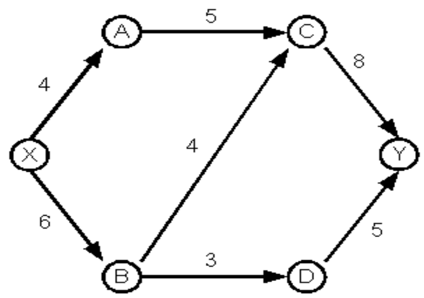
最後に，この 3 人を採用した場合，エリアＢは 3 人とも担当可能になっています. このようなダブりを許さないように，例題に「1 つのエリアはちょうど 1 人で担当する」という制約を加えてみるとどのようになるでしょうか. これは先ほどの SIMPLE モデル中の制約条件「sum(deliver[i,j]*x[i],i)>=1;」を「sum(deliver[i,j]*x[i],i)==1;」とすることで表現できます. 実行すると，鈴木・高橋・中村を採用すると良く，この場合の総コストは 650 であるという結果が得られます.

2.7 最大流問題

この節では,有向グラフをもとにしたデータ構造であるネットワークに関する基本的な問題の一つである最大流問題を取り上げます．最大流問題とは，ネットワーク上で始点（Source）から終点（Sink）まで流すことができる量の最大値を求める問題です．ただし，各辺には流すことができる量の上限が与えられています．さらに，始点と終点以外の点については,「流入してくる総量と流出していく総量は一致する」という関係（「保存則」と言います）が成り立つ必要があります．最大流問題は，配送や通信の分野などで応用が可能です．

（例題）

ある企業は,X 町にある製粉工場で製造した小麦粉を Y 市のレストランに納めています．この際，下の図のように，製粉工場からレストランまで輸送する間にいくつかの問屋を経由しています．また，2 つの地点の間で 1 日に運べる小麦粉の量の上限（kg）が図の数字の通りに決まっています．この時，製粉工場からレストランへ 1 日に納めることのできる小麦粉は最大で何 kg でしょうか．ただし，各問屋は他の地点から輸送されてきた小麦粉を全て他の地点に輸送するものとします．



まず変数として，ネットワーク上で始点（X 町の製粉工場）から終点（Y 市のレストラン）へ 1 日に輸送する小麦粉の量（今後は「総輸送量」と呼びます）と各辺に対し輸送する小麦粉の量を用意します．こうすると，目的関数は変数「総輸送量」自身となります．

次に，制約条件について考えます．まず，2 点間の輸送量については，負の値は許さず，対応する辺に与えられている上限を超えてはならないという制約条件が必要です．このことは，上下限制約として表現できます．さらに，各地点に対しては保存則が成立している必要があります．ただし，製粉工場については，各地点への輸送量の和が総輸送量と一致するということになります．また，レストランについては，各地点からの輸送量の和が総輸送量と一致しなければなりません．

以上のことから，この例題は次のように定式化できます．

変数	$total$	総輸送量
	f_{XA}	製粉工場から A 地点への輸送量
	f_{XB}	製粉工場から B 地点への輸送量
	f_{AC}	A 地点から C 地点への輸送量
	f_{BC}	B 地点から C 地点への輸送量
	f_{BD}	B 地点から D 地点への輸送量
	f_{CY}	C 地点からレストランへの輸送量
	f_{DY}	D 地点からレストランへの輸送量
目的関数（最大化）	$total$	総輸送量
制約条件	$total = f_{XA} + f_{XB}$	製粉工場での輸送量
	$f_{XA} = f_{AC}$	A 地点での輸送量保存則
	$f_{XB} = f_{BC} + f_{BD}$	B 地点での輸送量保存則
	$f_{AC} + f_{BC} = f_{CY}$	C 地点での輸送量保存則
	$f_{BD} = f_{DY}$	D 地点での輸送量保存則
	$f_{CY} + f_{DY} = total$	レストランでの輸送量
	$0 \le f_{XA} \le 4$	製粉工場から A 地点への輸送量の上下限
	$0 \le f_{XB} \le 6$	製粉工場から B 地点への輸送量の上下限
	$0 \le f_{AC} \le 5$	A 地点から C 地点への輸送量の上下限
	$0 \le f_{BC} \le 4$	B 地点から C 地点への輸送量の上下限
	$0 \le f_{BD} \le 3$	B 地点から D 地点への輸送量の上下限
	$0 \le f_{CY} \le 8$	C 地点からレストランへの輸送量の上下限
	$0 \le f_{DY} \le 5$	D 地点からレストランへの輸送量の上下限

これを SIMPLE で記述すると以下のようになります.

```
//変数の宣言
Variable total(name="製粉工場からレストランへの総輸送量");
Variable f_XA(name="製粉工場から地点Aへの輸送量");
Variable f_XB(name="製粉工場から地点Bへの輸送量");
Variable f_AC(name="地点Aから地点Cへの輸送量");
Variable f_BC(name="地点Bから地点Cへの輸送量");
Variable f_BD(name="地点Bから地点Dへの輸送量");
Variable f_CY(name="地点Cからレストランへの輸送量");
Variable f_DY(name="地点Dからレストランへの輸送量");
//総輸送量の最大化
Objective obj(name="製粉工場からレストランへの総輸送量",type=maximize);
obj=total;
//各地点での輸送量
total==f_XA+f_XB;
f_XA==f_AC;
f_XB==f_BC+f_BD;
f_AC+f_BC==f_CY;
f_BD==f_DY;
f_CY+f_DY==total;
//輸送量に関する上下限制約
0<=f_XA<=4;
0<=f_XB<=6;
0<=f_AC<=5;
0<=f_BC<=4;
0<=f_BD<=3;
0<=f_CY<=8;
0<=f_DY<=5;
//求解して値を出力する
solve();
f_XA.val.print();
f_XB.val.print();
f_AC.val.print();
f_BC.val.print();
f_BD.val.print();
f_CY.val.print();
f_DY.val.print();
total.val.print();
```

このモデルファイルを実行させると，

```
製粉工場から地点Ａへの輸送量=4
製粉工場から地点Ｂへの輸送量=6
地点Ａから地点Ｃへの輸送量=4
地点Ｂから地点Ｃへの輸送量=3.24901
地点Ｂから地点Ｄへの輸送量=2.75099
地点Ｃからレストランへの輸送量=7.24901
地点Ｄからレストランへの輸送量=2.75099
製粉工場からレストランへの総輸送量=10
```

という表示がされ，1 日に最大 10kg 小麦粉を納めることができるという結果を確認できます．

ここで，SIMPLE での記述について，より簡潔な形に書き直していくことにします．そのために，都市の集合という概念を導入します．また，2 点間の輸送量の上限値を定数として外部から与えることにします．すると，定式化については次のように書き直すことができます．

集合	$City = \{A, B, C, D, X, Y\}$	都市の集合
変数	$total$	総輸送量
	$f_{ij} \ , i \in City, j \in City$	ｉからｊへの輸送量
目的関数（最大化）	$total$	総輸送量
定数	$upper_{ij} \ , i \in City, j \in City$	ｉからｊへの輸送量の上限
制約条件	$total = \sum_{j \in City} f_{Xj}$	製粉工場での輸送量
	$\sum_{i} f_{ik} = \sum_{j} f_{kj}, \forall k \in City \setminus \{X, Y\}$	問屋での輸送量の保存則
	$\sum_{i \in City} f_{iY} = total$	レストランでの輸送量
	$0 \le f_{ij} \le upper_{ij}, \forall i, j \in City$	ｉからｊへの輸送量の上下限

これより，SIMPLE での記述は次のようなものになります.

```
//都市の集合の宣言
Set City;
Element i(set=City),j(set=City),k(set=City);
//変数の宣言
Variable total(name="製粉工場からレストランへの総輸送量");
Variable f(index=(i,j));
//2 点間の輸送量の上限をパラメータとして宣言
Parameter upper(name="輸送量の上限",index=(i,j));
//総輸送量の最大化
Objective obj(name="製粉工場からレストランへの総輸送量",type=maximize);
obj=total;
//各地点での輸送量
total==sum(f["X",j],j);
sum(f[k,j],j)==sum(f[j,k],j),k!="X",k!="Y";
sum(f[i,"Y"],i)==total;
//輸送量に関する上下限制約
0<=f[i,j]<=upper[i,j];
//求解して解を出力する
solve();
total.val.print();
```

なお，実行時には次の csv ファイルを与えます.

```
輸送量の上限,A,B,C,D,X,Y
A,0,0,5,0,0,0
B,0,0,4,3,0,0
C,0,0,0,0,0,8
D,0,0,0,0,0,5
X,4,6,0,0,0,0
Y,0,0,0,0,0,0
```

Vector/Matrix を使用した SIMPLE モデル

NUOPT V13 より導入された，Vector/Matrix クラスの記述を使ったモデルを紹介します．このモデルでは，グラフの接続行列を利用します．

有向グラフ $G(V,E)$ の接続行列とは，次のように定義されます．

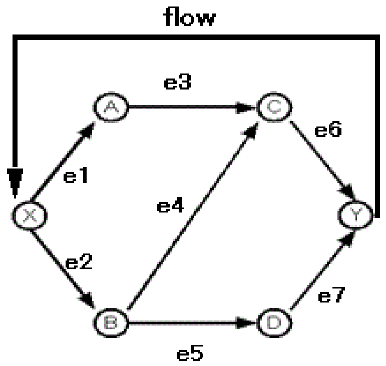
定義　接続行列

頂点集合 $V=\{v_1,...,v_n\}$ と有向辺集合 $E=\{e_1,...,e_m\}$ からなる有向グラフ $G(V,E)$ に対して，

頂点 v が 辺 e の始点であるとき $i_{ve}=1$ ，
頂点 v が 辺 e の終点であるとき $i_{ve}=-1$，
それ以外は $i_{ve}=0$

となる $n\times m$ 行列 $I(i_{ve})$ を，有向グラフ $G(V,E)$ の接続行列という．

最大流問題では，ネットワークの終点から始点へ向かう辺 $flow$ を加えた次の図を考えます．辺 $flow$ の輸送量の上限は，他の辺の輸送用の上限の総和とします．



図のグラフの接続行列を I ，各辺の輸送量を成分とするベクトルを $\boldsymbol{x}$ ，各辺の輸送量の上限値を成分とするベクトルを $\boldsymbol{upper}$ とおくと，

$$
\begin{aligned}
\max \quad & x_{flow} \\
\text{s.t} \quad & I\,\boldsymbol{x}=\boldsymbol{0}, \\
& \boldsymbol{x}\le \boldsymbol{upper}, \\
& \boldsymbol{x}\ge 0
\end{aligned}
$$

という線形計画問題に帰着できます．

定式化については以下のように書き直します．

集合	$City = \{A, B, C, D, X, Y\}$	都市の集合
	$Edge = \{e1, e2, e3, e4, e5, e6, e7, flow\}$	辺の集合
変数	x_e , $e \in Edge$	各辺の輸送量
定数	$I(v, e)$, $v \in City, e \in Edge$	接続行列
	$upper_e$, $e \in Edge$	各辺の輸送量の上限
目的関数（最大化）	x_{flow}	総輸送量
制約条件	$I\,\boldsymbol{x} = \mathbf{0}$	輸送量の保存則
	$\boldsymbol{x} \le \boldsymbol{upper}$	各輸送量の上限
	$\boldsymbol{x} \ge 0$	各輸送量の下限

これにより，SIMPLE での記述は次のようになります．

```
Set City, Edge;
Element v(set=City), e(set=Edge);

Matrix I((v,e));
Vector Upper(e);

Variable xvar(index=e);
Vector x(e);
x[e] = xvar[e]; // ベクトル x に変数 xvar を詰め込む

Objective total(type=maximize);
total = xvar["flow"];

I * x == zeros(City);
zeros(Edge) <= x;
x <= Upper;

options.method="simplex";

solve();
```

実行時の csv ファイルは以下の二つとなります．

```
I,e1,e2,e3,e4,e5,e6,e7,flow
A,-1,0,1,0,0,0,0,0
B,0,-1,0,1,1,0,0,0
C,0,0,-1,-1,0,1,0,0
D,0,0,0,0,-1,0,1,0
X,1,1,0,0,0,0,0,-1
Y,0,0,0,0,0,-1,-1,1
```

```
e,Upper
e1,4
e2,6
e3,5
e4,4
e5,5
e6,6
e7,7
flow,37
```

2.7.1 最小カット問題

最大流問題を，線形計画問題の一般形に定式化したことを利用して，その双対問題である，最小カット問題を SIMPLE で記述してみましょう．

最大流問題の制約式 $I\,\boldsymbol{x}=\boldsymbol{0}$ ，$\boldsymbol{x} \leq \boldsymbol{upper}$ に対する双対変数を，それぞれ $\boldsymbol{y}$，$\boldsymbol{z}$ とします．

また，最大流問題の目的関数 x_{flow} は，

$$
\begin{aligned}
&a_{flow}=1,\\
&a_e=0\,,\,e\neq\,flow
\end{aligned}
$$

であるようなベクトル $\boldsymbol{a}$ を用いて $\boldsymbol{a}\cdot\boldsymbol{x}$ と表すことができます．
よって最小カット問題は，

$$
\begin{aligned}
\min\quad & \boldsymbol{upper}\cdot\boldsymbol{z}\ (=\boldsymbol{0}\cdot\boldsymbol{y}+\boldsymbol{upper}\cdot\boldsymbol{z})\\
\text{s.t}\quad & I^{\mathrm{T}}\boldsymbol{y}+\boldsymbol{z}\geq\boldsymbol{a},\\
& \boldsymbol{z}\geq 0.
\end{aligned}
$$

のように記述することができます．

定式化については，次のようになります．

集合	$City=\{A,B,C,D,X,Y\}$	都市の集合
	$Edge=\{e1,e2,e3,e4,e5,e6,e7,\ flow\}$	辺の集合
変数	$y_v\,,\,v\in City$	
	$z_e\,,\,e\in Edge$	
定数	$I(v,e)\,,\,v\in City,\,e\in Edge$	接続行列
	$upper_e\,,\,e\in Edge$	各辺の輸送量の上限
	$a_e=0\,(e\neq\,flow),a_{flow}=1$	
目的関数（最大化）	$\boldsymbol{upper}\cdot\boldsymbol{z}$	カット総量
制約条件	$I^{\mathrm{T}}\boldsymbol{y}+\boldsymbol{z}\geq\boldsymbol{a}$	双対制約式
	$\boldsymbol{z}\geq 0$	不等号制約式に対する双対変数は 0 以上

これより，SIMPLE の記述は次のようになります.

```
Set City, Edge;
Element v(set=City), e(set=Edge);

Matrix I((v,e));
Vector Upper(e);

Variable yvar(index=v), zvar(index=(e));
Vector y(v), z(e);
y[v] = yvar[v]; // ベクトル y に変数 yvar を詰め込む
z[e] = zvar[e]; // ベクトル z に変数 zvar を詰め込む

Objective cut(name=”カットの総量”,type=minimize);
cut = inprod(Upper,z);

Vector a(e);
a = zeros(Edge);
a["flow"] = 1;

trans(I) * y + z >= a;
z >= zeros(Edge) ;

options.method="simplex";

solve();
cut.val.print();
```

入力データは，最大流問題と同じです.

このモデルファイルを実行すると，

```
カットの総量=10
```

という表示がされ，最大流問題と同じ目的関数の値を得ることができます.

2.8 最小費用流問題

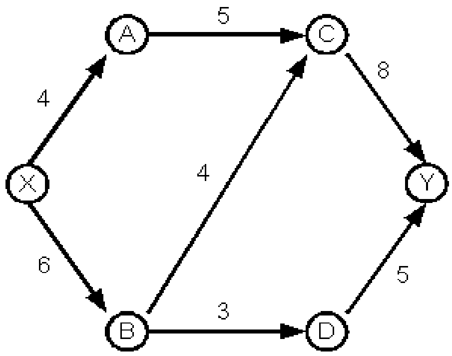
前節の最大流問題では,小麦粉の輸送を例にネットワークを流れる量をできるだけ多くしようという方針で最適化を行いました．このため，小麦粉を輸送する際にかかる燃料費などの様々なコストを考慮していませんでした．また，製粉工場から 1 日に出荷する**小麦粉**の量が決まっているような場合には別の方針で最適化を行う必要があります.

決まった量をネットワーク上で始点から終点まで流す際にかかる費用（コスト）を最小にしようという問題のことを最小費用流問題といいます．なお，最大流問題のときと同様に，保存則が成立する必要があります．また,「決まった量」が 1 である場合には最小路問題と呼ばれ，カーナビゲーションシステムでのルート探索や鉄道の経路案内などに応用されています.

（例題）

ある企業は,X 町の製粉工場で製造した小麦粉を Y 市のレストランまで 1 日に 8kg 納めています．この際，右下の図のように，製粉工場からレストランまで輸送する間にいくつかの問屋を経由しています．また，2 つの地点の間で 1 日に運べる小麦粉の量の上限（kg）が図の数字のように決まっています．さらに，2 つの地点の間で小麦粉を 1kg 輸送する毎に左下の表に書かれている費用がかかります．このとき，小麦粉を 8kg 製粉工場からレストランへ輸送する際にかかる総費用は最低いくらでしょうか．ただし，各問屋は他の地点から輸送されてきた小麦粉を全て他の地点に輸送するものとします.

輸送元	輸送先	1 kg あたりの費用
製粉工場 X	A	250
製粉工場 X	B	200
A	C	270
B	C	300
B	D	220
C	レストラン Y	190
D	レストラン Y	170



定式化について，前節の最大流問題の時との違いは

・総輸送量（前節の例題での total という変数）が 8 に決まっている.

・目的関数が総費用の最小化に変わっている.

の 2 点です．この 2 点以外については，前節の例題と同じ定式化となりますので必要に応じて前節を参考にしてください．なお，総費用は各輸送ルートについて「小麦粉 1kg あたりの費用」と「輸送した小麦粉の量」の積を求め総和をとることで求められます.

以上のことから，この例題は次のように定式化できます．

変数	f_{XA}	製粉工場から A 地点への輸送量
	f_{XB}	製粉工場から B 地点への輸送量
	f_{AC}	A 地点から C 地点への輸送量
	f_{BC}	B 地点から C 地点への輸送量
	f_{BD}	B 地点から D 地点への輸送量
	f_{CY}	C 地点からレストランへの輸送量
	f_{DY}	D 地点からレストランへの輸送量
目的関数（最小化）	$250f_{XA}+200f_{XB}+270f_{AC}+300f_{BC}+220f_{BD}+190f_{CY}+170f_{DY}$	総費用
制約条件	$8=f_{XA}+f_{XB}$	製粉工場での輸送量
	$f_{XA}=f_{AC}$	A 地点での輸送量の保存則
	$f_{XB}=f_{BC}+f_{BD}$	B 地点での輸送量の保存則
	$f_{AC}+f_{BC}=f_{CY}$	C 地点での輸送量の保存則
	$f_{BD}=f_{DY}$	D 地点での輸送量の保存則
	$f_{CY}+f_{DY}=8$	レストランでの輸送量
	$0\le f_{XA}\le 4$	製粉工場から A 地点への輸送量の上下限
	$0\le f_{XB}\le 6$	製粉工場から B 地点への輸送量の上下限
	$0\le f_{AC}\le 5$	A 地点から C 地点への輸送量の上下限
	$0\le f_{BC}\le 4$	B 地点から C 地点への輸送量の上下限
	$0\le f_{BD}\le 3$	B 地点から D 地点への輸送量の上下限
	$0\le f_{CY}\le 8$	C 地点からレストランへの輸送量の上下限
	$0\le f_{DY}\le 5$	D 地点からレストランへの輸送量の上下限

この結果を SIMPLE で記述すると以下のようになります.

```
//変数の宣言
Variable f_XA(name="製粉工場から地点 A への輸送量");
Variable f_XB(name="製粉工場から地点 B への輸送量");
Variable f_AC(name="地点 A から地点 C への輸送量");
Variable f_BC(name="地点 B から地点 C への輸送量");
Variable f_BD(name="地点 B から地点 D への輸送量");
Variable f_CY(name="地点 C からレストランへの輸送量");
Variable f_DY(name="地点 D からレストランへの輸送量");
//総費用の最小化
Objective totalcost(name="総費用",type=minimize);
totalcost=250*f_XA+200*f_XB+270*f_AC+300*f_BC+220*f_BD+190*f_CY+
170*f_DY;
//各地点での輸送量
8==f_XA+f_XB;
f_XA==f_AC;
f_XB==f_BC+f_BD;
f_AC+f_BC==f_CY;
f_BD==f_DY;
f_CY+f_DY==8;
//輸送量に関する上下限制約
0<=f_XA<=4;
0<=f_XB<=6;
0<=f_AC<=5;
0<=f_BC<=4;
0<=f_BD<=3;
0<=f_CY<=8;
0<=f_DY<=5;
//求解して値を出力する
solve();
f_XA.val.print();
f_XB.val.print();
f_AC.val.print();
f_BC.val.print();
f_BD.val.print();
f_CY.val.print();
f_DY.val.print();
totalcost.val.print();
```

このモデルを実行させると,

```
製粉工場から地点Ａへの輸送量=2
製粉工場から地点Ｂへの輸送量=6
地点Ａから地点Ｃへの輸送量=2
地点Ｂから地点Ｃへの輸送量=3
地点Ｂから地点Ｄへの輸送量=3
地点Ｃからレストランへの輸送量=5
地点Ｄからレストランへの輸送量=3
総費用=5260
```

という表示がされ，結果を確認できます.

ところで，このままの記述では都市の数が多い問題に応用することが困難です．そこで，都市の集合という概念を導入し,必要なデータを定数として外部から与えるようにすることで大規模な問題に対しても応用しやすいようにします．まず，この方針で定式化をしなおすと次のようになります．なお，保存則については「他の地点からの輸送量の和」と「他の地点への輸送量の和」の差のデータを外部から与える形式を取りました．ちなみに，この値が負の地点が製粉工場，正の地点がレストラン，そして０の地点が問屋ということになります.

集合	$City = \{A, B, C, D, X, Y\}$	都市の集合
変数	$f_{ij}\ , i \in City, j \in City$	ｉからｊへの輸送量
定数	$upper_{ij}\ , i \in City, j \in City$	ｉからｊへの輸送量の上限
	$cost_{ij}\ , i \in City, j \in City$	ｉからｊへの輸送の際に かかる小麦粉 1kg あたりの費用
	$supply_i\ , i \in City$	ｉでの流入量と流出量の差
目的関数 （最小化）	$\sum_{j \in City} \sum_{i \in City} cost_{ij} \cdot f_{ij}$	総費用
制約条件	$\sum_{i \in City} f_{ik} - \sum_{j \in City} f_{kj} = supply_k, \forall k \in City$	各地点での輸送量
	$0 \le f_{ij} \le upper_{ij}, \forall i, j \in City$	ｉからｊへの輸送量の上下限

この定式化をしなおした結果を SIMPLE で記述すると次のようになります.

```
//都市の集合の宣言
Set City;
Element i(set=City),j(set=City),k(set=City);

//変数の宣言
Variable f(name="輸送量",index=(i,j));

//パラメータの宣言
Parameter upper(name="輸送量の上限",index=(i,j));
Parameter cost(name="輸送費用",index=(i,j));
Parameter supply(name="supply",index=k);

//総費用の最小化
Objective total_cost(name="総費用",type=minimize);
total_cost=sum(cost[i,j]*f[i,j], (i,j));

//各地点での輸送量
sum(f[i,k],i)-sum(f[k,j],j)==supply[k];

//輸送量に関する上下限制約
0<=f[i,j]<=upper[i,j];

//求解して解を出力する
solve();
total_cost.val.print();
```

なお，実行時には次の 2 つの csv ファイル（左・中）と 1 つの dat ファイル（右）を与えます.

```
輸送量の上限,A,B,C,D,X,Y
A,0,0,5,0,0,0
B,0,0,4,3,0,0
C,0,0,0,0,0,8
D,0,0,0,0,0,5
X,4,6,0,0,0,0
Y,0,0,0,0,0,0
```

```
輸送費用,A,B,C,D,X,Y
A,0,0,270,0,0,0
B,0,0,300,220,0,0
C,0,0,0,0,0,190
D,0,0,0,0,0,170
X,250,200,0,0,0,0
Y,0,0,0,0,0, 0
```

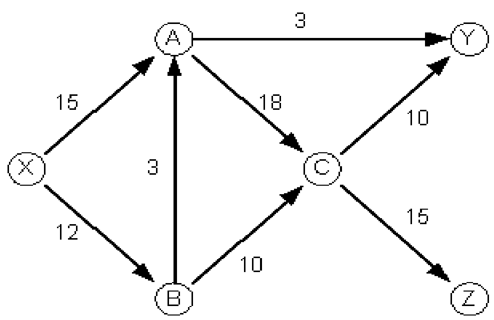
```
supply=
[A]0
[B]0
[C]0
[D]0
[X]-8
[Y]8;
```

2.9 多品種流問題

今まで述べてきた最大流問題や最小費用流問題では，ネットワーク上を１種類のものしか流れていませんでした．しかし，現実世界ではネットワーク上を複数のものが流れるというケースがよくあります．多品種流問題は，複数の品物をそれぞれ始点から終点まで輸送するという場合に利用される問題で，通信など多くの分野で応用されています．

（例題）

ある企業は，X 町の製粉工場で製造したパン用の小麦粉を Y 市のパン屋まで 1 日に 8kg 納めています．さらに，同じ製粉工場で製造したうどん用の小麦粉を Z 市のうどん屋まで 1 日に 13 kg 納めています．この際，下の図のように，製粉工場からパン屋，製粉工場からうどん屋まで輸送する間にいくつかの問屋を経由しています．また，2 つの地点の間で 1 日に運べるパン用の小麦粉とうどん用の小麦粉の合計量の上限（kg）が図の数字のように決まっています．さらに，2 つの地点の間でパン用の小麦粉やうどん用の小麦粉を 1kg 輸送する毎に下の表に書かれているだけの費用がかかります．このとき，パン用の小麦粉とうどん用の小麦粉を決められた量だけ納める際にかかる総費用は 1 日につき最低いくらでしょうか．



輸送元	輸送先	パン用の小麦粉の輸送費	うどん用の小麦粉の輸送費
製粉工場 X	A	12	11
製粉工場 X	B	10	12
A	C	4	5
A	パン屋 Y	11	-
B	A	6	7
B	C	11	9
C	パン屋 Y	9	-
C	うどん屋 Z	-	5

まず，各輸送ルートについてパン用の小麦粉の輸送量とうどん用の小麦粉の輸送量が変数として必要です．ただし，うどん屋にパン用の小麦粉を輸送したりパン屋にうどん用の小麦粉を輸送したりすることはありませんので，これらに対応する変数はここでは宣言しないことにします．

次に，目的関数については最小費用流問題の時と同様に考えることになります．ただし，パン用の小麦粉とうどん用の小麦粉で輸送コストが異なる点には注意が必要です．

最後に，制約条件を考えます．各輸送ルートについては問題文にあるようにパン用の小麦粉の輸送量とうどん用の小麦粉の輸送量の和が上限を超えないという制約条件が必要です．また，保存則については地点と商品に関し場合分けをする必要があります．さらに，「-1 個輸送する」ということはありえませんので変数について非負制約が必要です．

以上のことから，次のように定式化できます．

変数	$PWheat_{XA}, UWheat_{XA}$	製粉工場から A 地点への輸送量
	$PWheat_{XB}, UWheat_{XB}$	製粉工場から B 地点への輸送量
	$PWheat_{AC}, UWheat_{AC}$	A 地点から C 地点への輸送量
	$PWheat_{AY}$	A 地点からパン屋への輸送量
	$PWheat_{BA}, UWheat_{BA}$	B 地点から A 地点への輸送量
	$PWheat_{BC}, UWheat_{BC}$	B 地点から C 地点への輸送量
	$PWheat_{CY}$	C 地点からパン屋への輸送量
	$UWheat_{CZ}$	C 地点からうどん屋への輸送量
目的関数（最小化）	$12PWheat_{XA} + 10PWheat_{XB} + 4PWheat_{AC} + 11PWheat_{AY}$ $+6PWheat_{BA} + 11PWheat_{BC} + 9PWheat_{CY}$ $+11UWheat_{XA} + 12UWheat_{XB} + 5UWheat_{AC}$ $+7UWheat_{BA} + 9UWheat_{BC} + 5UWheat_{CZ}$	総輸送費
制約条件	$PWheat_{XA} + UWheat_{XA} \le 15, PWheat_{XB} + UWheat_{XB} \le 12,$ $PWheat_{AC} + UWheat_{AC} \le 18, PWheat_{AY} \le 3,$ $PWheat_{BA} + UWheat_{BA} \le 3, PWheat_{BC} + UWheat_{BC} \le 10,$ $PWheat_{CY} \le 10, UWheat_{CZ} \le 15$	各輸送ルートの輸送量の上限を超えない

$$PWheat_{XA} + PWheat_{XB} = 8 \ ,$$
$$PWheat_{AC} + PWheat_{AY} = PWheat_{XA} + PWheat_{BA} \ ,$$
$$PWheat_{BA} + PWheat_{BC} = PWheat_{XB} \ ,$$
$$PWheat_{CY} = PWheat_{AC} + PWheat_{BC} \ ,$$
$$8 = PWheat_{AY} + PWheat_{CY}$$

パン用の小麦粉に関する保存則

$$UWheat_{XA} + UWheat_{XB} = 13 \ ,$$
$$UWheat_{AC} + UWheat_{AY} = UWheat_{XA} + UWheat_{BA} \ ,$$
$$UWheat_{BA} + UWheat_{BC} = UWheat_{XB} \ ,$$
$$UWheat_{CZ} = UWheat_{AC} + UWheat_{BC} \ ,$$
$$13 = UWheat_{CZ}$$

うどん用の小麦粉に関する保存則

$$PWheat_{XA} \geq 0, PWheat_{XB} \geq 0, PWheat_{AC} \geq 0, PWheat_{AY} \geq 0,$$
$$PWheat_{BA} \geq 0, PWheat_{BC} \geq 0, PWheat_{CY} \geq 0,$$
$$UWheat_{XA} \geq 0, UWheat_{XB} \geq 0, UWheat_{AC} \geq 0,$$
$$UWheat_{BA} \geq 0, UWheat_{BC} \geq 0, UWheat_{CZ} \geq 0$$

輸送量に関する非負制約

これを SIMPLE で記述すると次のようになります.

```
//変数の宣言
Variable
  PWheat_XA,PWheat_XB,PWheat_AC,PWheat_AY,PWheat_BA,PWheat_BC,
  PWheat_CY,
  UWheat_XA,UWheat_XB,UWheat_AC,UWheat_BA,UWheat_BC,UWheat_CZ;
//目的関数の宣言
Objective total_cost(name="総輸送費",type=minimize);
total_cost=12*PWheat_XA+10*PWheat_XB+4*PWheat_AC+11*PWheat_AY
+6*PWheat_BA+11*PWheat_BC+9*PWheat_CY
+11*UWheat_XA+12*UWheat_XB+5*UWheat_AC+7*UWheat_BA
+9*UWheat_BC+5*UWheat_CZ;
//輸送ルートごとの輸送量の上限
PWheat_XA+UWheat_XA<=15;
PWheat_XB+UWheat_XB<=12;
PWheat_AC+UWheat_AC<=18;
PWheat_AY<=3;
PWheat_BA+UWheat_BA<=3;
PWheat_BC+UWheat_BC<=10;
PWheat_CY<=10;
UWheat_CZ<=15;
//パン用の小麦粉に関する保存則
PWheat_XA+PWheat_XB==8;
PWheat_AC+PWheat_AY==PWheat_XA+PWheat_BA;
PWheat_BA+PWheat_BC==PWheat_XB;
PWheat_CY==PWheat_AC+PWheat_BC;
8==PWheat_AY+PWheat_CY;
//うどん用の小麦粉に関する保存則
UWheat_XA+UWheat_XB==13;
UWheat_AC==UWheat_XA+UWheat_BA;
UWheat_BA+UWheat_BC==UWheat_XB;
UWheat_CZ==UWheat_AC+UWheat_BC;
13==UWheat_CZ;
//非負制約
PWheat_XA>=0;PWheat_XB>=0;PWheat_AC>=0;PWheat_AY>=0;
PWheat_BA>=0;PWheat_BC>=0;PWheat_CY>=0;
UWheat_XA>=0;UWheat_XB>=0;UWheat_AC>=0;UWheat_BA>=0;
UWheat_BC>=0;UWheat_CZ>=0;
```

ここで，SIMPLE での記述をより簡潔なものにしていくことにします．そのためには，都市の集合の概念を導入し，さまざまなデータに対応できるようにすることが有効です．この際，輸送量の上限値などの具体的なデータを出来るだけ外部から与えるようにしておきます．

すると，定式化については次のように表現できます．

集合	$City = \{A, B, C, X, Y, Z\}$	都市の集合
変数	$PWheat_{ij} \ , i \in City, j \in City$	i から j へのパン用の小麦粉の輸送量
	$UWheat_{ij} \ , i \in City, j \in City$	i から j へのうどん用の小麦粉の輸送量
定数	$upper_{ij} \ , i \in City, j \in City$	i から j への輸送量の上限
	$PWheat\_cost_{ij} \ , i \in City, j \in City$	i から j への輸送の際にかかるパン用の小麦粉 1kg あたりの費用
	$UWheat\_cost_{ij} \ , i \in City, j \in City$	i から j への輸送の際にかかるうどん用の小麦粉 1kg あたりの費用
	$PWheat\_supply_{i} \ , i \in City$	i でのパン用の小麦粉 1kg の流入量と流出量の差
	$UWheat\_supply_{i} \ , i \in City$	i でのうどん用の小麦粉 1kg の流入量と流出量の差
目的関数（最小化）	$\sum_{j \in City} \sum_{i \in City} (PWheat\_cost_{ij} \cdot PWheat_{ij} + UWheat\_cost_{ij} \cdot UWheat_{ij})$ 総輸送費	
制約条件	$PWheat_{ij} + UWheat_{ij} \le upper_{ij}, \forall i, j \in City$	i から j への輸送量の上限

$\sum_{i \in City} PWheat_{ik} - \sum_{j \in City} PWheat_{kj} = PWheat\_supply_k, \forall k \in City$	各地点でのパン用の小麦粉の輸送量
$\sum_{i \in City} UWheat_{ik} - \sum_{j \in City} UWheat_{kj} = UWheat\_supply_k, \forall k \in City$	各地点でのうどん用の小麦粉の輸送量
$0 \le PWheat_{ij}, \forall i, j \in City$	パン用の小麦粉の輸送量の非負制約
$0 \le UWheat_{ij}, \forall i, j \in City$	うどん用の小麦粉の輸送量の非負制約

SIMPLE で記述すると次のようになり, 先ほどのものより簡潔になっていることが分かります.

```
//都市の集合の宣言
Set City;
Element i(set=City),j(set=City),k(set=City);
//変数の宣言
Variable PWheat(name="パン用の小麦粉の輸送数",index=(i,j));
Variable UWheat(name="うどん用の小麦粉の輸送数",index=(i,j));
//パラメータの宣言
Parameter upper(name="輸送量の上限",index=(i,j));
Parameter PWheat_cost(name="パン用の小麦粉の輸送費用",index=(i,j));
Parameter UWheat_cost(name="うどん用の小麦粉の輸送費用",index=(i,j));
Parameter PWheat_supply(name="PWheat_supply",index=k);
Parameter UWheat_supply(name=" UWheat_supply",index=k);
//目的関数の宣言
Objective total_cost(name="総輸送費",type=minimize);
total_cost=sum(PWheat_cost[i,j]*PWheat[i,j]+UWheat_cost[i,j]*UWhe
at[i,j],(i,j));
//輸送ルートごとの輸送量の上限
PWheat[i,j]+UWheat[i,j]<=upper[i,j];
//パン用の小麦粉に関する保存則
sum(PWheat[i,k],i)-sum(PWheat[k,j],j)== PWheat_supply[k];
//うどん用の小麦粉に関する保存則
sum(UWheat[i,k],i)-sum(UWheat[k,j],j)== UWheat_supply[k];
//非負制約
PWheat[i,j]>=0;
UWheat[i,j]>=0;
```

なお，実行時には次の 3 つの csv ファイル（輸送量の上限・パン用の小麦粉の輸送費用・うどん用の小麦粉の輸送費用）と 1 つの dat ファイル（PWheat_supply および UWheat_supply）を与えます.

