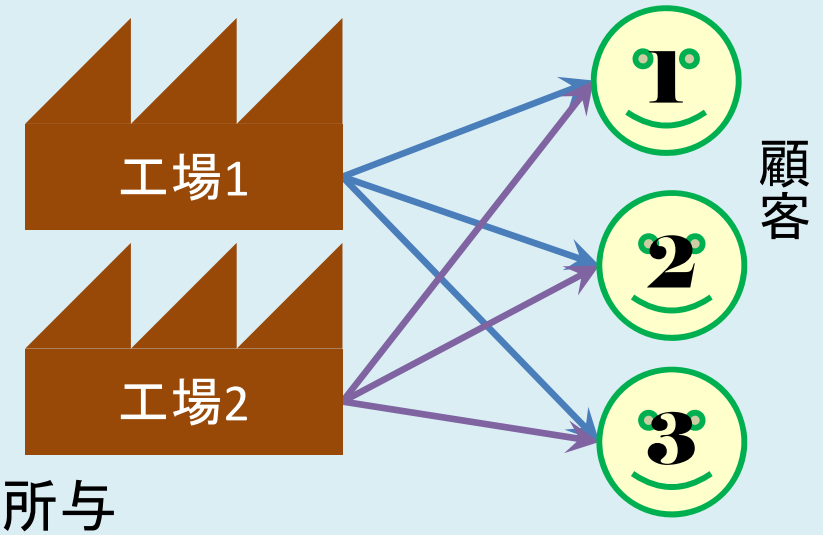
2026/5/25 Mon.

問題解決

最適化活用
整数計画ソルバーの利用

堀田 敬介

輸送問題を解く



- 輸送問題の最適化（例1）
  - 2工場で製品を供給できる
  - 3人の顧客がいて，需要がある
  - 2工場→3顧客への単位あたり輸送コストが所与
  - 輸送コストが最小となる配送計画をたてよ

	需要	50	80	60
供給	工場＼顧客	1	2	3
120	1	3	2	4
130	2	5	6	5

- 定式化（変数設定）
  - 変数 x_{ij} … 工場i→顧客jへの輸送量
  - 変数行列 X

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$$

- 定式化（係数表記）
  - 工場の供給を表す係数ベクトル s
  - 顧客の需要を表す係数ベクトル d
  - 輸送コストを表す係数行列 C

$$s = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 120 \\ 130 \end{pmatrix}$$

$$d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 50 \\ 80 \\ 60 \end{pmatrix}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} = \begin{pmatrix} 3 & 2 & 4 \\ 5 & 6 & 5 \end{pmatrix}$$

輸送問題を解く

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 120 \\ 130 \end{pmatrix}$$

$$\boldsymbol{d} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} 50 \\ 80 \\ 60 \end{pmatrix}$$

$$\boldsymbol{C} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} = \begin{pmatrix} 3 & 2 & 4 \\ 5 & 6 & 5 \end{pmatrix}$$

➢ 定式化（ベタ表記）

$$\begin{aligned} \text{min. } & 3x_{11}+2x_{12}+4x_{13}+5x_{21}+6x_{22}+5x_{23} \\ \text{s. t. } & x_{11}+x_{12}+x_{13} \leqq 120 \\ & x_{21}+x_{22}+x_{23} \leqq 130 \\ & x_{11}+x_{21} = 50 \\ & x_{12}+x_{22} = 80 \\ & x_{13}+x_{23} = 60 \\ & x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \geqq 0 \end{aligned}$$

➢ 定式化（Σ表記）

$$\begin{aligned} \text{min.} & \sum_{i=1}^{2}\sum_{j=1}^{3} c_{ij}x_{ij} \\ \text{s. t. } & \sum_{j=1}^{3} x_{ij} \leq s_i \ (i = 1,2) \\ & \sum_{i=1}^{2} x_{ij} = d_j \ (j = 1, \dots, 3) \\ & x_{ij} \geq 0 \ (i = 1,2; j = 1, \dots, 3) \end{aligned}$$

➢ 定式化（行列表記）

$$\begin{aligned} \text{min.} & \boldsymbol{C} \odot \boldsymbol{X} \\ \text{s. t. } & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{s} \\ & \boldsymbol{e}^T\boldsymbol{X} = \boldsymbol{d} \\ & \boldsymbol{X} \geq \boldsymbol{0} \end{aligned}$$

- ✓ $\odot$ はアダマール積(Hadamard product)
- ✓ $\boldsymbol{e}$は要素が全て1の適当な次元のベクトル

輸送問題：How to use CPLEX?

- 新規プロジェクトの作成
  - ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
  - ② [プロジェクト名] を記入（例：Transportation）し，3カ所にチェックする
    - ☑ デフォルトの実行構成の追加
    - ☑ モデルの作成
    - ☑ データの作成
  - ③ [終了]をクリック

プロジェクト名は自由だが，半角英数で何の問題を解こうとしているのかが分かる名前が良い

- プロジェクト内のいくつかの名前を変更
  - ✓ [構成1] → [config1]　※日本語を英語に変更しないと実行時エラーになる
  - ✓ モデルファイル [Transportation.mod] → [tr.mod]
  - ✓ データファイル [Transportation.dat] → [trex1.dat]

- モデルファイル・データファイルを記述し保存（次ページ参照）
- [config1]にモデルファイルとデータファイルをセットする

輸送問題：How to use CPLEX?

- tr.mod

```
int i_max = ...;// 行の添え字の最大値
int j_max = ...;// 列の添え字の最大値

range I = 1..i_max;// 行の添え字の範囲 [1..i_max]を指定
range J = 1..j_max;// 列の添え字の範囲 [1..j_max]を指定

int d[J] = ...;// 需要ベクトル設定 d = [d1,d2,d3]
int s[I] = ...;// 供給ベクトル設定 s = [s1,s2]
int c[I,J] = ...;// 輸送コスト設定 C サイズI×Jの行列

dvar float+ x[I,J];// 変数宣言：変数ベクトル(size: I×J)

minimize
  sum(i in I) sum(j in J) c[i,j]*x[i,j];
subject to{
  forall(i in I) {
    sum(j in J) x[i,j] <= s[i];
  };
  forall(j in J) {
    sum(i in I) x[i,j] == d[j];
  };
};
```

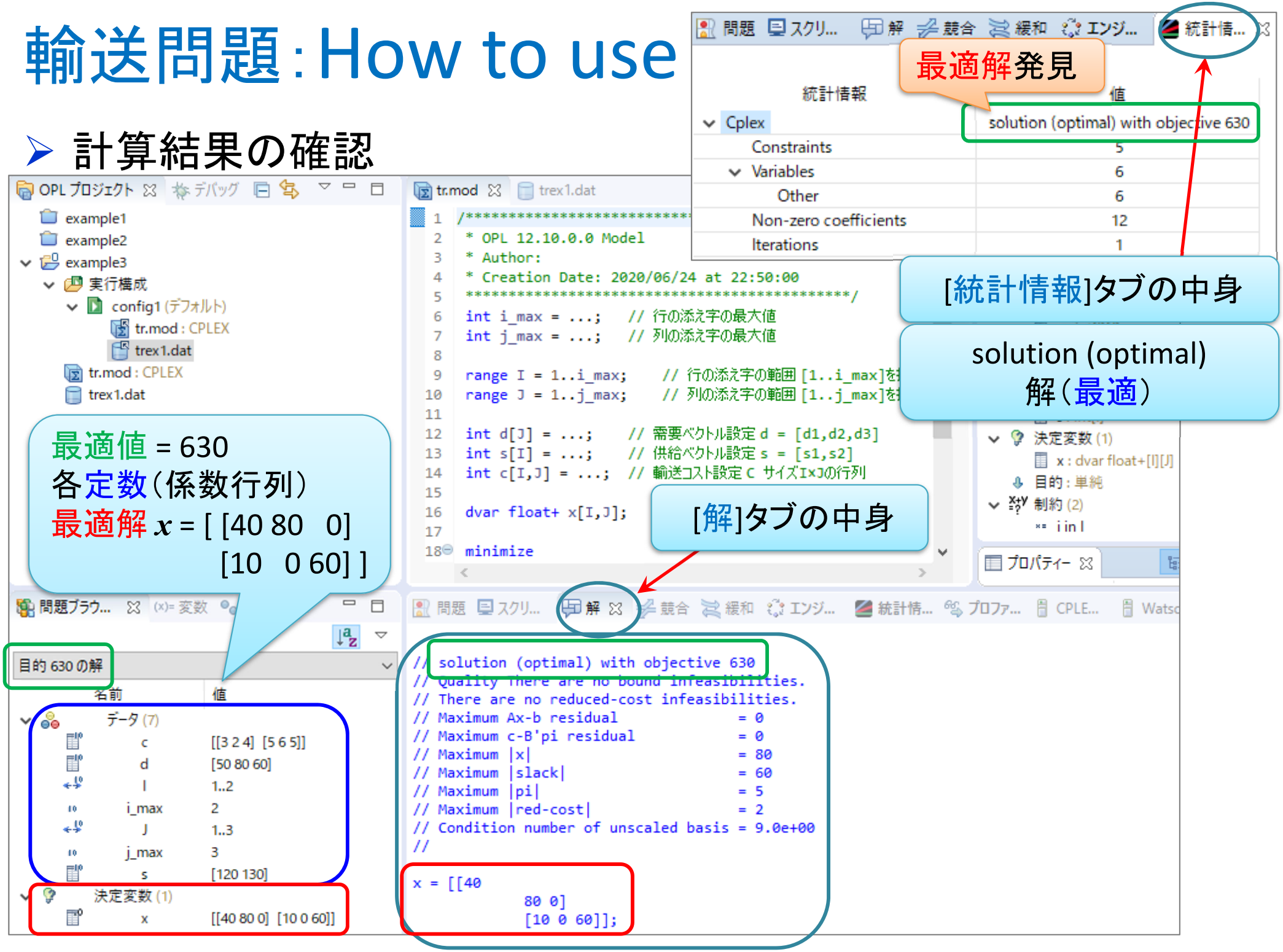
輸送問題：How to use CPLEX?

