Exercice 1. Affectation des ressources

Nous allons résoudre le problème suivant. Une entreprise fabrique deux types de produits : des téléphones fixes et des téléphones portables.

Chaque produit est assemblé et peint par l'entreprise, qui veut produire au moins 100 unités de chaque produit.

- Un téléphone fixe requiert 12 min. sur la machine d'assemblage, et 30 min. sur la machine de peinture.
- Un téléphone portable requiert 24 min. sur la machine d'assemblage, et 24 min. sur la machine de peinture.
- La machine de peinture est disponible 400 heures.
- La machine d'assemblage est disponible 490 heures.
- Chaque téléphone fixe produit génère un profit net de 12€, et chaque téléphone portable génère un profit net de 20€.

L'objectif du problème est de maximiser le profit total.

Question 1. Ecrire un programme linéaire modélisant le problème. Le modèle sera écrit d'abord pour le cas particulier et ensuite de manière générique (pour N produits et R ressources machine).

Exercice 2. Dimensionnement de lots

On s'intéresse ici à un problème de planification de la production. Dans ce problème, on suppose l'horizon temporel de planification partitionné ("discrétisé") en T périodes de temps, et on considère une usine pouvant fabriquer N produits différents (les quantités produites ne sont pas nécessaire entières).

Pour chacune des périodes de temps t et chaque produit i, on connaît :

- la demande des clients d_{it},
- le coût de fabrication unitaire α_{it}
- le coût de stockage unitaire γ_{it}

Les coûts de stockage sont comptabilisés à la fin de chaque période : le coût de stockage pour le produit i à la période t s'applique sur la quantité de produit i en stock à la fin de la période, qui est égale à la quantité en stock à la fin de la période précédente, plus la quantité produite lors de la période, moins la demande satisfaite au cours de la période. Le coût de stockage total est la somme des coûts de stockage pour chaque produit et chaque période.

Dans ce contexte, il est ainsi avantageux de produire aux périodes où la production est la moins coûteuse pour satisfaire des demandes ultérieures.

Le problème consiste à décider quelle quantité de chaque produit fabriquer et quelle quantité stocker pour chaque période, de manière à satisfaire toutes les demandes et en minimisant le coût total (coûts de production + coûts de stockage).

Question 1. Modéliser le problème sous forme de programme linéaire, en se basant sur des variables x_{it} représentant la quantité de produit i fabriqué en t, et des variables s_{it} représentant la quantité de produit i présente en stock à la fin de la période t.

La capacité de production de l'usine est en fait limitée pour chaque période de temps. Plus précisément, on connaît, pour chaque période t :

- la capacité (en heures) de l'usine C_t,
- la durée de fabrication unitaire de chaque produit i, v_{it},

Question 2. Modifier le modèle pour prendre ces contraintes en compte.

On apprend à présent que, chaque fois que la fabrication d'un type de produit est lancée durant une période, des réglages doivent être effectués. Plus précisément, pour chaque période t et chaque produit i, s'il y a une production de i en t :

- il faut comptabiliser f_{it} heures de réglage durant lesquelles l'usine ne fabrique rien, quelque soit le nombre d'unités de i fabriquées durant t,
- il y a un coût supplémentaire de β_{it}, quelque soit le nombre d'unités de i fabriquées durant t.

Lorsque la quantité fabriquée d'un produit lors d'une période est nulle, il n'y a pas de réglage associé et ces coût et durée fixes n'interviennent pas.

Question 3. Adapter le modèle pour se conformer au nouveau problème. On pourra utiliser une variable binaire y_{it} indiquant si la quantité de produit i fabriquée en t est nulle ou non. Si vous utilisez une constante M, indiquez-en une valeur valide.

Au vu des ressources limitées de l'usine et la demande croissante, on décide de ne pas satisfaire systématiquement la demande. Bien entendu, chaque unité de produit i dont la demande à la période t n'est pas satisfaite est pénalisée par un coût φ_{it} (qui correspond à sous-traiter la production ou à pénaliser la dégradation de l'image de l'entreprise en cas de rupture).

Question 4. Modifier le modèle pour prendre en compte cette nouvelle possibilité.

Exercice 3. Cutting-Stock

Nous avons des pièces qui ont toutes une longueur `RollWidth`. Chacune de ces pièces peut être découper en plusieurs pièces. Nous avons reçu `NbItems` demandes. A chaque demande `d` correspond une longueur particulière `Size[d]` (évidemment inférieure à `RollWidth`) et un nombre de pièces à fournir `Amount[d]`. Cependant une pièce originale ne peut être découpée que selon un des motifs possibles décrits dans le tableau `Patterns` ; l'objet `Pattern` qui représente un motif est un tuple composé de trois élément : son id, son coût, et un tableau qui contient pour chaque longueur donnée les nombres de pièces résultantes, suite au découpage de la pièce originale selon le motif en question. Nous supposons ici que le coût est constant ; il est égal à 1 pour tous les motifs. L'objectif est de répondre à toutes les demandes en utilisant le nombre minimum des pièces originales.

```
NbItems = 5;
RollWidth = 110;
Size = [20, 45, 50, 55, 75];
Amount = [48, 35, 24, 10, 8];

Patterns = { <0, 1, [1, 0, 0, 0, 1]>,
<1, 1, [0, 1, 0, 0, 0]>,
<2, 1, [0, 0, 1, 0, 0]>,
<3, 1, [1, 0, 0, 1, 0]>,
<4, 1, [0, 0, 0, 0, 1]> };
```

Question 1. Ecrire un programme linéaire modélisant le problème.

Exercice 4. Premiers pas avec Julia

Lancer `julia` dans un terminal puis executer les commandes (instructions) suivantes en lisant à chaque fois le retour.

```
# Vecteurs , Matrices et tableaux.
a = [1; 2; 3]
b = [4 5 6]
A = [1 2 3; 4 5 6]

A[1,3]
A[2,1]
transpose(A)
A'

a = [1; 2; 3]
c = [7; 8; 9]
a * c'
a' * c
dot(a,c)


eye(3)
zeros(2,3)
```

```
ones(3,2)

B = [1 3 2; 3 2 2; 1 1 1]
inv(B)
B * inv(B)
inv(B)[2,1]

a = [1; 2; 3]
b = [1.0; 2; 3]

d = Array{Float64}(3)
d[1] = 1
d[2] = 2
d[3] = 3
d


# Tuple
pairs = Array{Tuple{Int64, Int64}}(3)
pairs[1] = (1,2)
pairs[2] = (2,3)
pairs[3] = (3,4)
pairs
pairs = [ (1,2); (2,3); (3,4) ]

ijk_array = Array{Tuple{Int64, Int64, Int64}}(3)
ijk_array[1] = (1,4,2)

# Indices and Intervalles
a = [10; 20; 30; 40; 50; 60; 70; 80; 90]
a[1:3]
a[1:3:9]
a[end-2:end]

b = [200; 300; 400]
a[2:4] = b
a

c = 1:2:9
d = collect(1:2:9)

A = [1 2 3; 4 5 6; 7 8 9]
A[:,2]
A[3,:]

# Affichage de Messages
println("Hello World")
print("Hello "); print("World"); print(" Again")
println("Hello "); println("World"); println(" Again")

a = 123.0
println("The value of a = ", a)
println("a is $a, and a-10 is $(a-10).")
b = [1; 3; 10]
println("b is $b.")
println("The second element of b is $(b[2]).")
```