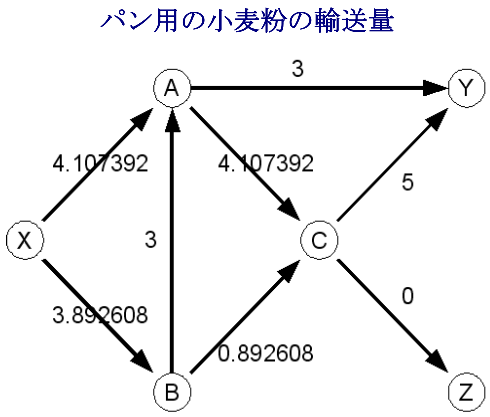
```
パン用の小麦粉の輸送費用,A,B,C,X,Y,Z
A,0,0,4,0,11,0
B,6,0,11,0,0,0
C,0,0,0,0,9,0
X,12,10,0,0,0,0
Y,0,0,0,0,0,0
Z,0,0,0,0,0,0
```

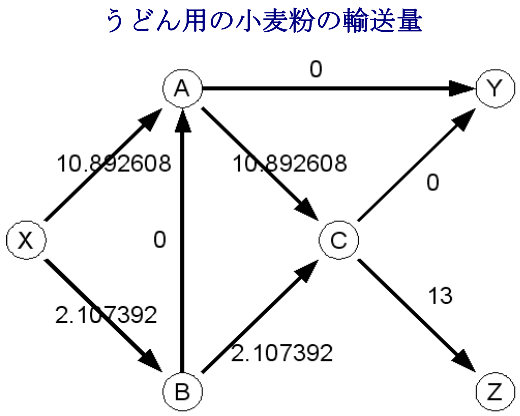
```
輸送量の上限,A,B,C,X,Y,Z
A,0,0,18,0,3,0
B,3,0,10,0,0,0
C,0,0,0,0,10,15
X,15,12,0,0,0,0
Y,0,0,0,0,0,0
Z,0,0,0,0,0,0
```

```
うどん用の小麦粉の輸送費用,A,B,C,X,Y,Z
A,0,0,5,0,0,0
B,7,0,9,0,0,0
C,0,0,0,0,0,5
X,11,12,0,0,0,0
Y,0,0,0,0,0,0
Z,0,0,0,0,0,0
```

```
PWheat_supply=[A]0 [B]0 [C]0
              [X]-8 [Y]8 [Z]0;
UWheat_supply=[A]0 [B]0 [C]0
          [X]-13 [Y]0 [Z]13;
```

このモデルを実行した結果から，総費用が 494 になるように輸送すると最適であり，例えば次の図のようにパン用の小麦粉とうどん用の小麦粉を輸送するとよいことが分かります．





2.10 p メディアン問題

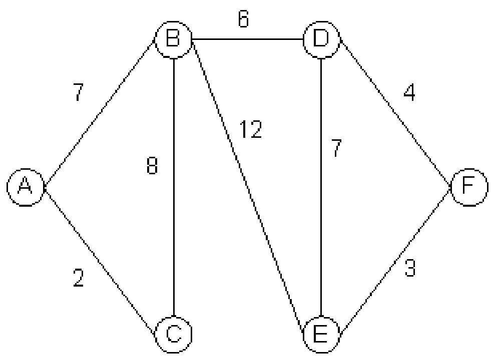
施設をどの地点に配置すると最適なのかを求める施設配置問題は，都市計画などに応用されています．施設配置問題の中から，この節で p メディアン問題を，次の節で p センター問題を取り上げます．なお，本チュートリアルでは各地点は需要を有する地点（今後は「需要点」と呼びます）でありさらに施設を配置することが可能な地点でもあると仮定します．

まず，メディアンについて説明します．メディアンとは各需要点から施設を配置した地点（今後は「施設点」と呼びます）への移動距離の総和を最小にする点のことです．そして，このメディアンを求める問題のことをメディアン問題と呼びます．なお，一般にはどの程度の需要があるかは需要点ごとに異なりますので重み付きの総移動距離を最小にすることになります．また，施設を 1 箇所ではなく p 箇所に配置するような場合に p メディアン問題と呼びます．

（例題）

ある企業は，A 市から F 市までの 6 市の中から 2 つの市に新たにデパートを出店しようと考えています．この時，出店地までの総移動距離を最小にするためにはどの市に出店すると良いでしょうか．なお，各市の人口は左下の表のようになっています．また，2 市間の距離は右下の図のようになっています．ただし，直接結ばれていない 2 都市間の距離は他の都市を経由したときの最短距離を採用します．

都市	人口
A 市	800
B 市	550
C 市	780
D 市	600
E 市	1020
F 市	360



まず，この問題では各都市に対しデパートを出店するのかどうかを表す変数が必要であることがわかります．しかし，これだけでは各都市からどちらの出店地が近いのかを求める必要があり，複雑な式になってしまいます．そこで，都市 i について都市 j に配分する場合 1，そうでない場合は 0 を取るような 0-1 変数 x_{ij} を導入します．なお，$x_{ii}=1$ の時には都市 i にデパートを出店し，$x_{ii}=0$ の時には出店しないという解釈をします．

次に，制約条件について考えていきます．先ほど述べたように，x_{ii} は都市 i に出店するかど

うかを表しています．この例題では 2 都市に出店しますので，$\sum_i x_{ii}$ は 2 である必要があります．また，各都市について近い方のデパートに配分しなければなりません（同距離の場合には任意にどちらかを選択することになります）．x_{ij} を用いて表現すると，各 i について $\sum_j x_{ij}$ が 1 であるということになります．最後に，今のままですと出店していない都市に配分されてしまう可能性があります．これを防ぐためには，$x_{jj}=0$ の場合に $x_{ij}=1$ となってはいけないという制約条件が必要です．言い換えると，x_{ij} は x_{jj} より大きい値をとらないということになります．よって，式で表すと $x_{ij} \le x_{jj}$ となります．ただし，この制約条件は $i=j$ の場合には無意味ですから $i \ne j$ の場合のみで十分であることに注意してください．

目的関数は，重み付きの総移動距離ということになります．まず，都市 i から都市 j への移動距離は $x_{ij}=1$ の場合にのみ考慮する必要があります．よって，まず 2 市 i,j 間の距離と x_{ij} の積を求めます．そして，j について和をとることで都市 i からの重みなしの移動距離が得られます．そして，この移動距離に重みである人口を掛けあわせることで都市 i からの重みをつけた移動距離を得ます．重みをつけた移動距離を全ての都市について求め，和をとることで目的関数である重み付きの総移動距離となります．なお，図より 2 都市間の距離は次の表の通りになることが分かります．

	A市	B市	C市	D市	E市	F市
A市	0	7	2	13	19	17
B市	7	0	8	6	12	10
C市	2	8	0	14	20	18
D市	13	6	14	0	7	4
E市	19	12	20	7	0	3
F市	17	10	18	4	3	0

以上のことをまとめると，次のように定式化できます.

0-1 変数 $x_{AA}, x_{AB}, x_{AC}, x_{AD}, x_{AE}, x_{AF}, x_{BA}, x_{BB}, x_{BC}, x_{BD}, x_{BE}, x_{BF}, x_{CA}, x_{CB}, x_{CC}, x_{CD}, x_{CE}, x_{CF},$
$x_{DA}, x_{DB}, x_{DC}, x_{DD}, x_{DE}, x_{DF}, x_{EA}, x_{EB}, x_{EC}, x_{ED}, x_{EE}, x_{EF}, x_{FA}, x_{FB}, x_{FC}, x_{FD}, x_{FE}, x_{FF}$

0 または 1 をとる出店するのかと最も近い出店地はどこなのかを表す変数

目 的 関 数（最小化） 重み付き総移動距離

$$\begin{aligned}
&800(7x_{AB}+2x_{AC}+13x_{AD}+19x_{AE}+17x_{AF})\\
&\quad+550(7x_{BA}+8x_{BC}+6x_{BD}+12x_{BE}+10x_{BF})\\
&\quad+780(2x_{CA}+8x_{CB}+14x_{CD}+20x_{CE}+18x_{CF})\\
&\quad+600(13x_{DA}+6x_{DB}+14x_{DC}+7x_{DE}+4x_{DF})\\
&\quad+1020(19x_{EA}+12x_{EB}+20x_{EC}+7x_{ED}+3x_{EF})\\
&\quad+360(17x_{FA}+10x_{FB}+18x_{FC}+4x_{FD}+3x_{FE})
\end{aligned}$$

制約条件

$$x_{AA}+x_{BB}+x_{CC}+x_{DD}+x_{EE}+x_{FF}=2 \qquad \text{2 都市に出店}$$
$$x_{AA}+x_{AB}+x_{AC}+x_{AD}+x_{AE}+x_{AF}=1 \qquad \text{A 市は 1 箇所に配分}$$
$$x_{BA}+x_{BB}+x_{BC}+x_{BD}+x_{BE}+x_{BF}=1 \qquad \text{B 市は 1 箇所に配分}$$
$$x_{CA}+x_{CB}+x_{CC}+x_{CD}+x_{CE}+x_{CF}=1 \qquad \text{C 市は 1 箇所に配分}$$
$$x_{DA}+x_{DB}+x_{DC}+x_{DD}+x_{DE}+x_{DF}=1 \qquad \text{D 市は 1 箇所に配分}$$
$$x_{EA}+x_{EB}+x_{EC}+x_{ED}+x_{EE}+x_{EF}=1 \qquad \text{E 市は 1 箇所に配分}$$
$$x_{FA}+x_{FB}+x_{FC}+x_{FD}+x_{FE}+x_{FF}=1 \qquad \text{F 市は 1 箇所に配分}$$

$$\begin{aligned}
&x_{AB}\le x_{BB}, x_{AC}\le x_{CC}, x_{AD}\le x_{DD}, x_{AE}\le x_{EE}, x_{AF}\le x_{FF},\\
&x_{BA}\le x_{AA}, x_{BC}\le x_{CC}, x_{BD}\le x_{DD}, x_{BE}\le x_{EE}, x_{BF}\le x_{FF},\\
&x_{CA}\le x_{AA}, x_{CB}\le x_{BB}, x_{CD}\le x_{DD}, x_{CE}\le x_{EE}, x_{CF}\le x_{FF},\\
&x_{DA}\le x_{AA}, x_{DB}\le x_{BB}, x_{DC}\le x_{CC}, x_{DE}\le x_{EE}, x_{DF}\le x_{FF},\\
&x_{EA}\le x_{AA}, x_{EB}\le x_{BB}, x_{EC}\le x_{CC}, x_{ED}\le x_{DD}, x_{EF}\le x_{FF},\\
&x_{FA}\le x_{AA}, x_{FB}\le x_{BB}, x_{FC}\le x_{CC}, x_{FD}\le x_{DD}, x_{FE}\le x_{EE}
\end{aligned}$$

出店している都市に配分

この結果をそのまま SIMPLE で記述すると次のようになります.

```
//変数の宣言
IntegerVariable
      x_AA(type=binary),x_AB(type=binary),x_AC(type=binary),
      x_AD(type=binary),x_AE(type=binary),x_AF(type=binary),
      x_BA(type=binary),x_BB(type=binary),x_BC(type=binary),
      x_BD(type=binary),x_BE(type=binary),x_BF(type=binary),
      x_CA(type=binary),x_CB(type=binary),x_CC(type=binary),
      x_CD(type=binary),x_CE(type=binary),x_CF(type=binary),
      x_DA(type=binary),x_DB(type=binary),x_DC(type=binary),
      x_DD(type=binary),x_DE(type=binary),x_DF(type=binary),
      x_EA(type=binary),x_EB(type=binary),x_EC(type=binary),
      x_ED(type=binary),x_EE(type=binary),x_EF(type=binary),
      x_FA(type=binary),x_FB(type=binary),x_FC(type=binary),
      x_FD(type=binary),x_FE(type=binary),x_FF(type=binary);
//重み付き総移動距離の最小化
Objective total_distance(type=minimize);
total_distance=800*(7*x_AB+2*x_AC+13*x_AD+19*x_AE+17*x_AF)
      +550*(7*x_BA+8*x_BC+6*x_BD+12*x_BE+10*x_BF)
      +780*(2*x_CA+8*x_CB+14*x_CD+20*x_CE+18*x_CF)
      +600*(13*x_DA+6*x_DB+14*x_DC+7*x_DE+4*x_DF)
      +1020*(19*x_EA+12*x_EB+20*x_EC+7*x_ED+3*x_EF)
      +360*(17*x_FA+10*x_FB+18*x_FC+4*x_FD+3*x_FE);
//２都市に出店する
x_AA+x_BB+x_CC+x_DD+x_EE+x_FF==2;
//各都市を配分する
x_AA+x_AB+x_AC+x_AD+x_AE+x_AF==1;
x_BA+x_BB+x_BC+x_BD+x_BE+x_BF==1;
x_CA+x_CB+x_CC+x_CD+x_CE+x_CF==1;
x_DA+x_DB+x_DC+x_DD+x_DE+x_DF==1;
x_EA+x_EB+x_EC+x_ED+x_EE+x_EF==1;
x_FA+x_FB+x_FC+x_FD+x_FE+x_FF==1;
//出店しない都市には配分しない
x_AB<=x_BB; x_AC<=x_CC; x_AD<=x_DD; x_AE<=x_EE; x_AF<=x_FF;
x_BA<=x_AA; x_BC<=x_CC; x_BD<=x_DD; x_BE<=x_EE; x_BF<=x_FF;
x_CA<=x_AA; x_CB<=x_BB; x_CD<=x_DD; x_CE<=x_EE; x_CF<=x_FF;
x_DA<=x_AA; x_DB<=x_BB; x_DC<=x_CC; x_DE<=x_EE; x_DF<=x_FF;
x_EA<=x_AA; x_EB<=x_BB; x_EC<=x_CC; x_ED<=x_DD; x_EF<=x_FF;
x_FA<=x_AA; x_FB<=x_BB; x_FC<=x_CC; x_FD<=x_DD; x_FE<=x_EE;
```

ところで，この記述ですと都市の数が増えた場合に記述が大変であるなどの理由で汎用的ではありません．そこで，モデルファイルを汎用的なものにしていくことにします．そのためには，生データをできるだけ外部から与える必要があります．ここでは，人口と距離に関するデータをデータファイルから与えることにします．この時，様々なデータに対応できるように都市の集合という概念を導入しておきます．すると，定式化について次のように書き直せます．

集合	$City = \{A, B, C, D, E, F\}$	都市の集合
0-1 変数	$x_{ij}, i \in City, j \in City$	0 または 1 をとる出店するのかとどこに配分されるのかを表す変数
定数	$distance_{ij}, i \in City, j \in City$	都市 i から都市 j までの距離
	$population_i, i \in City$	都市 i の人口
目的関数（最小化）	$\sum_{i \in City} \left(population_i \cdot \sum_{j \in City, i \neq j} (distance_{ij} \cdot x_{ij}) \right)$	重み付き総移動距離
制約条件	$\sum_{i \in City} x_{ii} = 2$	2 都市に出店
	$\sum_{j \in City} x_{ij} = 1, \forall i \in City$	各都市は 1 箇所に配分
	$x_{ij} \le x_{jj}, \forall i, j \in City, i \neq j$	出店している都市に配分

上記の式を SIMPLE で記述すると，次のようになります．なお，NUOPT での実行時には都市の集合の具体的な要素はデータファイルから自動的に認識します．また，simple_printf() を用い出店する都市のみを表示するように工夫しました．

```
//都市の集合の宣言
Set City;
Element i(set=City), j(set=City), k(set=City);
//パラメータの宣言
Parameter distance(name="2 都市間の距離",index=(i,j));
Parameter population(name="都市の人口",index=i);
//変数の宣言
IntegerVariable x(index=(i,j),type=binary);
//目的関数の宣言
Objective total_distance(name="総移動距離",type=minimize);
total_distance=sum(population[i]*sum(distance[i,j]*x[i,j],(j,i!=j
)),i);
//制約条件
sum(x[k,k],k)==2;
sum(x[i,j],j)==1;
x[i,k]<=x[k,k],i!=k;
//求解し出店する都市を出力する
solve();
simple_printf("%s 市に出店する. ¥n",k,x[k,k].val==1);
```

なお，実行時に次の 2 都市間の距離を表す csv ファイル（左）と都市の人口を表す dat ファイル（右）を与えます.

```
2 都市間の距離,A,B,C,D,E,F
A,0,7,2,13,19,17
B,7,0,8,6,12,10
C,2,8,0,14,20,18
D,13,6,14,0,7,4
E,19,12,20,7,0,3
F,17,10,18,4,3,0
```

```
都市の人口=
["A"]800
["B"]550
["C"]780
["D"]600
["E"]1020
["F"]360;
```

このモデルを実行すると次の表示がされ，A 市と E 市に出店すると良いことがわかります.

```
"A" 市に出店する.
"E" 市に出店する.
```

2.11　p センター問題

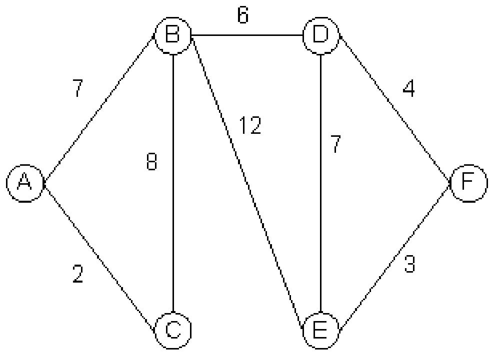
前節の　p　メディアン問題は，需要点と施設点の間の（重み付き）総移動距離を最小にするという目的で作られた問題でした．このため，需要点から施設点までの平均的な移動距離を最小にするような解を期待できます．しかし，解の中に需要点と施設点の間が極端に離れている組み合わせが含まれている可能性があります．この点，この節で扱う p センター問題は，（重み付き）移動距離が極端に大きい組み合わせを作らないように配置をする問題と見ることができます．

ここで，センターとは最も遠い需要点からの（重み付き）距離を最小にする施設点のことで，この点を求める問題のことをセンター問題といいます．また，p 箇所に施設を配置するような場合に　p　センター問題と呼び，消防署のような施設を配置するような場合に用いられています．

（例題）

ある企業は，A 市から F 市までの 6 市の中から 2 つの市に新たにデパートを出店しようと考えています．この時，都市から出店地までの重み付き移動距離の最大値を最小にするためにはどの市に出店すると良いでしょうか．なお，各市の人口は左下の表のようになっています．また，2 市間の距離は右下の図のようになっています．なお，直接結ばれていない 2 都市間の距離は他の都市を経由したときの最短距離を採用します．

都市	人口
A 市	800
B 市	550
C 市	780
D 市	600
E 市	1020
F 市	360



前節の例題との違いは目的関数の部分だけです．このため，前節の例題での変数と制約条件についてはこの例題でもそのまま使用しますので詳細は前節を参照してください．

目的関数については,都市から出店地までの重み付き移動距離の最大値ということになります．まず，都市 i から都市 j までの重み付き移動距離を求めます．これは $x_{ij}=1$ の時に意味を持つものですから i-j 間の距離，i の人口そして x_{ij} の 3 つを掛け合わせたものとなります．なお，$i=j$ の時にはこの積は常に 0 になりますので計算には含めないことにします．次に，今回のようなミニマックス問題の定式化での一つのテクニックとして変数 v を新たに導入します．この

時，重み付き移動距離がそれぞれ v 以下であるという制約条件を加えることで v に重み付き移動距離の最大値という意味を持たせることができます．よって，変数 v 自身が目的関数となります．

以上のことから，次のように定式化できます．

0-1 変数 $x_{AA},x_{AB},x_{AC},x_{AD},x_{AE},x_{AF},x_{BA},x_{BB},x_{BC},x_{BD},x_{BE},x_{BF},x_{CA},x_{CB},x_{CC},x_{CD},x_{CE},x_{CF},$
$x_{DA},x_{DB},x_{DC},x_{DD},x_{DE},x_{DF},x_{EA},x_{EB},x_{EC},x_{ED},x_{EE},x_{EF},x_{FA},x_{FB},x_{FC},x_{FD},x_{FE},x_{FF}$

0 または 1 をとる出店するのかと最も近い出店地はどこなのかを表す変数

連続変数 v 重み付き移動距離の最大値

目 的 関 数（最小化） v 重み付き移動距離の最大値

制約条件

$$v\ge 800\times 7x_{AB}, v\ge 800\times 2x_{AC}, v\ge 800\times 13x_{AD}, v\ge 800\times 19x_{AE}, v\ge 800\times 17x_{AF},$$
$$v\ge 550\times 7x_{BA}, v\ge 550\times 8x_{BC}, v\ge 550\times 6x_{BD}, v\ge 550\times 12x_{BE}, v\ge 550\times 10x_{BF},$$
$$v\ge 780\times 2x_{CA}, v\ge 780\times 8x_{CB}, v\ge 780\times 14x_{CD}, v\ge 780\times 20x_{CE}, v\ge 780\times 18x_{CF},$$
$$v\ge 600\times 13x_{DA}, v\ge 600\times 6x_{DB}, v\ge 600\times 14x_{DC}, v\ge 600\times 7x_{DE}, v\ge 600\times 4x_{DF},$$
$$v\ge 1020\times 19x_{EA}, v\ge 1020\times 12x_{EB}, v\ge 1020\times 20x_{EC}, v\ge 1020\times 7x_{ED}, v\ge 1020\times 3x_{EF},$$
$$v\ge 360\times 17x_{FA}, v\ge 360\times 10x_{FB}, v\ge 360\times 18x_{FC}, v\ge 360\times 4x_{FD}, v\ge 360\times 3x_{FE}$$

各重み付き移動距離は最大値を越えない

$x_{AA}+x_{BB}+x_{CC}+x_{DD}+x_{EE}+x_{FF}=2$ 2 都市に出店

$x_{AA}+x_{AB}+x_{AC}+x_{AD}+x_{AE}+x_{AF}=1$ A 市は 1 箇所に配分

$x_{BA}+x_{BB}+x_{BC}+x_{BD}+x_{BE}+x_{BF}=1$ B 市は 1 箇所に配分

$x_{CA}+x_{CB}+x_{CC}+x_{CD}+x_{CE}+x_{CF}=1$ C 市は 1 箇所に配分

$x_{DA}+x_{DB}+x_{DC}+x_{DD}+x_{DE}+x_{DF}=1$ D 市は 1 箇所に配分

$x_{EA}+x_{EB}+x_{EC}+x_{ED}+x_{EE}+x_{EF}=1$ E 市は 1 箇所に配分

$x_{FA}+x_{FB}+x_{FC}+x_{FD}+x_{FE}+x_{FF}=1$ F 市は 1 箇所に配分

$$x_{AB}\le x_{BB}, x_{AC}\le x_{CC}, x_{AD}\le x_{DD}, x_{AE}\le x_{EE}, x_{AF}\le x_{FF},$$
$$x_{BA}\le x_{AA}, x_{BC}\le x_{CC}, x_{BD}\le x_{DD}, x_{BE}\le x_{EE}, x_{BF}\le x_{FF},$$
$$x_{CA}\le x_{AA}, x_{CB}\le x_{BB}, x_{CD}\le x_{DD}, x_{CE}\le x_{EE}, x_{CF}\le x_{FF},$$
$$x_{DA}\le x_{AA}, x_{DB}\le x_{BB}, x_{DC}\le x_{CC}, x_{DE}\le x_{EE}, x_{DF}\le x_{FF},$$
$$x_{EA}\le x_{AA}, x_{EB}\le x_{BB}, x_{EC}\le x_{CC}, x_{ED}\le x_{DD}, x_{EF}\le x_{FF},$$
$$x_{FA}\le x_{AA}, x_{FB}\le x_{BB}, x_{FC}\le x_{CC}, x_{FD}\le x_{DD}, x_{FE}\le x_{EE}$$

出店している都市に配分

この結果を SIMPLE で記述すると次のようになります．

```
//変数の宣言
IntegerVariable
      x_AA(type=binary),x_AB(type=binary),x_AC(type=binary),
      x_AD(type=binary),x_AE(type=binary),x_AF(type=binary),
      x_BA(type=binary),x_BB(type=binary),x_BC(type=binary),
      x_BD(type=binary),x_BE(type=binary),x_BF(type=binary),
      x_CA(type=binary),x_CB(type=binary),x_CC(type=binary),
      x_CD(type=binary),x_CE(type=binary),x_CF(type=binary),
      x_DA(type=binary),x_DB(type=binary),x_DC(type=binary),
      x_DD(type=binary),x_DE(type=binary),x_DF(type=binary),
      x_EA(type=binary),x_EB(type=binary),x_EC(type=binary),
      x_ED(type=binary),x_EE(type=binary),x_EF(type=binary),
      x_FA(type=binary),x_FB(type=binary),x_FC(type=binary),
      x_FD(type=binary),x_FE(type=binary),x_FF(type=binary);
Variable v;
//重み付き移動距離の最大値の最小化
Objective max_distance(type=minimize);
max_distance=v;
v>=800*7*x_AB;  v>=800*2*x_AC;  v>=800*13*x_AD;  v>=800*19*x_AE;
v>=800*17*x_AF; v>=550*7*x_BA;  v>=550*8*x_BC;   v>=550*6*x_BD;
v>=550*12*x_BE; v>=550*10*x_BF; v>=780*2*x_CA;   v>=780*8*x_CB;
v>=780*14*x_CD; v>=780*20*x_CE; v>=780*18*x_CF; v>=600*13*x_DA;
v>=600*6*x_DB;  v>=600*14*x_DC; v>=600*7*x_DE;   v>=600*4*x_DF;
v>=1020*19*x_EA;v>=1020*12*x_EB;v>=1020*20*x_EC;v>=1020*7*x_ED;
v>=1020*3*x_EF; v>=360*17*x_FA; v>=360*10*x_FB; v>=360*18*x_FC;
v>=360*4*x_FD;  v>=360*3*x_FE;
//出店と配分に関する制約条件
x_AA+x_BB+x_CC+x_DD+x_EE+x_FF==2;
x_AA+x_AB+x_AC+x_AD+x_AE+x_AF==1;
x_BA+x_BB+x_BC+x_BD+x_BE+x_BF==1;
x_CA+x_CB+x_CC+x_CD+x_CE+x_CF==1;
x_DA+x_DB+x_DC+x_DD+x_DE+x_DF==1;
x_EA+x_EB+x_EC+x_ED+x_EE+x_EF==1;
x_FA+x_FB+x_FC+x_FD+x_FE+x_FF==1;
x_AB<=x_BB; x_AC<=x_CC; x_AD<=x_DD; x_AE<=x_EE; x_AF<=x_FF;
x_BA<=x_AA; x_BC<=x_CC; x_BD<=x_DD; x_BE<=x_EE; x_BF<=x_FF;
x_CA<=x_AA; x_CB<=x_BB; x_CD<=x_DD; x_CE<=x_EE; x_CF<=x_FF;
x_DA<=x_AA; x_DB<=x_BB; x_DC<=x_CC; x_DE<=x_EE; x_DF<=x_FF;
x_EA<=x_AA; x_EB<=x_BB; x_EC<=x_CC; x_ED<=x_DD; x_EF<=x_FF;
x_FA<=x_AA; x_FB<=x_BB; x_FC<=x_CC; x_FD<=x_DD; x_FE<=x_EE;
```

このままでは大変分かりにくいですので，前節と同様に SIMPLE での記述を簡潔なものにしていくことにします．この際，定式化については次のようなものにします．

集合	$City = \{A, B, C, D, E, F\}$	都市の集合
0-1 変数	$x_{ij}, i \in City, j \in City$	0または1をとる出店するのかと最も近い出店地はどこなのかを表す変数
連続変数	v	重み付き移動距離の最大値
定数	$distance_{ij}, i \in City, j \in City$	都市 i から都市 j までの距離
	$population_i, i \in City$	都市 i の人口
目的関数（最小化）	v	重み付き移動距離の最大値
制約条件	$v \geq population_i \cdot distance_{ij} \cdot x_{ij}, \forall i, j \in City, i \neq j$	各重み付き移動距離は最大値を越えない
	$\sum_{i \in City} x_{ii} = 2$	2 都市に出店
	$\sum_{j \in City} x_{ij} = 1, \forall i \in City$	各都市は 1 箇所に配分
	$x_{ij} \leq x_{jj}, \forall i, j \in City, i \neq j$	出店している都市に配分

すると，SIMPLE では次のような簡潔な記述になります.

```
//都市の集合の宣言
Set City;
Element i(set=City),j(set=City),k(set=City);
//パラメータの宣言
Parameter distance(name="２都市間の距離",index=(i,j));
Parameter population(name="都市の人口",index=i);
//変数の宣言
IntegerVariable x(index=(i,j),type=binary);
Variable v;
//目的関数の宣言
Objective max_distance(type=minimize);
max_distance=v;
//制約条件
v>=population[i]*distance[i,j]*x[i,j];
sum(x[k,k],k)==2;
sum(x[i,j],j)==1;
x[i,k]<=x[k,k],i!=k;
//求解し出店する都市を出力する
solve();
simple_printf("%s 市に出店する. ¥n", k,x[k,k].val==1);
```

実行する際には，前節の例題と同様に次の 2 都市間の距離を表す csv ファイル（左）と各都市の人口を表す dat ファイル（右）を与えます.

```
２都市間の距離,A,B,C,D,E,F
A,0,7,2,13,19,17
B,7,0,8,6,12,10
C,2,8,0,14,20,18
D,13,6,14,0,7,4
E,19,12,20,7,0,3
F,17,10,18,4,3,0
```

```
都市の人口=
["A"]800
["B"]550
["C"]780
["D"]600
["E"]1020
["F"]360;
```

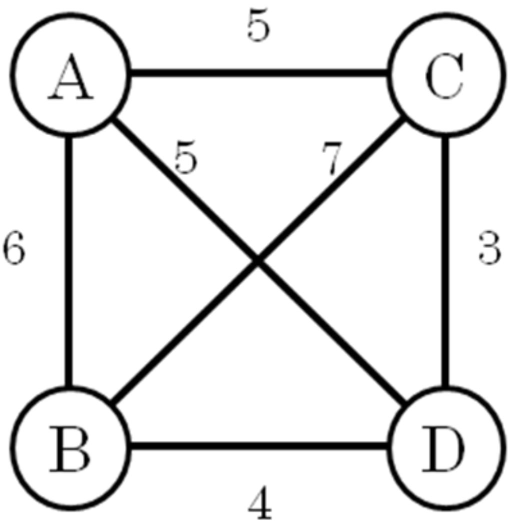
このモデルを実行すると，前節の p メディアン問題の時の結果とは異なり，A 市と F 市に出店すると良いことがわかります.

