M2 ID
TRANSPARENCE DES ALGORITHMES
POUR LA DÉCISION
PROJET

L'ALGORITHME DE CALCUL DU NUTRI-SCORE DES PRODUITS ALIMENTAIRES EST-IL TRANSPARENT ?

PARTIE 1

Année académique 2019/2020

L'objectif de ce projet est de sensibiliser les étudiants à une démarche d'analyse, d'interprétation et de transparence des modèles d'évaluation utilisés pour l'élaboration des classements ou classification de produits, d'organismes, de personnes, etc. Nous examinerons ici le cas du Nutri-score, un système d'étiquetage nutritionnel pour faciliter le choix d'achat du consommateur, au regard de la composition nutritionnelle des produits . Ce travail sera réalisé à l'aide d'algorithmes implémentés en langage Python.

1 Le Nutri-score, qu'est-ce que c'est ?

1.1 Introduction

Le Nutri-Score, est un système d'étiquetage nutritionnel qui répartit les produits en cinq classes (5 couleurs), du vert (catégorie A) pour les produits de très bonne qualité nutritionnelle au rouge (catégorie E) pour ceux dont il vaut mieux limiter la consommation. Il a été mis en place en janvier 2016, par le gouvernement français, dans le cadre de la loi de modernisation du système de santé. Son but est de faciliter le choix d'achat du consommateur, au regard de la composition nutritionnelle des produits. Ainsi, il favorise le choix de produits plus sains par les consommateurs et par conséquent, participe à la lutte contre l'augmentation des maladies cardiovasculaires, de l'obésité et du diabète.

Le logo Nutri-Score a été conçu par Santé publique France, à la demande de la Direction générale de la santé, en s'appuyant sur les travaux de l'équipe du Professeur Serge Hercberg (Université Paris 13), les expertises de l'Anses (Agence nationale de sécurité sanitaire de l'alimentation, de l'environnement et du travail) et du Haut Conseil de Santé Publique.

FIGURE 1 – Logo Nutri-score

L'ensemble des articles scientifiques et documents publiés relatifs au Nutri-Score est accessible sur le site internet `https://solidarites-sante.gouv.fr/prevention-en-sante/preserver-sa-sante/nutrition/article/articles-scientifiques-et-documents-publies-relatifs-au-nutri-score`.

1.2 Méthode de calcul du Nutriscore

Cette section est rédigée à partir du Rapport d'appui scientifique et technique de l'Anses sur le Nutri-score [1].

Le Nutri-score permet de calculer le score nutritionnel des produits alimentaires en se basant sur :

- **Les nutriments ou apports à favoriser** : fibres, protéines, fruits, légumes, légumineuses, noix ;
- **Les nutriments ou apports à limiter** : énergie (kcal), acides gras saturés, sucres, sel.

Cette note est proposée après analyse d'une portion de 100 grammes ou 100 ml s'il s'agit de boisson. Plus formellement, le Nutri-score est basé sur le score nutritionnel des aliments, tel que défini par Rayner et al. [2], est un score intégrant :

- Une composante dite «négative», calculée à partir des teneurs en nutriments dont la consommation doit être limitée : énergie (kJ/100g), sucres simples (g/100g), acides gras saturés (g/100g) et sodium (mg/100g) ;
- Une composante dite «positive», calculée en intégrant les teneurs en nutriment dont la consommation est recommandée : fibres (g/100g), protéines (g/100g) ;
- Une deuxième composante «positive», calculée à partir des teneurs d'une catégorie spécifique d'aliments : les fruits/légumes/fruits à coque (g/100g)

Les composantes "positives" et "négatives" sont chacune associées à un score plus ou moins important, en fonction de la composition nutritionnelle de l'aliment considéré :

- De 0 à 10 pour les nutriments de la composante "négative" (Voir Tableau de la Figure 3) ;
- De 0 à 5 pour les éléments de la composante " positive" (Voir Tableau de la Figure 2).

Ainsi, le score associé à la composante "négative" peut théoriquement aller de 0 à 40. Celui de la composante "positive" peut quant à lui théoriquement aller de 0 à 15.