- trex1.dat

```
i_max = 2;// 行数の最大値を指定
j_max = 3;// 列数の最大値を指定

d = [50 80 60];

c = [
[3 2 4]
[5 6 5]
];

s = [
120
130
];
```

輸送問題：How to use

➢ 計算結果の確認

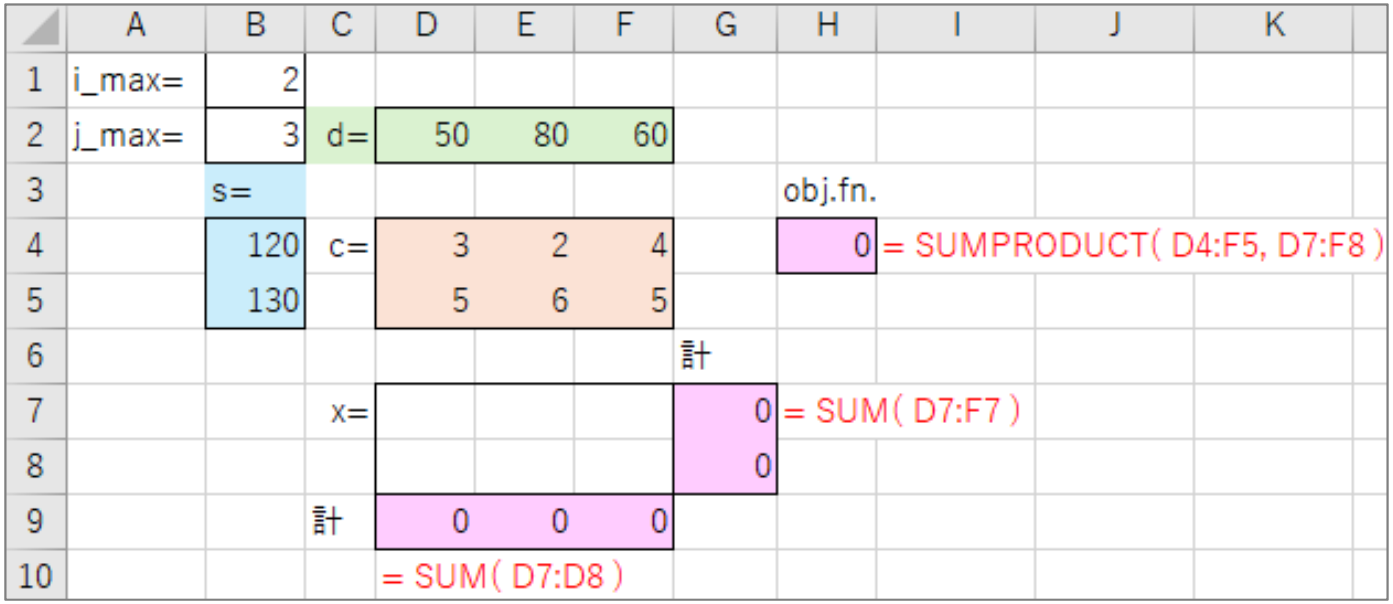


輸送問題：How to use CPLEX?

- 演習1）データファイル（trex1.dat）をExcel入出力に変更せよ
  - Excelファイル [trex.xlsx] を作成し，データを適切に記述
  - データファイル [trex1.dat] をコピーして [trex1xlsx.dat] 作成し，修正

輸送問題：How to use CPLEX?

- 演習1）解答例
  - trex.xlsx
    [Sheet1]



	A	B	C	D	E	F	G	H	I	J	K
1	i_max=	2									
2	j_max=	3	d=	50	80	60					
3		s=						obj.fn.			
4		120	c=	3	2	4		0	= SUMPRODUCT(D4:F5, D7:F8)		
5		130		5	6	5					
6							計				
7			x=				0	= SUM(D7:F7)			
8							0				
9			計	0	0	0					
10				= SUM(D7:D8)							

  - trex1xlsx.dat

```
SheetConnection sheet("trex.xlsx");

i_max from SheetRead(sheet, "Sheet1!B1");
j_max from SheetRead(sheet, "Sheet1!B2");
d from SheetRead(sheet, "Sheet1!D2:F2");
s from SheetRead(sheet, "Sheet1!B4:B5");
c from SheetRead(sheet, "Sheet1!D4:F5");

x to SheetWrite(sheet, "Sheet1!D7:F9");
```

輸送問題を解く

- 輸送問題の最適化（例2）
  - 工場が製品を供給，顧客が需要，工場→顧客コスト
  - 輸送コスト最小の配送計画を

	需要	40	30	70	80	60	50
供給	工場＼顧客	1	2	3	4	5	6
80	1	1	2	4	3	1	2
90	2	2	1	5	2	4	1
100	3	3	6	2	1	2	3
110	4	4	3	5	2	3	2

- 最適化問題で定式化（ベタ・Σ表記）する
  - 変数 x_{ij} … 工場i→顧客jへの輸送量
  - 係数vector ***s***:供給supply, ***d***:需要demand
  - 係数matrix ***C***:輸送コストcost
- CPLEXで解く（モデルファイル[tr.mod]は共通で使えるので[trex2.dat]のみ作り解く）

輸送問題：How to use CPLEX?

- trex2.dat

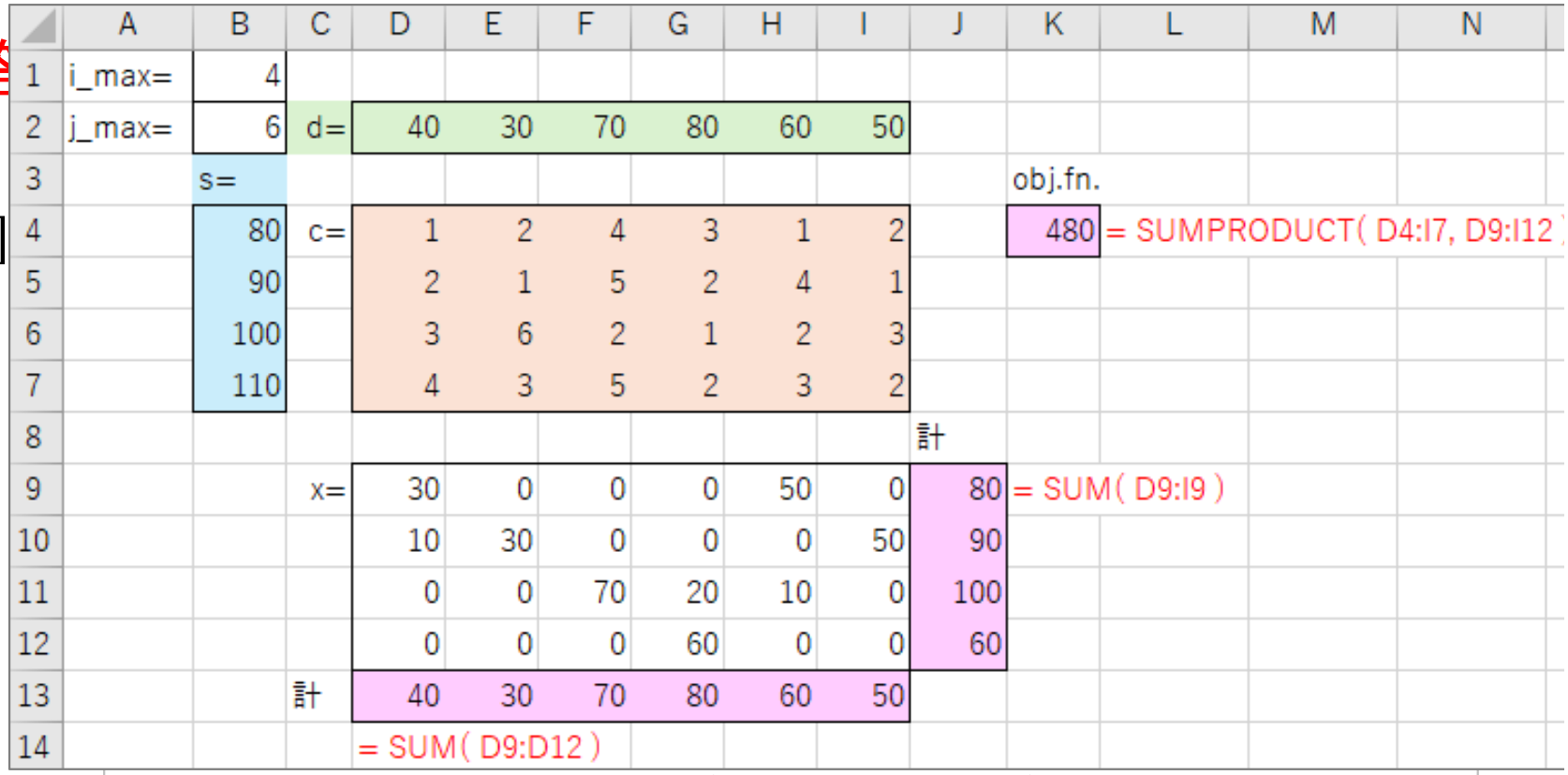
```
i_max = 4;// 行数の最大値を指定
j_max = 6;// 列数の最大値を指定

d = [40 30 70 80 60 50];
s = [80 90 100 110];
c = [
 [1 2 4 3 1 2]
 [2 1 5 2 4 1]
 [3 6 2 1 2 3]
 [4 3 5 2 3 2]
];
```

輸送問題：How to use CPLEX?

- 演習2）データファイル（trex2.dat）をExcel入出力に変更せよ
  - Excelファイル [trex.xlsx] にシート[Sheet2] を作成し，データを適切に記述
  - データファイル [trex1xlsx.dat] をコピーして [trex2xlsx.dat] 作成し，修正

輸送問題：How to use CPLEX?