2.12 巡回セールスマン問題

巡回セールスマン問題とは，セールスマンがある都市から出発し，全ての都市を訪問して，出発地点に帰還する場合，どのような順番で都市を回るのが最短経路であるか，という問題です．この問題は最後に訪れる都市が出発地点でなくてもよいという設定もありますが，ここでは，最後に訪れる都市は出発地点であるという設定の問題を紹介します．また，一度訪れた都市を経由して別の都市に移動するような場合は考えないこととします．

（例題）

次のグラフは都市 A，B，C，D と，そのリンクの距離関係を表した図です．



ある都市から出発し，全ての都市を訪問して，出発地点に帰還する場合，どのような順番で都市を回ると最短経路となるでしょうか．

この問題を NUOPT で解くために定式化を行います．本例題の定式化は文献 [4] からの引用です．“すべての都市を訪問して出発地点に帰還する”ということは，一筆書きの経路を考えるため，経路のどの都市を出発地点に解釈してもよいことになります．ここでは仮に都市 A を出発地点と解釈することにします．

変数について考えましょう．どの都市からどの都市に移っていくかを決定する問題ですので，ある都市から別のある都市に移動する場合に 1 をとり，移動しない場合に 0 をとるような 0-1 変数 $x_{AB}, x_{AC}, x_{AD}, x_{BC}, x_{BD}, x_{CD}, x_{BA}, x_{CA}, x_{DA}, x_{CB}, x_{DB}, x_{DC}$ を導入します．例えば，x_{AB} は都市 A から都市 B に移動する場合に 1 をとり，移動しない場合には 0 をとります．

次に，最小化すべき目的関数は経路長になりますので，各都市間の距離を用いて，

$$6x_{AB}+5x_{AC}+5x_{AD}+7x_{BC}+4x_{BD}+3x_{CD}+6x_{BA}+5x_{CA}+5x_{DA}+7x_{CB}+4x_{DB}+3x_{DC}$$

と表現することができます．

最後に制約条件です．巡回セールマン問題特有の制約，“すべての都市を訪問して帰還する”を定式化する必要があります．この制約は，“各都市から別都市の 1 つに移動する”，“各都市に別都市の 1 つから移動する”という制約を含みます．例えば，都市 A については，

$x_{AB}+x_{AC}+x_{AD}=1$：都市 A から別都市の 1 つに移動する

$x_{BA}+x_{CA}+x_{DA}=1$：都市 A に別都市の 1 つから移動する

という制約が必要です．

上記の制約だけでは不十分であり，もう少し制約を加える必要があります．なぜならば，上記の場合には， $x_{AB}=x_{BA}=1, x_{CD}=x_{DC}=1$ となるような解，つまり一筆書きで全都市を訪問することができないような経路が許されてしまうからです．巡回セールスマン問題では，一筆書きの閉路をツアーと呼び，「都市 A→都市 B→都市 A」のような，すべての都市を訪問していないツアーを特にサブツアーと呼びます．巡回セールスマン問題では，上記制約に加えて“すべてのサブツアーを排除する”という制約が必要になります．この制約を表現するために，中間変数と呼ばれる変数 y_B, y_C, y_D を導入し，都市 B，都市 C，都市 D が関わるツアーを排除するという制約を導入します．この制約は，

$y_B - y_C + 3x_{BC} \le 2$ ・・・(1)

$y_B - y_D + 3x_{BD} \le 2$ ・・・(2)

$y_C - y_B + 3x_{CB} \le 2$ ・・・(3)

$y_C - y_D + 3x_{CD} \le 2$ ・・・(4)

$y_D - y_B + 3x_{DB} \le 2$ ・・・(5)

$y_D - y_C + 3x_{DC} \le 2$ ・・・(6)

と表現することができます．上記制約から，例えば，(1)，(3)の制約の両辺を足し合わせると，

$3x_{BC}+3x_{CB} \le 4$

となり， $x_{BC}=x_{CB}=1$，すなわち「都市 B→都市 C→都市 B」のようなサブツアーを排除し，また，(1)，(４)，(5)の制約の両辺を足し合わせると，

$3x_{BC}+3x_{CD}+3x_{DB} \le 6$

となり， $x_{BC}=x_{CD}=x_{DB}=1$，すなわち，「都市 B→都市 C→都市 D→都市 B」のようなサブツアーを排除することができます．都市 A に関する情報が式に現れないのは，解となるツアーまでもを排除しないようにするためです．つまり，都市 A を含むツアーの内，都市 A 以外で構成されるツアーをすべて排除することを表現しています．

以上のことから，次のように定式化することができます．

0-1 変数	x_{AB}	都市Ａから都市Ｂへ移動するか否か
	x_{AC}	都市Ａから都市Ｃへ移動するか否か
	x_{AD}	都市Ａから都市Ｄへ移動するか否か
	x_{BC}	都市Ｂから都市Ｃへ移動するか否か
	x_{BD}	都市Ｂから都市Ｄへ移動するか否か
	x_{CD}	都市Ｃから都市Ｄへ移動するか否か
	x_{BA}	都市Ｂから都市Ａへ移動するか否か
	x_{CA}	都市Ｃから都市Ａへ移動するか否か
	x_{DA}	都市Ｄから都市Ａへ移動するか否か
	x_{CB}	都市Ｃから都市Ｂへ移動するか否か
	x_{DB}	都市Ｄから都市Ｂへ移動するか否か
	x_{DC}	都市Ｄから都市Ｃへ移動するか否か
変数	y_B	中間変数
	y_C	中間変数
	y_D	中間変数
目的関数(最小化)	$6x_{AB}+5x_{AC}+5x_{AD}+7x_{BC}+4x_{BD}+3x_{CD}$ $+6x_{BA}+5x_{CA}+5x_{DA}+7x_{CB}+4x_{DB}+3x_{DC}$	経路長
制約	$x_{AB}+x_{AC}+x_{AD}=1$	都市Ａから別都市の１つに移動する
	$x_{BA}+x_{BC}+x_{BD}=1$	都市Ｂから別都市の１つに移動する
	$x_{CA}+x_{CB}+x_{CD}=1$	都市Ｃから別都市の１つに移動する
	$x_{DA}+x_{DB}+x_{DC}=1$	都市Ｄから別都市の１つに移動する
	$x_{BA}+x_{CA}+x_{DA}=1$	都市Ａに別都市の１つから移動する
	$x_{AB}+x_{CB}+x_{DB}=1$	都市Ｂに別都市の１つから移動する
	$x_{AC}+x_{BC}+x_{DC}=1$	都市Ｃに別都市の１つから移動する
	$x_{AD}+x_{BD}+x_{CD}=1$	都市Ｄに別都市の１つから移動する
	$y_B-y_C+3x_{BC}\le 2$	サブツアー排除制約
	$y_B-y_D+3x_{BD}\le 2$	サブツアー排除制約
	$y_C-y_B+3x_{CB}\le 2$	サブツアー排除制約
	$y_C-y_D+3x_{CD}\le 2$	サブツアー排除制約
	$y_D-y_B+3x_{DB}\le 2$	サブツアー排除制約
	$y_D-y_C+3x_{DC}\le 2$	サブツアー排除制約

この問題は，変数に 0-1 変数と連続変数が混ざっており，目的関数，制約式全てが線形なので，混合整数線形計画問題となります．定式化した結果を SIMPLE で記述すると以下のようになります．

```
//変数
IntegerVariable x_AB(name="A->B",type=binary);
IntegerVariable x_AC(name="A->C",type=binary);
IntegerVariable x_AD(name="A->D",type=binary);
IntegerVariable x_BC(name="B->C",type=binary);
IntegerVariable x_BD(name="B->D",type=binary);
IntegerVariable x_CD(name="C->D",type=binary);
IntegerVariable x_BA(name="B->A",type=binary);
IntegerVariable x_CA(name="C->A",type=binary);
IntegerVariable x_DA(name="D->A",type=binary);
IntegerVariable x_CB(name="C->B",type=binary);
IntegerVariable x_DB(name="D->B",type=binary);
IntegerVariable x_DC(name="D->C",type=binary);
//目的関数
Objective route(name="経路長",type=minimize);
route=6*x_AB+5*x_AC+5*x_AD+7*x_BC+4*x_BD+3*x_CD+6*x_BA
+5*x_CA+5*x_DA+7*x_CB+4*x_DB+3*x_DC;
//中間変数
Variable y_B;
Variable y_C;
Variable y_D;
// 各都市から別都市の 1 つに移動する
x_AB+x_AC+x_AD==1;
x_BA+x_BC+x_BD==1;
x_CA+x_CB+x_CD==1;
x_DA+x_DB+x_DC==1;
// 各都市に別都市の 1 つから移動する
x_BA+x_CA+x_DA==1;
x_AB+x_CB+x_DB==1;
x_AC+x_BC+x_DC==1;
x_AD+x_BD+x_CD==1;
```

```
// サブツアーを排除する
y_B-y_C+3*x_BC<=2;
y_B-y_D+3*x_BD<=2;
y_C-y_B+3*x_CB<=2;
y_C-y_D+3*x_CD<=2;
y_D-y_B+3*x_DB<=2;
y_D-y_C+3*x_DC<=2;
//求解
solve();
//出力
x_AB.val.print();
x_AC.val.print();
x_AD.val.print();
x_BC.val.print();
x_BD.val.print();
x_CD.val.print();
x_BA.val.print();
x_CA.val.print();
x_DA.val.print();
x_CB.val.print();
x_DB.val.print();
x_DC.val.print();
```

より汎用的に問題を定式化すると以下のようになります.

集合	$City = \{A, B, C, D\}$	都市の集合
	$City\_ = \{B, C, D\}$	出発地点以外の都市の集合
定数	$dis_{c1c2}, c1 \in City, c2 \in City$	都市$c1$と都市$c2$の距離
	num	集合$City_$の要素数
0-1 変数	$x_{c1c2}, c1 \in City, c2 \in City$	都市$c1$から都市$c2$に移動するか否か
変数	$y_c, c \in City\_$	
目的関数（最小化）	$\sum_{c1 \in City, c2 \in City} dis_{c1c2} x_{c1c2}$	経路長
制約	$\sum_{c2 \in City} x_{c1c2} = 1, \forall c1 \in City$	都市$c1$から別都市の 1 つに移動
	$\sum_{c1 \in City} x_{c1c2} = 1, \forall c2 \in City$	都市$c2$に別都市の 1 つから移動
	$y_{c1} - y_{c2} + num \cdot x_{c1c2} \leq num - 1, c1 \in City\_, c2 \in City\_, c1 \neq c2$	サブツアーを排除

次に，定数（各都市間の距離）をデータファイルから与える SIMPLE モデルを示します．このようにモデルとデータを分離することにより,都市数が変わったとしてもデータファイルを変更するだけで対応できるようになります.

```
//集合と添え字
Set City(name="都市");
Element c1(set=City);
Element c2(set=City);
Set City_(name="出発地点を除く都市");
Element c_(set=City_);
Element c1_(set=City_);
Element c2_(set=City_);
//パラメータ
Parameter dis(name="距離",index=(c1,c2));
Parameter num = City_.card();
//変数
IntegerVariable x(name="移動",index=(c1,c2), type=binary);
Variable y(name="順番",index=c_);
//目的関数
Objective route(name="経路長",type=minimize);
route = sum(x[c1,c2]*dis[c1,c2], ((c1,c2),c1!=c2));
// 各都市から 1 つの別都市へ移動する
sum(x[c1,c2],(c2,c1!=c2))==1;
// 各都市に 1 つの別都市から移動する
sum(x[c1,c2],(c1,c1!=c2))==1;
// サブツアーを排除する
y[c1_] - y[c2_] + num*x[c1_,c2_] <= num-1, c1_!=c2_;
//求解
solve();
//出力
x.val.print();
y.val.print();
route.val.print();
```

データファイル（.dat 形式）は以下のようになります.

```
"都市" = A B C D ;
"出発地点を除く都市" = B C D ;

"距離" =
  [A,B] 6  [A,C] 5  [A,D] 5
  [B,C] 7  [B,D] 4
  [C,D] 3
  [B,A] 6  [C,A] 5  [D,A] 5
  [C,B] 7  [D,B] 4
  [D,C] 3
  ;
```

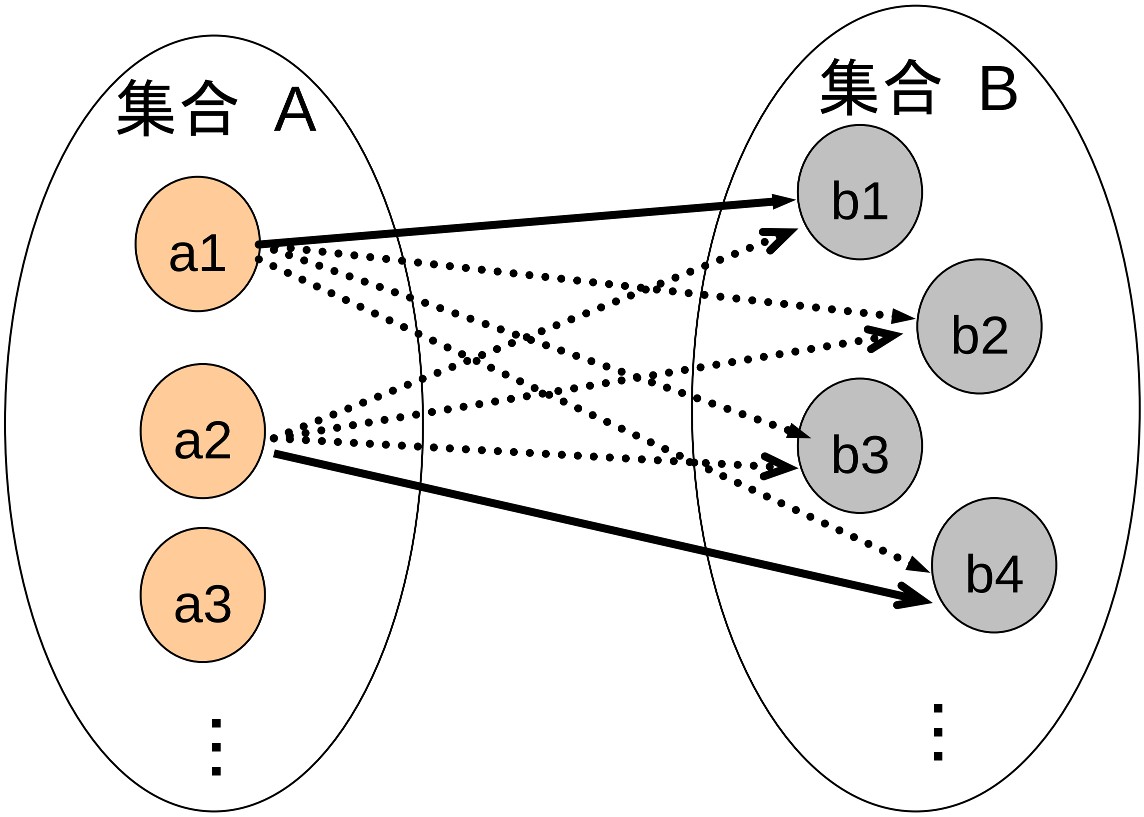
このモデルを実行すると，都市 A→都市 B→都市 D→都市 C→都市 A の順番で移動するのが最適解となり，その経路長は 18 であることがわかります.

2.13 割当問題

割当問題は現在多くの業務に利用されています. ここでは基本的な割当問題の例題を取り上げ, 割当問題の定式化のテクニック, モデル化の例, 大規模な問題に対するアプローチ法を紹介します.

2.13.1 割当問題とは

割当問題とは, 集合 A の要素を集合 B の要素のどれに割り当てるかを決定する問題です.



また, 割当問題はマス目を埋める問題に置き換えることができます. 上記の図を, マス目を埋めるイメージで表すと以下の図のようになります.

	a1	a2	a3		
b1	○				
b2			○		
b3					
b4		○			

割当問題はマスの埋め方のルールを制約条件として与えることによって,個々の問題に対応する定式化を行うことができます．割当問題の応用例として，例えば以下のものがあります.

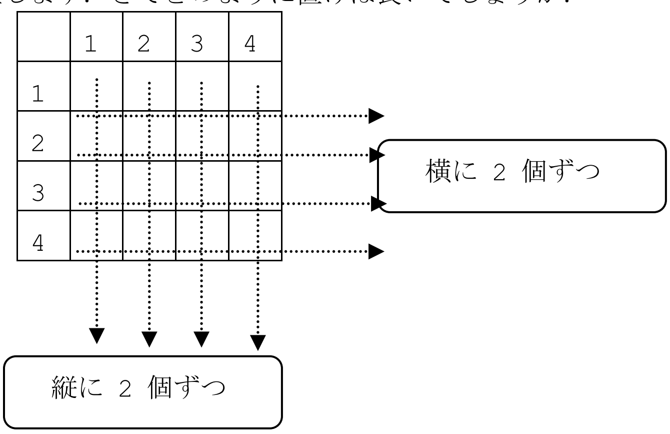
- 人員配置問題
- シフトスケジューリング問題
- 配車計画問題
- 訪問スケジューリング決定問題
- マーケットテリトリー決定問題

2.13.2 基礎的なマス埋め割当問題

ここではまず，マス目を埋めるだけの問題を考えていきます.

（例題）

４×４のマス目があります．このマスに石を置きたいのですが，全ての縦横に関して石の数が２つになるように配置します．さてどのように置けば良いでしょうか.



この問題を定式化するためには， 0-1 変数を用いる必要があります．この問題の場合以下の図のように，各マスに対して 0-1 変数を対応させます．

	1	2	3	4
1	x11	x12	x13	X14
2	x21	x22	x23	X24
3	x31	x32	x33	X34
4	x41	x42	x43	X44

定式化をすると以下のようになります．

0-1 変数	$x_{11}, x_{12}, x_{13}, x_{14}$ $x_{21}, x_{22}, x_{23}, x_{24}$ $x_{31}, x_{32}, x_{33}, x_{34}$ $x_{41}, x_{42}, x_{43}, x_{44}$	それぞれのマスに石を置くならば 1 置かないならば 0 ．
目的関数		この問題には目的関数はない
制約条件	$x_{11} + x_{12} + x_{13} + x_{14} = 2$	１行目を横に見たときに石を２つ置く
	$x_{21} + x_{22} + x_{23} + x_{24} = 2$	２行目を横に見たときに石を２つ置く
	$x_{31} + x_{32} + x_{33} + x_{34} = 2$	３行目を横に見たときに石を２つ置く
	$x_{41} + x_{42} + x_{43} + x_{44} = 2$	４行目を横に見たときに石を２つ置く
	$x_{11} + x_{21} + x_{31} + x_{41} = 2$	１列目を縦に見たときに石を２つ置く
	$x_{12} + x_{22} + x_{32} + x_{42} = 2$	２列目を縦に見たときに石を２つ置く
	$x_{13} + x_{23} + x_{33} + x_{43} = 2$	３列目を縦に見たときに石を２つ置く
	$x_{14} + x_{24} + x_{34} + x_{44} = 2$	４列目を縦に見たときに石を２つ置く

定式化した結果を SIMPLE で記述すると以下のようになります.

```
IntegerVariable x_11(type = binary);
IntegerVariable x_21(type = binary);
IntegerVariable x_31(type = binary);
IntegerVariable x_41(type = binary);

IntegerVariable x_12(type = binary);
IntegerVariable x_22(type = binary);
IntegerVariable x_32(type = binary);
IntegerVariable x_42(type = binary);

IntegerVariable x_13(type = binary);
IntegerVariable x_23(type = binary);
IntegerVariable x_33(type = binary);
IntegerVariable x_43(type = binary);

IntegerVariable x_14(type = binary);
IntegerVariable x_24(type = binary);
IntegerVariable x_34(type = binary);
IntegerVariable x_44(type = binary);

x_11 + x_12 + x_13 + x_14 == 2;
x_21 + x_22 + x_23 + x_24 == 2;
x_31 + x_32 + x_33 + x_34 == 2;
x_41 + x_42 + x_43 + x_44 == 2;

x_11 + x_21 + x_31 + x_41 == 2;
x_12 + x_22 + x_32 + x_42 == 2;
x_13 + x_23 + x_33 + x_43 == 2;
x_14 + x_24 + x_34 + x_44 == 2;

// 以下は出力用のプログラム
solve();
simple_printf("%d %d %d %d¥n",x_11,x_12,x_13,x_14);
simple_printf("%d %d %d %d¥n",x_21,x_22,x_23,x_24);
simple_printf("%d %d %d %d¥n",x_31,x_32,x_33,x_34);
simple_printf("%d %d %d %d¥n",x_41,x_42,x_43,x_44);
```

このモデルを NUOPT で実行すると，最後に

```
1 0 1 0
1 1 0 0
0 1 0 1
0 0 1 1
```

という表示がされ，この例題の答えを確認できます．どの縦横にも二箇所ずつ石が置いてある（1 と表示されている）のが分かります．

さて，次にこの問題を SIMPLE の添字機能を用いてモデル化してみます．定式化は以下のように変更します．

集合	$I = \{1,2,3,4\}$	行
	$J = \{1,2,3,4\}$	列
0-1 変数	$x_{ij}, i \in I, j \in J$	マス (i, j) に石を置くならば $x_{ij} = 1$，置かないならば $x_{ij} = 0$ とする．
目的関数		この問題には目的関数はない
制約条件	$\sum_i x_{ij} = 2, \forall j \in J$	各列 j は横に見たときに石を２つ置く
	$\sum_j x_{ij} = 2, \forall i \in I$	各行 i は縦に見たときに石を２つ置く

添字を用いることにより，以下のように簡単にモデル記述することができます．

```
Set I = "1 2 3 4";
Set J = "1 2 3 4";
Element i(set = I);
Element j(set = J);
IntegerVariable x(type = binary,index = (i,j));
sum(x[i,j],i) == 2;
sum(x[i,j],j) == 2;

// 以下は出力用のプログラム
solve();
simple_printf("%d %d %d %d¥n",
x[i,j],x[i,j+1],x[i,j+2],x[i,j+3],j == 1);
```

2.13.3 仕事割当問題

ここでは，仕事を人に効率よく割り当てる問題を取り上げます．次の例題を考えます．

（例題）「安藤」「佐藤」「鈴木」「山本」「渡辺」 の 5 人に仕事を割り当てます．仕事は 「接客」「厨房」「レジ打ち」「発注」「ごみ捨て」「買出し」「掃除」「仕込み」の 8 つです．各人を仕事に割り当てるにはコストがかかり，それは個人・仕事によって決まります．また，各人はそれぞれの仕事に対して熟練度があり，熟練度が高いほどコストがかかる傾向があります．以下は熟練度とコストをまとめたものです．

熟練度	安藤	佐藤	鈴木	山本	渡辺
接客	-1	3	-2	3	-4
厨房	5	-2	3	-4	5
レジ打ち	0	3	-2	3	-1
発注	-3	-1	1	1	2
ごみ捨て	1	-1	3	1	-1
買出し	-1	-1	1	0	2
掃除	2	-2	2	-3	4
仕込み	5	-2	0	1	5

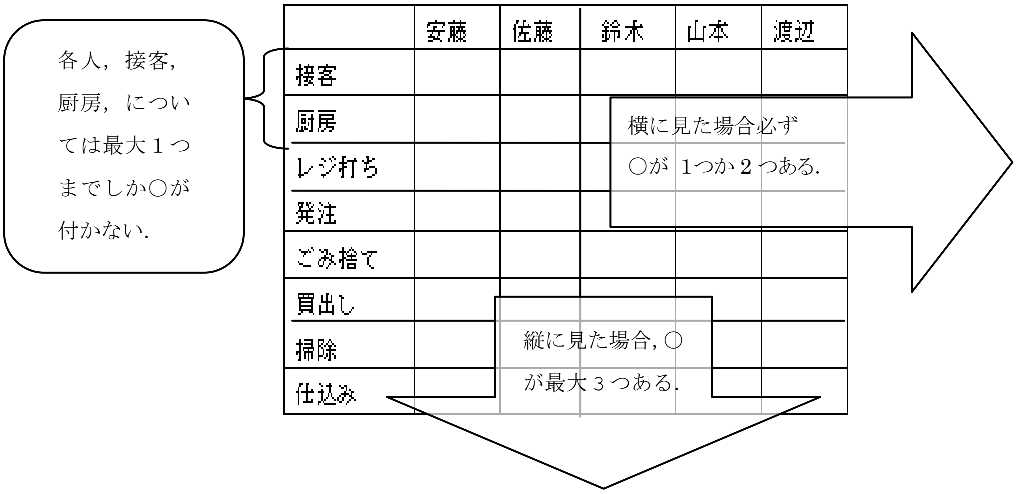
コスト	安藤	佐藤	鈴木	山本	渡辺
接客	570	1400	520	1410	450
厨房	1800	1000	1700	1050	2300
レジ打ち	800	1500	500	1500	600
発注	500	600	1000	1000	1200
ごみ捨て	800	600	1200	800	600
買出し	600	600	800	700	1300
掃除	1200	500	1200	500	1300
仕込み	1500	1000	1200	1200	1500

また，以下の点を守らなくてはいけません．

- 各人に割り振る仕事は，最大で３つまでとする．
- 「接客」「厨房」「レジ打ち」「掃除」「仕込み」は２人を割り当てる．
- 「発注」「ごみ捨て」「買出し」は１人を割り当てる．
- 「接客」「厨房」は別の人が担当する．
- 各仕事において，仕事を担当する人の熟練度の和を，その仕事のクオリティーとする．
- 各仕事のクオリティーは負になってはいけない．

このとき，コストの合計を最小にするような割り当て方を求めて下さい．

この問題も前節の問題と同様に以下のようなマス目に○をつける問題として考えることができます.



以上を踏まえて，以下のように定式化することができます.

集合	JOB＝｛**接客，厨房，レジ打ち，発注，ごみ捨て，買出し，掃除，仕込み**｝ **仕事の集合** $PEOPLE$＝｛安藤，佐藤，鈴木，山本，渡辺｝ 人の集合	
0-1 変数	$x_{jp}, j \in JOB, p \in PEOPLE$	仕事 j を人 p に割り当てるならば $x_{jp}=1$，そうでないならば $x_{jp}=0$ とする
定数	$cost_{jp}, j \in JOB, p \in PEOPLE$ $jyukuren_{jp}, j \in JOB, p \in PEOPLE$	仕事 j を人 p に割り当てる際のコスト 仕事 j を人 p が行う際の熟練度
目的関数 （最小化）	$\sum_{j,p} cost_{jp} \times x_{jp}$	コストの総和

制約条件	$\sum_{p} x_{jp} = 2, \forall j \in$ ｛接客，厨房，レジ打ち，掃除，仕込み｝ 「接客」「厨房」「レジ打ち」「掃除」「仕込み」は２人を割り当てる．
	$\sum_{p} x_{jp} = 1, \forall j \in$ ｛発注，ごみ捨て，買出し｝ 「発注」「ごみ捨て」「買出し」は１人を割り当てる．
	$\sum_{j} x_{jp} \le 3, \forall p \in PEOPLE$ 各人には，最大３つまでの仕事を割り当てることができる
	$\sum_{p} jyukuren_{jp} \times x_{jp} \ge 0, \forall j \in JOB$ 各仕事のクオリティーが負になってはいけない
	$\sum_{j,\, j \in \{接客,\, 厨房\}} x_{jp} \le 1, \forall p \in PEOPLE$ 接客，厨房は違う人が担当する（同じ人が接客と厨房を兼ねない）．

今回は定数に関してはファイルからデータを与えてみます. モデル部分は以下のようになります.

```
Set Job;
Set People;
Element j(set = Job);
Element p(set = People);

IntegerVariable x(type = binary, index = (j,p));
Parameter cost(index = (j,p), name = "コスト");
Parameter jyukuren(index = (j,p), name = "熟練度");

sum(x[j,p],p) == 2,
  j == "接客" || j == "厨房" || j == "レジ打ち" ||
  j == "掃除" || j == "仕込み";
sum(x[j,p],p) == 1,
  j == "発注" || j == "ごみ捨て" || j == "買出し" ;
sum(x[j,p],j) <= 3;
sum(x[j,p]*jyukuren[j,p],p) >= 0;
sum(x[j,p],(j, j == "接客" || j == "厨房")) <= 1;

Objective total_cost(type = minimize, name = "総コスト");
total_cost = sum(cost[j,p]*x[j,p],(j,p));

// 以下出力用プログラム
solve();
total_cost.val.print();
simple_printf("%s,%s¥n",j,p,x[j,p].val == 1);
simple_printf("¥n");
simple_printf("%s のクオリティーは  %d¥n",
j,sum(jyukuren[j,p]*x[j,p].val,p));
```

以下が csv 形式の入力ファイルです．コストに関してと熟練度に関しての二種類のファイルを用意する必要があります．

```
熟練度,安藤,佐藤,鈴木,山本,渡辺
接客,-1,3,-2,3,-4
厨房,5,-2,3,-4,5
レジ打ち,0,3,-2,3,-1
発注,-3,-1,1,1,2
ごみ捨て,1,-1,3,1,-1
買出し,-1,-1,1,0,2
掃除,2,-2,2,-3,4
仕込み,5,-2,0,1,5
```

```
コスト,安藤,佐藤,鈴木,山本,渡辺
接客,570,1400,520,1410,450
厨房,1800,1000,1700,1050,2300
レジ打ち,800,1500,500,1500,600
発注,500,600,1000,1000,1200
ごみ捨て,800,600,1200,800,600
買出し,600,600,800,700,1300
掃除,1200,500,1200,500,1300
仕込み,1500,1000,1200,1200,1500
```

このモデルを NUOPT で実行すると，最後に

```
総コスト=13380
"ごみ捨て","安藤"
"レジ打ち","佐藤"
"レジ打ち","渡辺"
"仕込み","山本"
"仕込み","鈴木"
"厨房","佐藤"
"厨房","鈴木"
"接客","安藤"
"接客","山本"
"掃除","安藤"
"掃除","佐藤"
"買出し","山本"
"発注","鈴木"

"ごみ捨て"のクオリティーは　1
"レジ打ち"のクオリティーは　2
"仕込み"のクオリティーは　1
"厨房"のクオリティーは　1
"接客"のクオリティーは　2
"掃除"のクオリティーは　0
"買出し"のクオリティーは　0
"発注"のクオリティーは　1
```

という表示がされ，この例題の答えを確認できます．

（例題）先ほどの問題に以下の条件を付け加えて下さい．

安藤に「接客」をさせることはできない
佐藤に「厨房」をさせることはできない
鈴木に「レジ打ち」をさせることはできない
山本に「発注」をさせることはできない
渡辺に「ごみ捨て」をさせることはできない

さて,この問題ですが通常に考えると先ほどの問題に以下の制約を付け加えることで解くことができます．また記述する位置は x の宣言以降，solve() の手前であればどこでもかまいません.

```
x["接客,安藤"] == 0;
x["厨房,佐藤"] == 0;
x["レジ打ち,鈴木"] == 0;
x["発注,山本"] == 0;
x["ごみ捨て,渡辺"] == 0;
```

上記記述を追加し，モデルを実行すると結果は以下のように変わります.

```
総コスト=13470
"ごみ捨て","安藤"
"レジ打ち","佐藤"
"レジ打ち","渡辺"
"仕込み","山本"
"仕込み","鈴木"
"厨房","安藤"
"厨房","山本"
"接客","佐藤"
"接客","鈴木"
"掃除","安藤"
"掃除","佐藤"
"買出し","山本"
"発注","鈴木"

"ごみ捨て"のクオリティーは 1
"レジ打ち"のクオリティーは 2
"仕込み"のクオリティーは 1
"厨房"のクオリティーは 1
"接客"のクオリティーは 1
"掃除"のクオリティーは 0
"買出し"のクオリティーは 0
"発注"のクオリティーは 1
```

結果を見ると，総コストが多くなっていることが分かります.

