Production de bureaux et de tables

Une usine fabrique plusieurs produits à vendre (des bureaux et des tables). La réalisation de chaque produit nécessite des ressources:

La production d'un bureau nécessite trois unités de bois, une heure de travail et 50 minutes de temps machine. La production d'une table nécessite cinq unités de bois, deux heures de travail et 20 minutes de temps de machine.

Chaque jour, L'usine dispose de 200 ouvriers qui travaillent huit heures chacun, 50 machines qui tournent chacune pendant 16 heures, et un approvisionnement de 3600 unités de bois. Les bureaux et les tables sont vendus à 700 et 900 l'unité, respectivement. L'usine peut vendre tout ce qu'elle arrive à produire.

x1= Le nombre de bureaux à produire chaque jour

x2= Le nombre de tables à produire chaque jour

On considère le PL suivant:

```
Max    700x1 + 900x2

s.t.    3x1 + 5x2 <= 3600    (Bois)
          x1 + 2x2 <= 1600    (Main d'oeuvre)
        50x1 + 20x2 <= 48000  (Machine)
          x1>=0 et x2>=0
```

1-Le programme de résolution

Pour créer un nouveau modèle avec gurobipy, ouvrez l'environnement de développement et saisissez ces codes dans un nouveau fichier python :

Sauvegardez le fichier et exécutez-le. Pour exécuter un fichier Python dans Jupyter Notebook, vous pouvez utiliser la commande magique %run en suivant les étapes suivantes:

- Ouvrez un notebook Jupyter et accédez au répertoire où votre fichier Python est enregistré.
- Créez une nouvelle cellule dans le notebook en cliquant sur l'icône plus dans la barre d'outils ou en appuyant sur la touche B de votre clavier.
- Dans la nouvelle cellule, tapez %run nom_du_fichier.py
- Exécutez la cellule en cliquant sur le bouton "Run" ou en appuyant sur

Shift+Enter sur votre clavier.

2- La résolution détaillée

Nous allons construire pas à pas le PL avec Gurobi. Il faut d'abord importer le package d'optimisation Gurobi appelé gurobipy.

In [1]:
```
from gurobipy import *
```

Utilisez les fonctions Gurobi **Model** et **addvar** pour créer un nouveau modèle et ajouter des variables. N'oubliez pas de définir les paramètres appropriés comme entrées de fonction.

Dans **addvar**, utilisez **lb** pour définir la limite inférieure, **vtype** pour définir le type et **name** pour définir le nom de la variable.

La valeur du paramètre **GRB.CONTINOUS** signifie que la variable est un nombre continu.

In [2]:
```
#Construire un nouveau modèle

eg1=Model("eg1")  #Le modèle est appelé "eg1"
x1=eg1.addVar(lb=0, vtype= GRB.CONTINUOUS, name="x1") #"x1" est le nombre
x2=eg1.addVar(lb=0, vtype= GRB.CONTINUOUS, name="x2") #"x2" est le nombre
```

```
Set parameter Username
```

Utilisez la fonction **setObjective** et **addConstr** pour définir la fonction objectif et ajouter des contraintes. Il est nécessaire de préciser s'il s'agit d'un problème de maximisation ou de minimisation. N'oubliez pas non plus de donner à toutes les contraintes, variables et modèles des noms distincts.

In [3]:
```
#Définissez la fonction objectif
#Utilisez GRB.MAXIMIZE pour un problème de maximisation
eg1.setObjective(700*x1+900*x2, GRB.MAXIMIZE)

#Ajoutez des contraintes et nommez les
eg1.addConstr(3*x1+5*x2<=3600,"Bois")
eg1.addConstr(x1+2*x2<=1600,"Main d'oeuvre")
eg1.addConstr(50*x1+20*x2<=48000,"Machine")
```

Out[3]:
```
<gurobi.Constr *Awaiting Model Update*>
```

Utilisez **optimize** pour exécuter et résoudre le modèle.

In [4]:
```
eg1.optimize()
```

```
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i7-4650U CPU @ 1.70GHz
Thread count: 2 physical cores, 4 logical processors, using up to 4 threa
ds

Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0xfc16a329
Coefficient statistics:
  Matrix range     [1e+00, 5e+01]
  Objective range  [7e+02, 9e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+03, 5e+04]
Presolve time: 0.04s
Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.0000000e+32   3.593750e+30   2.000000e+02      0s
       3    7.8947368e+05   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.07 seconds (0.00 work units)
Optimal objective  7.894736842e+05
```

Enfin, pour voir la solution vous pouvez utiliser **getVars** pour lister toutes les variables et utiliser **objVal** pour obtenir la valeur objective.