- 演習2）解答
  - trex.xlsx
    [Sheet2]



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	i_max=	4												
2	j_max=	6	d=	40	30	70	80	60	50					
3		s=									obj.fn.			
4		80	c=	1	2	4	3	1	2		480	= SUMPRODUCT(D4:I7, D9:I12)		
5		90		2	1	5	2	4	1					
6		100		3	6	2	1	2	3					
7		110		4	3	5	2	3	2					
8										計				
9			x=	30	0	0	0	50	0	80	= SUM(D9:I9)			
10				10	30	0	0	0	50	90				
11				0	0	70	20	10	0	100				
12				0	0	0	60	0	0	60				
13			計	40	30	70	80	60	50					
14				= SUM(D9:D12)										

  - trex2xlsx.dat

```
SheetConnection sheet("trex.xlsx");

i_max from SheetRead(sheet, "Sheet2!B1");
j_max from SheetRead(sheet, "Sheet2!B2");
d from SheetRead(sheet, "Sheet2!D2:I2");
s from SheetRead(sheet, "Sheet2!B4:B7");
c from SheetRead(sheet, "Sheet2!D4:I7");

x to SheetWrite(sheet, "Sheet2!D9:I12");
```

輸送問題：How to use gurobi?

- 輸送問題の最適化（例1）
  - Google colaboratory / Python & gurobi

```
!pip install gurobipy
```

```
d = [50,80,60]     # 顧客3人の需要
c = [[3,2,4],      # 工場i→顧客jへの輸送コスト
     [5,6,5]]
s = [120,130]      # 工場2つの供給量
I = range(len(s))# 工場2つの範囲I={0,1}
J = range(len(d))# 顧客3人の範囲J={0,1,2}

from gurobipy import *
m = Model("Transportation")  # モデルの設定

x = {}
for i in I:
  for j in J:
    x[i,j] = m.addVar(lb=0.0, vtype="C",name="x(%s,%s)" %(i+1,j+1)) # 変数設定
for i in I:
  m.addConstr(quicksum(x[i,j] for j in J) <= s[i])  # 供給制約
for j in J:
  m.addConstr(quicksum(x[i,j] for i in I) == d[j])   # 需要制約
m.setObjective(quicksum(c[i][j]*x[i,j] for i in I for j in J), GRB.MINIMIZE)

m.optimize()  # 最適化実行
m.printAttr('X') # 最適解の表示
```

	需要	50	80	60
供給	工場＼顧客	1	2	3
120	1	3	2	4
130	2	5	6	5

輸送問題：How to use gurobi?

- 輸送問題の最適化（例1）
  - 実行結果（一部）

```
Solved in 1 iterations and 0.02 seconds (0.00 work units)
Optimal objective 6.300000000e+02
Variable X
-------------------------
x(1,1) 40
x(1,2) 80
x(2,1) 10
x(2,3) 60
```

輸送問題：How to use gurobi?

- 輸送問題の最適化（例2）
  - Google colaboratory / Python & gurobi

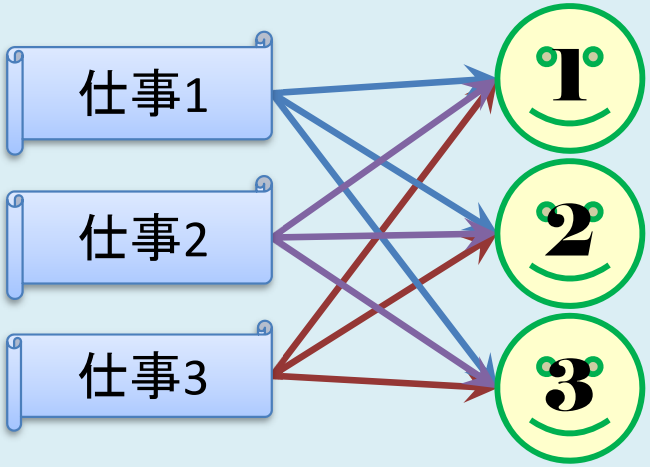
```
d = [40, 30, 70, 80, 60, 50]
c = [[1, 2, 4, 3, 1, 2],
     [2, 1, 5, 2, 4, 1],
     [3, 6, 2, 1, 2, 3],
     [4, 3, 5, 2, 3, 2]]
s = [80, 90, 100, 110]
I = range(len(s))
J = range(len(d))

from gurobipy import *

    …（中略）…   例1と同じ

m.optimize()   # 最適化実行
m.printAttr('X') # 最適解の表示
```

	需要	40	30	70	80	60	50
供給	工場＼顧客	1	2	3	4	5	6
80	1	1	2	4	3	1	2
90	2	2	1	5	2	4	1
100	3	3	6	2	1	2	3
110	4	4	3	5	2	3	2

  - 実行結果（一部）

```
Solved in 4 iterations and 0.01 seconds (0.00 work units)
Optimal objective 4.800000000e+02
Variable X
-------------------------
x(1,1) 30 x(1,5) 50 x(2,1) 10 x(2,2) 30 x(2,6) 50
x(3,3) 70 x(3,4) 20 x(3,5) 10 x(4,4) 60
```

割当問題を解く



- 割当問題の最適化（例1）
  - 今すべき仕事は3つあり，3人の部下がいる
  - 各部下は1つの仕事を引き受けられる
  - 上司は各仕事を任せたときの部下の出来具合を5段階で評価済（右下表）
  - この部署のパフォーマンスが最大になるように部下へ仕事を割り当てよ

仕事＼部下	1	2	3
1	2	2	4
2	5	3	2
3	4	1	3

- 定式化（変数設定）
  - 0-1変数 $x_{ij} = \begin{cases} 1 \ldots \text{仕事}i\text{を部下}j\text{へ割り当てる} \\ 0 \ldots \text{仕事}i\text{を部下}j\text{へ割り当てない} \end{cases}$
  - 変数行列X

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$

- 定式化（係数表記）
  - 評価を表す係数行列 $\boldsymbol{C}$

$$\boldsymbol{C} = \begin{pmatrix} 2 & 2 & 4 \\ 5 & 3 & 2 \\ 4 & 1 & 3 \end{pmatrix}$$

割当問題を解く

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}$$

$$\boldsymbol{C} = \begin{pmatrix} 2 & 2 & 4 \\ 5 & 3 & 2 \\ 4 & 1 & 3 \end{pmatrix}$$

➢ 定式化（ベタ表記）

$$\begin{aligned} \max.\ & 2x_{11}+2x_{12}+4x_{13}+5x_{21}+3x_{22}+2x_{23}+4x_{31}+1x_{32}+3x_{33} \\ \text{s. t.}\ & x_{11}+x_{12}+x_{13}=1 \\ & x_{21}+x_{22}+x_{23}=1 \\ & x_{31}+x_{32}+x_{33}=1 \\ & x_{11}+x_{21}+x_{31} \leqq 1 \\ & x_{12}+x_{22}+x_{32} \leqq 1 \\ & x_{13}+x_{23}+x_{33} \leqq 1 \\ & x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \in \{0,1\} \end{aligned}$$

➢ 定式化（Σ表記）

$$\begin{aligned} \max.\ & \sum_{i=1}^{3}\sum_{j=1}^{3} c_{ij}x_{ij} \\ \text{s. t.}\ & \sum_{j=1}^{3} x_{ij} = 1\ (i = 1, \dots, 3) \\ & \sum_{i=1}^{3} x_{ij} \leq 1\ (j = 1, \dots, 3) \\ & x_{ij} \in \{0,1\}(i = 1, \dots, 3; j = 1, \dots, 3) \end{aligned}$$

➢ 定式化（行列表記）

$$\begin{aligned} \max.\ & \boldsymbol{C} \odot \boldsymbol{X} \\ \text{s. t.}\ & \boldsymbol{X}\boldsymbol{e} = \boldsymbol{e} \\ & \boldsymbol{e}^T\boldsymbol{X} \leq \boldsymbol{e} \\ & \boldsymbol{X} \in \{0,1\}^{3\times 3} \end{aligned}$$

- ✓ $\odot$ はアダマール積(Hadamard product)
- ✓ $\boldsymbol{e}$は要素が全て1の適当な次元のベクトル

割当問題:How to use CPLEX?

- 新規プロジェクトの作成
  - ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
  - ② [プロジェクト名] を記入（例：Assignment）し，3カ所にチェックする
    - ☑ デフォルトの実行構成の追加
    - ☑ モデルの作成
    - ☑ データの作成
  - ③ [終了]をクリック

プロジェクト名は自由だが，半角英数で何の問題を解こうとしているのかが分かる名前が良い

- プロジェクト内のいくつかの名前を変更
  - ✓ [構成1] → [config1]　※日本語を英語に変更しないと実行時エラーになる
  - ✓ モデルファイル [Assignment.mod] → [as.mod]
  - ✓ データファイル [Assignment.dat] → [asex1.dat]

- モデルファイル・データファイルを記述し保存（次ページ参照）
- [config1]にモデルファイルとデータファイルをセットする

割当問題:How to use CPLEX?

- as.mod

```
int i_max = ...;// 行の添え字の最大値
int j_max = ...;// 列の添え字の最大値

range I = 1..i_max;// 行の添え字の範囲 [1..i_max]を指定
range J = 1..j_max;// 列の添え字の範囲 [1..j_max]を指定

int c[I,J] = ...;// 評価値Cij(size:I×J)

dvar int+ x[I,J] in 0..1;// 変数宣言:0-1変数(size:I×J)

maximize
  sum(i in I) sum(j in J) c[i,j]*x[i,j];
subject to{
  forall(i in I) {
    sum(j in J) x[i,j] == 1;
  };
  forall(j in J) {
    sum(i in I) x[i,j] <= 1;
  };
};
```

割当問題:How to use CPLEX?