以上の考え方は以下の図のように変数を準備し，色が付いているところが 0 であるという制約を設けたと考えることができます．（なお図中のレジはレジ打ち，ごみはごみ捨て，買出は買出し，仕込は仕込みです）

	安藤	佐藤	鈴木	山本	渡辺
接客	x[接客，安藤]	x[接客，佐藤]	x[接客，鈴木]	x[接客，山本]	x[接客，渡辺]
厨房	x[厨房，安藤]	x[厨房，佐藤]	x[厨房，鈴木]	x[厨房，山本]	x[厨房，渡辺]
レジ打ち	x[レジ，安藤]	x[レジ，佐藤]	x[レジ，鈴木]	x[レジ，山本]	x[レジ，渡辺]
発注	x[発注，安藤]	x[発注，佐藤]	x[発注，鈴木]	x[発注，山本]	x[発注，渡辺]
ごみ捨て	x[ごみ，安藤]	x[ごみ，佐藤]	x[ごみ，鈴木]	x[ごみ，山本]	x[ごみ，渡辺]
買出し	x[買出，安藤]	x[買出，佐藤]	x[買出，鈴木]	x[買出，山本]	x[買出，渡辺]
掃除	x[掃除，安藤]	x[掃除，佐藤]	x[掃除，鈴木]	x[掃除，山本]	x[掃除，渡辺]
仕込み	x[仕込，安藤]	x[仕込，佐藤]	x[仕込，鈴木]	x[仕込，山本]	x[仕込，渡辺]

ここで，上の図の色のついている部分は 1 になることがないので，はじめから変数を準備する必要がないと考えることが出来ます．特に大規模の割当問題においては，無駄な変数を準備することにより不用意に問題の規模を大きくしてしまうと，計算のパフォーマンスを著しく低下させてしまう要因になります．ここでは明らかな制約に関しては「制約を付け加えることなく，そのような選択肢を元々準備しない」という方法を紹介します．以下の図のようなイメージです．

	安藤	佐藤	鈴木	山本	渡辺
接客		x[接客，佐藤]	x[接客，鈴木]	x[接客，山本]	x[接客，渡辺]
厨房	x[厨房，安藤]		x[厨房，鈴木]	x[厨房，山本]	x[厨房，渡辺]
レジ打ち	x[レジ，安藤]	x[レジ，佐藤]		x[レジ，山本]	x[レジ，渡辺]
発注	x[発注，安藤]	x[発注，佐藤]	x[発注，鈴木]		x[発注，渡辺]
ごみ捨て	x[ごみ，安藤]	x[ごみ，佐藤]	x[ごみ，鈴木]	x[ごみ，山本]	
買出し	x[買出，安藤]	x[買出，佐藤]	x[買出，鈴木]	x[買出，山本]	x[買出，渡辺]
掃除	x[掃除，安藤]	x[掃除，佐藤]	x[掃除，鈴木]	x[掃除，山本]	x[掃除，渡辺]
仕込み	x[仕込，安藤]	x[仕込，佐藤]	x[仕込，鈴木]	x[仕込，山本]	x[仕込，渡辺]

定式化する際に，上記を表す集合 *JPPair* を準備する必要があります．

- $(j, p) \in JPPair \Leftrightarrow$ j を p に割り当てることができる

以下が $JPPair$ を用いた定式化です.

集合	$JOB=$｛**接客，厨房，レジ打ち，発注，ごみ捨て，買出し，掃除，仕込み**｝ **仕事の集合，** $j \in JOB$	
	$PEOPLE=$｛安藤，佐藤，鈴木，山本，渡辺｝ 人の集合， $p \in PEOPLE$ $(j,p) \in JPPair \Leftrightarrow$ j を p に割り当てることができる	
0-1 変数	$x_{jp},(j,p) \in JPPair$	仕事 j を人 p に割り当てるならば $x_{jp}=1$，そうでないならば $x_{jp}=0$ とする
定数	$cost_{jp},(j,p) \in JPPair$	仕事 j を人 p に割り当てる際のコスト
	$jyukuren_{jp},(j,p) \in JPPair$	仕事 j を人 p が行う際の熟練度
目的関数 （最小化）	$\sum_{j,p,(j,p)\in JPPair} cost_{jp} \times x_{jp}$ コストの総和	
制約条件	$\sum_{p,(j,p)\in JPPair} x_{jp}=2, \forall j \in$｛接客，厨房，レジ打ち，掃除，仕込み｝ 「接客」「厨房」「レジ打ち」「掃除」「仕込み」は２人を割り当てる. $\sum_{p,(j,p)\in JPPair} x_{jp}=1, \forall j \in$｛発注，ごみ捨て，買出し｝ 「発注」「ごみ捨て」「買出し」は１人を割り当てる $\sum_{j,(j,p)\in JPPair} x_{jp} \le 3$ 各人には，最大３つまで仕事を割り当てることができる $\sum_{p,(j,p)\in JPPair} jyukuren_{jp} \times x_{jp} \ge 0$ 各仕事のクオリティーが負になってはいけない $\sum_{(j,p),j\in\{接客,厨房\},(j,p)\in JPPair} x_{jp} \le 1$ 接客，厨房は違う人が担当する.	

モデルは以下のようになります.

```
Set Job;
Set People;
Element j(set = Job);
Element p(set = People);

Set JPPair(dim = 2,superSet = (Job,People));
IntegerVariable x(type = binary, index = JPPair);
Parameter cost(index = JPPair, name = "コスト");
Parameter jyukuren(index = JPPair, name = "熟練度");

sum(x[j,p],(p,(j,p)<JPPair)) == 2,
  j == "接客" || j == "厨房" || j == "レジ打ち" ||
  j == "掃除" || j == "仕込み";
sum(x[j,p],(p,(j,p)<JPPair)) == 1,
  j == "発注" || j == "ごみ捨て" || j == "買出し" ;
sum(x[j,p],(j,(j,p)<JPPair)) <= 3;
sum(x[j,p]*jyukuren[j,p],(p,(j,p)<JPPair)) >= 0;
sum(x[j,p],(j,(j,p)<JPPair,j == "接客" || j == "厨房")) <= 1;

Objective total_cost(type = minimize, name = "総コスト");
total_cost = sum(cost[j,p]*x[j,p],(j,p,(j,p)<JPPair));

// 以下出力用プログラム
solve();
total_cost.val.print();
x.val.print();
```

上記記述内の JPPair には無駄な [j,p] のペアを入れてはいけないので，入力ファイルもそれに伴い以下のように変わります．

```
j,p,熟練度,コスト
接客,佐藤,3,1400
接客,鈴木,-2,520
接客,山本,3,1410
接客,渡辺,-4,450
厨房,安藤,5,1800
厨房,鈴木,3,1700
厨房,山本,-4,1050
厨房,渡辺,5,2300
レジ打ち,安藤,0,800
レジ打ち,佐藤,3,1500
レジ打ち,山本,3,1500
レジ打ち,渡辺,-1,600
発注,安藤,-3,500
発注,佐藤,-1,600
発注,鈴木,1,1000
発注,渡辺,2,1200
ごみ捨て,安藤,1,800
ごみ捨て,佐藤,-1,600
ごみ捨て,鈴木,3,1200
ごみ捨て,山本,1,800
買出し,安藤,-1,600
買出し,佐藤,-1,600
買出し,鈴木,1,800
買出し,山本,0,700
買出し,渡辺,2,1300
掃除,安藤,2,1200
掃除,佐藤,-2,500
掃除,鈴木,2,1200
掃除,山本,-3,500
掃除,渡辺,4,1300
仕込み,安藤,5,1500
仕込み,佐藤,-2,1000
仕込み,鈴木,0,1200
仕込み,山本,1,1200
仕込み,渡辺,5,1500
```

またこのモデルを制約充足問題ソルバ wcsp を用いる場合にはモデルを以下のように変更する必要があります.

```
options.method = "wcsp";// 解法に wcsp を用いる指定
options.maxtim = 10;// 計算時間の設定

Set Job;
Set People;
Element j(set = Job);
Element p(set = People);

Set JPPair(dim = 2,superSet = (Job,People));
IntegerVariable x(type = binary,index = JPPair);
Parameter cost(index = JPPair,name = "コスト");
Parameter jyukuren(index = JPPair,name = "熟練度");

sum(x[j,p],(p,(j,p)<JPPair)) == 2,
  j == "接客" || j == "厨房" ||
  j == "レジ打ち" || j == "掃除" || j == "仕込み";

// wcsp 特有の関数 selection
selection(x[j,p],(p,(j,p)<JPPair)),
  j == "発注" || j == "ごみ捨て" || j == "買出し";

sum(x[j,p],(j,(j,p)<JPPair)) <= 3;
sum(x[j,p]*jyukuren[j,p],(p,(j,p)<JPPair)) >= 0;
sum(x[j,p],(j,(j,p)<JPPair,j == "接客" || j == "厨房")) <= 1;

Objective total_cost(type = minimize,name = "総コスト");
total_cost = sum(cost[j,p]*x[j,p],(j,p,(j,p)<JPPair));

// 以下出力用プログラム
solve();
total_cost.val.print();
x.val.print();
```

大規模な割当問題を解く際の問題規模に対する対応として上記で紹介した手法を試してみて下さい.

2.13.4 施設配置問題

ここでは，どこに施設を配置するかを決定する問題を取り上げます．次の例題を考えます．

（例題） 施設を候補地のどこに建てるかを決定します．施設として緊急時の収容所を考えます．つまり，火事での火傷や脳卒中の患者の受け入れ等を考慮するため，エリアの合計人数を考慮するのではなく，各エリアの対象予想数をカバーできればよく，例えば施設 1 がエリア A，B，C を担当するときには，施設 1 の容量に関する制約は A，B，C の合計人数ではなく，A，B，C エリアのそれぞれの人数をカバーできればよいということになります．これは，異なるエリアで同時に事象が起こりえないモデルとして考えることを意味します．問題を以下に整理します．

- 緊急時の収容を考慮する（異なる地域を同時に収容しない）
- 各施設は受け入れ容量を持っている
- 各地域は最寄の施設に対応づけられる
- 容量をオーバーする場合は距離に比例するコストを支払えば 3 人まで超過することができる
- 施設を建設するにはコストがかかる
- 施設の建設候補地は 7 地点である
- 地域は 5 × 5 の 25 地点である

このとき，容量超過コストと施設建設コストの和を最小化するような建設場所を決定したいとします．

以上を踏まえて，以下のように定式化することができます．但し M は大きな定数とします．

集合	I = {**1 ... 7** } **施設の集合** J = {1 ... 25} 地域の集合	
0-1 変数	$x_i, i \in I$	施設 i を建設するならば $x_i = 1$，そうでないならば $x_i = 0$ とする
離散変数	$y_j, j \in J$	エリア j に対しての超過人数．0 ... 3 の値をとる
定数	$dist_{ij}, i \in I, j \in J$	施設 i から地域 j までの距離．今回は座標を与え直線距離をモデル内で求める
	num_j	地域 j で発生する緊急患者数
	$capa_i$	施設 i が受け入れることのできる人数
	$cost_i$	施設 i の建設コスト
目的関数 （最小化）	$\sum_i cost_i x_i + \sum_j y_j \min\left(dist_{ij} + M\left(1 - x_i\right), i\right)$	コストの総和
制約条件	下記参照	

制約式は「各地域は最寄の施設に収容される」ということと，「施設の収容には容量があり，3 人までは容量の超過が認められる」ということです．これは WCSP 特有の関数である argmin 関数を用いることによって 1 つの制約式で表現することができます．これは実際にモデルで確認をして下さい．

今回は定数に関してはファイルからデータを与えてみます．モデル部分は以下のようになります．

```
// 施設の集合
Set I;
Element i(set = I);
// 地域の集合
Set J;
Element j(set = J);
IntegerVariable x(index = i,type = binary,name = "x");

Set S = "0 ... 3"; // 離散変数用の集合
DiscreteVariable y(dom = S,index = j,name = "y");

// dist はモデル内で計算する
Parameter dist(index = (i,j),name = "dist");

// 実際には以下の座標を入力する
Parameter pos_xi(index = i,name = "pos_xi");
Parameter pos_xj(index = j,name = "pos_xj");
Parameter pos_yi(index = i,name = "pos_yi");
Parameter pos_yj(index = j,name = "pos_yj");
// 距離は直線距離を考える
dist[i,j] = sqrt(pow((pos_xi[i] - pos_xj[j]),2)
               +pow((pos_yi[i] - pos_yj[j]),2));

Parameter num(index = j,name = "num");
Parameter n(index = (i,j),name = "n");
Parameter capa(index = i,name = "capa");
Parameter cost(index = i,name = "cost");
Parameter M;
M = 100;

// 施設の容量超過分のための計算
n[i,j] = capa[i] - num[j];

Objective total_cost(type = minimize);
total_cost = sum(cost[i]*x[i],i) +
sum(y[j]*min(dist[i,j]+ M*(1 - x[i]),i),j);

// 制約式，超過人数 y についての記述
n[argmin((x[i] - 1)*M + dist[i,j],i),j] + y[j] >= 0;

options.maxtim = 5;
options.method = “wcsp”;
solve();
```

以下が csv 形式の入力ファイルです. 施設に関してと地域に関しての二種類のファイルを用意する必要があります.

```
i,pos_xi,pos_yi,capa,cost
1,13,18,6,24
2,14,4,9,20
3,19,15,7,41
4,3,10,5,13
5,18,11,7,23
6,7,5,10,18
7,6,1,7,20
```

```
j,pos_xj,pos_yj,num
1,0,0,3
2,0,5,6
3,0,10,1
4,0,15,7
5,0,20,3
6,5,0,5
7,5,5,8
8,5,10,12
9,5,15,3
10,5,20,4
11,10,0,5
12,10,5,9
13,10,10,2
14,10,15,1
15,10,20,11
16,15,0,3
17,15,5,4
18,15,10,5
19,15,15,9
20,15,20,1
21,20,0,5
22,20,5,7
23,20,10,8
24,20,15,2
25,20,20,11
```

このように，WCSP 特有の関数を用いると煩雑な制約式でも簡単に記述できる場合があります．次にこの問題で「最寄の施設まで 10.5 以上距離がある地域は 10 地域までしか許さない」という制約を記述することを考えてみます．これは WCSP 用の count 関数を用いれば簡単に記述することができます．この場合どの施設がどのエリアを担当するという変数 z を加えて以下のように記述することができます．

```
IntegerVariable z(index = (i,j),type = binary);
selection(z[i,j],i);
Element ii(set = I);
sum(dist[i,j]*z[i,j],i) <= min(dist[ii,j]+(1 - x[ii])*M,ii);
semiHardConstraint();
count( 10.5 <= z[i,j]*dist[i,j],(i,j)) <= 10;
```

ここで semiHardConstraint を用いたのは万が一 10 地域以内でおさまらなかった場合に「なるべく 10 地域以内で」ということを表現するためです．

2.14 二次割当問題

二次割当問題とは，1957 年 Koopmans と Beckmann により考案された，目的関数が二次式となる割当問題です．NUOPT V13 より導入された Vector/Matrix クラスの表記を用いて，次の例題を考えてみます．

（例題）
工場 1，..，5 を地区 I，..，V に配置することを考えます．ただし，各工場間での物資を輸送する量，頻度などのフロー量，および各地区間の距離が，以下のように与えられているとします．

各工場のフロー						各地域間の距離					
	1	2	3	4	5		I	II	III	IV	V
1	0	15	12	8	7	I	0	4	7	6	10
2	15	0	14	2	4	II	4	0	2	5	4
3	12	14	0	6	6	III	7	2	0	8	7
4	8	2	6	0	11	IV	6	5	8	0	12
5	7	4	6	11	0	V	10	4	7	12	0

物資の輸送コストを（フロー）×（距離）で与えるとき，最もコストがかからない配置を求めなさい．

この問題は，工場間のフロー行列を $F(f_{ij})$，地区間の距離行列を $D(d_{kl})$ とすると，

$$\min_{\phi \in S_n} \sum_i \sum_j f_{ij}\, d_{\phi(i)\phi(j)} \qquad (1$$

と表現されます．ここで，S_n は n 次の置換群（この問題では $n=5$ ）です．

ここで，置換 $\phi \in S_n$ に対し，次のような変数 x_{ik}

$$x_{ik} = \begin{cases} 1 & \text{if } \phi(i) = k \\ 0 & \text{otherwise.} \end{cases}$$

を成分に持つ $n \times n$ 行列 X を用いることで，(1) における $d_{\phi(i)\phi(j)}$ は

$$(d_{\phi(i)\phi(j)}) = XDX^{\mathrm{T}}$$

と表すことができます．

よって(1)は，次のような整数計画問題に書き換えることができます.

$$\min F \bullet (XDX^{\mathrm{T}}) = \sum_{i,j} (f_{ij} d_{\phi(i)\phi(j)})$$

$$\text{s.t.} \ \sum_i x_{ik} = 1, \ k = 1, 2, \ldots, n,$$

$$\sum_k x_{ik} = 1, \ i = 1, 2, \ldots, n,$$

$$x_{ik} = \{0,1\}, \ i, k = 1, 2, \ldots, n.$$

以上より，定式化は次のようになります.

集合	Fac	工場の集合
	Loc	地区の集合
定数	$F_{ij}, \ i, j \in Fac$	各工場間のフロー
	$D_{mn}, m, n \in Loc$	各地区間の距離
変数	$X_{im}, i \in Fac, m \in Loc$	工場 i を地区 m に配置する場合 1 それ以外は 0
目的関数（最小化）	$F \bullet (XDX^{\mathrm{T}})$	$\bullet$ は要素ごとの積の和
制約条件	$Xe = e$ $X^{\mathrm{T}} e = e$	e は全成分 1 のベクトル

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 集合と添字
Set Fac,Loc;
Element i(set=Fac),j(set=Fac),m(set=Loc),n(set=Loc);

// パラメータ
Matrix F((i,j)),D((m,n));

// 変数
IntegerVariable x(index=(i,m),type=binary);
Matrix X((i,m));
X[i,m] = x[i,m];// 変数を行列に詰める

// 目的関数
Objective total_cost(type=minimize);
total_cost = inprod(F,X*D*trans(X));

// 制約条件
X*ones(Loc) == ones(Fac);
trans(X)*ones(Fac)== ones(Loc);

options.method = “wcsp”;
options.maxtim = 10;

solve();
simple_printf(“工場 %d を地区 %s に配置する¥n”,i,m,X[i,m].val == 1);
```

V12 までの SIMPLE では，この目的関数を

```
total_cost = sum(f[i,j]*d[k,l]*x[i,k]*x[j,l],(i,j,k,l));
```

と記述します．Vector/Matrix クラスにより，より簡潔に記述できることがわかります.

以下が csv 形式の入力ファイルです.

```
F,1,2,3,4,5
1,0,15,12,8,7
2,15,0,14,2,4
3,12,14,0,6,6
4,8,2,6,0,11
5,7,4,6,11,0
```

```
D,I,II,III,IV,V
I,0,4,7,6,10
II,4,0,2,5,4
III,7,2,0,8,7
IV,6,5,8,0,12
V,10,4,7,12,0
```

このモデルを実行すると次のような結果が得られます.

```
工場 1 を地区 II に配置する
工場 2 を地区 V に配置する
工場 3 を地区 III に配置する
工場 4 を地区 IV に配置する
工場 5 を地区 I に配置する
```

2.15 設備計画問題

設備計画問題として以下の例題では，電力を購入して生産を行っている工場が，自家発電用の発電機を導入する問題を考えます．

（例題）

電力を購入して生産を行っているある工場が，自家発電用の発電機の導入を計画しています．ただしその際，購入したスポット電力と自家発電の電力の和が，想定されるすべての時刻で電力需要を上回らなければなりません．なお，購入を想定するスポット電力商品は次の 3 つとします．スポット電力ですので，時刻毎に購入する商品を変更することが可能です．ただし，各時刻において必ず 1 つの商品を選択して購入しなければなりません．

商品	1	2	3
出力電力(kWh)	5	10	20
単位時間あたりのコスト	10	20	30

また，新たに導入する候補となる発電機は，以下の 6 種類とします．発電機を導入しますと，以下のような定常的な出力が得られるものとします．

発電機	1	2	3	4	5	6
出力電力(kWh)	5	6	8	10	15	20
導入コスト	100	120	150	160	280	300

時刻 t = 1,…,24 について電力需要予想 D_t は以下のように与えられているものとします．スポット電力購入コストと発電機導入コストの和を最小化するような，発電機の導入方法およびスポット電力の購入方法を求めて下さい．

t	1	2	3	4	5	6	7	8	9	10	11	12
D_t	10	10	12	12	15	20	30	30	35	40	50	56
t	13	14	15	16	17	18	19	20	21	22	23	24
D_t	52	47	40	35	42	50	48	40	30	20	15	12

この問題の定式化は，以下のようになります．

集合	$Product$	商品集合
	$Dynamo$	発電機集合
	$Time$	時刻集合
定数	$costS_i,\quad i \in Product$	各商品に対する単位時間あたりのコスト
	$costP_j,\quad j \in Dynamo$	各発電機に対する導入コスト
	$p_i,\quad i \in Product$	各商品の単位時間あたりの出力電力
	$q_j,\quad j \in Dynamo$	各発電機の単位時間あたりの出力電力
	$D_t,\quad t \in Time$	時刻ごとの電力需要
0-1 整数変数	$u_i^{\mathrm{t}},\quad i \in Product,\quad t \in Time$	各時刻ごとに各商品について，利用するならば 1，利用しないならば 0
	$x_j,\quad j \in Dynamo$	各発電機に関して，導入するならば 1，しないならば 0
目的関数（最小化）	$\sum_{i,t} costS_i \times u_i^t + \sum_j costP_j \times x_j$	総コスト（スポット電力購入コストと発電機導入コストの和）の最小化
制約条件	$\sum_i u_i^t = 1,\ \forall t \in Time$	各時刻における商品の使用は 1 つのみ
	$\max_t\{D_t - \sum_i p_i \times u_i^t\} \le \sum_j q_j \times x_j$	すべての時刻において，購入したスポット電力と自家発電の出力電力の和が電力需要を上回らないといけない

定式化のポイントとしましては，すべての時刻における電力の総和と電力需要に関する制約式を，時刻ごとに設けるのでなく，時刻に関する max 関数を用いて表現する点にあります．そのため制約充足問題ソルバ wcsp を用いて解くべく，SIMPLE では以下のように記述します．なお，max 関数に関しましては，NUOPT_SIMPLE_マニュアルの記述もあわせてご覧下さい．

```
// 商品集合
Set Product;
Element i(set=Product);
// 時刻集合
Set Time;
Element t(set=Time);
// 発電機集合
Set Dynamo;
Element j(set=Dynamo);

// コスト
// 各商品に対する単位時間あたりのコスト
Parameter costS(name="costS", index=i);
// 各発電機に対する導入コスト
Parameter costP(name="costP", index=j);

// 出力電力
// 各商品の単位時間あたりの出力電力
Parameter p(name="p", index=i);
// 各発電機の単位時間あたりの出力電力
Parameter q(name="q", index=j);

// 時刻ごとの電力需要
Parameter D(name="D", index=t);

// 変数
// 各時間ごとに，どの商品を利用するか
IntegerVariable u(name=”商品”, index=(i,t), type=binary);
// どの発電機を導入するか
IntegerVariable x(name=”発電機”, index=j, type=binary);
```

```
// 目的関数（総コストの最小化）
Objective cost(name="総コスト", type=minimize);
cost = sum(costS[i]*u[i,t], (i,t)) + sum(costP[j]*x[j], j);

// 制約式
// 各時間における商品の使用は 1 つのみ
selection(u[i,t], i);
// すべての時刻において，購入したスポット電力と自家発電の出力電力の和が
// 電力需要を上回らないといけない
max(D[t] - sum(p[i]*u[i,t], i), t) <= sum(q[j]*x[j], j);

// 最大求解時間の設定
options.maxtim = 5;

// 求解
solve();

// 結果の出力
cost.val.print();
x.val.print();
u.val.print();
```

データファイル（dat 形式）は以下のようになります．

```
costS = [1] 10 [2] 20 [3] 30;
costP = [1] 100 [2] 120 [3] 150 [4] 160 [5] 280 [6] 300;

p = [1] 5 [2] 10 [3] 20;
q = [1] 5 [2] 6 [3] 8 [4] 10 [5] 15 [6] 20;

D = [1] 10 [2] 10 [3] 12 [4] 12 [5] 15 [6] 20 [7] 30 [8] 30
[9] 35 [10] 40 [11] 50 [12] 56 [13] 52 [14] 47 [15] 40 [16] 35
[17] 42 [18] 50 [19] 48 [20] 40 [21] 30 [22] 20 [23] 15 [24] 12;
```

このモデルを実行すると，以下のような解が得られ，発電機の導入方法および電力の購入方法が求まりました．

```
総コスト=950
発電機[1]=0
発電機[2]=0
発電機[3]=1
発電機[4]=1
発電機[5]=0
発電機[6]=1
```

```
商品[1,1]=1
商品[1,2]=1
商品[1,3]=1
商品[1,4]=1
商品[1,5]=1
商品[1,6]=1
商品[1,7]=1
商品[1,8]=1
商品[1,9]=1
商品[1,10]=1
商品[1,11]=0
商品[1,12]=0
商品[1,13]=0
商品[1,14]=0
商品[1,15]=1
商品[1,16]=1
商品[1,17]=1
商品[1,18]=0
商品[1,19]=0
商品[1,20]=1
商品[1,21]=1
商品[1,22]=1
商品[1,23]=1
商品[1,24]=1
```

```
商品[2,1]=0
商品[2,2]=0
商品[2,3]=0
商品[2,4]=0
商品[2,5]=0
商品[2,6]=0
商品[2,7]=0
商品[2,8]=0
商品[2,9]=0
商品[2,10]=0
商品[2,11]=0
商品[2,12]=0
商品[2,13]=0
商品[2,14]=1
商品[2,15]=0
商品[2,16]=0
商品[2,17]=0
商品[2,18]=0
商品[2,19]=1
商品[2,20]=0
商品[2,21]=0
商品[2,22]=0
商品[2,23]=0
商品[2,24]=0
```

```
商品[3,1]=0
商品[3,2]=0
商品[3,3]=0
商品[3,4]=0
商品[3,5]=0
商品[3,6]=0
商品[3,7]=0
商品[3,8]=0
商品[3,9]=0
商品[3,10]=0
商品[3,11]=1
商品[3,12]=1
商品[3,13]=1
商品[3,14]=0
商品[3,15]=0
商品[3,16]=0
商品[3,17]=0
商品[3,18]=1
商品[3,19]=0
商品[3,20]=0
商品[3,21]=0
商品[3,22]=0
商品[3,23]=0
商品[3,24]=0
```

2.16 最小二乗問題

最小二乗問題は，科学や工学の分野でよく利用される基本的な問題です．この問題の典型的な例としては，曲線の当てはめがあります．実験により得られたデータをモデル曲線に当てはめ定数を推定していくことを考えます．すると，一般に各観測点においてデータとモデルの間には誤差が発生してしまいます．この時，「各観測点における誤差の二乗和を最小にする」という基準を採用しなるべく良い曲線に当てはめようとすると最小二乗問題となります．

（例題）

A さんは，実験開始から t 秒後の y という値を計測するという実験を行いました．その結果をまとめると，下の表のようになりました．A さんはこの結果をプロットしたところ， $f(t)=at^2+bt+c$ という二次関数モデルに当てはめられることに気付きました．そこで，各計測時刻でのデータ値とモデルから得られる値との誤差の二乗和が最小になるように a,b,c を推定して下さい．

t	y
0.5	1.22470
1.0	0.87334
1.5	0.99577
2.0	1.34215
2.5	2.12172

t	y
3.0	3.22933
3.5	4.44744
4.0	6.13509
4.5	8.14697
5.0	10.50759

この例題において，変数となるのは推定する a,b,c の 3 つです．また，ある計測時刻 t でのデータ値 $y(t)$ とモデルから得られる値 $f(t)$ との誤差は $at^2+bt+c-y(t)$ となります．よって，各計測時刻について誤差の二乗を求め，その総和をとることで最小化する目的関数が得られます．

以上のことから，この例題は次のように定式化できます.

変数	a	2 次の項の係数
	b	1 次の項の係数
	c	定数項

目的関数（最小化）

$$
\begin{aligned}
&(0.5^2 a + 0.5b + c - 1.22470)^2 + (1.0^2 a + 1.0b + c - 0.87334)^2 \\
&\quad + (1.5^2 a + 1.5b + c - 0.99577)^2 + (2.0^2 a + 2.0b + c - 1.34215)^2 \\
&\quad + (2.5^2 a + 2.5b + c - 2.12172)^2 + (3.0^2 a + 3.0b + c - 3.22933)^2 \\
&\quad + (3.5^2 a + 3.5b + c - 4.44744)^2 + (4.0^2 a + 4.0b + c - 6.13509)^2 \\
&\quad + (4.5^2 a + 4.5b + c - 8.14697)^2 + (5.0^2 a + 5.0b + c - 10.50759)^2
\end{aligned}
$$

各計測時刻における誤差の二乗和

ここで，この結果を SIMPLE で何も工夫せずに記述すると次のようになります.