Points	Fruits, leg (%)	Fibres (g) AOAC	Protéine (g)
0	≤ 40	≤ 0.9	≤ 1,6
1	> 40	> 0.9	> 1,6
2	> 60	> 1.9	> 3,2
3	-	> 2.8	> 4,8
4	-	> 3.7	> 6,4
5	> 80	> 4.7	> 8,0

FIGURE 2 – Table de Calcul des points attribués à chacun des éléments de la composante dite "positive" selon la méthodologie développée par Rayner et al.

Pour chaque aliment, le score nutritionnel est ensuite calculé selon la méthodologie présentée à la Figure 1 5.

Dans la plupart des cas, le score est calculé en retranchant le score de la composante "positive" à celui de la composante "négative" (voir Figure 4).

$$\text{Nutri-Score} = \text{nutriments à limiter (a)} - \text{nutriments à favoriser (b)} \quad (1)$$

Cependant, si le score de la composante "négative" est supérieur ou égal à 11 et que la teneur en fruits/légumes/fruits à coque ne dépasse pas 80%, alors les protéines ne sont plus prises en compte dans le calcul du score nutritionnel.

1. Évaluation de la faisabilité du calcul d'un score nutritionnel tel qu'élaboré par Rayner et al. (pages 12-15 et 27-30). Mars 2015
2. Application of the Nutrient profiling model : Definition of "fruit, vegetables and nuts" and guidance on quantifying the fruit, vegetable and nut content of a processed product - Peter Scarborough, Mike Rayner, Anna Boxer and Lynn Stockley - British Heart Foundation - Health Promotion Research Group, Department of Public Health, University of Oxford - December 2005.

Points	Valeur énergétique (kJ/100g)	Acides gras saturés (g/100g)	Sucres (g/100g)	Sodium (mg/100g)
0	≤335	≤1	≤4,5	≤90
1	>335	>1	>4,5	>90
2	>670	>2	>9	>180
3	>1005	>3	>13,5	>270
4	>1340	>4	>18	>360
5	>1675	>5	>22,5	>450
6	>2010	>6	>27	>540
7	>2345	>7	>31	>630
8	>2680	>8	>36	>720
9	>3015	>9	>40	>810
10	>3350	>10	>45	>900

FIGURE 3 – Table de Calcul des points attribués à chacun des nutriments de la composante dite “négative” selon la méthodologie développée par Rayner et al.

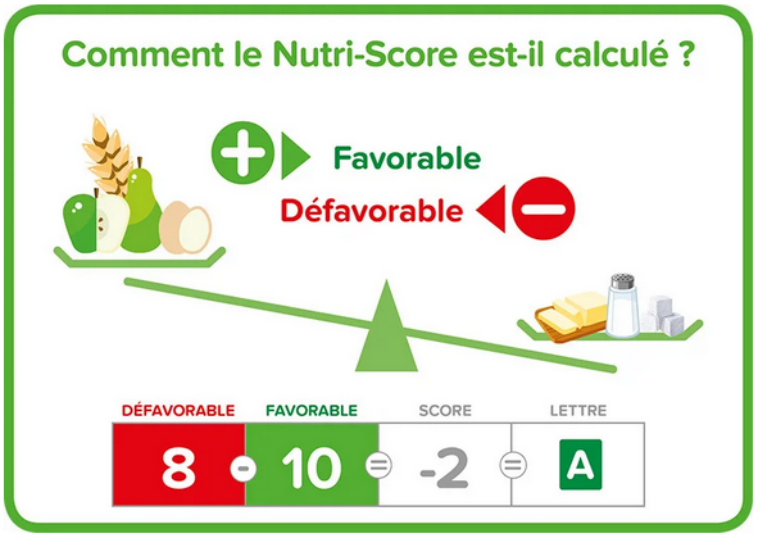

FIGURE 4 – Formule générale du Nutri-score

Le score nutritionnel final peut ainsi théoriquement varier de -15 à $+40$ en fonction des produits. **Plus il est faible, plus le produit est considéré comme ayant un profil nutritionnel favorable**.

L’affectation d’un produit à une classe, à partir de son score Nutri-score, se fait suivant le tableau de la Figure 6 où sont mentionnés les seuils des 5 catégories.