- asex1.dat

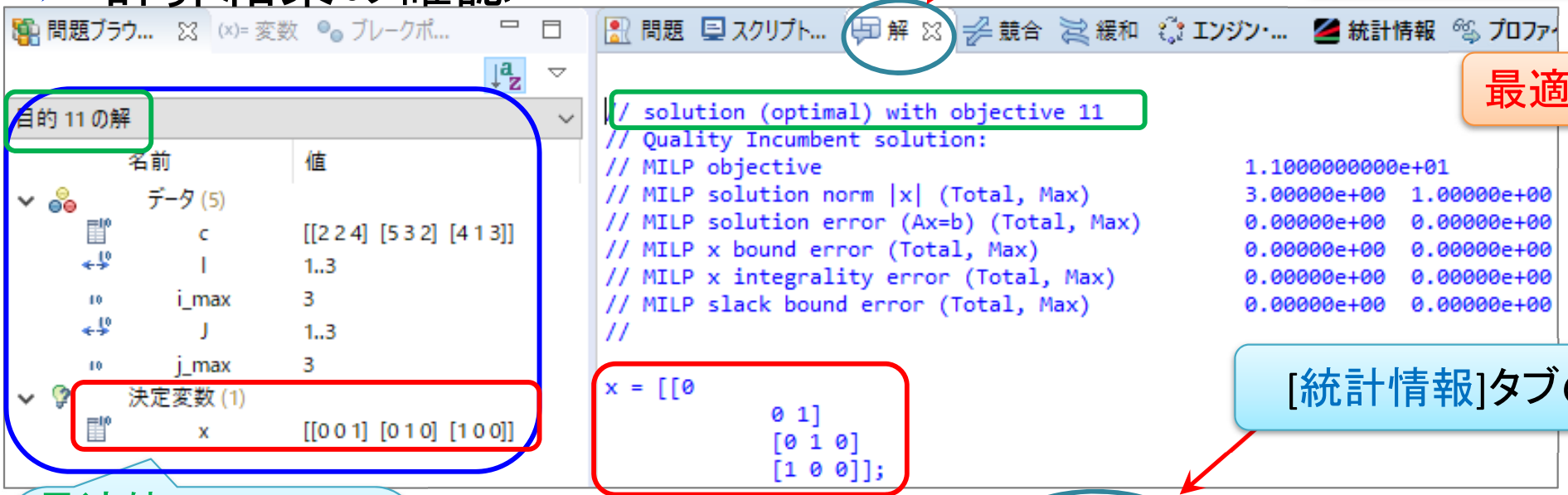
```
i_max = 3;
j_max = 3;

c = [
[2 2 4]
[5 3 2]
[4 1 3]
];
```

割当問題:How to use

[解]タブの中身

solution (optimal)
解(最適)

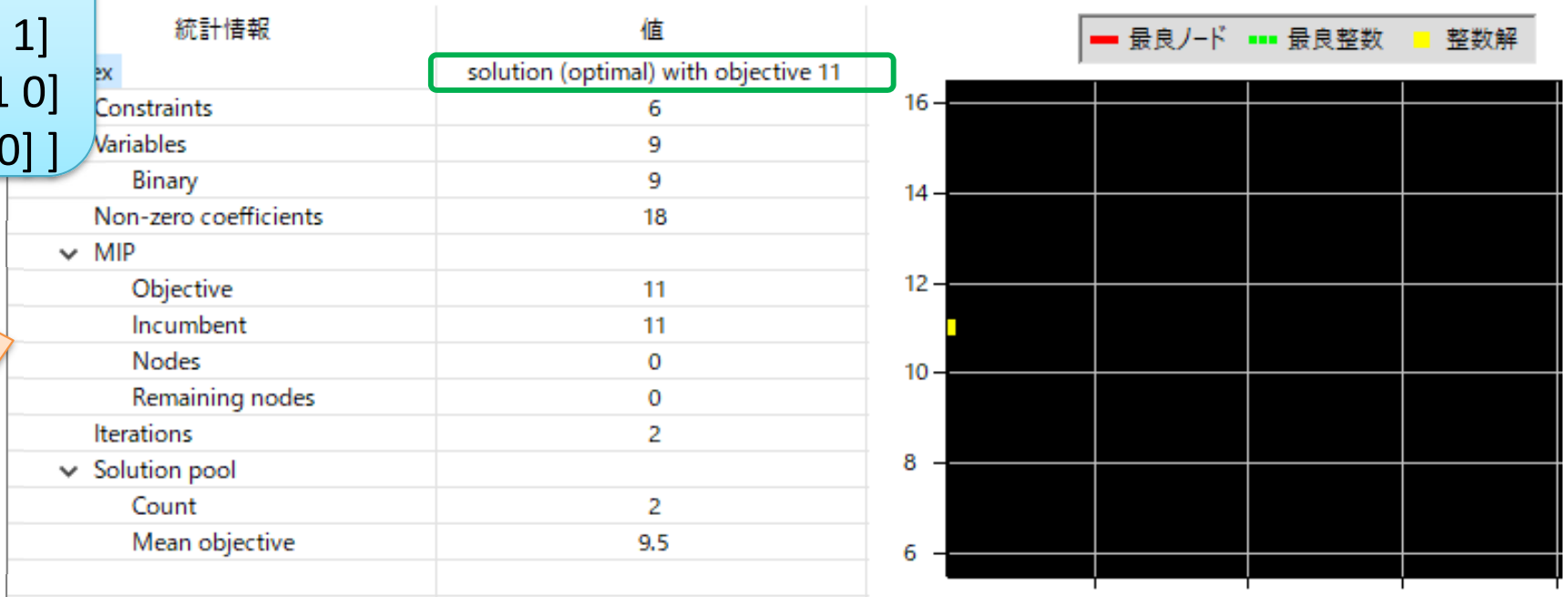
➢ 計算結果の確認



最適解発見

[統計情報]タブの中身

最適値 = 11
各定数(係数行列)
最適解 x = [[0 0 1]
[0 1 0]
[1 0 0]]



LPではなく
IPなので
情報が少し
違う

割当問題：How to use CPLEX?

- 演習1）データファイル（asex1.dat）をExcel入出力に変更せよ
  - Excelファイル [asex.xlsx] を作成し，データを適切に記述
  - データファイル [asex1.dat] をコピーして [asex1xlsx.dat] 作成し，修正

割当問題：How to use CPLEX?

- 演習1）解答例
  - asex.xlsx
    [Sheet1]

	A	B	C	D	E	F	G	H	I
1	i_max=	3							
2	j_max=	3				*obj.fn.*			
3	c=	2	2	4		11	= SUMPRODUCT(B3:D5, B6:D8)		
4		5	3	2					
5		4	1	3					
6	x=	0	0	1					
7		0	1	0					
8		1	0	0					

  - asex1xlsx.dat

```
SheetConnection sheet("asex.xlsx");

i_max from SheetRead(sheet, "Sheet1!B1");
j_max from SheetRead(sheet, "Sheet1!B2");
c from SheetRead(sheet, "Sheet1!B3:D5");

x to SheetWrite(sheet, "Sheet1!B6:D8");
```

割当問題を解く

- 割当問題の最適化（例2）
  - 仕事は4つ，部下は7人，各部下は1つの仕事を引き受けられる
  - 上司は部下を5段階[1,2,3,4,5]で評価済（下表）
  - この部署のパフォーマンスが最大になるように部下へ仕事を割り当てよ

- 最適化問題の定式化（ベタ・Σ表記）
  - 0-1変数 x_{ij} ＝1 … 仕事iを部下jへ割り当てる，x_{ij} ＝0 … 割り当てない
  - 係数matrix C:上司による部下評価値

仕事＼部下	1	2	3	4	5	6	7
1	2	4	2	1	3	1	4
2	3	1	3	2	5	4	2
3	1	3	4	4	4	3	3
4	4	2	1	4	5	2	1

- CPLEXで解く（モデルファイル[as.mod]は共通で使えるので[asex2.dat]のみ作り解く）

割当問題:How to use CPLEX?

- asex2.dat

```
i_max = 4;
j_max = 7;

c = [
[2 4 2 1 3 1 4]
[3 1 3 2 5 4 2]
[1 3 4 4 4 3 3]
[4 2 1 4 5 2 1]
];
```

割当問題：How to use CPLEX?

- 演習2）データファイル（asex2.dat）をExcel入出力に変更せよ
  - Excelファイル [asex.xlsx] にシート[Sheet2] を作成し，データを適切に記述
  - データファイル [asex2.dat] をコピーして [asex2xlsx.dat] 作成し，修正

割当問題：How to use CPLEX?

- 演習2）解答例
  - asex.xlsx
    [Sheet2]

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	i_max=	4											
2	j_max=	7								*obj.fn.*			
3	c=	2	4	2	1	3	1	4		17	= SUMPRODUCT(B3:H6, B7:H10)		
4		3	1	3	2	5	4	2					
5		1	3	4	4	4	3	3					
6		4	2	1	4	5	2	1					
7	x=	0	0	0	0	0	0	1					
8		0	0	0	0	1	0	0					
9		0	0	1	0	0	0	0					
10		0	0	0	1	0	0	0					

  - asex2xlsx.dat

```
SheetConnection sheet("asex.xlsx");

i_max from SheetRead(sheet, "Sheet2!B1");
j_max from SheetRead(sheet, "Sheet2!B2");
c from SheetRead(sheet, "Sheet2!B3:H6");

x to SheetWrite(sheet, "Sheet2!B7:H10");
```

割当問題：How to use gurobi?