In [5]:
```
for var in eg1.getVars():
    print(var.varName, '=', var.x)
    print("Objective value =", eg1.objVal)
```

```
x1 = 884.2105263157895
Objective value = 789473.6842105263
x2 = 189.47368421052633
Objective value = 789473.6842105263
```

2- Découpage des données

Pour rendre les programmes Python flexibles et extensibles, vous devez découper les données d'un modèle. Pour le faire, il faut préparer plusieurs listes pour stocker les données.

- La liste contient les paramètres de l'instance
- La partie modèle ne contient qu'un modèle abstrait

Dans cet exemple, les listes sont dans le programme Python avant la partie modèle.

2.1- La partie données

Essayez maintenant de découpler les données du modèle:

In [6]:
```
produits = range(2)    # les deux produits
ressources = range(3)  # les ressources
prix = [700,900]

ressources_consommations = [[3,5],
                            [1,2],
                            [50,20]]
ressources_disponibilité = [3600,1600,48000]
```

2.2- La partie modèle

Vous pouvez maintenant écrire le modèle de manière plus simple.

In [7]:
```
eg1_de=Model("eg1_decoupage")

x=[]
for i in produits:
    x.append(eg1_de.addVar(lb=0, vtype = GRB.CONTINUOUS, name='x'+str(i))

eg1_de.setObjective(quicksum(prix[i]*x[i] for i in produits), GRB.MAXIMIZ

#ajoutez des contraintes et nommez-les.
eg1_de.addConstrs((quicksum(ressources_consommations[j][i]*x[i] for i in
    <= ressources_disponibilité[j] for j in ressources), "Limitation des

eg1_de.optimize()

for var in eg1_de.getVars():
    print(var.varName, '=', var.x)
    print("Objective value =", eg1_de.objVal)
```

```
Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (mac64[x86])

CPU model: Intel(R) Core(TM) i7-4650U CPU @ 1.70GHz
Thread count: 2 physical cores, 4 logical processors, using up to 4 threa
ds

Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0xfc16a329
Coefficient statistics:
  Matrix range     [1e+00, 5e+01]
  Objective range  [7e+02, 9e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [2e+03, 5e+04]
Presolve time: 0.01s
Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.0000000e+32   3.593750e+30   2.000000e+02      0s
       3    7.8947368e+05   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.02 seconds (0.00 work units)
Optimal objective  7.894736842e+05
x0 = 884.2105263157895
Objective value = 789473.6842105263
x1 = 189.47368421052633
Objective value = 789473.6842105263
```

- Utilisez des listes pour stocker les données.
- La longueur de la liste et de la matrice doit être cohérente.
- Utilisez **.append()** pour ajouter un nouvel élément dans une liste.
- Faites attention aux **[ ]**, **:**, et à l'espace et au moment de leur utilisation.
- Dans **setObjective**, utilisez **GRB.MAXIMIZE** pour les problèmes de maximisation.
- Utilisez **quicksum** et **for** additionner un groupe de variables.
- Utilisez **addConstrs** et **for** pour ajouter plusieurs contraintes en une seule commande. Attention à la longueur des listes

3- Remarques

- RQ1: Python est sensible à la casse (case-sensitive): Le comportement du système différencie les lettres majuscules et minuscules. Par exemple, "Hello" et "hello" seraient considérés comme deux chaînes de caractères différentes.
- RQ2: Le découpage modèle-données rend le modèle plus extensible: Il ne doit pas y avoir de numéro codé en dur (Hard-coded number) dans la partie modèle. "Nombre codé en dur" désigner un nombre qui est directement intégré dans le code source d'un programme plutôt que d'être stocké dans une variable ou récupéré à partir d'une entrée utilisateur ou d'une source externe. ill ne peut être modifié sans modifier le code source. Cette pratique peut être utile pour des valeurs qui ne changent jamais, comme une constante mathématique
- RQ3: La solution optimale consiste à produire chaque jour 884,2 bureaux, et 189,5 tables. Cela ne semble pas si raisonnable car un bureau est un bureau, on ne peut pas faire 0,2 bureaux. On peut dire alors que ce PL n'est pas réaliste et la solution ne peut pas être exécutée. Parfois, cela peut être vrai, mais en fait, dans la plupart des cas, ce n'est pas le cas. Parce que réellement dans une usine, on continue à travailler jusqu'à finalement arriver en fin de journée à fabriquer la moitié d'un tableau. Puis demain on commence à faire la deuxième partie et arriver à construire un bureau complet. Cela signifie que 189,5 est un concept de moyenne. Si on prend cela en considération, on se rend compte que c'est réalisable.