1.3 Exemple de calcul de Nutri-score : Yaourt Le Nature - Danone-500 g (4 × 125 g)

- Kilojoules (Kj/100g) : 188 $\Longrightarrow$ **0 point**

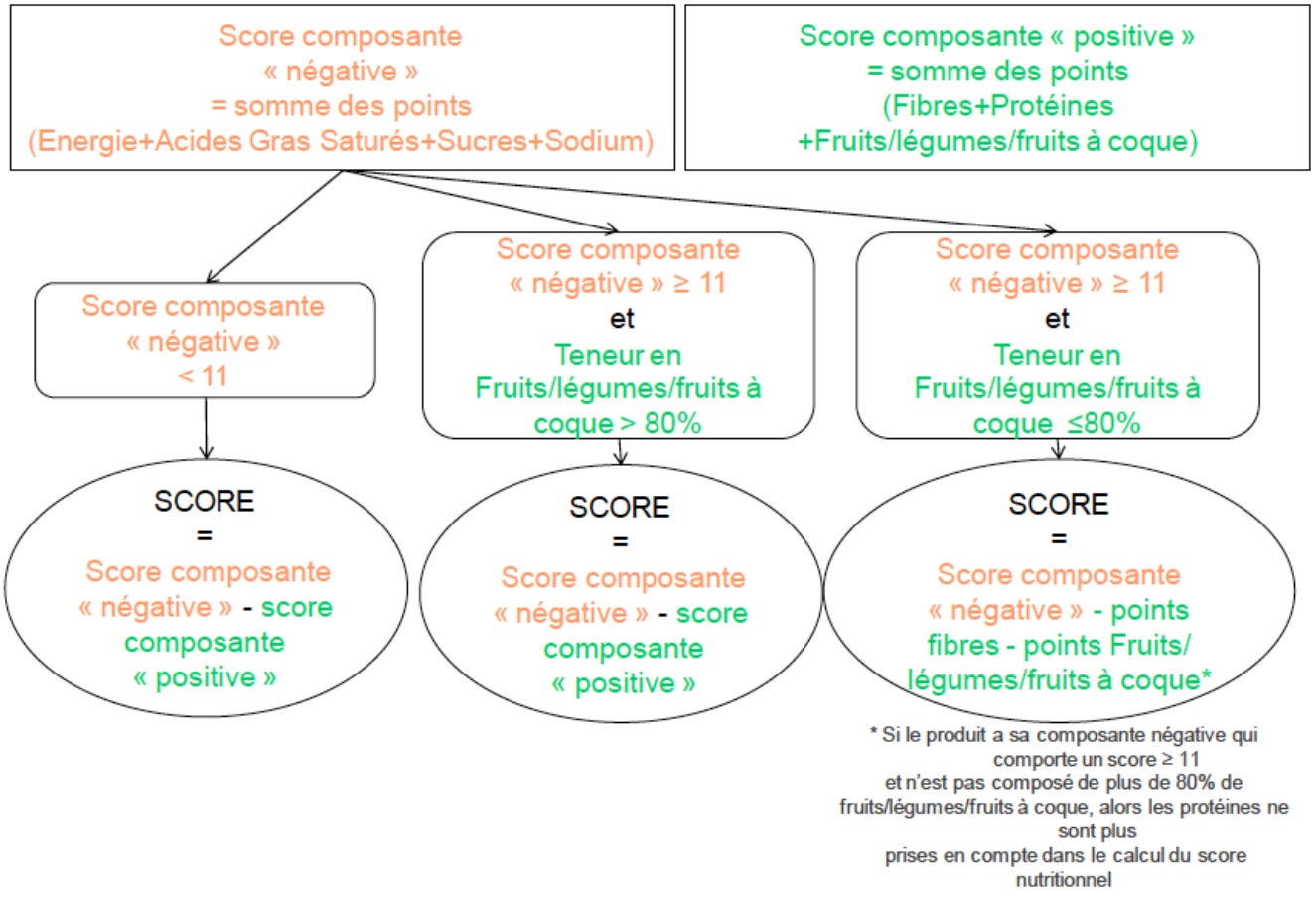

FIGURE 5 – Calcul du score nutritionnel selon la méthodologie développée par Rayner et al.