```
//変数の宣言
Variable a(name="２次の項の係数");
Variable b(name="１次の項の係数");
Variable c(name="定数項");
//誤差の二乗和の最小化
Objective err(type=minimize);
err=(0.5*0.5*a+0.5*b+c-1.2247)*(0.5*0.5*a+0.5*b+c-1.2247)
+(1.0*1.0*a+1.0*b+c-0.87334)*(1.0*1.0*a+1.0*b+c-0.87334)
+(1.5*1.5*a+1.5*b+c-0.99577)*(1.5*1.5*a+1.5*b+c-0.99577)
+(2.0*2.0*a+2.0*b+c-1.34215)*(2.0*2.0*a+2.0*b+c-1.34215)
+(2.5*2.5*a+2.5*b+c-2.12172)*(2.5*2.5*a+2.5*b+c-2.12172)
+(3.0*3.0*a+3.0*b+c-3.22933)*(3.0*3.0*a+3.0*b+c-3.22933)
+(3.5*3.5*a+3.5*b+c-4.44744)*(3.5*3.5*a+3.5*b+c-4.44744)
+(4.0*4.0*a+4.0*b+c-6.13509)*(4.0*4.0*a+4.0*b+c-6.13509)
+(4.5*4.5*a+4.5*b+c-8.14697)*(4.5*4.5*a+4.5*b+c-8.14697)
+(5.0*5.0*a+5.0*b+c-10.50759)*(5.0*5.0*a+5.0*b+c-10.50759);
//求解し結果を表示する
solve();
a.val.print();
b.val.print();
c.val.print();
```

これでは，何回も実験を行うような場合モデルファイルの修正に大変な手間がかかってしまい効率的ではありません．そこで，SIMPLE の特長を生かし様々な測定結果に対応できるように定式化から見直していきます．

まず，ある計測時刻 t でのデータ値 $y(t)$ とモデルから得られる値 $f(t)$ との誤差を $e(t)$ と表すことにします．すると，先ほどの議論から $e(t) = at^2 + bt + c - y(t)$ となります．これより，目的関数は $e(t)^2$ を t について和をとったものと言えます．SIMPLE においては，Expression を用いることで数式を宣言できます．よって，$e(t)$ を Expression で記述することでモデルファイルをより簡潔なものにできます．また，二乗和を求める部分についても観測時間の集合を導入することで sum() を用い簡単に記述できます．最後に，各計測時刻 t における $y(t)$ の値は直接モデルファイルに記述するのではなく csv ファイルから与えることにします．

以上のことから，次のように定式化しなおすことができます．

集合	$ObserveTime = \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0\}$	観測時間の集合
変数	a	2 次の項の係数
	b	1 次の項の係数
	c	定数項
定数	$y(t), t \in ObserveTime$	時刻 t における観測値
目的関数（最小化）	$\sum_{t \in ObserveTime} e(t)^2 = \sum_{t \in ObserveTime} (at^2 + bt + c - y(t))^2$	各観測時刻における誤差の二乗和

この結果を SIMPLE で記述すると次のようになり，先ほどのものに比べ汎用性が高くさらに見やすいものになっています.

```
//観測時間の集合の宣言
Set ObserveTime;
Element t(set=ObserveTime);
//観測値をパラメータとして宣言
Parameter y(index=t);
//変数の宣言
Variable a(name="２次の項の係数");
Variable b(name="１次の項の係数");
Variable c(name="定数項");
//誤差を Expression を用いて表現する
Expression e(index=t);
e[t]=a*t*t+b*t+c-y[t];
//誤差の二乗和の最小化
Objective err(type=minimize);
err=sum(e[t]*e[t],t);
//求解して結果を出力する
solve();
a.val.print();
b.val.print();
c.val.print();
```

なお，実行させる際には次のような csv ファイルを与えます.

```
t,y
0.5,1.22470
1.0,0.87334
1.5,0.99577
2.0,1.34215
2.5,2.12172
3.0,3.22933
3.5,4.44744
4.0,6.13509
4.5,8.14697
5.0,10.50759
```

このモデルを実行すると，最後に

```
２次の項の係数=0.644402
１次の項の係数=-1.47656
定数項=1.76057
```

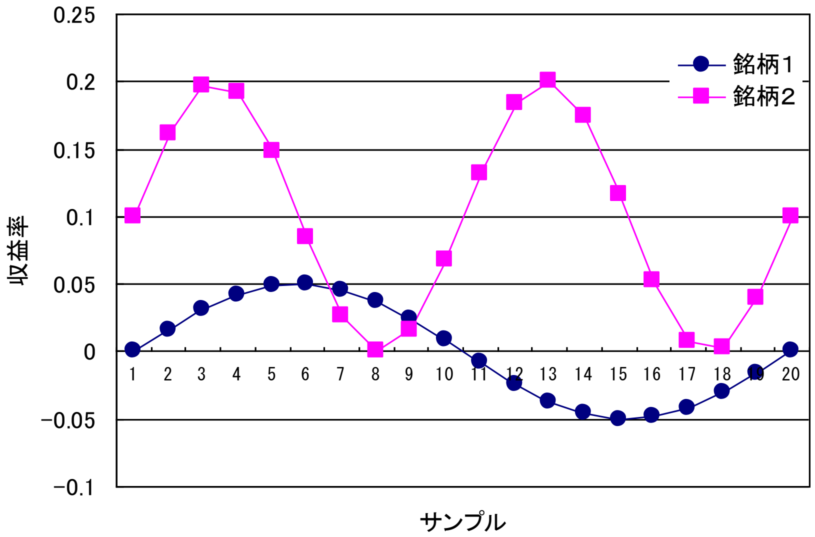
という表示がされます．このことから， $f(t) = 0.644402t^2 \text{-} 1.47656t + 1.76057$ という二次関数モデルが推定できたことになります．

2.17 ポートフォリオ最適化問題

ここでは,ポートフォリオが与える収益率の分布を評価する統計量として平均と分散を用いるマルコビッツモデルの例を示します.具体的にはポートフォリオのもたらす収益率の変動の大きさ（リスク）を分散で計測することにし，それを最小化するような資産配分を求めます.

（例題）

銘柄 1,2 に対する収益率のサンプル 20 期分が以下の図のように得られているとします.



この収益率のサンプルを用いてポートフォリオの収益率の分散が最小となる投資配分を求めます．ただし，空売りはできないものとし，またポートフォリオの収益率の期待値は 0 以上とします.

この問題を定式化すると以下のようになります.

集合	$Asset = \{1, 2\}$	銘柄の集合
	$Sample = \{1, 2, 3, \ldots, 20\}$	サンプルの集合
定数	$r_{tj}, t \in Sample, j \in Asset$	サンプル t における銘柄 j の収益率
	$\overline{r}_j = \sum_{t \in Sample} r_{tj} / \lvert Sample \rvert$ $, j \in Asset$	銘柄 j の平均収益率
変数	$x_j, j \in Asset$	銘柄 j の組入比率
目的関数（最小化）	$\sum_{t \in Sample} dev_t^2 / \lvert Sample \rvert$	ポートフォリオが与える期待収益率の分散（リスク）
	$dev_t = \sum_{j \in Asset} r_{tj} x_j - \overline{rp}$ $, \forall t \in Sample$	サンプル t における平均からのぶれ
	$\overline{rp} = \sum_{j \in Asset} \overline{r}_j x_j$	ポートフォリオの期待収益率
制約条件	$x_j \geq 0, \forall j \in Asset$	非負制約（空売り禁止）
	$\sum_{j \in Asset} x_j = 1$	組入比率の総和は 1
	$\overline{rp} \geq 0$	ポートフォリオが与える期待収益率は 0 以上

この問題は目的関数が二次であり制約式は全て線形ですので二次計画問題になります.

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 集合と添字
Set Asset(name="銘柄");
Element j(set=Asset);
Set Sample(name="サンプル");
Element t(set=Sample);

// パラメータ
Parameter r(name="収益率", index=(t,j));
Parameter rb(name="平均収益率", index=j);
rb[j] = sum(r[t,j],t)/Sample.card();// Sample.card()：集合 Sample の
要素数

// 変数
Variable x(name="組入比率", index=j);

// 式
Expression rpb(name="期待収益率");
rpb = sum(rb[j]*x[j],j);
Expression dev(name="偏差", index=t);
dev[t] = sum(r[t,j]*x[j],j)-rpb;

// 目的関数
Objective V(name="リスク");
V = sum(dev[t]*dev[t],t)/Sample.card();

// 制約
x[j] >= 0.0;
sum(x[j],j) == 1.0;
rpb >= 0.0;

// 求解
solve();

// 出力
V.val.print();
rpb.val.print();
x.val.print();
```

データファイル（.dat 形式）は以下のようになります.

```
収益率 =
[ 1,1]  0.0000 [ 1,2] 0.100
[ 2,1]  0.0162 [ 2,2] 0.161
[ 3,1]  0.0307 [ 3,2] 0.197
[ 4,1]  0.0419 [ 4,2] 0.192
[ 5,1]  0.0485 [ 5,2] 0.148
[ 6,1]  0.0498 [ 6,2] 0.084
[ 7,1]  0.0458 [ 7,2] 0.026
[ 8,1]  0.0368 [ 8,2] 0.000
[ 9,1]  0.0238 [ 9,2] 0.016
[10,1]  0.0082 [10,2] 0.068
[11,1] -0.0082 [11,2] 0.132
[12,1] -0.0238 [12,2] 0.184
[13,1] -0.0368 [13,2] 0.200
[14,1] -0.0458 [14,2] 0.174
[15,1] -0.0498 [15,2] 0.116
[16,1] -0.0485 [16,2] 0.052
[17,1] -0.0419 [17,2] 0.008
[18,1] -0.0307 [18,2] 0.003
[19,1] -0.0162 [19,2] 0.039
[20,1]  0.0000 [20,2] 0.100
;
```

このモデルを実行すると以下のような解が得られます.

```
リスク=0.000951734
期待収益率=0.0199126
組入比率[1]=0.800874
組入比率[2]=0.199126
```

Vector/Matrix を使用した SIMPLE モデル

NUOPT V13 より導入された，Vector/Matrix クラスの記述を使ったモデルを紹介します.

```
Set Asset(name=”銘柄”), Sample(name=”サンプル”);
Element j(set=Asset), t(set=Sample);

// パラメータ行列，ベクトル
Matrix R(name=”収益率”,(Sample,Asset)),Rbar((Sample,Asset));
Vector r(Asset);
r[j] = inprod(ones(Sample),R.col(j))/Sample.card();
Rbar[t,j] = R[t,j] − r[j];

// 変数，変数ベクトル
Variable x(name=”組入比率”,index=j);
Vector xv(Asset),
dev(Sample);// 偏差ベクトル
xv[j] = x[j];// 変数 x をベクトル xv に詰め込む
dev = Rbar*xv;

Expression rpb(name=”期待収益率”);
rpb = inprod(r,xv);

Objective V(name=”リスク”,type=minimize);
V = inprod(dev,dev)/Sample.card();

xv >= 0.0;
inprod(ones(Asset),xv) == 1.0;
rpb >= 0.0;

solve();

V.val.print();
rpb.val.print();
x.val.print();
```

実行時には，同じデータファイルを与えます.

2.18 ロジスティック回帰モデル

本章では，ロジスティック回帰モデルによるパラメータ推定方法を，アヤメの識別問題を例に紹介します.

（例題）
同じアヤメ科の Iris setosa と Iris versicolor に関して，がくの長さ，がくの幅，花弁の長さ，花弁の幅に関する各々50 個のデータが存在する．これらの 4 つを説明変数として，Iris setosa と Iris versicolor を判別するようなロジスティック回帰モデルを構築せよ.

この問題の定式化は次のようになります.

集合	I	サンプル集合
	J	説明変数集合
定数	$d_{i,j}, \quad i \in I, j \in J$	計測データ
	$k_i, \quad i \in I$	アヤメの種類 （0:Iris setosa, 1:Iris versicolor）
変数	$a0$ $a_j, \quad j \in J$	推定したいパラメータ
	$x_i, \quad i \in I$ （中間変数） $\left(x_i = a0 + \sum_j a_j d_{ij} \right)$	
目的関数（最大化）	$\sum_i \left(k_i x_i - \log\left(1 + \exp\left(x_i\right)\right)\right)$	対数尤度

サンプル集合の要素$i \in I$に対し，ロジスティック関数$L(x_i)$は，以下のような式で表わすことができます.

$$L(x_i) = \frac{\exp(x_i)}{1+\exp(x_i)}$$

ただしx_iは，$x_i = a0 + \sum_j a_j d_{ij}$ で表わされるものとします.

次に，ロジスティック関数を用いた $a0, a_j$ のパラメータ推定方法の説明を行います．対象データ全てに対する尤もらしさを考え，それが最大となるような $a0, a_j$ を求めます．即ち，目的関数fを

$$f = \prod_i L(x_i)^{k_i}\left(1 - L(x_i)\right)^{1-k_i}$$

で定義し，これを最大化する問題を考えることにより，$a0, a_j$ の推定を行います.

なお，このままでは目的関数の形状が複雑なため，目的関数fに対して対数をとった$\tilde{f}$

$$\begin{aligned}
\tilde{f} &= \log f \\
&= \sum_i \left(k_i \log L(x_i) + (1-k_i)\log\left(1-L(x_i)\right)\right) \\
&= \sum_i \left(k_i \log \frac{\exp(x_i)}{1+\exp(x_i)} + (1-k_i)\log\frac{1}{1+\exp(x_i)}\right) \\
&= \sum_i \left(k_i \log\left(\exp(x_i)\right) + \log\frac{1}{1+\exp(x_i)}\right) \\
&= \sum_i \left(k_i x_i - \log\left(1+\exp(x_i)\right)\right)
\end{aligned}$$

で目的関数を置きなおしても一般性は失われない性質を利用して,対象となる問題の置き換えを行います.

以上が定式化の説明となります.

次に，アヤメの計測データですが，これは [5] の文献にあるデータ（Iris setosa，Iris versicolor，各々50 個）を使用します.

具体的には，以下のような 2 種類の csv ファイルを用意します.

（csv 形式ファイル；1 種類目）

```
計測データ,がくの長さ,がくの幅,花弁の長さ,花弁の幅
1,5.1,3.5,1.4,0.2
2,4.9,3,1.4,0.2
3,4.7,3.2,1.3,0.2
4,4.6,3.1,1.5,0.2
5,5,3.6,1.4,0.2
6,5.4,3.9,1.7,0.4
7,4.6,3.4,1.4,0.3
8,5,3.4,1.5,0.2
9,4.4,2.9,1.4,0.2
10,4.9,3.1,1.5,0.1
11,5.4,3.7,1.5,0.2
12,4.8,3.4,1.6,0.2
13,4.8,3,1.4,0.1
14,4.3,3,1.1,0.1
15,5.8,4,1.2,0.2
16,5.7,4.4,1.5,0.4
17,5.4,3.9,1.3,0.4
18,5.1,3.5,1.4,0.3
19,5.7,3.8,1.7,0.3
20,5.1,3.8,1.5,0.3
21,5.4,3.4,1.7,0.2
22,5.1,3.7,1.5,0.4
23,4.6,3.6,1,0.2
24,5.1,3.3,1.7,0.5
25,4.8,3.4,1.9,0.2
26,5,3,1.6,0.2
27,5,3.4,1.6,0.4
28,5.2,3.5,1.5,0.2
29,5.2,3.4,1.4,0.2
30,4.7,3.2,1.6,0.2
31,4.8,3.1,1.6,0.2
32,5.4,3.4,1.5,0.4
33,5.2,4.1,1.5,0.1
```

（左段の続き）

```
34,5.5,4.2,1.4,0.2
35,4.9,3.1,1.5,0.2
36,5,3.2,1.2,0.2
37,5.5,3.5,1.3,0.2
38,4.9,3.6,1.4,0.1
39,4.4,3,1.3,0.2
40,5.1,3.4,1.5,0.2
41,5,3.5,1.3,0.3
42,4.5,2.3,1.3,0.3
43,4.4,3.2,1.3,0.2
44,5,3.5,1.6,0.6
45,5.1,3.8,1.9,0.4
46,4.8,3,1.4,0.3
47,5.1,3.8,1.6,0.2
48,4.6,3.2,1.4,0.2
49,5.3,3.7,1.5,0.2
50,5,3.3,1.4,0.2
51,7,3.2,4.7,1.4
52,6.4,3.2,4.5,1.5
53,6.9,3.1,4.9,1.5
54,5.5,2.3,4,1.3
55,6.5,2.8,4.6,1.5
56,5.7,2.8,4.5,1.3
57,6.3,3.3,4.7,1.6
58,4.9,2.4,3.3,1
59,6.6,2.9,4.6,1.3
60,5.2,2.7,3.9,1.4
61,5,2,3.5,1
62,5.9,3,4.2,1.5
63,6,2.2,4,1
64,6.1,2.9,4.7,1.4
65,5.6,2.9,3.6,1.3
66,6.7,3.1,4.4,1.4
67,5.6,3,4.5,1.5
68,5.8,2.7,4.1,1
69,6.2,2.2,4.5,1.5
```

（中段の続き）

```
70,5.6,2.5,3.9,1.1
71,5.9,3.2,4.8,1.8
72,6.1,2.8,4,1.3
73,6.3,2.5,4.9,1.5
74,6.1,2.8,4.7,1.2
75,6.4,2.9,4.3,1.3
76,6.6,3,4.4,1.4
77,6.8,2.8,4.8,1.4
78,6.7,3,5,1.7
79,6,2.9,4.5,1.5
80,5.7,2.6,3.5,1
81,5.5,2.4,3.8,1.1
82,5.5,2.4,3.7,1
83,5.8,2.7,3.9,1.2
84,6,2.7,5.1,1.6
85,5.4,3,4.5,1.5
86,6,3.4,4.5,1.6
87,6.7,3.1,4.7,1.5
88,6.3,2.3,4.4,1.3
89,5.6,3,4.1,1.3
90,5.5,2.5,4,1.3
91,5.5,2.6,4.4,1.2
92,6.1,3,4.6,1.4
93,5.8,2.6,4,1.2
94,5,2.3,3.3,1
95,5.6,2.7,4.2,1.3
96,5.7,3,4.2,1.2
97,5.7,2.9,4.2,1.3
98,6.2,2.9,4.3,1.3
99,5.1,2.5,3,1.1
100,5.7,2.8,4.1,1.3
```

（csv 形式ファイル;2 種類目）

```
i,種類
1,1
2,1
3,1
4,1
5,1
6,1
7,1
8,1
9,1
10,1
11,1
12,1
13,1
14,1
15,1
16,1
17,1
18,1
19,1
20,1
21,1
22,1
23,1
24,1
25,1
26,1
27,1
28,1
29,1
30,1
31,1
32,1
33,1
```

（左段の続き）

```
34,1
35,1
36,1
37,1
38,1
39,1
40,1
41,1
42,1
43,1
44,1
45,1
46,1
47,1
48,1
49,1
50,1
51,0
52,0
53,0
54,0
55,0
56,0
57,0
58,0
59,0
60,0
61,0
62,0
63,0
64,0
65,0
66,0
67,0
```

（中段の続き）

```
68,0
69,0
70,0
71,0
72,0
73,0
74,0
75,0
76,0
77,0
78,0
79,0
80,0
81,0
82,0
83,0
84,0
85,0
86,0
87,0
88,0
89,0
90,0
91,0
92,0
93,0
94,0
95,0
96,0
97,0
98,0
99,0
100,0
```

以上をもとに SIMPLE で記述すると，以下のようになります．

```
// サンプル集合
Set I(name="サンプル集合"); Element i(set=I);
// 説明変数集合
Set J(name="説明変数集合"); Element j(set=J);
// データ
Parameter d(name="計測データ", index=(i,j));
// アヤメの種類
// 0 : Iris setosa, 1 : Iris versicolor
Parameter k(name="種類", index=i);
// 推定したいパラメータ
Variable a(name="a", index=j);
Variable a0(name="a0");
// 中間変数
Variable x(name="x", index=i);
x[i] == a0 + sum(a[j] * d[i,j], j);

// 対数尤度（目的関数）
Objective f(name="対数尤度", type=maximize);
f = sum(k[i]*x[i] - log(1+exp(x[i])), i);

//求解
solve();

// 結果の標準出力
a0.val.print();
a.val.print();
```

このモデルを実行すると，以下のような解が得られます．

```
a0=-3.80539
a["がくの長さ"]=6.94731
a["がくの幅"]=6.08239
a["花弁の長さ"]=-14.0887
a["花弁の幅"]=-19.5508
```

2.19 イールドカーブ推定問題

イールドカーブとは，償還期間の異なる利回りをもつ債権等について，利回りと償還期間の相関を描いた曲線（縦軸：利回り，横軸：償還期間）のことをいいます．ここでは，この利回りがスポットレート（現時点から将来のある時点まで保有される資産にかかる金利のこと）である場合を対象とします．なお，イールドカーブの形状には順イールド（右上がりの曲線）と逆イールド（右下がりの曲線）があり，前者は短期金利よりも長期金利のほうが高くなることを意味し，後者は逆に短期金利よりも長期金利のほうが低くなることを意味しています．

以下の例題では，実際の観測点からスポットレートを推定する問題を考え，ひいては，イールドカーブの推定を行います．

（例題）

期間（償還期間）が 1〜10 期まであり，各々に対するスポットレート r_t を用いて，関数 S が

$$S(t;r_1,\ldots,r_t) \equiv 1/\sum_{i=1}^{t} d_i\ ,\ d_i \equiv 1/(1+0.01\cdot r_i)^i$$

のように定義されているとする．観測結果 $(t,S(t))$ の組

$$(t_i,S_i)\quad i\in\{1,\ldots,60\}$$

が与えられているとき，

$$\sum_i \left\{S(t_i)-S_i\right\}^2$$

を最小化するようなスポットレートを推定せよ．

この問題を定式化すると以下のようになります．
なお，求めるスポットレートの単位は%（パーセント）とし，非負であるものとします．

集合	$Term$	期間（償還期間）集合
	$Point$	観測点の集合
定数	$tvalue_i, \quad i \in Point$	観測点が何期目か
	$Svalue_i, \quad i \in Point$	観測値
変数	$r_t, \quad t \in Term$	t 期に対するスポットレート
目的関数（最小化）	$\sum_i \left\{S(tvalue_i) - Svalue_i\right\}^2,$ $S(t; r_1, \ldots, r_t) \equiv$ $1/\sum_{i=1}^{t}(1/(1+0.01 \cdot r_i)^i)$	理論値と観測値の誤差の二乗和
制約条件	$0 \le r_t, \quad \forall t \in Term$	スポットレートの非負条件

この問題は目的関数が非線形ですので，非線形計画問題になります．

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// スポットレートの設定
Set Term(name=”期間”); // 期間集合
Term = "1 .. 10";
Element t(set=Term);
Variable r(name="スポットレート", index=t);

// 観測点
Set Point;
Element i(set=Point);
Parameter tvalue(name="tvalue", index=i); // 何期目か
Parameter Svalue(name="Svalue", index=i); // 観測値(S)

// 関数設定
Expression d(index=t);
d[t] = 1 / pow(1+0.01*r[t], t);
// tvalue[i] 期における S の理論値
Expression S(index=i);
S[i] = 1 / sum(d[t], (t, t<=tvalue[i])); // tvalue[i] 期までの和
Expression diff(index=i);
diff[i] = S[i] - Svalue[i];

// 目的関数
Objective err(name="理論値と観測値の誤差の二乗和", type=minimize);
err = sum(pow(diff[i], 2), i);

// 制約条件
0 <= r[t]; // スポットレートの非負条件

// 求解
solve();

// 結果の標準出力
err.val.print();
simple_printf("¥n");
r.val.print();
```

データファイル（csv 形式）は以下のようになります.

```
i,tvalue,Svalue
1,1,1.020476
2,1,1.020582
3,1,1.021692
4,1,1.020304
5,1,1.021171
6,1,1.020387
7,2,0.520158
8,2,0.518884
9,2,0.519507
10,2,0.518841
11,2,0.519271
12,2,0.519520
13,3,0.355121
14,3,0.353832
15,3,0.355088
16,3,0.353742
17,3,0.354565
18,3,0.354443
19,4,0.271774
20,4,0.271318
```

（左段の続き）

```
21,4,0.272115
22,4,0.272244
23,4,0.272818
24,4,0.271190
25,5,0.222502
26,5,0.223564
27,5,0.222312
28,5,0.223726
29,5,0.222489
30,5,0.223251
31,6,0.190145
32,6,0.190989
33,6,0.191429
34,6,0.191156
35,6,0.191077
36,6,0.191451
37,7,0.168971
38,7,0.167869
39,7,0.168913
40,7,0.168672
```

（中段の続き）

```
41,7,0.167762
42,7,0.167502
43,8,0.152239
44,8,0.150571
45,8,0.151664
46,8,0.151251
47,8,0.152121
48,8,0.151681
49,9,0.138188
50,9,0.138396
51,9,0.139097
52,9,0.139389
53,9,0.138134
54,9,0.138999
55,10,0.128923
56,10,0.129079
57,10,0.129807
58,10,0.129427
59,10,0.128610
60,10,0.129465
```

このモデルを実行すると，以下のような解が得られます．

```
理論値と観測値の誤差の二乗和=1.74889e-05

スポットレート[1]=2.07687
スポットレート[2]=2.82648
スポットレート[3]=3.7392
スポットレート[4]=3.94718
スポットレート[5]=4.37854
スポットレート[6]=4.91763
スポットレート[7]=5.05042
スポットレート[8]=5.5363
スポットレート[9]=5.76144
スポットレート[10]=5.74989
```

この節の最後に，実際に推定したスポットレートをもとに，Excel 上でイールドカーブを描きます．そのために，スポットレートを Excel 連係機能により，Excel 上に表示します．

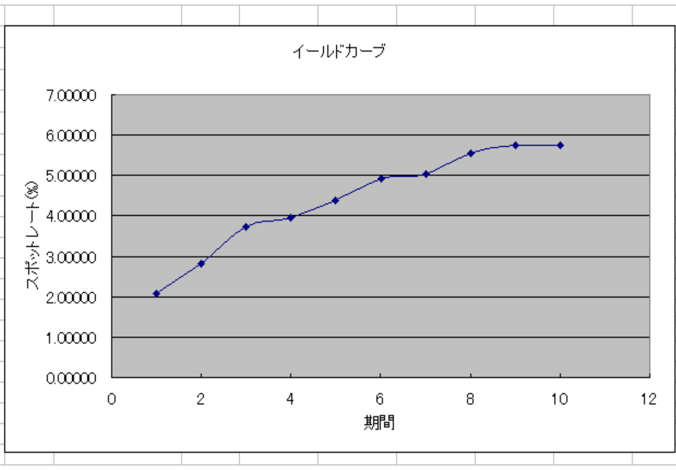
モデルの結果出力の部分を以下のように変更する必要があります．

```
// 結果出力
err.val.dump();
r.val.dump();
```

結果は Excel 上に，例えば下記のように表示されます．

理論値と観測値の誤差の二乗和
1.7489E-05

期間	スポットレート
1	2.07687
2	2.82648
3	3.73920
4	3.94718
5	4.37854
6	4.91763
7	5.05042
8	5.53630
9	5.76144
10	5.74989



2.20 格付け推移行列推定問題

格付け（rating）とは，企業の発行する社債の元本，利息の支払い能力をランク形式で表示したものです．格付け会社は独自の調査結果のもと，A，B，C や ＋，－ などの記号を用いて対象社債のリスク度合いを示します．格付けは，社債の購入者側からすると購入判断時の評価指標になりますし，発行体となる企業側からすると，資金調達の際の利回り決定の基準となります．

格付け推移行列とは，ある格付け評価を受けている企業が，一定期間後にどのような格付けとなるかについての確率を表す行列のことをいいます．

以下では，この格付け推移行列の推定に関する例題を考えます．

（例題）

格付けとして，{AAA，AA，A，BBB，BB，B，CCC，CC，C} の 9 種類があるものとして，格付けに関する 1 年後の推移行列 Q_0 が以下のように与えられているものとする．

		推移後の格付け								
	Q0	AAA	AA	A	BBB	BB	B	CCC	CC	C
推移前の格付け	AAA	0.9651	0.0349	0	0	0	0	0	0	0
	AA	0.0356	0.9382	0.0262	0	0	0	0	0	0
	A	0.0014	0.0433	0.9364	0.0186	0.0003	0	0	0	0
	BBB	0	0	0.0352	0.9456	0.0154	0.0038	0	0	0
	BB	0	0	0	0.1078	0.872	0.0049	0.0153	0	0
	B	0	0	0	0	0.0747	0.8762	0.041	0.0081	0
	CCC	0	0	0	0	0	0.0278	0.9654	0.0068	0
	CC	0	0	0	0	0	0	0.0083	0.9675	0.0242
	C	0	0	0	0	0	0	0	0.0266	0.9734

Q_0 を基に，1 ヶ月単位期間の格付け推移行列 Q を推定せよ．

推移確率行列 Q はマルコフ過程に従うものとします．そのとき，題意より行列 Q^{12} は行列 Q_0 と本来一致するはずです．従って，行列 Q を推定するとは，$\left\|Q_0 - Q^{12}\right\|_F$ を最小にするような Q を求めるということになります．なお $\|\bullet\|_F$ は，Frobenius（フロベニウス）ノルムのことで，行列の各成分の二乗和の平方根を表すものとします．

また，格付け推移行列の性質として，行列 Q の各行の和は 1 で各成分は非負であるものとします．

以上を踏まえると，定式化は以下のようになります．

順序集合	$Rating$	格付けの集合
定数	${q^0}_{ij}, \quad i, j \in Rating$	1 年後の格付け推移行列（Q_0）の各要素
変数	$q_{ij}, \ i, j \in Rating$	1 ヶ月単位の格付け推移行列（Q）の各要素
目的関数（最小化）	$\left\|Q_0 - Q^{12}\right\|_F^2$	行列 Q_0 と Q^{12} の差のフロベニウスノルムの二乗
制約条件	$\sum_{j \in Rating} q_{ij} = 1, \forall i \in Rating$	行列 Q の各行の和は 1
	$0 \le q_{ij} \le 1, \forall i, j \in Rating$	行列 Q の各要素条件

この問題は目的関数が非線形ですので，非線形計画問題になります．

定式化した結果を SIMPLE で記述すると以下のようになります.

なお，行列 Q^{12} に関しては,

$$Q^{12} = Q^8 \cdot Q^4 = (Q^2 \cdot Q^2)^2 \cdot (Q^2 \cdot Q^2)$$

で表現することにより，演算の回数を節約することができます.

また, 局所解に陥ることを回避するため, および収束までの反復回数を軽減するため, 行列 Q の初期値として，対角成分に 0.9 を与え，非対角要素の上限値を 0.05 としておきます.

```
// 集合・要素・変数・パラメータ
OrderedSet Rating; // 格付け集合
Element i(set=Rating), j(set=Rating), k(set=Rating);
Variable q(name="Q", index=(i,j)); // 1ヶ月単位の格付け推移行列の各要素
Parameter q0(name="Q0", index=(i,j)); // 1年後の格付け推移行列の各要素
// 定式の定義に利用する演算の回数を節約する記法
Expression q2(name="q2", index=(i,j));
Expression q4(name="q4", index=(i,j));
Expression q8(name="q8", index=(i,j));
Expression q12(name="q12", index=(i,j));
q2[i,j] = sum(q[i,k]*q[k,j], k);
q4[i,j] = sum(q2[i,k]*q2[k,j], k);
q8[i,j] = sum(q4[i,k]*q4[k,j], k);
q12[i,j] = sum(q8[i,k]*q4[k,j], k);
Expression diff(name="diff", index=(i,j));
diff[i,j] = q0[i,j] - q12[i,j];
// 目的関数
Objective diffnrm(name="行列 Q0 と 行列 Q の 12 乗 の差ノルムの二乗",
type=minimize);
diffnrm = sum(pow(diff[i,j], 2), (i,j));
// 制約条件
sum(q[i,j], j) == 1; // 行列 Q について，各行の和は1
0 <= q[i,i] <= 1; // 行列 Q の対角要素条件
0 <= q[i,j] <= 0.05, i != j; // 行列 Q の非対角要素条件
// 初期値設定
q[k,k]=0.9;
// 求解
solve();
```

```
// 結果出力
diffnrm.val.print();
simple_printf("¥n");
simple_printf("Q");
simple_printf(",%s", k);
simple_printf("¥n");
for( k=Rating.first(); k<Rating; k=Rating.next(k) ){
  simple_printf("%s", k);
  simple_printf(",%7.5f", q[k,j]);
  simple_printf("¥n");
}
```

データファイル（csv 形式）は以下のようになります.

```
Q0,AAA,AA,A,BBB,BB,B,CCC,CC,C
AAA,0.9651,0.0349,0,0,0,0,0,0,0
AA,0.0356,0.9382,0.0262,0,0,0,0,0,0
A,0.0014,0.0433,0.9364,0.0186,0.0003,0,0,0,0
BBB,0,0,0.0352,0.9456,0.0154,0.0038,0,0,0
BB,0,0,0,0.1078,0.872,0.0049,0.0153,0,0
B,0,0,0,0,0.0747,0.8762,0.041,0.0081,0
CCC,0,0,0,0,0,0.0278,0.9654,0.0068,0
CC,0,0,0,0,0,0,0.0083,0.9675,0.0242
C,0,0,0,0,0,0,0,0.0266,0.9734
```

このモデルを実行すると以下のような解が得られます.