- 割当問題の最適化（例2）
  - Google colaboratory / Python & gurobi

仕事＼部下	1	2	3	4	5	6	7
1	2	4	2	1	3	1	4
2	3	1	3	2	5	4	2
3	1	3	4	4	4	3	3
4	4	2	1	4	5	2	1

```
!pip install gurobipy
```

```
c = [[2,4,2,1,3,1,4],
     [3,1,3,2,5,4,2],
     [1,3,4,4,4,3,3],
     [4,2,1,4,5,2,1]]
I = range(len(c))    # 仕事4つの範囲I={0,1,2,3}
J = range(len(c[0]))# 部下7人の範囲J={0,1,…,6}

from gurobipy import *
m = Model("Assignment") # モデル(割り当て問題)の設定

x = {}
for i in I:
  for j in J:
    x[i,j] = m.addVar(vtype="B",name="x[%s,%s]" % (i+1,j+1))  # 仕事iを部下jが担当
for i in I:
  m.addConstr(quicksum(x[i,j] for j in J) == 1) # 仕事制約:全ての仕事は誰かが担当
for j in J:
  m.addConstr(quicksum(x[i,j] for i in I) <= 1) # 部下制約:各部下は高々1つの仕事を担当
m.setObjective(quicksum(c[i][j]*x[i,j] for i in I for j in J), GRB.MAXIMIZE)

m.optimize()  # 最適化実行
m.printAttr('X') # 最適解の表示
```

輸送問題：How to use gurobi?

- 輸送問題の最適化（例1）
  - 実行結果（一部）

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.700000000000e+01, best bound 1.700000000000e+01, gap 0.0000%

    Variable            X
-------------------------
      x[1,7]            1
      x[2,5]            1
      x[3,4]            1
      x[4,1]            1
```

生産計画をたてる

材料＼製品	α	β	γ	δ	ε	量
A	3	0	1	3	1	80
B	1	2	0	3	2	75
C	0	4	2	5	0	95
D	2	1	0	1	2	70
利益	6	7	3	10	5	

- 生産計画の最適化（例1）
  - 4種の材料 A, B, C, D がある
  - 5種の製品 $\alpha, \beta, \gamma, \delta, \varepsilon$ をつくる
  - 各材料の所有数，各製品を1単位作るのに必要な材料数，利益は右表
  - **総利益**を**最大**にする生産計画をたてたい

- 定式化（変数設定）
  - 変数 x_i … 製品i の作成数（単位）
  - 変数ベクトル $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$

- 定式化（係数表記）
  - 利益を表す係数ベクトル c
  - 材料所持数を表す係数ベクトル b
  - 必要材料数を表す係数行列 A

$$\boldsymbol{c} = (c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5) = (6 \quad 7 \quad 3 \quad 10 \quad 5)$$

$$\boldsymbol{b} = (b_1 \quad b_2 \quad b_3 \quad b_4) = (80 \quad 75 \quad 95 \quad 70)$$

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} = \begin{pmatrix} 3 & 0 & 1 & 3 & 1 \\ 1 & 2 & 0 & 3 & 2 \\ 0 & 4 & 2 & 5 & 0 \\ 2 & 1 & 0 & 1 & 2 \end{pmatrix}$$

生産計画をたてる

- 定式化（ベタ表記）

$$\begin{aligned}
&\text{max. } 6x_1+7x_2+3x_3+10x_4+5x_5 \\
&\text{s. t. } 3x_1+0x_2+1x_3+3x_4+1x_5 \leqq 80 \\
&\quad 1x_1+2x_2+0x_3+3x_4+2x_5 \leqq 75 \\
&\quad 0x_1+4x_2+2x_3+5x_4+0x_5 \leqq 95 \\
&\quad 2x_1+1x_2+0x_3+1x_4+2x_5 \leqq 70 \\
&\quad x_1, x_2, x_3, x_4, x_5 \geqq 0
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{c} &= (c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5) \\
&= (6 \quad 7 \quad 3 \quad 10 \quad 5) \\
\boldsymbol{b} &= (b_1 \quad b_2 \quad b_3 \quad b_4) \\
&= (80 \quad 75 \quad 95 \quad 70) \\
A &= \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} \\
&= \begin{pmatrix} 3 & 0 & 1 & 3 & 1 \\ 1 & 2 & 0 & 3 & 2 \\ 0 & 4 & 2 & 5 & 0 \\ 2 & 1 & 0 & 1 & 2 \end{pmatrix}
\end{aligned}$$

- 定式化（Σ表記）

$$\begin{aligned}
&\text{max.} \sum_{j=1}^{5} c_j x_j \\
&\text{s. t. } \sum_{j=1}^{5} a_{ij} x_j \leq b_i \ (i = 1, \ldots, 4) \\
&\quad x_j \geq 0 \ (j = 1, \ldots, 5)
\end{aligned}$$

- 定式化（行列表記）

$$\begin{aligned}
&\text{max.} \boldsymbol{c}^T \boldsymbol{x} \\
&\text{s. t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\
&\quad \boldsymbol{x} \geq \boldsymbol{0}
\end{aligned}$$

生産計画:How to use CPLEX?

- 新規プロジェクトの作成
  - ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
  - ② [プロジェクト名] を記入（例：ProductionPlanning）し，3カ所にチェックする
    - ☑ デフォルトの実行構成の追加
    - ☑ モデルの作成
    - ☑ データの作成
  - ③ [終了]をクリック

プロジェクト名は自由だが，半角英数で何の問題を解こうとしているのかが分かる名前が良い

- プロジェクト内のいくつかの名前を変更
  - ✓ [構成1] → [config1]　※日本語を英語に変更しないと実行時エラーになる
  - ✓ モデルファイル [ProductionPlanning.mod] → [pp.mod]
  - ✓ データファイル [ProductionPlanning.dat] → [ppex1.dat]

- モデルファイル・データファイルを記述し保存（次ページ参照）
- [config1]にモデルファイルとデータファイルをセットする

生産計画:How to use CPLEX?

- pp.mod

```
int i_max = ...;// 材料数
int j_max = ...;// 製品数

range I = 1..i_max;// 材料集合の範囲
range J = 1..j_max;// 製品集合の範囲

int a[I,J] = ...;// 各製品1単位あたりの必要材料数を表す行列
int b[I] = ...;// 材料i の所持数
int c[J] = ...;// 製品j の1単位あたり利益

dvar float+ x[J];// 製品j の生産量(非負)

maximize
  sum(j in J) c[j]*x[j];
subject to {
  forall(i in I) {
    sum(j in J) a[i,j]*x[j] <= b[i];
  }
}
```

生産計画:How to use CPLEX?

- ppex1.dat

```
i_max = 4;
j_max = 5;

c = [6 7 3 10 5];
b = [80 75 95 70];

a = [
[3 0 1 3 1]
[1 2 0 3 2]
[0 4 2 5 0]
[2 1 0 1 2]
];
```

生産計画:How to use CPLEX?

- 計算結果の確認([解]タブの中身)

最適値

```
// solution (optimal) with objective 316
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b residual                  = 0
// Maximum c-B'pi residual                = 0
// Maximum |x|                            = 45.5
// Maximum |slack|                        = 4
// Maximum |pi|                           = 2.2
// Maximum |red-cost|                     = 0.2
// Condition number of unscaled basis = 1.2e+01
//

x = [0 1 45.5 0 34.5];
```

optimal solution
最適解

生産計画：How to use CPLEX?

- 演習1）データファイル（ppex1.dat）をExcel入出力に変更せよ
  - Excelファイル [ppex.xlsx] を作成し，データを適切に記述
  - データファイル [ppex1.dat] をコピーして [ppex1xlsx.dat] 作成し，修正

生産計画：How to use CPLEX?

- 演習1）解答例
  - ppex.xlsx
    [Sheet1]

	A	B	C	D	E	F	G	H	I	J	K	L
1	i_max=	4						[H3]	= SUMPRODUCT(B3:F3, B$9:F$9)			
2	j_max=	5						*obj.fn.*		→[H4:H7]へコピペ		
3	c=	6	7	3	10	5		316		b=		
4	A =	3	0	1	3	1		80	≦	80		
5		1	2	0	3	2		71	≦	75		
6		0	4	2	5	0		95	≦	95		
7		2	1	0	1	2		70	≦	70		
8												
9	x=	0.0	1.0	45.5	0.0	34.5						

  - ppex1xlsx.dat

```
SheetConnection sheet("ppex.xlsx");

i_max from SheetRead(sheet, "Sheet1!B1");
j_max from SheetRead(sheet, "Sheet1!B2");
c from SheetRead(sheet, "Sheet1!B3:F3");
a from SheetRead(sheet, "Sheet1!B4:F7");
b from SheetRead(sheet, "Sheet1!J4:J7");

x to SheetWrite(sheet, "Sheet1!B9:F9");
```

生産計画をたてる

- 生産計画の最適化（例2）
  - 7種の材料 A, B, …, G がある
  - 5種の製品 $\alpha, \beta, \gamma, \delta, \varepsilon$ をつくる
  - 各材料の所有数，各製品を1単位作るのに必要な材料数，利益は右表
  - **総利益最大**の生産計画をたてよ

材料＼製品	α	β	γ	δ	ε	量
A	*2*	*0*	*1*	*3*	*1*	*80*
B	*3*	*1*	*0*	*3*	*1*	*75*
C	*0*	*2*	*3*	*5*	*0*	*95*
D	*3*	*2*	*0*	*1*	*2*	*70*
E	*1*	*0*	*3*	*0*	*3*	*60*
F	*0*	*3*	*0*	*2*	*4*	*80*
G	*1*	*5*	*0*	*4*	*1*	*90*
利益	*8*	*7*	*3*	*9*	*6*	

- 線形最適化問題として定式化し，CPLEXで解く
  - モデルファイル[pp.mod]は共通なので，データファイル[ppex2.dat]を作り解く

生産計画：How to use CPLEX?

- 演習2）データファイル（ppex2.dat）をExcel入出力に変更せよ
  - Excelファイル [ppex.xlsx] にシート[Sheet2] を作成し，データを適切に記述
  - データファイル [ppex2.dat] をコピーして [ppex2xlsx.dat] 作成し，修正

生産計画：How to use CPLEX?

- 演習2）解答例
  - ppex.xlsx
    [Sheet2]

	A	B	C	D	E	F	G	H	I	J	K	L
1	i_max=	7						[H3] = SUMPRODUCT(B3:F3, B$9:F$9)				
2	j_max=	5						*obj.fn.*		→[H4:H10]へコピペ		
3	c=	6	7	3	10	5		0		b=		
4	A =	2	0	1	3	1		0	≦	80		
5		3	1	0	3	1		0	≦	75		
6		0	2	3	5	0		0	≦	95		
7		3	2	0	1	2		0	≦	70		
8		1	0	3	0	3		0	≦	60		
9		0	3	0	2	4		0	≦	80		
10		1	5	0	4	1		0	≦	90		
11												
12	x=											

- ppex2xlsx.dat