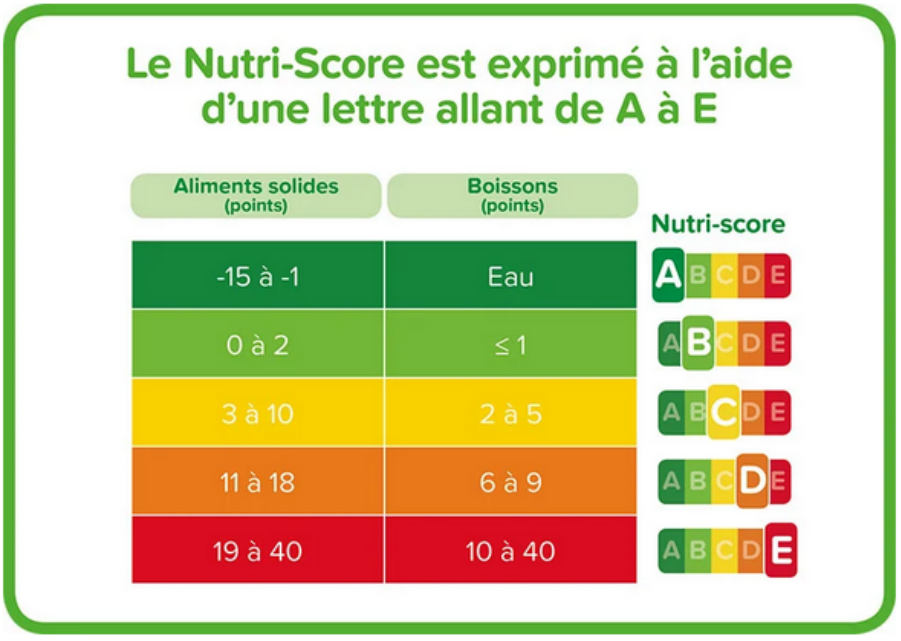

FIGURE 6 – Définition des 5 classes utilisées dans le cadre du Nutri-score

- Sucres (g/100g) : 5.1 $\Longrightarrow$ **1 points**
- Acides gras saturés (g/100g) : 0.6 $\Longrightarrow$ **0 points**

- Sodium (mg/100g) : 0.04 $\Longrightarrow$ **0 points**
- Protéines (g/100g) : 3.8 $\Longrightarrow$ **2 points**
- Fibres AOAC (g/100g) : 0 $\Longrightarrow$ **0 points**

Le score du produit est égal à $[(0+1+0+0)-(2+0)] = -1$ points, et donc il appartient à la classe A.

1.4 Exemple de calcul de Nutri-score : le cas des Céréales pour petit déjeuner (marque CIQUAL)

- Kilojoules (Kj/100g) : 1669 $\Longrightarrow$ **4 point**
- Sucres (g/100g) : 26.3 $\Longrightarrow$ **5 points**
- Acides gras saturés (g/100g) : 2.37 $\Longrightarrow$ **2 points**
- Sodium (mg/100g) : 328 $\Longrightarrow$ **3 points**
- Protéines (g/100g) : 7.99 $\Longrightarrow$ **0 points** (ici on attribuera 0 point aux protéines, au lieu de 4 points, car comme le stipule l'algorithme, la partie négative a plus de 11 points)
- Fibres AOAC (g/100g) : 4.83 $\Longrightarrow$ **5 points**

Le score du produit est égal à $[(4+5+2+3)-(0+5)] = 9$ points, et donc il appartient à la classe C.

1. *A la lecture de cette section, ainsi que de vos propres ressources documentaires, que pensez-vous de la transparence sur l'étiquetage des aliments par le calcul du Nutri-score (par exemple, du point de vue consommateur et/ou informaticien, ...) ? Vous pouvez appuyez votre analyse en indiquant (et en justifiant) selon vous les points qui vous semblent positifs et/ou insuffisants.*

2 Le Nutri-score vu comme un problème d'Aide Multicritères à la Décision

Dans la suite, nous considèrerons le calcul du Nutri-Score comme un problème d'Aide à la Décision Multicritères où :

- L'ensemble X des alternatives correspond à l'ensemble des produits à analyser.
- Les six critères (constituant l'ensemble N) à prendre en compte pour une quantité de 100 g seront :
  1. **La valeur énergétique** (Kcal/KJ) (critère à minimiser)
  2. **La quantité d'acides gras saturés** (g) (critère à minimiser)
  3. **La quantité de sucres** (g) (critère à minimiser)
  4. **La quantité de Sodium** (g/mg) (critère à maximiser)
  5. **La quantité de protéines** (g) (critère à maximiser)
  6. **La quantité de Fibres** (g) (critère à maximiser)