```
行列 Q0 と 行列 Q の 12 乗 の差ノルムの二乗=0.000154701

Q,"AAA","AA","A","BBB","BB","B","CCC","CC","C"
"AAA",0.99680,0.00285,0.00006,0.00004,0.00007,0.00006,0.00004,0.00004,0.00004
"AA",0.00298,0.99447,0.00218,0.00006,0.00008,0.00008,0.00005,0.00005,0.00005
"A",0.00007,0.00371,0.99435,0.00152,0.00010,0.00009,0.00006,0.00006,0.00006
"BBB",0.00007,0.00006,0.00289,0.99505,0.00158,0.00013,0.00006,0.00009,0.00007
"BB",0.00008,0.00008,0.00005,0.00966,0.98839,0.00033,0.00123,0.00010,0.00008
"B",0.00008,0.00006,0.00008,0.00003,0.00684,0.98880,0.00348,0.00056,0.00008
"CCC",0.00005,0.00005,0.00006,0.00005,0.00006,0.00235,0.99688,0.00046,0.00005
"CC",0.00005,0.00005,0.00006,0.00006,0.00008,0.00008,0.00059,0.99708,0.00194
"C",0.00004,0.00004,0.00004,0.00005,0.00007,0.00007,0.00004,0.00210,0.99754
```

下記では，このモデルを Excel 連係機能を用いて Excel 上に表示させます．そのために，モデルの結果出力の部分を以下のように変更する必要があります．

```
// 結果出力
diffnrm.val.dump();
q.val.dump();
```

Excel 連係の詳細につきましては，「NUOPT Excel 連係マニュアル」をご覧下さい．結果は Excel 上に，例えば下記のように表示されます．

	行列 Q0 と行列 Q の 12 乗 の差ノルムの二乗									
	0.000154701									
		推移後の格付け								
	Q	AAA	AA	A	BBB	BB	B	CCC	CC	C
推移前の格付け	AAA	0.99680	0.00285	0.00006	0.00004	0.00007	0.00006	0.00004	0.00004	0.00004
	AA	0.00298	0.99447	0.00218	0.00006	0.00008	0.00008	0.00005	0.00005	0.00005
	A	0.00007	0.00371	0.99435	0.00152	0.00010	0.00009	0.00006	0.00006	0.00006
	BBB	0.00007	0.00006	0.00289	0.99505	0.00158	0.00013	0.00006	0.00009	0.00007
	BB	0.00008	0.00008	0.00005	0.00966	0.98839	0.00033	0.00123	0.00010	0.00008
	B	0.00008	0.00006	0.00008	0.00003	0.00684	0.98880	0.00348	0.00056	0.00008
	CCC	0.00005	0.00005	0.00006	0.00005	0.00006	0.00235	0.99688	0.00046	0.00005
	CC	0.00005	0.00005	0.00006	0.00006	0.00008	0.00008	0.00059	0.99708	0.00194
	C	0.00004	0.00004	0.00004	0.00005	0.00007	0.00007	0.00004	0.00210	0.99754

Vector/Matrix を用いた SIMPLE モデル

NUOPT V13 より導入された，Vector/Matrix クラスを用いたモデルを紹介します.

```
Set Rating;
Element i(set=Rating), j(set=Rating);

Matrix Q0((i,j));
Variable q(name="Q", index=(i,j));
Matrix Q((i,j)); Q[i,j] = q[i,j];

Matrix Q2((i,j)), Q4((i,j)), Q8((i,j)), Q12((i,j));
Q2 = Q*Q;
Q4 = Q2*Q2;
Q8 = Q4*Q4;
Q12 = Q8*Q4;

Matrix D((i,j));
D = Q0 - Q12;

Objective diffnrm(type=minimize);
diffnrm = inprod(D,D);

Q*ones(Rating) == ones(Rating);
0.00 <= q[i,i] <= 1;
0 <= q[i,j] <= 0.05 , i!=j;

q[i,i]=0.9;

solve();
```

実行時には，同じデータファイルを与えます.

2.21 相関行列取得問題

複数の株価の時系列データ等，相関を持った乱数列を発生させる際に，元の相関行列内の要素をある程度任意に変更したい場合があります．しかし，相関を持った乱数列を適切に発生させるためには，相関行列は半正定値である必要があります．そこで，与えられた行列に一番近く，かつ，半正定値であるような相関行列を作成する問題を考えます．

（例題）

以下の対称行列Aに対してフロベニウスノルムの意味で一番近く，かつ，半正定値であるような相関行列Xを求めよ．

A	1	2	3	4	5	6	7	8	9	10
1	1.00	0.17	0.17	0.90	0.90	0.00	0.10	0.00	0.80	0.70
2	0.17	1.00	0.10	0.90	0.20	0.40	0.00	0.00	0.00	0.00
3	0.17	0.10	1.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00
4	0.90	0.90	0.20	1.00	0.20	0.20	0.20	0.20	0.00	0.00
5	0.90	0.20	0.00	0.20	1.00	0.40	0.00	0.00	0.00	0.00
6	0.00	0.40	0.00	0.20	0.40	1.00	0.00	0.00	0.00	0.00
7	0.10	0.00	0.00	0.20	0.00	0.00	1.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.20	0.00	0.00	0.00	1.00	0.00	0.00
9	0.80	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
10	0.70	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

この問題を定式化すると以下のようになります.

集合	$N = \{1, 2, \ldots, 10\}$	相関行列の行及び列の集合
定数	$A_{ij}, i \in N, j \in N$	所与の行列の要素
	$minEig$	相関行列の最小固有値
変数	$X_{ij}, i \in N, j \in N$	相関行列の要素
目的関数（最小化）	$\sum_{i,j \in N} \left(X_{ij} - A_{ij} \right)^2$	元の行列と相関行列の差のフロベニウスノルムの二乗
制約条件	$X \succeq minEig$	半正定値制約
	$X_{ii} = 1, \quad \forall i \in N$	対角要素は 1

この問題は半正定値制約が入っているので，半正定値計画問題になります.

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 集合と添字
OrderedSet N;
Element i(set=N),j(set=N);

// パラメータ
Parameter A(index=(i,j)); // 与えられた行列の要素
Parameter minEig;        // 出力される相関行列の最小固有値
minEig = 1.0e-3;

// 変数
Variable  X(index=(i,j));

// 対称行列
SymmetricMatrix M((i,j));
M[i,j] = X[i,j], i <= j; // 上三角部分のみ定義

// 目的関数
Objective diffnrm(type=minimize); // 差の行列のノルム
diffnrm = sum((X[i,j]-A[i,j])*(X[i,j]-A[i,j]),(i,j));

// 制約条件
M >= minEig;             // 半正定値制約
X[j,i] == X[i,j], i < j; // X は対称行列
X[i,i] == 1;             // 対角要素は 1

// 求解
solve();

// 出力（CSV 形式）
simple_printf(" X");
simple_printf(",%5d", i);
simple_printf("¥n");
for(i=N.first(); i<N; i=N.next(i)){
  simple_printf("%2d", i);
  simple_printf(",%5.2f", X[i,j]);
  simple_printf("¥n");
}
```

データファイル（.csv 形式）は以下のようになります.

```
A,1,2,3,4,5,6,7,8,9,10
1,1.00 ,0.17 ,0.17 ,0.90 ,0.90 ,0.00 ,0.10 ,0.00 ,0.80 ,0.70
2,0.17 ,1.00 ,0.10 ,0.90 ,0.20 ,0.40 ,0.00 ,0.00 ,0.00 ,0.00
3,0.17 ,0.10 ,1.00 ,0.20 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00
4,0.90 ,0.90 ,0.20 ,1.00 ,0.20 ,0.20 ,0.20 ,0.20 ,0.00 ,0.00
5,0.90 ,0.20 ,0.00 ,0.20 ,1.00 ,0.40 ,0.00 ,0.00 ,0.00 ,0.00
6,0.00 ,0.40 ,0.00 ,0.20 ,0.40 ,1.00 ,0.00 ,0.00 ,0.00 ,0.00
7,0.10 ,0.00 ,0.00 ,0.20 ,0.00 ,0.00 ,1.00 ,0.00 ,0.00 ,0.00
8,0.00 ,0.00 ,0.00 ,0.20 ,0.00 ,0.00 ,0.00 ,1.00 ,0.00 ,0.00
9,0.80 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,1.00 ,0.00
10,0.70 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,0.00 ,1.00
```

このモデルを実行すると以下のような解が得られます.

```
 X,    1,    2,    3,    4,    5,    6,    7,    8,    9,   10
 1, 1.00, 0.27, 0.15, 0.61, 0.65, 0.06, 0.10, 0.03, 0.57, 0.50
 2, 0.27, 1.00, 0.10, 0.85, 0.15, 0.41, 0.00, 0.01,-0.04,-0.04
 3, 0.15, 0.10, 1.00, 0.21, 0.01,-0.00,-0.00,-0.00, 0.01, 0.01
 4, 0.61, 0.85, 0.21, 1.00, 0.34, 0.17, 0.20, 0.18, 0.12, 0.11
 5, 0.65, 0.15, 0.01, 0.34, 1.00, 0.37,-0.00,-0.01, 0.10, 0.09
 6, 0.06, 0.41,-0.00, 0.17, 0.37, 1.00, 0.00, 0.00,-0.02,-0.02
 7, 0.10, 0.00,-0.00, 0.20,-0.00, 0.00, 1.00, 0.00,-0.00,-0.00
 8, 0.03, 0.01,-0.00, 0.18,-0.01, 0.00, 0.00, 1.00,-0.01,-0.01
 9, 0.57,-0.04, 0.01, 0.12, 0.10,-0.02,-0.00,-0.01, 1.00, 0.08
10, 0.50,-0.04, 0.01, 0.11, 0.09,-0.02,-0.00,-0.01, 0.08, 1.00
```

2.22 ロバストポートフォリオ最適化問題

ポートフォリオのリスクを投資対象の収益率の分散共分散行列を用いて計測するマルコビッツモデルにおいて，与えられた分散共分散行列は，しばしば不確実性を伴います．この不確実性に対してロバストな解を得る以下の問題を考えます．

以下の典型的な平均・分散モデルを考えます．

$$\begin{aligned} &\max_{x} \ \mu^T x - \lambda x^T \Sigma x \\ &s.t. \quad x^T e = 1 \end{aligned} \qquad (1)$$

ここで，μ を期待リターン，Σ をリターンの分散共分散行列，x をポートフォリオの重み，λ をリスク回避係数とします．

分散共分散行列 Σ に不確実性が伴うとして，以下のロバストポートフォリオ最適化問題を考えます．なお U_Σ は，不確実性が伴うことによって取りうる Σ に関する行列の集合とします．

$$\begin{aligned} &\max_{x} \left\{ \mu^T x - \lambda \max_{\Sigma \in \mathrm{U}_\Sigma} \left\{ x^T \Sigma x \right\} \right\} \\ &s.t. \quad x^T e = 1 \end{aligned} \qquad (2)$$

U_Σ として，分散共分散行列 Σ の各要素が

$$\underline{\Sigma} \le \Sigma \le \overline{\Sigma}$$

のような区間を持つとします．このとき(2)は，新たに行列 U, L を用いて以下のような問題に置き換えることができます[6]．なお行列 A，B に対し，$A \bullet B$ は，A と B の内積を表すものとします．

$$\begin{aligned} &\max_{x} \mu^T x - \lambda \left(\overline{\Sigma} \bullet U - \underline{\Sigma} \bullet L \right) \\ &s.t. \quad x^T e = 1 \\ &\qquad \begin{pmatrix} U - L & x \\ x^T & 1 \end{pmatrix} \succeq 0 \\ &\qquad U \ge 0, L \ge 0 \end{aligned} \qquad (3)$$

（例題）

分散共分散行列Σの各要素の下限を表す行列$sigL$，上限を表す行列$sigU$が以下のように与えられているとき，上記(3)を解きなさい.

sigL	1	2	3	4	5	6	7	8
1	1.900	−1.400	−1.000	−1.000	−1.000	−0.500	−1.000	0.100
2	−1.400	3.500	−1.500	0.500	−1.200	−1.200	0.400	0.600
3	−1.000	−1.500	5.000	−1.125	1.100	0.900	0.300	−0.200
4	−1.000	0.500	−1.125	6.000	1.500	0.400	1.200	−0.900
5	−1.000	−1.200	1.100	1.500	4.500	−1.400	0.150	−2.000
6	−0.500	−1.200	0.900	0.400	−1.400	9.000	−1.000	0.500
7	−1.000	0.400	0.300	1.200	0.150	−1.000	5.500	−1.250
8	0.100	0.600	−0.200	−0.900	−2.000	0.500	−1.250	11.500

sigU	1	2	3	4	5	6	7	8
1	3.000	1.000	2.500	−0.900	1.000	−0.400	2.500	2.000
2	1.000	4.500	−1.400	1.200	0.500	−1.100	0.500	2.600
3	2.500	−1.400	6.000	−0.500	1.200	1.000	0.400	−0.100
4	−0.900	1.200	−0.500	6.500	1.600	0.500	1.300	−0.800
5	1.000	0.500	1.200	1.600	5.500	−1.300	0.160	−1.900
6	−0.400	−1.100	1.000	0.500	−1.300	11.000	−0.900	0.600
7	2.500	0.500	0.400	1.300	0.160	−0.900	6.500	−1.200
8	2.000	2.600	−0.100	−0.800	−1.900	0.600	−1.200	13.500

ただし，$\lambda=1$，$\mu^T=\left(0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1\right)$とします.

この問題を定式化すると以下のようになります.

集合	$A = \{1, 2, \ldots, 8\}$	投資対象の集合
定数	$sigL_{ij}, i \in A, j \in A$	分散共分散行列の各要素の下限
	$sigU_{ij}, i \in A, j \in A$	分散共分散行列の各要素の上限
	λ	リスク回避係数
	$\mu_i, i \in A$	各投資対象の平均収益率
変数	$x_i, i \in A$	組入比率
	$L_{ij}, i \in A, j \in A$	
	$U_{ij}, i \in A, j \in A$	
目的関数（最大化）	$\mu^T x$ $-\lambda\left(sigU \bullet U - sigL \bullet L\right)$	$\bullet$ は要素ごとの積の和
制約条件	$\begin{pmatrix} U-L & x \\ x^T & 1 \end{pmatrix} \succeq 0$	半正定値制約
	$\sum_{i \in A} x_i = 1$	組入比率の総和は１
	$L_{ij} \geq 0, \quad \forall i \in A, j \in A$	行列 L の各要素に関する非負条件
	$U_{ij} \geq 0, \quad \forall i \in A, j \in A$	行列 U の各要素に関する非負条件

この問題は半正定値制約が入っているので，半正定値計画問題になります.

定式化した結果を SIMPLE で記述すると以下のようになります.

```
// 集合と添字
Parameter nA; // 銘柄数
Sequence Asset(from=1,to=nA);
Element i(set=Asset);
Element j(set=Asset);
// パラメータ
Parameter sigL(index=(i,j));
Parameter sigU(index=(i,j));
Parameter lambda;
Parameter mu(index=i);
// 変数
Variable U(index=(i,j));
Variable L(index=(i,j));
Variable x(index=i);
// 対称行列
Sequence V(from=1,to=nA+1);
Element v(set=V);
Element w(set=V);
SymmetricMatrix M((v,w));
// Mの要素の定義（上三角部分のみ）
M[i,j] = U[i,j] - L[i,j], i <= j; // 左上（の上三角部分）
M[j,nA+1] = x[j];        // 右上
M[nA+1,nA+1] = 1;        // 右下
// 目的関数
Objective f(type=maximize);
f = sum(mu[i]*x[i],i)
  - lambda*sum(sigU[i,j]*U[i,j]-sigL[i,j]*L[i,j],(i,j));
// 制約条件
M >= 0; // 半正定値制約
sum(x[i],i) == 1;
U[i,j] == U[j,i], i > j;
L[i,j] == L[j,i], i > j;
U[i,j] >= 0;
L[i,j] >= 0;
// 求解
solve();
// 出力
x.val.print();
```

データファイルは以下の三つです.

(csv 形式)

```
sigL,1,2,3,4,5,6,7,8
1,1.9,-1.4,-1,-1,-1,-0.5,-1,0.1
2,-1.4,3.5,-1.5,0.5,-1.2,-1.2,0.4,0.6
3,-1,-1.5,5,-1.125,1.1,0.9,0.3,-0.2
4,-1,0.5,-1.125,6,1.5,0.4,1.2,-0.9
5,-1,-1.2,1.1,1.5,4.5,-1.4,0.15,-2
6,-0.5,-1.2,0.9,0.4,-1.4,9,-1,0.5
7,-1,0.4,0.3,1.2,0.15,-1,5.5,-1.25
8,0.1,0.6,-0.2,-0.9,-2,0.5,-1.25,11.5
```

(csv 形式)

```
sigU,1,2,3,4,5,6,7,8
1,3,1,2.5,-0.9,1,-0.4,2.5,2
2,1,4.5,-1.4,1.2,0.5,-1.1,0.5,2.6
3,2.5,-1.4,6,-0.5,1.2,1,0.4,-0.1
4,-0.9,1.2,-0.5,6.5,1.6,0.5,1.3,-0.8
5,1,0.5,1.2,1.6,5.5,-1.3,0.16,-1.9
6,-0.4,-1.1,1,0.5,-1.3,11,-0.9,0.6
7,2.5,0.5,0.4,1.3,0.16,-0.9,6.5,-1.2
8,2,2.6,-0.1,-0.8,-1.9,0.6,-1.2,13.5
```

(dat 形式)

```
nA = 8;
lambda = 1;
mu = [1] 0.1 [2] 0.1 [3] 0.1 [4] 0.1 [5] 0.1 [6] 0.1 [7] 0.1 [8] 0.1;
```

このモデルを実行すると以下のような解が得られます.

```
x[1]=6.46383e-006
x[2]=0.233187
x[3]=0.175312
x[4]=0.0633794
x[5]=0.171046
x[6]=0.131894
x[7]=0.152881
x[8]=0.0722949
```

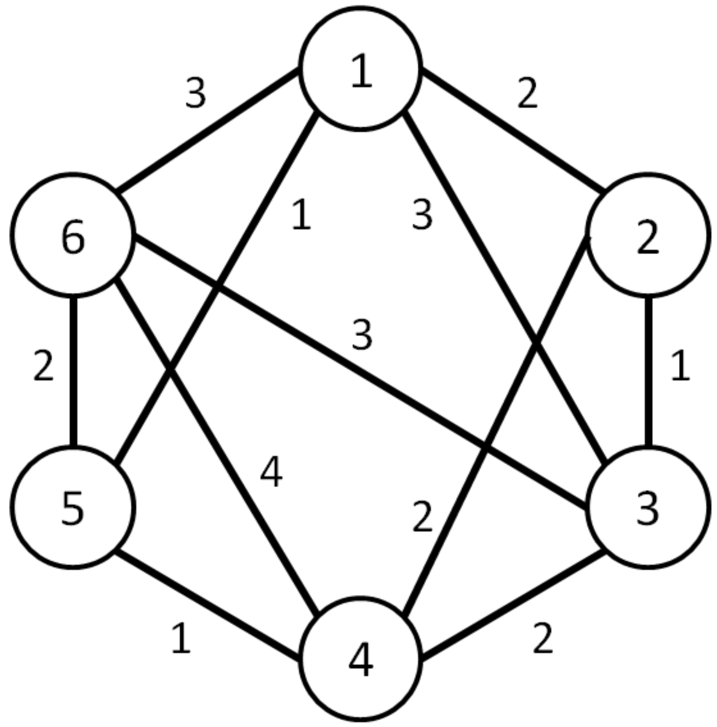
2.23 隣接行列（最大カット問題）

最大カット問題とは，重み付きグラフのノードを２つのグループに分割する問題で，グループ分けをした際にカットされるエッジの重み総和を最大化する問題です．ここで，２つのノードが異なるグループに属したとき，その２点をつなぐエッジがカットされると言います．

この問題は，グラフ構造が隣接行列で与えられるという特徴を持ちます．また，数理計画法としては， SDP 緩和問題として良質な上界値を求めることができる例題として知られています．

（例題）

次のグラフは，ノードとして 1， 2， 3， 4， 5， 6 を持つ重み付きグラフです．エッジがない場合は，重みを 0 とします．



ノードを二つのグループに分けて，カットされるエッジの重みの総和を最大化したい場合，どのようなグループとなるでしょうか．

この問題を NUOPT で解くために定式化を行いますが，まずは SDP 緩和問題として定式化し，目的関数値の上界値を求めてみましょう．NUOPT V13 から Vector/Matrix クラスが利用できるようになったため，ラプラス行列の計算を簡単に表現することができるようになりました．ここでラプラス行列とは，グラフの次数行列と隣接行列が与えられたとき，次数行列から隣接行列を引いた行列で定義されます．

以下で紹介する定式化は参考文献[7]によります．まず，ノードの集合として N を与え，変数として $x_i \in \{-1,1\}, (i \in N)$ を与えます．このとき，ノード $i, j \in N$ に対して， $x_i = x_j$ で i, j

が同じグループに属していることを，$x_i \neq x_j$ で i, j が異なるグループに属していることを表すことにすると，次のように定式化することができます.

$$\begin{aligned} \max \quad & \frac{1}{2}\sum_{i,j\in N, i<j} G_{ij}(1 - x_i x_j) \\ s.t. \quad & x_i \in \{-1,1\} \quad \forall i \in N \end{aligned}$$

上記定式化に対し， 変数 $x_i, i \in N$ を $X_{ij} = x_i x_i$ で置き換える変数 $X_{ij}, i \in N, j \in N$ を導入すると，次の問題と同値になることが知られています.

$$\begin{aligned} \max \quad & \frac{1}{2}\sum_{i,j\in N, i<j} G_{ij}(1 - X_{ij}) \\ s.t. \quad & X \succeq 0 \\ & \mathrm{rank}(X) = 1 \\ & X_{ii} = 1 \quad \forall i \in N \end{aligned}$$

$x_i \in \{-1,1\}$ は，$X_{ii}(= x_i x_i) = 1$ に対応し，x_i，x_j の値の整合性は $X \succeq 0$，$\mathrm{rank}(X) = 1$ に対応します．ここで，を緩和した問題が次の SDP 緩和問題となります.

$$\begin{aligned} \max \quad & \frac{1}{2}\sum_{i,j\in N, i<j} G_{ij}(1 - X_{ij}) \\ s.t. \quad & X \succeq 0 \\ & X_{ii} = 1 \quad \forall i \in N \end{aligned}$$

上記定式化の目的関数を

$$\begin{aligned} \frac{1}{2}\sum_{i,j\in N, i<j} G_{ij}(1 - X_{ij}) &= \frac{1}{4}\sum_{i,j\in N} G_{ij}(1 - X_{ij}) \\ &= \frac{1}{4}\sum_{i\in N}\left(\sum_{j\in N} G_{ij} X_{ii} - \sum_{j\in N} G_{ij} X_{ij}\right) \\ &= \frac{1}{4}\left(\mathrm{diag}(Ge) - G\right) \bullet X \end{aligned}$$

と式変形をすることで以下の定式化が得られます.

集合	$N = \{1, 2, 3, 4, 5, 6\}$	ノードの集合
定数	$G_{ij}, i \in N, j \in N$	ノード $i \in N$ とノード $j \in N$ がつくるエッジの重みを表現する隣接行列

	$L = diag(Ge) - G$	グラフ G のラプラス行列 （e は単位ベクトル）
変数	$X_{ij}, i \in N, j \in N$	緩和前ではノード $i \in N$ とノード $j \in N$ がカットされている場合に-1, カットされていない場合に 1 をとる変数
目的関数（最大化）	$\frac{1}{4} L \bullet X$	緩和前ではカットされるエッジの重みの総和
制約	$X \succeq 0$ $X_{ii} = 1, \quad \forall i \in N$	半正定値制約 対角要素は 1

次に，行列をデータファイルから与える SIMPLE モデルを示します．以下のモデルでは不要な変数を可能な限り消去するため，変数を上三角部分しか定義していないモデルになります．

```
// 集合と添え字
Set N;
Element i(set=N);
Element j(set=N);

// 行列パラメータ
Matrix G((i,j)); // 重み付き隣接行列
Matrix L((i,j)); // グラフ G のラプラス行列
L = diag(G*ones(N)) - G;

Set NN(dim=2); // 上三角部分
NN = setOf((i,j), i<=j);
Element ij(set=NN);

//変数
Variable x(index=ij);
SymmetricMatrix X((i,j));
X[i,j] = x[i,j], i<=j;
X[j,i] = x[i,j], i<j;
x[i,i] == 1; // 対角要素は 1
X >= 0;      // 半正定値制約

// 目的関数
Objective obj(type=maximize);
obj = 0.25 * inprod(L,X);

//  求解
solve();

// 出力
obj.val.print();
```

データファイル（.csv 形式）は以下のようになります.

```
G, 1, 2, 3, 4, 5, 6
1, 0, 2, 3, 0, 1, 3
2, 2, 0, 1, 2, 0, 0
3, 3, 1, 0, 2, 0, 3
4, 0, 2, 2, 0, 1, 4
5, 1, 0, 0, 1, 0, 2
6, 3, 0, 3, 4, 2, 0
```

このモデルを実行すると，目的関数値 18.7437 が得られます．これは最大化問題の緩和問題ですので，本問題の上界値となります.

次に 0-1 整数計画問題としての定式化を紹介します．SDP 緩和問題で得られた上界値の精度を調べてみましょう.

まず変数は，各ノードに対し 0-1 変数 $x_i, (i \in N)$ を用意して「2 つのグループに分割する」を「0 の値をとるグループ」と「1 の値をとるグループ」に対応させます．また，ノード i とノード j が異なるグループのときに 1 をとり，同じグループのときに 0 をとる変数 $y_{ij}, (i \in N, j \in N)$ を用意します．このとき，目的関数はカットされたエッジの重みの総和ですから，$\sum_{i<j} G_{ij} y_{ij}$ で表現されます．注意するべきこととして x_i と y_{ij} の整合性をとる制約を追加する必要があります．具体的に $(x_i, x_j, y_{ij}) = (0,1,1), (1,0,1), (0,0,0), (1,1,0)$ となる組合せを残せばよいので，この組合せの補集合を考えて $(x_i, x_j, y_{ij}) = (0,1,0), (1,0,0), (0,0,1), (1,1,1)$ となる組合せを排除する制約を導入します．このような制約を導入した定式化は次のようになります.

集合	$N = \{1,2,3,4,5,6\}$	ノードの集合
行列	$G_{ij}, i \in N, j \in N$	ノード $i \in N$ とノード $j \in N$ がつくるエッジの重みを表現する隣接行列
0-1 変数	$x_i, i \in N$	0-1 の 2 つのグループのどちらに入るかを表す変数

変数	$y_{ij}, i \in N, j \in N$	カットされたエッジの情報を x_i, x_j から得るための変数
目的関数（最大化）	$\sum_{i<j} G_{ij} y_{ij}$	カットされるエッジの重みの総和
制約	$-x_i - x_j + y_{ij} \le 0 \quad (1)$	$(x_i, x_j, y_{ij}) = (0,0,1)$ の排除
	$-x_i + x_j - y_{ij} \le 0 \quad (2)$	$(x_i, x_j, y_{ij}) = (0,1,0)$ の排除
	$x_i - x_j - y_{ij} \le 0 \quad (3)$	$(x_i, x_j, y_{ij}) = (1,0,0)$ の排除
	$x_i + x_j + y_{ij} \le 2 \quad (4)$	$(x_i, x_j, y_{ij}) = (1,1,1)$ の排除

上記にて変数 y_{ij} は連続変数として定義されていますが，実際には 0–1 しかとらない変数となります．これは，式(1)(4)より $y_{ij} \ge 0$ が，式(2)(3)より，$y_{ij} \le 1$ が導かれ，さらに (x_i, x_j) の組合せが $(0,0),(0,1),(1,0),(1,1)$ しかとらないことと式(1)(2)(3)(4)から，$y_{ij} = 0\ or\ 1$ が導かれます．また，上記の不必要な組合せを排除する方法の代用として 0–1 変数 z_{ij} を追加して，

$$x_i + x_j + y_{ij} = 2z_{ij}$$

と表現する方法もあります．次に， SIMPLE モデルを示します． 以下のモデルでは不要な変数を可能な限り消去するため，変数を上三角部分しか定義していないモデルになります．

```
// 集合
Set N;
Element i(set=N);
Element j(set=N);

// 定数
Matrix G((i,j));

Set NN(dim=2); // 対角成分
NN = setOf((i,j),i<j);
Element ij(set=NN);

// 変数
IntegerVariable x(index=i,type=binary);
Variable y(index=ij);

// x[i] x[j] y[i,j] の整合性をとる制約
- x[i] - x[j] + y[i,j] <= 0,(i,j)<NN;
- x[i] + x[j] - y[i,j] <= 0,(i,j)<NN;
  x[i] - x[j] - y[i,j] <= 0,(i,j)<NN;
  x[i] + x[j] + y[i,j] <= 2,(i,j)<NN;

// 目的関数
Objective obj(type=maximize);
obj = sum(G[ij]*y[ij],ij);

// 大規模問題には以下の近似解法を用いる
// ※ その際，y を 0-1 整数変数にする必要がある
// options.method="wcsp";

// 求解
solve();

// 出力
x.val.print();
obj.val.print();
```

このモデルを実行すると，目的関数値 18，ノード 1，4，5 のグループとノード 2,3,6 のグループが得られます．SDP 緩和問題で求められた目的関数値が 18.7437 であったことを考えると，SDP 緩和問題が非常に良質な上界値を与えていることがわかります．

上記のように SDP 緩和問題で良質な上界値を得られることがわかっていれば，上界値を用いて WCSP アルゴリズムの精度保証に利用したり，終了条件を設定する際にも利用できます．参考までに SDP 緩和問題，単体法（＋分枝限定法），WCSP でそれぞれ解いた結果を以下に示します．以下の表は SDP 緩和の上界値の精度と近似解法 WCSP の安定性を示す十分な結果を表しています．

	SDP		SIMPLEX		WCSP	
Graph	TIME	Upper	TIME	Optimal	TIME	Lower
G20.csv	0.2	464.39	0.2	452	0.05	452
G30.csv	1.2	825.64	1.2	797	2.1	797
G40.csv	6.5	1215.86	6.0	1164	1.7	1164
G50.csv	23.1	1736.33	23.7	1651	7.6	1651
G60.csv	55.5	2156.71	153.2	2068	27.4	2068

Graph データ G**.csv の「**」はノード数を表す．上記データはすべてノードに対して平均 10 のリンクが存在するように乱数を発生させたデータ．

WCSP アルゴリズムの設定は，最大探索時間 10 秒，乱数を用いた 3 回の探索試行を行った．

2.24 セミナー割当問題

資源制約付きスケジューリング問題の例として,本節ではセミナー会場提供会社におけるスケジュール計画を以下で考えていきます.

（例題 1）

あるセミナー会場提供会社は，以下の 1～6 までのセミナーに対し，会場を提供するものとします.

セミナー名	各セミナーのコマ数
1	4
2	6
3	5
4	3
5	10
6	7

例題 1 では，各セミナーにおいて 1 日あたりに消化できるコマ数は 1 つとします. 別のセミナーを同一の会場で行うことはできないものとし，1 日あたりの会場使用数の最大を 4 とするとき，これらすべてのセミナーが完了する時刻が最小となるようなスケジュールを求めて下さい.

なお，すべてのセミナーが終了するまでの日数は最大でも 20 日までとします.