```
SheetConnection sheet("ppex.xlsx");

i_max from SheetRead(sheet, "Sheet2!B1");
j_max from SheetRead(sheet, "Sheet2!B2");
c from SheetRead(sheet, "Sheet2!B3:F3");
a from SheetRead(sheet, "Sheet2!B4:F10");
b from SheetRead(sheet, "Sheet2!J4:J10");

x to SheetWrite(sheet, "Sheet2!B12:F12");
```

生産計画：How to use gurobi?

➢ 生産計画の最適化（例1）

➢ Google colaboratory / Python & gurobi

材料＼製品	α	β	γ	δ	ε	量
A	3	0	1	3	1	80
B	1	2	0	3	2	75
C	0	4	2	5	0	95
D	2	1	0	1	2	70
利益	6	7	3	10	5	

```
!pip install gurobipy
```

```
c = [6,7,3,10,5]
b = [80,75,95,70]
A = [[3,0,1,3,1],
     [1,2,0,3,2],
     [0,4,2,5,0],
     [2,1,0,1,2]]
I = range(len(A))    # 材料4つの範囲I={0,1,2,3}
J = range(len(A[0]))# 製品5つの範囲J={0,1,2,3,4}

from gurobipy import *
m = Model("ProductPlanning") # モデル(生産計画)の設定

x = {}
for j in J:
    x[j] = m.addVar(lb=0.0, vtype="C",name="x[%s]" % (j+1))  # 変数設定:製品j生産量

for i in I:
  m.addConstr(quicksum(A[i][j]*x[j] for j in J) <= b[i]) # 材料制約
m.setObjective(quicksum(c[j]*x[j] for j in J), GRB.MAXIMIZE) # 目的関数:利益最大

m.optimize()  # 最適化実行
m.printAttr('X') # 最適解の表示
```

生産計画：How to use gurobi?

- 生産計画の最適化（例1）
  - 実行結果（一部）

```
Solved in 5 iterations and 0.02 seconds (0.00 work units)
Optimal objective  3.160000000e+02

    Variable             X
-------------------------
        x[2]             1
        x[3]          45.5
        x[5]          34.5
```

栄養問題を解く

栄養素＼食品	α	β	γ	δ	ε	必要量
A	2.5	0.7	1.3	3.1	4.1	70
B	11	21	5	13	12	300
C	0.6	4.2	2.5	5.1	0.8	50
D	23	16	15	0	12	150
コスト(円)	52	64	32	46	38	

- 栄養問題の最適化(例1)
  - 4種の栄養素 A～D がある
  - 5種の食品 α～ε がある
  - 各食品を1単位摂取すると

  得られる栄養素，コストは表の通り
  - 摂取必要量を満たし，コスト最小となる各食品の摂取量を知りたい

- 定式化(変数設定)
  - 変数 x_j … 食品 j の摂取量(単位)
  - 変数ベクトル $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)$

- 定式化(係数表記)
  - 摂取必要量を表す係数ベクトル $\boldsymbol{b}$
  - コストを表す係数ベクトル $\boldsymbol{c}$
  - 含有栄養素を表す係数行列 A

$$\boldsymbol{b} = (b_1 \quad b_2 \quad b_3 \quad b_4) = (70 \quad 300 \quad 50 \quad 150)$$

$$\boldsymbol{c} = (c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5) = (52 \quad 64 \quad 32 \quad 46 \quad 38)$$

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} = \begin{pmatrix} 2.5 & 0.7 & 1.3 & 3.1 & 4.1 \\ 11 & 21 & 5 & 13 & 12 \\ 0.6 & 4.2 & 2.5 & 5.1 & 0.8 \\ 23 & 16 & 15 & 0 & 12 \end{pmatrix}$$

栄養問題を解く

$$
\begin{aligned}
\boldsymbol{b} &= (b_1 \quad b_2 \quad b_3 \quad b_4) \\
&= (70 \quad 300 \quad 50 \quad 150) \\
\boldsymbol{c} &= (c_1 \quad c_2 \quad c_3 \quad c_4 \quad c_5) \\
&= (52 \quad 64 \quad 32 \quad 46 \quad 38) \\
A &= \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} \\
&= \begin{pmatrix} 2.5 & 0.7 & 1.3 & 3.1 & 4.1 \\ 11 & 21 & 5 & 13 & 12 \\ 0.6 & 4.2 & 2.5 & 5.1 & 0.8 \\ 23 & 16 & 15 & 0 & 12 \end{pmatrix}
\end{aligned}
$$

定式化（ベタ表記）

$$
\begin{aligned}
&\text{min. } 52x_1+64x_2+32x_3+46x_4+38x_5 \\
&\text{s. t. } 2.5x_1+0.7x_2+1.3x_3+3.1x_4+4.1x_5 \geqq 70 \\
&\quad 11x_1+21x_2+5x_3+13x_4+12x_5 \geqq 300 \\
&\quad 0.6x_1+4.2x_2+2.5x_3+5.1x_4+0.8x_5 \geqq 50 \\
&\quad 23x_1+16x_2+15x_3+0x_4+12x_5 \geqq 150 \\
&\quad x_1,\ x_2,\ x_3,\ x_4,\ x_5 \geqq 0
\end{aligned}
$$

定式化（Σ表記）

$$
\begin{aligned}
&\text{min.} \sum_{j=1}^{5} c_j x_j \\
&\text{s. t. } \sum_{j=1}^{5} a_{ij} x_j \geq b_i \ (i = 1, \ldots, 4) \\
&\quad x_j \geq 0 \ (j = 1, \ldots, 5)
\end{aligned}
$$

定式化（行列表記）

$$
\begin{aligned}
&\text{min.} \boldsymbol{c}^T \boldsymbol{x} \\
&\text{s. t. } \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\
&\quad \boldsymbol{x} \geq \boldsymbol{0}
\end{aligned}
$$

栄養問題:How to use CPLEX?

- 新規プロジェクトの作成
  - ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
  - ② [プロジェクト名] を記入（例：Nutrients）し，3カ所にチェックする
    - ☑ デフォルトの実行構成の追加
    - ☑ モデルの作成
    - ☑ データの作成
  - ③ [終了]をクリック

プロジェクト名は自由だが，半角英数で何の問題を解こうとしているのかが分かる名前が良い

- プロジェクト内のいくつかの名前を変更
  - ✓ [構成1] → [config1]　※日本語を英語に変更しないと実行時エラーになる
  - ✓ モデルファイル [Nutrients.mod] → [nt.mod]
  - ✓ データファイル [Nutrients.dat] → [ntex1.dat]

- モデルファイル・データファイルを記述し保存（次ページ参照）
- [config1]にモデルファイルとデータファイルをセットする

栄養問題:How to use CPLEX?

- nt.mod

```
int i_max = ...;// 栄養素数
int j_max = ...;// 食品数

range I = 1..i_max;// 栄養素集合の範囲
range J = 1..j_max;// 食品集合の範囲

float a[I,J] = ...;// 各食品1単位あたりの栄養素含有量
float c[J] = ...;// 各食品1単位あたりの摂取カロリー
float b[I] = ...;// 各栄養素の摂取必要量

dvar float+ x[J];// 食品j の摂取量(非負)

minimize
  sum(j in J) c[j]*x[j];
subject to {
  forall(i in I) {
    sum(j in J) a[i,j]*x[j] >= b[i];
  }
}
```

栄養問題:How to use CPLEX?

- ntex1.dat

```
i_max = 4;
j_max = 5;

c = [52 64 32 46 38];
b = [70 300 50 150];

a = [
[2.5 0.7 1.3 3.1 4.1]
[11 21 5 13 12]
[0.6 4.2 2.5 5.1 0.8]
[23 16 15 0 12]
];
```

栄養問題:How to use CPLEX?

- 計算結果の確認([解]タブの中身)

最適値

```
// solution (optimal) with objective 960.810287123481
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Max. unscaled (scaled) Ax-b resid.          = 2.84217e-14 (1.77636e-15)
// Max. unscaled (scaled) c-B'pi resid.        = 7.10543e-15 (7.10543e-15)
// Max. unscaled (scaled) |x|                  = 13.0703 (13.0703)
// Max. unscaled (scaled) |slack|              = 72.681 (4.54256)
// Max. unscaled (scaled) |pi|                 = 2.72066 (43.5306)
// Max. unscaled (scaled) |red-cost|           = 18.6449 (26.8182)
// Condition number of scaled basis            = 6.6e+00
//

x = [0 4.1148 0 4.365 13.07];
```

optimal solution
最適解

栄養問題：How to use CPLEX?

- 演習1）データファイル（ntex1.dat）をExcel入出力に変更せよ
  - Excelファイル [ntex.xlsx] を作成し，データを適切に記述
  - データファイル [ntex1.dat] をコピーして [ntex1xlsx.dat] 作成し，修正

栄養問題：How to use CPLEX?

- 演習1）解答例
  - ntex.xlsx
    [Sheet1]

	A	B	C	D	E	F	G	H	I	J	K
1	ex1	4	i_max			[H4]	= SUMPRODUCT(C4:G4, C$10:G$10)				
2		5	j_max				→[H5:H8]へコピー				
3		栄養素＼食品	α	β	γ	δ	ε	摂取総量		必要量	
4		A	2.5	0.7	1.3	3.1	4.1	70.000	≧	70	
5		B	11	21	5	13	12	300.000	≧	300	
6		C	0.6	4.2	2.5	5.1	0.8	50.000	≧	50	
7		D	23	16	15	0	12	222.681	≧	150	
8		コスト（円）	¥52	¥64	¥32	¥46	¥38	960.810	min.		
9								*obj.fn.*			
10		摂取量x	0.000	4.115	0.000	4.365	13.070				

  - ntex1xlsx.dat