2. *Construire la fonction* `criteriaSet(fichier)` *qui retournera, à partir d'un fichier Excel ou csv, l'ensemble N des critères à prendre en compte dans notre problème. On supposera donc que le fichier Excel ou csv contient n critères et pour chaque critère l'information indiquant s'il est à minimiser ou à maximiser.*

   L'ensemble N retournée pourra par exemple être un dictionnaire (ou une liste ou une matrice, etc).

   Dans notre exemple à 6 critères, on obtiendrai par exemple :

```
criteriaSet(fichier)={"La valeur énergétique": "Min";
"La quantité d'acides gras saturés": "Min";
"La quantité de sucres": "Min"; "La quantité de Sodium": "Min";
"La quantité de protéines": "Max"; "La quantité de Fibres": "Max";}
```

3. Construire la fonction `performanceMatrix(fichier)` qui retournera, à partir d'un fichier Excel ou csv, la matrice de performance (encore appelée tableau de performance) associée à notre problème. Cette matrice pourra par exemple être un dictionnaire ou une liste ou une matrice, etc.

3 Élaboration d'une base de données de produits alimentaires

Chaque groupe devra constituer une base de données de 50 produits alimentaires. Il est conseillé de travailler avec des aliments solides ou liquides. Le nombre de produits alimentaires commun entre deux groupes ne devra pas excéder 20.

Pour récupérer des informations sur les produits alimentaires (liste des ingrédients, Nutri-Score, ...), vous pouvez vous rendre dans des supermarchés ou encore sur le site internet de l'association OPEN FOOD FACT `https://fr.openfoodfacts.org/`.

La base de données ainsi constituée permettra de construire un fichier Excel (ou équivalent) de données, lequel fichier pourra servir de paramètre d'entrée aux algorithmes développées dans le cadre de ce projet.

4 Le Nutri-score expliqué par une fonction d'utilités additive ?

Nous supposons dans cette partie que le Nutri-score d'un aliment $x = (x_1, \ldots, x_n)$ est donné par la note globale $F(u_1(x_1), \ldots, u_n(x_n))$ obtenue par la formule suivante :

$$F((u_1(x_1), \ldots, u_n(x_n)) = \sum_{i=1}^{n} u_i(x_i) \tag{2}$$

où

- x_i est la valeur de l'aliment x associée au critère i.
- La fonction $u_i : X \longrightarrow \mathbb{R}$, $i = 1, \ldots, n$, est appelée fonction d'utilité marginale associée au critère i. Cette fonction sera croissante ou décroissante, selon que le critère i est maximiser ou à minimiser.
- F est appelée fonction d'agrégation additive.

Pour déterminer les fonctions $u_i,\ i = 1, \ldots, n$, il suffit d'exprimer chaque score d'un aliment comme équation linéaire et les informations préférentielles issues des données sous forme d'inéquations linéaires.

4. *Existe-t-il une fonction additive F, comme définie à l'équation* (2)*, compatible avec les informations nutritionnelles des aliments et leur Nutri-score (aliments recensés dans votre base de données) ?*
5. *Le modèle obtenu est-il plus avantageux, en termes de transparence, par rapport au modèle officiel du Nutri-Score ? Justifiez votre réponse.*

Calcul scientifique en Python

Deux modules de calcul scientifique du langage Python pourront nous être utiles dans la modélisation des préférences du projet : NumPy et SciPy. La distribution Canopy contient les deux modules.