この問題を定式化すると，以下のようになります．なお，セミナー会場に関しては区別がないので，一つの資源とみなすことに注意して下さい．

セミナー集合	Se	セミナー名 1〜6
資源	1	セミナー会場，供給量:常に 4
モード	A_1	セミナー1 のコマ数消化方法 処理時間:4 必要量：資源 1 を処理時間中常に 1
	A_2	セミナー2 のコマ数消化方法 処理時間:6 必要量：資源 1 を処理時間中常に 1
	A_3	セミナー3 のコマ数消化方法 処理時間:5 必要量：資源 1 を処理時間中常に 1
	A_4	セミナー4 のコマ数消化方法 処理時間:3 必要量：資源 1 を処理時間中常に 1
	A_5	セミナー5 のコマ数消化方法 処理時間:10 必要量：資源 1 を処理時間中常に 1
	A_6	セミナー6 のコマ数消化方法 処理時間:7 必要量：資源 1 を処理時間中常に 1
モード集合族	$MM_i, \quad i \in Se$	セミナーi がとりうるモード集合
アクティビティ	$act_i, \quad i \in Se$	セミナーi をこなす作業 処理モード： A_i
目的関数 (最小化)	completionTime	最後の作業の完了時刻
スケジュール期間	T	最大で 20 日とする

定式化した結果を SIMPLE で記述すると以下のようになります．

```
// 全体のスケジューリング期間
Set T(name="T");
Element t(set=T);

// 作業(セミナー)集合
Set Se(name="Se");
Element se(set=Se);

// モード集合
Set M(name="M");
Element m(set=M);
// 各セミナーがとりうるモード
Set MM(name="MM", index=se);

// 資源(部屋)集合
Set Ro(name="Ro");
Element ro(set=Ro);

// 各モード中の最大開催期間までの期間集合
Set TT(name="TT");
Element tt(set=TT);

// 必要資源
ResourceRequire req(name="req", mode=M, resource=Ro, duration=TT);

// 資源供給量
ResourceCapacity cap(name="cap", resource=Ro, timeStep=T);

// アクティビティ
Activity act(name="セミナー", index=se, mode=MM[se]);

// 目的関数（すべてのセミナーの完了時刻の最小化）
Objective f(name="最小完了日数", type=minimize);
f = completionTime;

// 求解最大時間
options.maxtim = 5;
```

```
// 求解
solve();

// 結果の出力
f.val.print();
simple_printf("セミナー[%s] = %d¥n", se, act[se].startTime);

// ガントチャート出力
Gantt g;
g.add(act[se], se);
g.dump();
```

入力データ（dat 形式）は，以下のようになります.

```
cap = [1,0] 4 [1,1] 4 [1,2] 4 [1,3] 4 [1,4] 4 [1,5] 4 [1,6] 4
      [1,7] 4 [1,8] 4 [1,9] 4 [1,10] 4 [1,11] 4 [1,12] 4
      [1,13] 4 [1,14] 4 [1,15] 4 [1,16] 4 [1,17] 4 [1,18] 4
      [1,19] 4 [1,20] 4;
MM  = [1] A1 [2] A2 [3] A3 [4] A4 [5] A5 [6] A6;
req = [A1,1,1] 1 [A1,1,2] 1 [A1,1,3] 1 [A1,1,4] 1
      [A2,1,1] 1 [A2,1,2] 1 [A2,1,3] 1 [A2,1,4] 1 [A2,1,5] 1
      [A2,1,6] 1
      [A3,1,1] 1 [A3,1,2] 1 [A3,1,3] 1 [A3,1,4] 1 [A3,1,5] 1
      [A4,1,1] 1 [A4,1,2] 1 [A4,1,3] 1
      [A5,1,1] 1 [A5,1,2] 1 [A5,1,3] 1 [A5,1,4] 1 [A5,1,5] 1
      [A5,1,6] 1 [A5,1,7] 1 [A5,1,8] 1 [A5,1,9] 1 [A5,1,10] 1
      [A6,1,1] 1 [A6,1,2] 1 [A6,1,3] 1 [A6,1,4] 1 [A6,1,5] 1
      [A6,1,6] 1 [A6,1,7] 1;
```

なお，入力データを Excel 連係機能を用いて与えることも可能です．具体的には次図のようなデータを Excel から NUOPT に渡します．データ受け渡し方法につきましては「NUOPT Excel 連係マニュアル」をご覧下さい．

		cap
1	0	4
	1	4
	2	4
	3	4
	4	4
	5	4
	6	4
	7	4
	8	4
	9	4
	10	4
	11	4
	12	4
	13	4
	14	4
	15	4
	16	4
	17	4
	18	4
	19	4
	20	4

se	MM
1	A1
2	A2
3	A3
4	A4
5	A5
6	A6

	req	1	2	3	4	5	6	7	8	9	10
A1	1	1	1	1	1						
A2		1	1	1	1	1	1				
A3		1	1	1	1	1					
A4		1	1	1							
A5		1	1	1	1	1	1	1	1	1	1
A6		1	1	1	1	1	1	1			

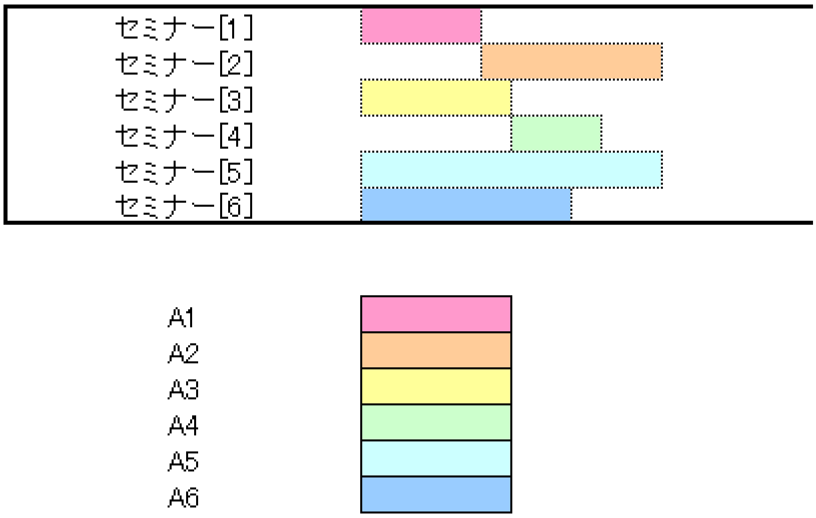
結果は，以下のようになります．

```
最小完了日数=10
セミナー[1] = 0
セミナー[2] = 4
セミナー[3] = 0
セミナー[4] = 5
セミナー[5] = 0
セミナー[6] = 0
```

上記結果のセミナーの値は，各セミナーの開始日時を表しています．

なお，ガントチャートによる解出力の可視化を行うと，次のようになります．

例題 1 の解の出力例：



例題 1 では，各セミナーにおけるモードは 1 種類（一日一コマずつ消化していく）のみでしたが，続く例題 2 では，各セミナーに対していくつかのモードを用意し，例題 1 からの結果の推移を検証します.

（例題 2）

例題 1 の条件のもと，各セミナーにおいて，コマ数の消化方法の選択肢をいくつか与えるものとします．具体的には，以下の表の通りです．

セミナー名	モード	セミナーコマ数の消化方法
1	A1_1	1-1-1-1
2	A2_1	1-1-1-1-1-1
	A2_2	2-2-2
	A2_3	1-1-2-2
3	A3_1	1-1-1-1-1
	A3_2	2-2-1
4	A4_1	1-1-1
5	A5_1	1-1-1-1-1-1-1-1-1-1
	A5_2	1-2-2-2-2-1
	A5_3	2-2-2-2-2
6	A6_1	1-1-1-1-1-1-1
	A6_2	1-2-2-2
	A6_3	2-2-2-1

表の見方ですが，例えば A2_2 でしたら，セミナー2 が全コマ数 6 を 3 日に渡って各々 2 ずつ消化する，ということになります．各セミナーは，そのセミナー特有に用意されたモードのうちどれか一つを選択しなければなりません．例えばセミナー3 に対するモードは A3_1，A3_2 の 2 つがあるので，それらのうちのどちらかを選択してセミナーのコマ数を消化するということになります．

すべてのセミナーが完了する時刻が最小となるように，適切なモードを選択してスケジュールを求めて下さい．

定式化に関しましては，例題 1 と同様です．ですので，モデルファイルの記述も例題 1 と同一になります．

例題 1 と異なるのは，入力データのみです．Excel 連係機能を用いて Excel から NUOPT へデータを渡す場合には，次のようなデータを与えます．

		cap
1	0	4
	1	4
	2	4
	3	4
	4	4
	5	4
	6	4
	7	4
	8	4
	9	4
	10	4
	11	4
	12	4
	13	4
	14	4
	15	4
	16	4
	17	4
	18	4
	19	4
	20	4

se	MM		
1	A1_1		
2	A2_1	A2_2	A2_3
3	A3_1	A3_2	
4	A4_1		
5	A5_1	A5_2	A5_3
6	A6_1	A6_2	A6_3

	req	1	2	3	4	5	6	7	8	9	10
A1_1	1	1	1	1	1						
A2_1		1	1	1	1	1	1				
A2_2		2	2	2							
A2_3		1	1	2	2						
A3_1		1	1	1	1	1					
A3_2		2	2	1							
A4_1		1	1	1							
A5_1		1	1	1	1	1	1	1	1	1	1
A5_2		1	2	2	2	2	1				
A5_3		2	2	2	2	2					
A6_1		1	1	1	1	1	1	1			
A6_2		1	2	2	2						
A6_3		2	2	2	1						

例題 1 と比較すると，各セミナーにおいてとりうるモードの数が増えたことがおわかりになると思います．

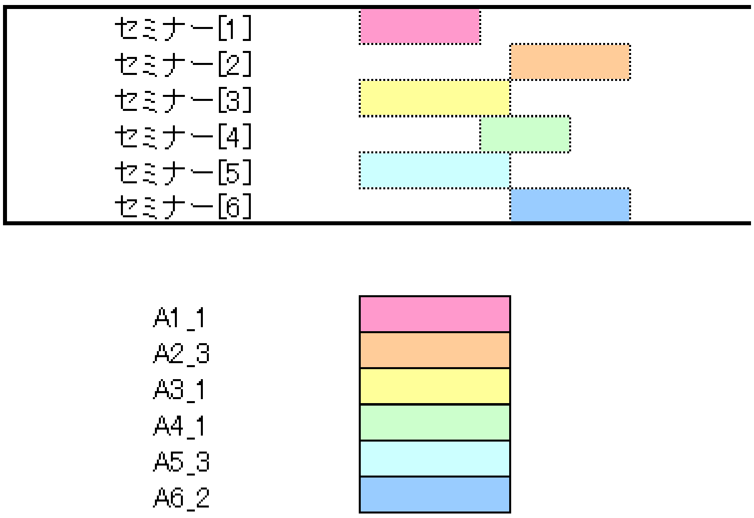
結果は，以下のようになります．

```
最小完了日数=9
セミナー[1] = 0
セミナー[2] = 5
セミナー[3] = 0
セミナー[4] = 4
セミナー[5] = 0
セミナー[6] = 5
```

結果から明らかなように，例題 1 に比べて，例題 2 ではセミナー完了時刻を短縮できるようなスケジュール計画を立てることができました．これは，各セミナーがとりうるモードの数が例題 1 と比較して増えたことによるためです．

なお，ガントチャートによる解出力の結果は以下のようになります．

例題 2 の解の出力例：



この節の最後に，今までは資源となる会場は一つのみでしたが，もう一つ資源となる会場を追加してみましょう．セミナーを実施するメイン会場（大部屋）とサブ会場（小部屋）のようなものを想定していただければ，おわかりになるかと思います．

（例題 3）

例題 2 の条件のもと，さらにメイン会場を用意します．

各セミナーともにメイン会場で 1 日セミナーを行った後，その他の会場（以下では，サブ会場と呼ぶことにします）において各個別のセミナーを行うものとします．サブ会場でのセミナー実施方法は，例題 2 と同一とします．1 日あたりのサブ会場使用数の最大に関しましても，例題 1・例題 2 同様，4 とします．

メイン会場では 1 日 1 種類のセミナーしか行うことができず，しかもメイン会場は各セミナーのはじめの 1 コマのみしか利用することができないものとするとき，すべてのセミナーが完了する時刻が最小となるようなスケジュールを求めて下さい．

定式化としては以下のように，各セミナーともに，メイン会場でセミナーを行った後にサブ会場で行うという先行制約が加わります．なお先行関係は≺により表すものとします．

資源	0 1	メイン会場，供給量:常に 1 サブ会場，供給量:常に 4
先行制約	$act_{i,0} \prec act_{i,1}, \quad \forall i \in Se$	任意のセミナー i において，メイン会場でのセミナーを終えた後にメイン以外の会場でのセミナーに移行する

上記定式化の追加を反映した SIMPLE でのモデル記述は，以下のようになります．

```
// 全体のスケジューリング期間
Set T(name="T");
Element t(set=T);

// 作業(セミナー)集合
Set Se(name="Se");
Element se(set=Se);

// モード集合
Set M(name="M");
Element m(set=M);

// 資源(部屋)集合
Set Ro(name="Ro");
Element ro(set=Ro);

// 各セミナーがとりうるモード
Set MM(name="MM", index=(se,ro));

// 各モード中の最大開催期間までの期間集合
Set TT(name="TT");
Element tt(set=TT);

// 必要資源
ResourceRequire req(name=”req”, mode=M, resource=Ro, duration=TT);
```

```
// アクティビティ
Activity act(name="セミナー", index=(se,ro), mode=MM[se,ro]);
// 先行制約
act[se,ro-1] < act[se,ro], ro > 0;

// 目的関数（すべてのセミナーの完了時刻の最小化）
Objective f(name=”最小完了日数”, type=minimize);
f = completionTime;

// 求解最大時間
options.maxtim = 5;

// 求解
solve();

// 結果の出力
f.val.print();
simple_printf("セミナー[%s,%s] = %d¥n", se, ro, act[se,ro].startTime);

// ガントチャート出力
Gantt g;
g.add(act[se,ro],se,ro);
g.dump();
```

入力データは，Excel 連係機能を用いて以下のようなデータを NUOPT に渡します.

メイン会場に対するモードは,各セミナーともに large_room として与えていることに注意して下さい.

	req	1	2	3	4	5	6	7	8	9	10
large_room	0	1									
A1_1	1	1	1	1	1						
A2_1		1	1	1	1	1	1				
A2_2		2	2	2							
A2_3		1	1	2	2						
A3_1		1	1	1	1	1					
A3_2		2	2	1							
A4_1		1	1	1							
A5_1		1	1	1	1	1	1	1	1	1	1
A5_2		1	2	2	2	2	1				
A5_3		2	2	2	2	2					
A6_1		1	1	1	1	1	1	1			
A6_2		1	2	2	2						
A6_3		2	2	2	1						

		MM		
1	0	large_room		
	1	A1_1		
2	0	large_room		
	1	A2_1	A2_2	A2_3
3	0	large_room		
	1	A3_1	A3_2	
4	0	large_room		
	1	A4_1		
5	0	large_room		
	1	A5_1	A5_2	A5_3
6	0	large_room		
	1	A6_1	A6_2	A6_3

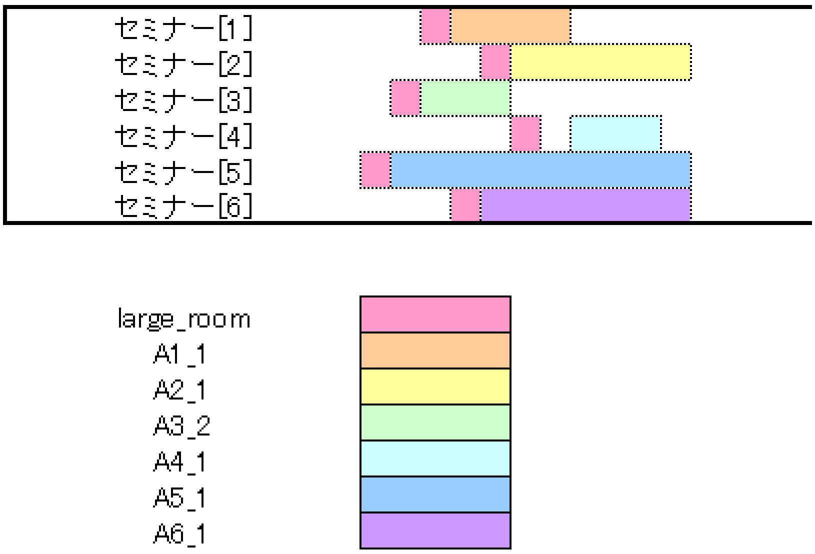
cap	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

結果は，以下のようになります．

```
最小完了日数=11
セミナー[1,0] = 2
セミナー[1,1] = 3
セミナー[2,0] = 4
セミナー[2,1] = 5
セミナー[3,0] = 1
セミナー[3,1] = 2
セミナー[4,0] = 5
セミナー[4,1] = 7
セミナー[5,0] = 0
セミナー[5,1] = 1
セミナー[6,0] = 3
セミナー[6,1] = 4
```

では，この結果をガントチャートによる解出力で可視化して検証しましょう．確かに各セミナーともに，メイン会場でのセミナー消化後，サブ会場でセミナーを行っているのが確認できます．

例題 3 の解の出力例：


セミナー[1]
セミナー[2]
セミナー[3]
セミナー[4]
セミナー[5]
セミナー[6]
large_room
A1_1
A2_1
A3_2
A4_1
A5_1
A6_1

2.25 ジョブショップスケジューリング問題

ジョブショップスケジューリング問題とは，生産計画等の現場で現れ，仕事(ジョブ)を機械に効率的に割り振る事で，完了時刻，納期遅れ等の最小化を目的としたスケジューリング問題です．ここでは，特に機械の数が複数で，仕事が複数の作業(オペレーション)から構成され，各仕事の全ての作業が処理されると，仕事の処理が完了する直列機械(serial machines)について言及します．本問題は，作業が処理される順序によって，以下の 3 つに大別されます．

(1)　オープンショップ問題（2.25.1 節）

各作業が処理される機械の順序は決まっていない．

(2)　フローショップ問題（2.25.2 節）

各作業が処理される機械の順序が同じ．

(3)　ジョブショップ問題（2.25.3 節）

各作業が処理される機械の順序が決められているが，これらの順序は仕事ごとに異なっていてもよい．フローショップ問題の一般形．

以下で，上記(1)～(3)の例題を紹介します．必要に応じて「NUOPT_SIMPLE_マニュアル」の第 6 章も併せてご覧下さい．

2.25.1 オープンショップ問題

（例題）オープンショップ問題

3 つの作業から構成される，3 つの仕事があるとします．各作業を処理する機械と処理時間は以下のように与えられています．

	作業 1(機械 1)	作業 2(機械 2)	作業 3(機械 3)
仕事 a	5 時間	8 時間	4 時間
仕事 b	5 時間	4 時間	6 時間
仕事 c	6 時間	5 時間	5 時間

各機械が作業を処理している間は，他の作業を処理する事は出来ません．また，各作業を処理する順番に制限はありません．この場合の最後の作業の完了時刻が最小となるようにするには，各仕事をどのように機械に割り振ればいいでしょうか．なお，すべての仕事を終えるまでの所要時間は最大で 30 日までとする．

この問題を資源制約付きスケジューリング問題で定式化すると，例えば以下の様になります．各仕事の各作業は同時には 2 つ以上処理出来ない事を，ダミー資源を用いて表現している事に注意して下さい．

資源	machine1	機械 1，供給量:常に 1
	machine2	機械 2，供給量:常に 1
	machine3	機械 3，供給量:常に 1
	d1	ダミー資源 1，供給量:常に 1
	d2	ダミー資源 2，供給量:常に 1
	d3	ダミー資源 3，供給量:常に 1
モード	mode_a_1	処理時間:5 必要量：machine1,d1 を処理時間中常に 1
	mode_a_2	処理時間:8 必要量：machine2,d1 を処理時間中常に 1
	mode_a_3	処理時間:4 必要量：machine3,d1 を処理時間中常に 1
	mode_b_1	処理時間:5 必要量：machine1,d2 を処理時間中常に 1
	mode_b_2	処理時間:4 必要量：machine2,d2 を処理時間中常に 1
	mode_b_3	処理時間:6 必要量：machine3,d2 を処理時間中常に 1
	mode_c_1	処理時間:6 必要量：machine1,d3 を処理時間中常に 1
	mode_c_2	処理時間:5 必要量：machine2,d3 を処理時間中常に 1
	mode_c_3	処理時間:5 必要量：machine3,d3 を処理時間中常に 1
アクティビティ	$act_{a,1}$	仕事 a の作業 1 処理モード：mode_a_1
	$act_{a,2}$	仕事 a の作業 2 処理モード：mode_a_2
	$act_{a,3}$	仕事 a の作業 3 処理モード：mode_a_3
	$act_{b,1}$	仕事 b の作業 1 処理モード：mode_b_1
	$act_{b,2}$	仕事 b の作業 2 処理モード：mode_b_2

	$act_{b,3}$	仕事 b の作業 3 処理モード：mode_b_3
	$act_{c,1}$	仕事 c の作業 1 処理モード：mode_c_1
	$act_{c,2}$	仕事 c の作業 2 処理モード：mode_c_2
	$act_{c,3}$	仕事 c の作業 3 処理モード：mode_c_3
目的関数（最小化）	completionTime	最後の作業の完了時刻
スケジュール期間	T	最大で 30 時間とする

これを SIMPLE で記述すると次のようになります.

なお，アクティビティの引数に渡すモード集合ですが，仕事・作業の各組み合わせに対してどのモードが対応するかという集合 AvailMode を用意し，それを引数として渡します.

```
// 作業集合
Set J; // 仕事
Element j(set=J);
Set S; // 作業
Element s(set=S);
// モード集合
Set M;
Element m(set=M);
Set AvailMode(name="AvailMode", index=(j,s)); // 各仕事のオペレーションにおいて処理されるモード
// 資源集合
Set R;
Element r(set=R);
// 作業時間集合
Set D; // 各モードの作業時間の最大
Element d(set=D);
// 期間集合
Set T; // スケジュール期間
T = "0 .. 30";
Element t(set=T);
```

```
// アクティビティ(変数)
Activity act(name="act", index=(j,s), mode=AvailMode[j,s]);

// 定数
// 必要資源量
ResourceRequire req(name="req", mode=M, resource=R, duration=D);
// 資源供給量
ResourceCapacity cap(name="cap", resource=R, timeStep=T);
cap[r,t] = 1;

// 目的関数
Objective f(type=minimize);
f = completionTime; // 最後の作業の完了時刻最小化

// 求解最大時間の設定
options.maxtim = 15;

// 求解
solve();

// 結果の標準出力
simple_printf("act[%s,%d] = %d¥n", j, s, act[j,s].startTime);
```

データファイルは次のようになります.

```
AvailMode =
[a,1] mode_a_1
[a,2] mode_a_2
[a,3] mode_a_3
[b,1] mode_b_1
[b,2] mode_b_2
[b,3] mode_b_3
[c,1] mode_c_1
[c,2] mode_c_2
[c,3] mode_c_3
;
```

```
req =
[mode_a_1,machine1,1] 1 [mode_a_2,machine2,1] 1  [mode_a_3,machine3,1] 1
[mode_a_1,machine1,2] 1 [mode_a_2,machine2,2] 1  [mode_a_3,machine3,2] 1
[mode_a_1,machine1,3] 1 [mode_a_2,machine2,3] 1  [mode_a_3,machine3,3] 1
[mode_a_1,machine1,4] 1 [mode_a_2,machine2,4] 1  [mode_a_3,machine3,4] 1
[mode_a_1,machine1,5] 1 [mode_a_2,machine2,5] 1
                        [mode_a_2,machine2,6] 1
                        [mode_a_2,machine2,7] 1
                        [mode_a_2,machine2,8] 1

[mode_b_1,machine1,1] 1 [mode_b_2,machine2,1] 1  [mode_b_3,machine3,1] 1
[mode_b_1,machine1,2] 1 [mode_b_2,machine2,2] 1  [mode_b_3,machine3,2] 1
[mode_b_1,machine1,3] 1 [mode_b_2,machine2,3] 1  [mode_b_3,machine3,3] 1
[mode_b_1,machine1,4] 1 [mode_b_2,machine2,4] 1  [mode_b_3,machine3,4] 1
[mode_b_1,machine1,5] 1                          [mode_b_3,machine3,5] 1
                                                 [mode_b_3,machine3,6] 1

[mode_c_1,machine1,1] 1 [mode_c_2,machine2,1] 1 [mode_c_3,machine3,1] 1
[mode_c_1,machine1,2] 1 [mode_c_2,machine2,2] 1 [mode_c_3,machine3,2] 1
[mode_c_1,machine1,3] 1 [mode_c_2,machine2,3] 1 [mode_c_3,machine3,3] 1
[mode_c_1,machine1,4] 1 [mode_c_2,machine2,4] 1 [mode_c_3,machine3,4] 1
[mode_c_1,machine1,5] 1 [mode_c_2,machine2,5] 1 [mode_c_3,machine3,5] 1
[mode_c_1,machine1,6] 1

[mode_a_1,d1,1] 1 [mode_a_2,d1,1] 1  [mode_a_3,d1,1] 1
[mode_a_1,d1,2] 1 [mode_a_2,d1,2] 1  [mode_a_3,d1,2] 1
[mode_a_1,d1,3] 1 [mode_a_2,d1,3] 1  [mode_a_3,d1,3] 1
[mode_a_1,d1,4] 1 [mode_a_2,d1,4] 1  [mode_a_3,d1,4] 1
[mode_a_1,d1,5] 1 [mode_a_2,d1,5] 1
                  [mode_a_2,d1,6] 1
                  [mode_a_2,d1,7] 1
                  [mode_a_2,d1,8] 1
```

```
[mode_b_1,d2,1] 1 [mode_b_2,d2,1] 1 [mode_b_3,d2,1] 1
[mode_b_1,d2,2] 1 [mode_b_2,d2,2] 1 [mode_b_3,d2,2] 1
[mode_b_1,d2,3] 1 [mode_b_2,d2,3] 1 [mode_b_3,d2,3] 1
[mode_b_1,d2,4] 1 [mode_b_2,d2,4] 1 [mode_b_3,d2,4] 1
[mode_b_1,d2,5] 1                   [mode_b_3,d2,5] 1
                                     [mode_b_3,d2,6] 1

[mode_c_1,d3,1] 1 [mode_c_2,d3,1] 1 [mode_c_3,d3,1] 1
[mode_c_1,d3,2] 1 [mode_c_2,d3,2] 1 [mode_c_3,d3,2] 1
[mode_c_1,d3,3] 1 [mode_c_2,d3,3] 1 [mode_c_3,d3,3] 1
[mode_c_1,d3,4] 1 [mode_c_2,d3,4] 1 [mode_c_3,d3,4] 1
[mode_c_1,d3,5] 1 [mode_c_2,d3,5] 1 [mode_c_3,d3,5] 1
[mode_c_1,d3,6] 1
;
```

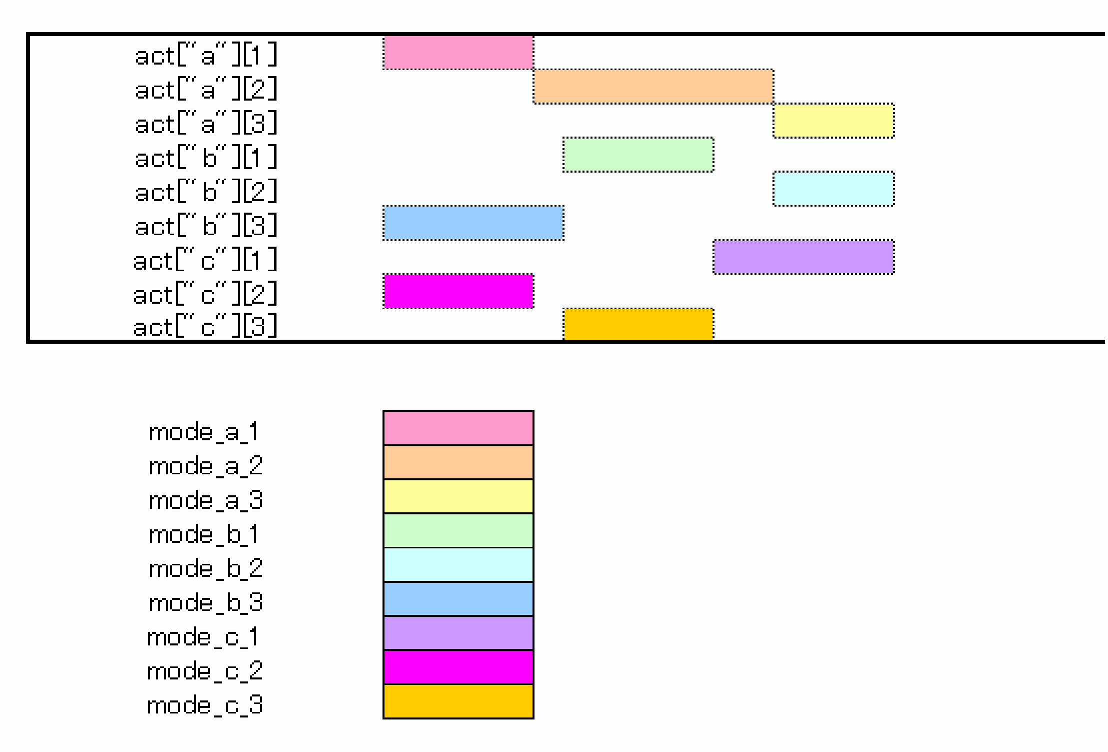
実行すると，各作業の開始時刻

```
act["a",1] = 0
act["a",2] = 5
act["a",3] = 13
act["b",1] = 6
act["b",2] = 13
act["b",3] = 0
act["c",1] = 11
act["c",2] = 0
act["c",3] = 6
```

が出力されます.

以下では，ガントチャートにより解の出力の可視化を行います.

例題（オープンショップ問題）の解の出力例：



2.25.2 フローショップ問題

（例題）フローショップ問題
2.23.1 節のオープンショップ問題の条件の下，各仕事は，作業 1，作業 2，作業 3 の順で処理されなければならないとします．この場合の最後の作業の完了時刻が最小となるようにするには，どのように機械を割り振ればよいでしょうか．

この問題を記述するには,各作業の処理順序を制約する以下の先行制約を加える必要があります．なお，以下で用いている $\prec$ は先行関係を表す記号とします．

先行制約	$act_{a,1} \prec act_{a,2}$	仕事 a の作業 1 は，仕事 a の作業 2 に先行する
	$act_{a,2} \prec act_{a,3}$	仕事 a の作業 2 は，仕事 a の作業 3 に先行する
	$act_{b,1} \prec act_{b,2}$	仕事 b の作業 1 は，仕事 b の作業 2 に先行する
	$act_{b,2} \prec act_{b,3}$	仕事 b の作業 2 は，仕事 b の作業 3 に先行する
	$act_{c,1} \prec act_{c,2}$	仕事 c の作業 1 は，仕事 c の作業 2 に先行する
	$act_{c,2} \prec act_{c,3}$	仕事 c の作業 2 は，仕事 c の作業 3 に先行する

上記の先行制約を加えた SIMPLE のモデルは以下の様になります．尚，各仕事の各作業を同時に 2 つ以上処理出来ない事は，先行制約で表現されている事に注意して下さい．

```
// 作業集合
Set J; // 仕事
Element j(set=J);
Set S; // 作業
Element s(set=S);
// モード集合
Set M;
Element m(set=M);
```

```
Set AvailMode(name="AvailMode", index=(j,s)); // 各仕事のオペレーションにおいて処理されるモード
// 資源集合
Set R;
Element r(set=R);
// 作業時間集合
Set D; // 各モードの作業時間の最大
Element d(set=D);
// 期間集合
Set T; // スケジュール期間
T = "0 .. 30";
Element t(set=T);

// アクティビティ(変数)
Activity act(name="act", index=(j,s), mode=AvailMode[j,s]);

// 定数
// 必要資源量
ResourceRequire req(name="req", mode=M, resource=R, duration=D);
// 資源供給量
ResourceCapacity cap(name="cap", resource=R, timeStep=T);
cap[r,t] = 1;

// 目的関数(最後の作業の完了時刻の最小化)
Objective f(type=minimize);
f = completionTime;

// 先行制約
act[j,s-1] < act[j,s], 1 < s;

// 求解最大時間の設定
options.maxtim = 15;

// 求解
solve();

// 結果の標準出力
simple_printf("act[%s,%d] = %d¥n", j, s, act[j,s].startTime);
```

先行制約を設けた事により，ダミー資源を用いる必要が無くなった為，データファイルの１つが以下の様に修正されます.