```
SheetConnection sheet("ntex.xlsx");

i_max from SheetRead(sheet, "Sheet1!B1");
j_max from SheetRead(sheet, "Sheet1!B2");
c from SheetRead(sheet, "Sheet1!C8:G8");
a from SheetRead(sheet, "Sheet1!C4:G7");
b from SheetRead(sheet, "Sheet1!J4:J7");

x to SheetWrite(sheet, "Sheet1!C10:G10");
```

栄養問題を解く

- 栄養問題の最適化（例2）　※データはExcelファイル
  - 18種の栄養素がある
  - 23種の食品がある
  - 各食品を1単位摂取すると得られる栄養素，カロリーはデータ表の通り
  - 摂取必要量を満たし，コスト最小の食品摂取量を求めよ

- 線形最適化問題として定式化し，CPLEXで解く
  - モデルファイル[nt.mod]は共通なので，データファイル[ntex2.dat]を作り解く

栄養問題：How to use gurobi?

- 栄養問題の最適化（例1）
  - Google colaboratory / Python & g

栄養素＼食品	α	β	γ	δ	ε	必要量
A	2.5	0.7	1.3	3.1	4.1	70
B	11	21	5	13	12	300
C	0.6	4.2	2.5	5.1	0.8	50
D	23	16	15	0	12	150
コスト（円）	52	64	32	46	38	

```
!pip install gurobipy
```

```
c = [52,64,32,46,38]
b = [70,300,50,150]
A = [[2.5,0.7,1.3,3.1,4.1],
     [11,21,5,13,12],
     [0.6,4.2,2.5,5.1,0.8],
     [23,16,15,0,12]
]
I = range(len(A))    # 栄養素4つの範囲I={0,1,2,3}
J = range(len(A[0]))# 食品5つの範囲J={0,1,2,3,4}

from gurobipy import *
m = Model("Nutrients") # モデル（栄養問題）の設定

x = {}
for j in J:
    x[j] = m.addVar(lb=0.0, vtype="C",name="x[%s]" % (j+1))   # 変数：食品jの摂取量
for i in I:
  m.addConstr(quicksum(A[i][j]*x[j] for j in J) >= b[i]) # 栄養素制約
m.setObjective(quicksum(c[j]*x[j] for j in J), GRB.MINIMIZE) # 目的関数：コスト最小

m.optimize()  # 最適化実行
m.printAttr('X') # 最適解の表示
```

栄養問題：How to use gurobi?

- 栄養問題の最適化（例1）
  - 実行結果（一部）

```
Solved in 3 iterations and 0.02 seconds (0.00 work units)
Optimal objective  9.608102871e+02

    Variable             X
-------------------------
        x[2]       4.11485
        x[4]       4.36498
        x[5]       13.0703
```

多品種輸送問題を解く

- 多品種輸送問題の最適化（例1）
  - 3つの工場で4種の製品A,B,C,Dを作っている．ただし，工場1はBとD，工場2はA,B,C，工場3はB,C,D のみを生産できる
  - 各工場の生産可能量は製品の種類に関係なく，それぞれ最大1200個
  - 5人の顧客に必要量を輸送する．輸送コスト最小となる輸送計画をたてたい

＜各工場で生産可能な製品と合計生産可能量＞

工場＼製品	A	B	C	D	生産可能量
1	—	OK	—	OK	1200
2	OK	OK	OK	—	1200
3	—	OK	OK	OK	1200

＜製品1単位あたり輸送コストと需要＞

	工場輸送コスト			製品需要			
顧客	1	2	3	A	B	C	D
α	4	6	9	80	85	300	6
β	5	4	7	270	160	400	7
γ	6	3	4	250	130	350	4
δ	8	5	3	160	60	200	3
ε	10	8	4	180	40	150	5

- 定式化（変数/係数）
  - 変数x_{ijk}：工場j→顧客i の製品k輸送量
  - 生産可能量の係数ベクトル b
  - 輸送コストを表す係数行列 C
  - 需要を表す係数行列 D

多品種輸送問題を解く

➢ 定式化（ベタ表記）

$$\min.\ 4(x_{111}+x_{112}+x_{113}+x_{114}) + \ldots + 4(x_{531}+x_{532}+x_{533}+x_{534})$$

$$\text{s. t. } x_{111}+x_{121}+x_{131}=80,\ x_{112}+x_{122}+x_{132}=85$$

$$x_{113}+x_{123}+x_{133}=300,\ x_{114}+x_{124}+x_{134}=6$$

$$\ldots$$

$$(x_{111}+x_{112}+x_{113}+x_{114})+\ldots+(x_{511}+x_{512}+x_{513}+x_{514}) \leqq 1200$$

$$(x_{121}+x_{122}+x_{123}+x_{124})+\ldots+(x_{521}+x_{522}+x_{523}+x_{524}) \leqq 1200$$

$$(x_{131}+x_{132}+x_{133}+x_{134})+\ldots+(x_{531}+x_{532}+x_{533}+x_{534}) \leqq 1200$$

$$(x_{111}+x_{211}+x_{311}+x_{411}+x_{511})+(x_{113}+x_{213}+x_{313}+x_{413}+x_{513})$$
$$+(x_{124}+x_{224}+x_{324}+x_{424}+x_{524})$$
$$+(x_{131}+x_{231}+x_{331}+x_{431}+x_{531}) = 0$$

$$x_{111},\ldots,x_{534} \geqq 0$$

各工場の生産不可製品の輸送量を0に

➢ 定式化（Σ表記）

$$\min.\sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} c_{ij}x_{ijk}$$

$$\text{s. t. } \sum_{j=1}^{3} x_{ijk} = d_{ik}\ (i = 1,\ldots,5;\ k = 1,\ldots,4)$$

$$\sum_{i=1}^{5}\sum_{k=1}^{4} x_{ijk} \leq b_j\ (j = 1,\ldots,3)$$

$$\sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} m_{jk}x_{ijk} = 0$$

$$x_{ijk} \geq 0\ (i = 1,\ldots,5;\ j = 1,\ldots,3;\ k = 1,\ldots,4)$$

$$\boldsymbol{b} = (b_1 \quad b_2 \quad b_3) = (1200 \quad 1200 \quad 1200)$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \\ c_{41} & c_{42} & c_{43} \\ c_{51} & c_{52} & c_{53} \end{pmatrix} = \begin{pmatrix} 4 & 6 & 9 \\ 5 & 4 & 7 \\ 6 & 3 & 4 \\ 8 & 5 & 3 \\ 10 & 8 & 4 \end{pmatrix}$$

$$D = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \\ d_{51} & d_{52} & d_{53} & d_{54} \end{pmatrix} = \begin{pmatrix} 80 & 85 & 300 & 6 \\ 270 & 160 & 400 & 7 \\ 250 & 130 & 350 & 4 \\ 160 & 60 & 200 & 3 \\ 180 & 40 & 150 & 5 \end{pmatrix}$$

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

多品種輸送問題:How to use CPLEX?

- 新規プロジェクトの作成
  - ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
  - ② [プロジェクト名] を記入（例：MultiTransport）し，3カ所にチェックする
    - ☑ デフォルトの実行構成の追加
    - ☑ モデルの作成
    - ☑ データの作成
  - ③ [終了]をクリック

プロジェクト名は自由だが，半角英数で何の問題を解こうとしているのかが分かる名前が良い

- プロジェクト内のいくつかの名前を変更
  - ✓ [構成1] → [config1]　※日本語を英語に変更しないと実行時エラーになる
  - ✓ モデルファイル [MultiTransport.mod] → [mt.mod]
  - ✓ データファイル [MultiTransport.dat] → [mtex1.dat]

- モデルファイル・データファイルを記述し保存（次ページ参照）
- [config1]にモデルファイルとデータファイルをセットする

多品種輸送問題:How to use CPLEX?

- mt.mod

```
int i_max = ...;// 顧客数
int j_max = ...;// 工場数
int k_max = ...;// 製品数

range I = 1..i_max;// 顧客集合の範囲
range J = 1..j_max;// 工場集合の範囲
range K = 1..k_max;// 製品集合の範囲

int c[I,J] = ...; // 工場 j → 顧客 i への製品1単位あたり輸送コスト
int d[I,K] = ...; // 顧客 i の 製品 k の需要量
int b[J] = ...;   // 工場 j の生産可能合計量
int m[J,K] = ...; // 工場 j の 製品 k の生産有無を表す係数行列

dvar float+ x[I,J,K];// 工場 j → 顧客 i への 製品 k の輸送量(非負)

minimize
  sum(i in I) sum(j in J) sum(k in K) c[i,j]*x[i,j,k];
subject to {
  forall(i in I) {
    forall(k in K) {
      sum(j in J) x[i,j,k] == d[i,k];
    }
  }
  forall(j in J) {
    sum(i in I) sum(k in K) x[i,j,k] <= b[j];
    sum(i in I) sum(k in K) m[j,k]*x[i,j,k] == 0;
 }
}
```

多品種輸送問題:How to use CPLEX?

- mtex1.dat

```
i_max = 5;
j_max = 3;
k_max = 4;

b = [1200 1200 1200];
c = [
[4 6 9]
[5 4 7]
[6 3 4]
[8 5 3]
[10 8 4]
];
d = [
[ 80  85 300 6]
[270 160 400 7]
[250 130 350 4]
[160  60 200 3]
[180  40 150 5]
];
m = [
[1 0 1 0]
[0 0 0 1]
[1 0 0 0]
];
```

多品種輸送問題:How to

工場＼製品	A	B	C	D	生産可能量
1	—	OK	—	OK	1200
2	OK	OK	OK	—	1200
3	—	OK	OK	OK	1200

- 計算結果の確認([解]タブ)

最適値