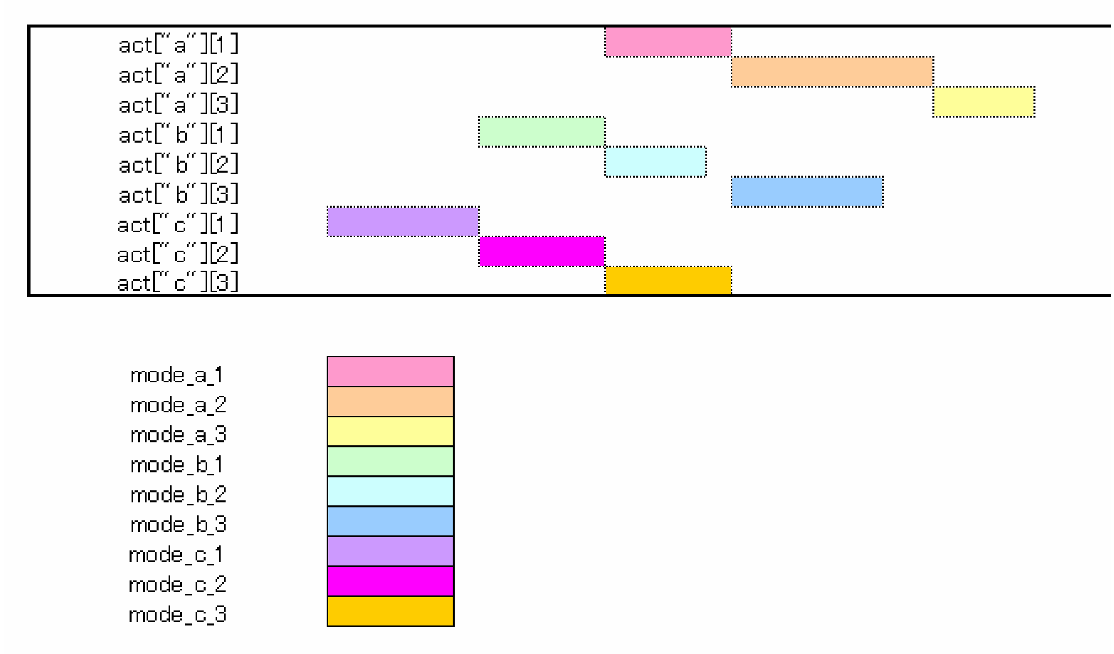
```
req =
[mode_a_1,machine1,1] 1 [mode_a_2,machine2,1] 1  [mode_a_3,machine3,1] 1
[mode_a_1,machine1,2] 1 [mode_a_2,machine2,2] 1  [mode_a_3,machine3,2] 1
[mode_a_1,machine1,3] 1 [mode_a_2,machine2,3] 1  [mode_a_3,machine3,3] 1
[mode_a_1,machine1,4] 1 [mode_a_2,machine2,4] 1  [mode_a_3,machine3,4] 1
[mode_a_1,machine1,5] 1 [mode_a_2,machine2,5] 1
                        [mode_a_2,machine2,6] 1
                        [mode_a_2,machine2,7] 1
                        [mode_a_2,machine2,8] 1

[mode_b_1,machine1,1] 1 [mode_b_2,machine2,1] 1  [mode_b_3,machine3,1] 1
[mode_b_1,machine1,2] 1 [mode_b_2,machine2,2] 1  [mode_b_3,machine3,2] 1
[mode_b_1,machine1,3] 1 [mode_b_2,machine2,3] 1  [mode_b_3,machine3,3] 1
[mode_b_1,machine1,4] 1 [mode_b_2,machine2,4] 1  [mode_b_3,machine3,4] 1
[mode_b_1,machine1,5] 1                          [mode_b_3,machine3,5] 1
                                                 [mode_b_3,machine3,6] 1

[mode_c_1,machine1,1] 1 [mode_c_2,machine2,1] 1 [mode_c_3,machine3,1] 1
[mode_c_1,machine1,2] 1 [mode_c_2,machine2,2] 1 [mode_c_3,machine3,2] 1
[mode_c_1,machine1,3] 1 [mode_c_2,machine2,3] 1 [mode_c_3,machine3,3] 1
[mode_c_1,machine1,4] 1 [mode_c_2,machine2,4] 1 [mode_c_3,machine3,4] 1
[mode_c_1,machine1,5] 1 [mode_c_2,machine2,5] 1 [mode_c_3,machine3,5] 1
[mode_c_1,machine1,6] 1
;
```

実行すると，各作業の開始時刻が以下のように出力されます.

```
act["a",1] = 11
act["a",2] = 16
act["a",3] = 24
act["b",1] = 6
act["b",2] = 11
act["b",3] = 16
act["c",1] = 0
act["c",2] = 6
act["c",3] = 11
```

以下では，2.25.1 節同様，結果のガントチャート出力を行います．

例題（フローショップ問題）の解の出力例：



2.25.3 ジョブショップ問題

（例題）ジョブショップ問題
2.23.2 節のオープンショップ問題の条件の下，各仕事は以下の順に処理されなければならないものとします.

	作業 1 (機械 1)	作業 2 (機械 2)	作業 3 (機械 3)
仕事 a	1	3	2
仕事 b	1	2	3
仕事 c	2	1	3

この場合の最後の作業の完了時刻が最小となるようにするには，どのように機械を割り振ればよいでしょうか.

この問題は，先行制約が以下の様に変更されます.

先行制約	$act_{a,1} \prec act_{a,3}$	仕事 a の作業 1 は，仕事 a の作業 3 に先行する
	$act_{a,3} \prec act_{a,2}$	仕事 a の作業 3 は，仕事 a の作業 2 に先行する
	$act_{b,1} \prec act_{b,2}$	仕事 b の作業 1 は，仕事 b の作業 2 に先行する
	$act_{b,2} \prec act_{b,3}$	仕事 b の作業 2 は，仕事 b の作業 3 に先行する
	$act_{c,2} \prec act_{c,1}$	仕事 c の作業 2 は，仕事 c の作業 1 に先行する
	$act_{c,1} \prec act_{c,3}$	仕事 c の作業 1 は，仕事 c の作業 3 に先行する

上記の先行制約をデータから与えられるように SIMPLE のモデルとデータを修正します.

```
// 作業集合
Set J; // 仕事
Element j(set=J);
Set S; // 作業
Element s(set=S);
// モード集合
Set M;
Element m(set=M);
Set AvailMode(name="AvailMode", index=(j,s)); // 各仕事のオペレーションにおいて処理されるモード
// 資源集合
Set R;
Element r(set=R);
// 作業時間集合
Set D; // 各モードの作業時間の最大
Element d(set=D);
// 期間集合
Set T; // スケジュール期間
T = "0 .. 30";
Element t(set=T);

// アクティビティ(変数)
Activity act(name="act", index=(j,s), mode=AvailMode[j,s]);

// 定数
// 必要資源量
ResourceRequire req(name="req", mode=M, resource=R, duration=D);
// 資源供給量
ResourceCapacity cap(name="cap", resource=R, timeStep=T);
cap[r,t] = 1;

// 目的関数(最後の作業の完了時刻の最小化)
Objective f(type=minimize);
f = completionTime;

// 先行制約
Set Prec(name="Prec", dim=3);
Element u(set=S);
Element v(set=S);
act[j,u] < act[j,v], (j,u,v)<Prec;
```

```
// 求解最大時間の設定
options.maxtim = 15;

// 求解
solve();

// 結果の標準出力
simple_printf("act[%s,%d] = %d¥n", j, s, act[j,s].startTime);
```

データファイルの 1 つが以下の様に修正されます.

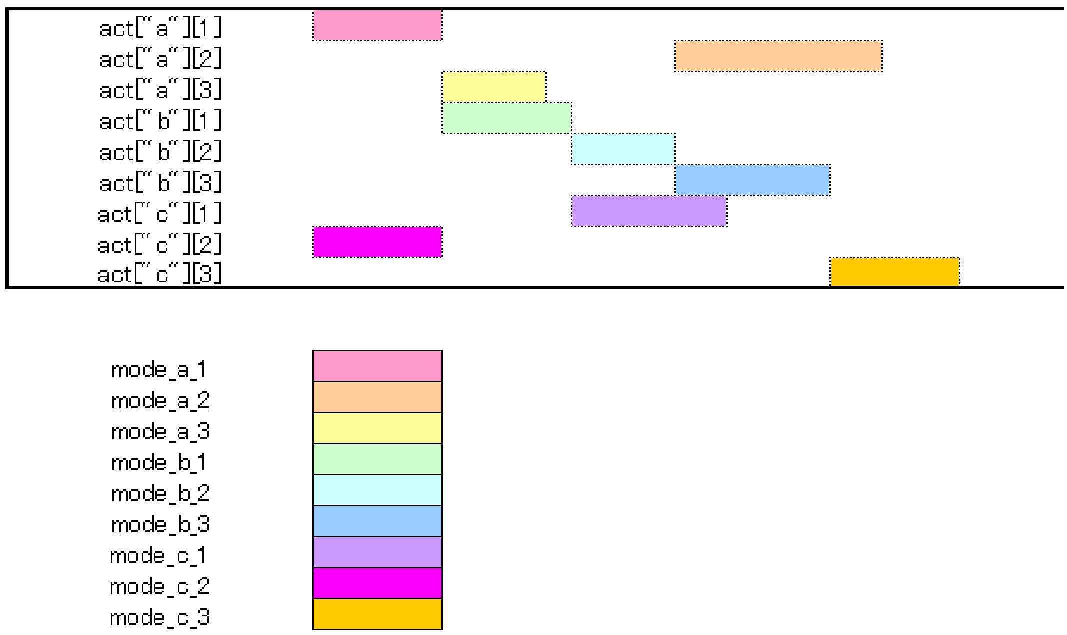
```
AvailMode =
[a,1] mode_a_1
[a,2] mode_a_2
[a,3] mode_a_3
[b,1] mode_b_1
[b,2] mode_b_2
[b,3] mode_b_3
[c,1] mode_c_1
[c,2] mode_c_2
[c,3] mode_c_3
;

Prec =
a 1 3
a 3 2
b 1 2
b 2 3
c 2 1
c 1 3
;
```

実行すると，各作業の開始時刻が以下のように出力されます.

```
act["a",1] = 0
act["a",2] = 14
act["a",3] = 5
act["b",1] = 5
act["b",2] = 10
act["b",3] = 14
act["c",1] = 10
act["c",2] = 0
act["c",3] = 20
```

2.25.2 節同様，結果のガントチャート出力を以下のように行います．

例題（ジョブショップ問題）の解の出力例：



2.25.4 リスケジューリング問題

実際の現場では，ある時機械が故障した等の突発事故が起こる事がしばしばあります．その場合，過去のスケジュールを固定し，条件を変更した後，再度スケジュールを組み直します．このような問題をリスケジューリング問題と呼ぶ事にします．

（例題）リスケジューリング問題
2.23.3 節のジョブショップ問題を解いた結果，各作業の開始時刻は，

	作業 1(機械 1)	作業 2(機械 2)	作業 3(機械 3)
仕事 a	0	14	5
仕事 b	5	10	14
仕事 c	10	0	20

となりました．このスケジュールに基づいて機械を運転していた所，10 時間が過ぎた所で，機械 2 が故障し，5 時間停止する事になりました．この場合の最後の作業の完了時刻が最小となるようにはどのようにスケジュールを組み直せばよいでしょうか．

この問題を記述するには，過去のスケジュールを固定する為，アクティビティ固定関数（fixActivity）を使用する必要があります．また，未来のスケジュールが過去に来ないようにする為，全ての作業に先行する sourceActivity(自動で定義されている)という先行制約を記述する必要があります．

以上のことを踏まえると，制約としては新たに以下のものが加わります．

アクティビティの開始時刻固定	$fixActivity(act_{a,1}.startTime)$	仕事 a の作業 1 の開始時刻を 0 に固定する
	$fixActivity(act_{a,3}.startTime)$	仕事 a の作業 3 の開始時刻を 5 に固定する
	$fixActivity(act_{b,1}.startTime)$	仕事 b の作業 1 の開始時刻を 5 に固定する
	$fixActivity(act_{c,2}.startTime)$	仕事 c の作業 2 の開始時刻を 0 に固定する
アクティビティのモード固定	$fixActivity(act_{a,1})$	仕事 a の作業 1 のモードを mode_a_1 に固定する
	$fixActivity(act_{a,3})$	仕事 a の作業 3 のモードを mode_a_3 に固定する
	$fixActivity(act_{b,1})$	仕事 b の作業 1 のモードを mode_b_1 に固定する
	$fixActivity(act_{c,2})$	仕事 c の作業 2 のモードを mode_c_2 に固定する
先行制約	$sourceActivity \prec act_{a,2}, 10$	仕事 a の作業 2 は，時刻 10 以前には処理されない
	$sourceActivity \prec act_{b,2}, 10$	仕事 b の作業 2 は，時刻 10 以前には処理されない
	$sourceActivity \prec act_{b,3}, 10$	仕事 b の作業 3 は，時刻 10 以前には処理されない
	$sourceActivity \prec act_{c,1}, 10$	仕事 c の作業 1 は，時刻 10 以前には処理されない
	$sourceActivity \prec act_{c,3}, 10$	仕事 c の作業 3 は，時刻 10 以前には処理されない

また，機械 2 は，時刻 10 から 5 時間停止するので，資源が以下の様に修正されます.

資源	machine1	機械 1，供給量:常に 1
	machine2	機械 2，供給量:時刻 10 以上 15 未満 0，その他常に 1
	machine3	機械 3，供給量:常に 1

これを SIMPLE で記述すると以下の様になります.

```
// 作業集合
Set J; // 仕事
Element j(set=J);
Set S; // 作業
Element s(set=S);
// モード集合
Set M;
Element m(set=M);
Set AvailMode(name="AvailMode", index=(j,s)); // 各仕事のオペレーション
において処理されるモード
// 資源集合
Set R;
Element r(set=R);
// 作業時間集合
Set D; // 各モードの作業時間の最大
Element d(set=D);
// 期間集合
Set T; // スケジュール期間
T = "0 .. 30";
Element t(set=T);

// アクティビティ(変数)
Activity act(name="act", index=(j,s), mode=AvailMode[j,s]);

// 必要資源量（定数)
ResourceRequire req(name="req", mode=M, resource=R, duration=D);
// 資源供給量（定数)
ResourceCapacity cap(name="cap", resource=R, timeStep=T);
cap[r,t] = 1;
cap["machine2", t] = 0, 10<=t<=14; // 故障に対応

// 目的関数(最後の作業の完了時刻の最小化)
Objective f(type=minimize);
f = completionTime;

// 先行制約
Set Prec(name="Prec", dim=3);
Element u(set=S);
Element v(set=S);
act[j,u] < act[j,v], (j,u,v)<Prec;
```

```
/* --- リスケジューリングの為の制約 --- */
// 過去(0 .. 10) のスケジュールを固定
Set FixAct(name="FixAct", dim=2);
Parameter fixTime(name="fixTime", index=(j,s));
act[j,s].startTime = fixTime[j,s], (j,s) < FixAct;
Parameter fixMode(name="fixMode", index=(j,s));
act[j,s] = fixMode[j,s], (j,s) < FixAct;
fixActivity(act[j,s].startTime, (j,s) < FixAct);
fixActivity(act[j,s], (j,s) < FixAct);
// 過去のジョブ以外はステップ 10 以前に来てはならない
Set NotFixAct(name="NotFixAct", dim=2);
NotFixAct = setOf((j,s), (j < J, s < S)) - FixAct;
sourceActivity < act[j,s], (j,s) < NotFixAct, 10;
/* ------------------------------------ */

options.maxtim = 15;
solve();

simple_printf("act[%s,%d] = %d¥n", j, s, act[j,s].startTime);
```

データファイルの 1 つが以下の様に修正されます.

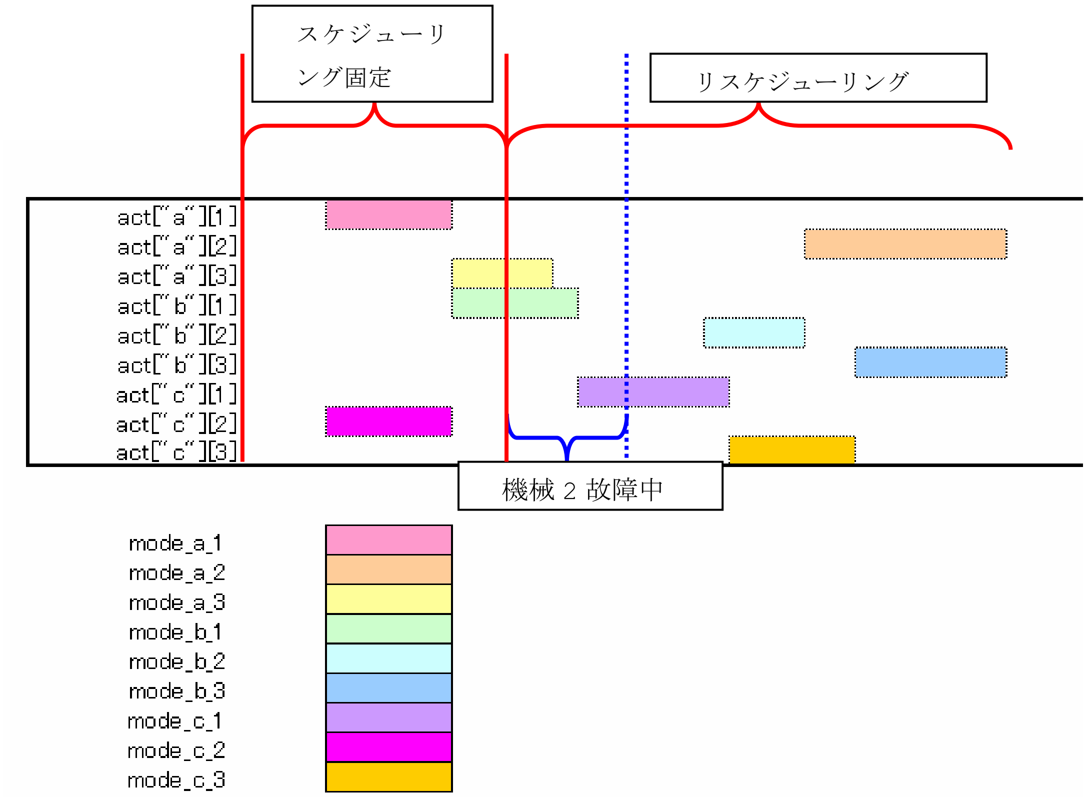
```
AvailMode =
[a,1] mode_a_1
[a,2] mode_a_2
[a,3] mode_a_3
[b,1] mode_b_1
[b,2] mode_b_2
[b,3] mode_b_3
[c,1] mode_c_1
[c,2] mode_c_2
[c,3] mode_c_3
;

Prec =
a 1 3
a 3 2
b 1 2
b 2 3
c 2 1
c 1 3
;

FixAct =
a 1
a 3
b 1
c 2
;

fixTime = [a,1] 0 [a,3] 5 [b,1] 5 [c,2] 0;
fixMode = [a,1] mode_a_1 [a,3] mode_a_3 [b,1] mode_b_1 [c,2] mode_c_2;
```

実行すると，各作業の開始時刻が以下のように出力されます．

```
act["a",1] = 0
act["a",2] = 19
act["a",3] = 5
act["b",1] = 5
act["b",2] = 15
act["b",3] = 21
act["c",1] = 10
act["c",2] = 0
act["c",3] = 16
```

2.25.3 節同様，結果のガントチャート出力を以下のように行います．

例題（リスケジューリング問題）の解の出力例：



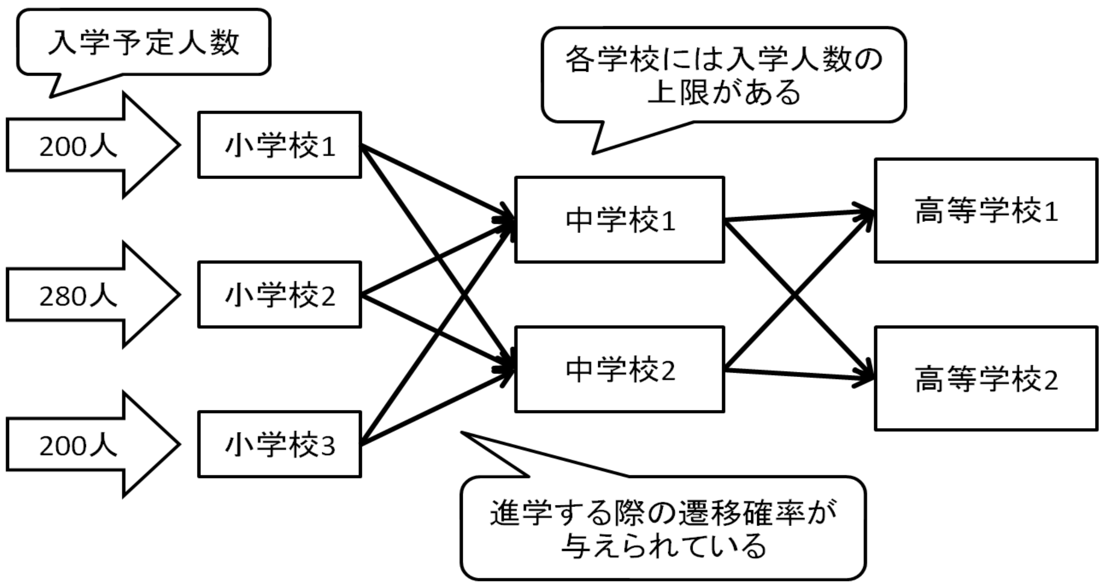
2.26 遷移確率行列

ここでは，遷移確率行列を用いた問題を紹介します．NUOPT V13 から導入された Vector/Matrix クラスを利用すると，遷移確率行列を用いた問題が記述し易くなります．

（例題）

ある地域には小学校が 3 校，中学校が 2 校，高等学校が 2 校ある．それぞれの学校には入学者の人数制限が決められており，どの学校へ進学するかの遷移確率がわかっているとする．

いま，ある年度の小学校入学予定者人数を予測すると，小学校 2 の入学者上限を大幅に超えてしまったため，小学校 2 から小学校 1，小学校 3 への入学者人数の調整を行うことにした．この年度の児童が与えられた遷移確率に従って中学校，高等学校に進学すると仮定したとき，入学者上限制約を満たし，かつ最小の人数調整で解決したい場合，どのような人数調整を行えばよいか．



この問題を NUOPT で解くために定式化を行います．

変数は，小学校 2 から小学校 1 への調整人数 x と小学校 2 から小学校 3 への調整人数 y とします．目的関数は調整人数の最小化ですので $x+y$ となります．また，現実問題を考慮すると変数は整数値をとるが，ここでは連続変数として扱い，得られた解の値を整数値に丸めて取り扱うことにします．

次に制約条件を考えます．まず，調整人数についての制約として，非負制約，および調整人数の総和が小学校 2 の入学予定人数を超えてはならないという制約があります．また，各遷移確

率に従って進学をした場合にそれぞれの学校の入学人数制限を満たすという制約があります.

以上をふまえて次のように定式化されます.

集合	$E = \{E1, E2, E3\}$	小学校の集合
	$J = \{J1, J2\}$	中学校の集合
	$H = \{H1, H2\}$	高等学校の集合
定数	$EJ_{ej}, e \in E, j \in J$	各小学校から各中学校への遷移確率行列
	$JH_{jh}, j \in J, h \in H$	各中学校から各高等学校への遷移確率行列
	$flow_e, e \in E$	各小学校への入学予定人数ベクトル
	$Eup_e, e \in E$	各小学校の入学者上限ベクトル
	$Jup_j, j \in J$	各中学校の入学者上限ベクトル
	$Hup_h, h \in H$	各高等学校の入学者上限ベクトル
変数	x	小学校２から小学校１への調整人数
	y	小学校２から小学校３への調整人数
	$X_e, e \in E$	人数調整後の各小学校への入学人数ベクトル
	$X_{E1} = flow_{E1} + x$	
	$X_{E2} = flow_{E2} - x - y$	
	$X_{E3} = flow_{E3} + y$	
目的関数（最小化）	$x + y$	調整人数の最小化
制約	$x, y \geq 0$	非負制約
	$x + y \leq flow_{E2}$	調整人数の総和は小学校 2 への入学予定人数を超えてはならない
	$X \leq Eup$	各小学校の入学人数の制限

$(X^T \cdot EJ)^T \le Jup$	各中学校の入学人数の制限
$(X^T \cdot EJ \cdot JH)^T \le Hup$	各高等学校の入学人数の制限

次に，ベクトルと行列をデータファイルから与える SIMPLE モデルを示します.

```
// 集合と添え字
Set E; Element e(set=E); // 小学校
Set J; Element j(set=J); // 中学校
Set H; Element h(set=H); // 高等学校

// 行列パラメータとベクトルパラメータ
Matrix EJ((e,j)), JH((j,h));
Vector flow(e), Eup(e), Jup(j), Hup(h);

// 変数
Variable  x,y;
0 <= x;
0 <= y;
x + y <= flow["E2"];

// 変数ベクトル
Vector X(e);
X["E1"] = flow["E1"] + x;
X["E2"] = flow["E2"] - x - y;
X["E3"] = flow["E3"] + y;

// 制約
X <= Eup;
trans(trans(X) * EJ) <= Jup;
trans(trans(X) * EJ * JH) <= Hup;

// 目的関数値
Objective obj(type=minimize);
obj = x + y;
// 求解
solve();

// 出力
x.val.print();
y.val.print();
```

データファイル（.csv 形式）は以下のようになります.

各小学校への入学人数と，各学校への進学の遷移確率行列

```
e,flow
E1,200
E2,280
E3,200
```

```
EJ,J1,J2
E1,0.8,0.2
E2,0.4,0.6
E3,0.3,0.7
```

```
JH,H1,H2
J1,0.6,0.4
J2,0.45,0.55
```

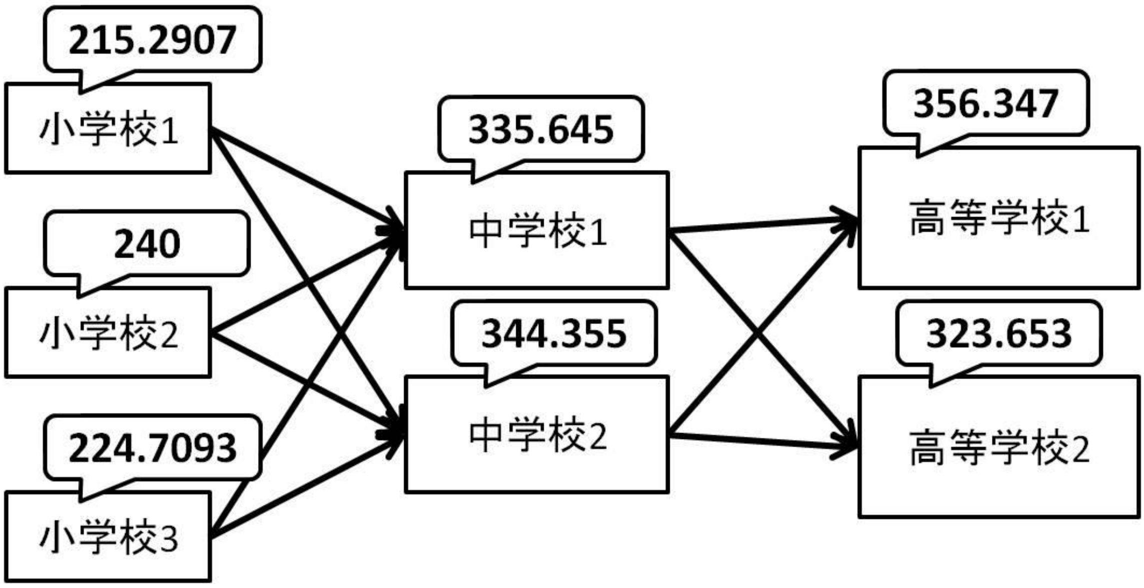
各学校の入学上限

```
e,Eup
E1,240
E2,240
E3,240
```

```
j,Jup
J1,360
J2,360
```

```
h,Hup
H1,360
H2,360
```

このモデルを実行すると， $x = 15.2907$ ， $y = 24.7093$ が求まりますので，整数値に丸めて小学校 2 から小学校 1 への調整人数 15 人と小学校 2 から小学校 3 への調整人数 25 人が得られます．また，その結果，以下のように進学予定人数がわかります.



2.27 ムーア・ペンローズ一般逆行列

NUOPT V13 から新たに導入された Vector/Matrix クラスの使い方に慣れるため，最適化問題ではありませんが，ムーア・ペンローズ一般逆行列を求めてみましょう．この問題は[8]を参考文献としています．

（例題）
次の行列 A の，ムーア・ペンローズ一般逆行列 A^{+} を求めよ．

$$A = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

A を $m \times n$ 行列とする線型方程式 $A\boldsymbol{x} = \boldsymbol{y}$ が解 $\boldsymbol{x}$ を持つような $\boldsymbol{y}$ に対して，$\boldsymbol{x} = A^{-}\boldsymbol{y}$ がこの方程式の一つの解となる場合，$n \times m$ 行列 A^{-} を A の一般逆行列といいます．

A の一般逆行列は，次の式から求めることができます．

$$AA^{-}A = A$$

一般的に A^{-} は一意に定まりません．

行列 M の任意の一般逆行列を M^{-} と表すとき，A のムーア・ペンローズ一般逆行列 A^{+} は

$$A^{+} = A^{\mathrm{T}}A\left(A^{\mathrm{T}}AA^{\mathrm{T}}A\right)^{-}A^{\mathrm{T}}$$

と書くことができ，これは一意に定まります．ここで，A^{T} は A の転置行列です．

よって，$A^{\mathrm{T}}AA^{\mathrm{T}}A$ に対する一般逆行列を一つ求めればよいことがわかります．

以上より，次のように定式化できます．

集合	M	行
	N	列
定数	$A_{mn}, m \in M, n \in N$	定数行列成分
変数	$X_{ij}, i \in N, j \in N$	変数行列成分
目的関数		この問題に目的関数は無い
制約条件	$A^{\mathrm{T}}AA^{\mathrm{T}}A\,X\,A^{\mathrm{T}}AA^{\mathrm{T}}A = A^{\mathrm{T}}AA^{\mathrm{T}}A$	X は $A^{\mathrm{T}}AA^{\mathrm{T}}A$ の一般逆行列

定式化した結果を SIMPLE で記述すると以下のようになります.

```
Set M, N;
Element m(set = M), i(set = N), j(set = N);

// 定数行列
Matrix A((m,i)), B((i,j));
B = trans(A)*A*trans(A)*A;

// 変数
Variable x(index = (i,j));
Matrix X((i,j));
X[i,j] = x[i,j];// 行列 X に変数 x を詰め込む

// 制約条件
B*X*B == B;

solve();

// ムーア・ペンローズ一般逆行列
Matrix A_plus((i,m));
A_plus = trans(A)*A*X*trans(A);

for(int k = 1; k <= N.card(); k++){
simple_printf("%5.4f ", A_plus[k,m].val);
simple_printf("¥n");
}
```

csv 形式の入力ファイルは以下のようになります,

```
A,1,2,3
1,1,-1,1
2,1,1,-1
3,-1,1,1
4,1,1,1
```

このモデルを NUOPT で実行すると，最後に

```
0.2500 0.2500 -0.2500 0.2500
-0.2500 0.2500 0.2500 0.2500
0.2500 -0.2500 0.2500 0.2500
```

という表示がされて，この例題の答えが

$$A^{+}=\begin{pmatrix} 1/4 & 1/4 & -1/4 & 1/4 \\ -1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & -1/4 & 1/4 & 1/4 \end{pmatrix}$$

であることがわかります.

参考文献

[1] George B. Dantzig, Mukund N.Thapa, Linear Programming 1:Introduction, Springer, 1997

[2] 福島雅夫, 数理計画入門, 朝倉書店, 1996

[3] 刀根薫, 数理計画, 朝倉書店, 2007

[4] H.P.ウイリアムス, 数理計画モデルの作成法, 産業図書, 1995

[5] Fisher,R.A., The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics, 7, 179-188, 1936

[6] Frank J.Fabozzi, Petter N.Kolm, Dessislava A. Pachamanova, Sergio M. Focardi, Robust Portfolio Optimization and Management, John Wiley & Sons, Inc., 2007

[7] Christoph Helmberg, Semidefinite Programming for Combinatorial Optimization, Konrad-Zuse-Zentrum fur Informationstechnik Berlin,January 2000

[8] 柳井春夫・竹内啓, 射影行列 一般逆行列 特異値分解, 東京大学出版会

索引

W

い

え

お

か

こ

さ

し

す

せ

そ

た

ち

つ

て

な

に

は

ひ

ふ

へ

ほ

ま

み

む

も

ゆ

ら

り

ろ

わ