```
// solution (optimal) with objective 12014
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b residual                = 0
// Maximum c-B'pi residual              = 0
// Maximum |x|                          = 400
// Maximum |slack|                      = 2902
// Maximum |pi|                         = 8
// Maximum |red-cost|                   = 8
// Condition number of unscaled basis = 1.4e+02
x = [[[   0   85    0   6]      …工場1(B=85,D=6)
      [  80    0  300   0]      …工場2(A=80,C=300)
      [   0    0    0   0]]     …工場3(0)
     [[   0    0    0   7]
      [270  160  400   0]
      [   0    0    0   0]]
     [[   0    0    0   0]
      [250  130  350   0]
      [   0    0    0   4]]
     [[   0    0    0   0]
      [160    0    0   0]
      [   0   60  200   3]]
     [[   0    0    0   0]
      [180    0    0   0]
      [   0   40  150   5]]];
```

顧客αへの輸送

Optimal solution 最適解

	製品需要			
顧客	A	B	C	D
α	80	85	300	6
β	270	160	400	7
γ	250	130	350	4
δ	160	60	200	3
ε	180	40	150	5

多品種輸送問題：How to use CPLEX?

- 演習1）データファイル（ntex1.dat）をExcel入出力に変更せよ
  - モデルファイル [nt.mod] をコピーして [ntxlsx.mod] 作成し，修正
  - Excelファイル [ntex.xlsx] を作成し，データを適切に記述
  - データファイル [ntex1.dat] をコピーして [ntex1xlsx.dat] 作成し，修正

添え字3つ（3次元）では
Excelへ入出力出来ない
ので，変数・定式化を
2次元版に修正する

工場jから顧客iへの製品kの輸送量

変数$x_{i,j,k}$ → x_{qk}

工場jと顧客iとの組合せ$q=3(i-1)+j$についての製品kの輸送量

i	j	→	q
1	1		1
1	2		2
1	3		3
2	1		4
2	2		5
2	3		6
⋮	⋮		⋮
5	1		13
5	2		14
5	3		15

多品種輸送問題：How to use CPLEX?

- 変数$x_{i,j,k} \rightarrow x_{qk}$　$q=3(i-1)+j$

- 3次元変数 $x_{i,j,k}$ 版の定式化（Σ表記）

$$\min. \sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} c_{ij}x_{ijk}$$
$$\text{s. t. } \sum_{j=1}^{3} x_{ijk} = d_{ik} (i = 1, \dots, 5; k = 1, \dots, 4)$$
$$\sum_{i=1}^{5}\sum_{k=1}^{4} x_{ijk} \leq b_j (j = 1, \dots, 3)$$
$$\sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} m_{jk}x_{ijk} = 0$$
$$x_{ijk} \geq 0 \ (i = 1, \dots, 5; j = 1, \dots, 3; k = 1, \dots, 4)$$

- 2次元変数 x_{qk} 版の定式化（Σ表記）

$$\min. \sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} c_{ij}x_{3(i-1)+j,k}$$
$$\text{s. t. } \sum_{j=1}^{3} x_{3(i-1)+j,k} = d_{ik} (i = 1, \dots, 5; k = 1, \dots, 4)$$
$$\sum_{i=1}^{5}\sum_{k=1}^{4} x_{3(i-1)+j,k} \leq b_j (j = 1, \dots, 3)$$
$$\sum_{i=1}^{5}\sum_{j=1}^{3}\sum_{k=1}^{4} m_{jk}x_{3(i-1)+j,k} = 0$$
$$x_{3(i-1)+j,k} \geq 0 \ (i = 1, \dots, 5; j = 1, \dots, 3; k = 1, \dots, 4)$$

多品種輸送問題:How to use CPLEX?

- mt2.mod

2次元変数 x_{qk} 版

```
int i_max = ...;// 顧客数
int j_max = ...;// 工場数
int k_max = ...;// 製品数
range I = 1..i_max;// 顧客集合の範囲
range J = 1..j_max;// 工場集合の範囲
range Q = 1..i_max*j_max; // q=3(i-1)+j に変換する
range K = 1..k_max;// 製品集合の範囲

int c[I,J] = ...;// 工場 j → 顧客 i への製品1単位あたり輸送コスト
int d[I,K] = ...;// 顧客 i の 製品 k の需要量
int b[J] = ...;// 工場 j の生産可能合計量
int m[J,K] = ...;// 工場 j の 製品 k の生産有無を表す係数行列

dvar float+ x[Q,K];// 工場 j → 顧客 i への 製品 k の輸送量(非負)

minimize
  sum(i in I) sum(j in J) sum(k in K) c[i,j]*x[3*(i-1)+j,k];
subject to {
  forall(i in I) {
    forall(k in K) {
      sum(j in J) x[3*(i-1)+j,k] == d[i,k];
    }
  }
  forall(j in J) {
    sum(i in I) sum(k in K) x[3*(i-1)+j,k] <= b[j];
    sum(i in I) sum(k in K) m[j,k]*x[3*(i-1)+j,k] == 0;
 }
}
```

修正箇所
(mt.mod参照)

多品種輸送問題：How to use CPLEX?

- mtex1.xlsx

[Sheet1]

多品種輸送

	輸送コスト			製品需要			
顧客	1	2	3	A	B	C	D
α	4	6	9	80	85	300	6
β	5	4	7	270	160	400	7
γ	6	3	4	250	130	350	4
δ	8	5	3	160	60	200	3
ε	10	8	4	180	40	150	5

工場＼製品	A	B	C	D	生産可能量
1	1	0	1	0	1200
2	0	0	0	1	1200
3	1	0	0	0	1200

1=生産不可 / 0=生産可

i_max=	5
j_max=	3
k_max=	4

目的関数

総輸送コ

			x						
i	j	q	1	2	3	4	輸送コ	輸送量	14482
1	1	1	0	85	0	6	4	91	364
1	2	2	80	0	260	0	6	340	2040
1	3	3	0	0	40	0	9	40	360
2	1	4	0	160	0	7	5	167	835
2	2	5	270	0	0	0	4	270	1080
2	3	6	0	0	400	0	7	400	2800
3	1	7	0	130	0	4	6	134	804
3	2	8	250	0	0	0	3	250	750
3	3	9	0	0	350	0	4	350	1400
4	1	10	0	48	0	0	8	48	384
4	2	11	160	0	0	0	5	160	800
4	3	12	0	12	200	3	3	215	645
5	1	13	0	0	0	0	10	0	0
5	2	14	180	0	0	0	8	180	1440
5	3	15	0	40	150	5	4	195	780

- mtex1xlsx.dat

```
SheetConnection sheet("mtex1.xlsx");
i_max from SheetRead(sheet, "Sheet1!L1");
j_max from SheetRead(sheet, "Sheet1!L2");
k_max from SheetRead(sheet, "Sheet1!L3");
c from SheetRead(sheet, "Sheet1!C4:E8");
d from SheetRead(sheet, "Sheet1!F4:I8");
m from SheetRead(sheet, "Sheet1!F11:I13");
b from SheetRead(sheet, "Sheet1!J11:J13");
x to SheetWrite(sheet, "Sheet1!Q4:T18");
```

多品種輸送問題：How to use gurobi?

- 多品種輸送問題の最適化（例1）
  - Google colaboratory / Python & gurobi

```
!pip install gurobipy
```

```
# 多品種輸送問題
b = [1200,1200,1200]  # 3工場の供給量
c = [[4,6,9],         # 工場j → 顧客i への単位当たり輸送コスト
     [5,4,7],
     [6,3,4],
     [8,5,3],
     [10,8,4]]
d = [[ 80, 85,300,6], # 顧客i の 製品k の需要
     [270,160,400,7],
     [250,130,350,4],
     [160, 60,200,3],
     [180, 40,150,5]]
p = [[1,0,1,0],       # 工場j が製品kを生産不可=1
     [0,0,0,1],
     [1,0,0,0]]
I = range(len(c))     # 顧客数5人の範囲I={0,1,2,3,4}
J = range(len(c[0]))  # 工場数3の範囲J={0,1,2}
K = range(len(d[0]))  # 製品数4の範囲K={0,1,2,3}
```

（次ページへ続く…）

多品種輸送問題：How to use gurobi?

- 多品種輸送問題の最適化（例1）
  - Google colaboratory / Python & gurobi

```
from gurobipy import *
m = Model("MultiTransport") # モデル(生産計画)の設定

x = {}                      # 変数設定:工場j→顧客iへの製品kの輸送量
for i in I:
  for j in J:
    for k in K:
      x[i,j,k] = m.addVar(lb=0.0, vtype="C",name="x[%s,%s,%s]" % (i+1,j+1,k+1))

for i in I:
  for k in K:
    m.addConstr(quicksum(x[i,j,k] for j in J) == d[i][k]) # 需要制約
for j in J:
  m.addConstr(quicksum(x[i,j,k] for i in I for k in K) <= b[j]) # 供給制約
  m.addConstr(quicksum(p[j][k]*x[i,j,k] for i in I for k in K) == 0) # 生産不可制約

m.setObjective(quicksum(c[i][j]*x[i,j,k] for i in I for j in J for k in K),
GRB.MINIMIZE) # 目的関数:輸送コスト最小

m.optimize()  # 最適化実行
m.printAttr('X') # 最適解の表示
```

多品種輸送問題：How to use gurobi?

- 多品種輸送問題の最適化（例1）
  - 実行結果（一部）

```
Solved in 3 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.448200000e+04
    Variable             X
-------------------------
    x[1,1,2]            85
    x[1,1,4]             6
    x[1,2,1]            80
    x[1,3,3]           300
    x[2,1,2]           160
    x[2,1,4]             7
    x[2,2,1]           270
    x[2,2,3]           260
    x[2,3,3]           140
    x[3,1,2]           130
    x[3,1,4]             4
    x[3,2,1]           250
    x[3,3,3]           350
    x[4,1,2]            45
    x[4,1,4]             3
    x[4,2,1]           160
    x[4,3,2]            15
    x[4,3,3]           200
    x[5,2,1]           180
    x[5,3,2]            40
    x[5,3,3]           150
    x[5,3,4]             5
```