Module NumPy

NumPy propose des tableaux multidimensionnels pour Python ainsi qu'une large gamme d'opérations efficaces sur ces tableaux : arithmétique, fonctions mathématiques, opérations structurales, etc. Les opérations sont inspirées par des langages comme APL ou Matlab. On y trouve aussi des fonctions permettant de résoudre des problèmes d'algèbre linéaire tels que la résolution de système d'équations. NumPy est donc la bibliothèque de base pour toute application de Python dans le domaine du calcul scientifique. Il s'agit d'un module stable, bien testé et relativement bien documenté : `http://docs.scipy.org/doc/`,`http://docs.scipy.org/doc/numpy/reference/`. Pour importer ce module, on recommande d'utiliser

```
>>> import numpy as np
```

Toutes les fonctions NumPy seront alors préfixées par `np`. Exécutez et analysez le code suivant sur la manipulation simple et efficace des tableaux :

```
>>> x = np.arange(0,2.0,0.1) # De 0 (inclus) à 2 (exclus) par pas de 0.1
>>> x
array([ 0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ,
1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9])
>>> np.size(x) # Sa taille
20
>>> x[0] # Le premier élément
0.0
>>> x[1] # Le deuxième élément
0.10000000000000001
>>> x[19] # Le dernier élément
1.9000000000000001
>>> x[20] # Pas un élément !
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError : index 20 is out of bounds for axis 0 with size 20
>>> a = np.array ([[1,2,3], [4,5,6], [7,8,9]])
>>> a
array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
>>> b = 2 * a # Multiplication de chaque terme
>>> c = a + b # Sommation terme à terme
>>> np.dot(a, b) # Produit de matrices
array([[ 60, 72, 84],
[132, 162, 192],
[204, 252, 300]])
>>> a * b # Produit terme à terme
array([[ 2, 8, 18],
```

```
[ 32, 50, 72],
[ 98, 128, 162]])
>>> a=np.array([1,2])
>>> np.outer(a,a) # Calcule le produit a*transpose(a), soit une matrice.
array([[1, 2],
[2, 4]])
```

On peut facilement effectuer des coupes dans un tableau Numpy. Cette fonctionnalité est particulièrement importante en calcul scientifique pour éviter l'utilisation de boucles.

```
>>> t = np.array([1,2,3,4,5,6])
>>> t[1 :4] # de l'indice 1 à l'indice 4 exclu !!!ATTENTION!!!
array([2, 3, 4])
>>> t[:4] # du debut à l'indice 4 exclu
array([1, 2, 3, 4])
>>> t[4 :] # de l'indice 4 inclus à la fin
array([5, 6])
>>> t[:-1] # excluant le dernier element
array([1, 2, 3, 4, 5])
>>> t[1 :-1] # excluant le premier et le dernier
array([2, 3, 4, 5])
>>> a = np.arange(10)
>>> np.sum(a)
45
>>> np.mean(a)
4.5
>>> M=np.array([[0,1],[2,3]])
>>> M
array([[0, 1],
[2, 3]])
>>> M.transpose() # Calcule la transposée de la matrice M.
array([[0, 2],
[1, 3]])
```

Il est également possible de résoudre des systèmes linéaires. Par exemple, pour résoudre le système d'équations $\begin{cases} 3x_0 + x_1 = 9 \\ x_0 + 2x_1 = 8 \end{cases}$, on peut exécuter :

```
>>> a = np.array([[3,1], [1,2]])
>>> b = np.array([9,8])
>>> x = np.linalg.solve(a, b)
>>> x
array([ 2., 3.])
```

Module SciPy

Scipy est un projet visant à unifier et fédérer un ensemble de bibliothèques Python à usage scientifique. Cette distribution de modules est destinée à être utilisée avec le langage interprété Python afin de créer un environnement de travail scientifique très similaire à celui offert par Matlab. Il contient par exemple des modules pour l'optimisation, l'algèbre linéaire, les statistiques ou encore le traitement du signal. Il offre également des possibilités avancés

de visualisation grâce au module “Matplotlib”. SciPy est un module stable, bien testé et relativement bien documenté. `http://docs.scipy.org/doc/`,`http://docs.scipy.org/doc/scipy/reference/`.

Le module SciPy réalise les différentes opérations sur des tableaux numériques (`ndarray`) de NumPy. On peut donc directement utiliser ces tableaux comme arguments pour les différentes fonctions.

```
>>> import scipy
>>> from scipy import linalg
>>> mat = np.array([[1, 2], [2, 4]])
>>> mat
array([[1, 2],
[2, 4]])
>>> linalg.det(mat)
0.0
```

Exécutez et analysez le code sur la résolution du programme linéaire suivant : $\begin{cases} \min 70\ x_1 + 80\ x_2 + 85\ x_3 \\ s.t. \\ x_1 + x_2 + x_3 = 999 \\ x_1 + 4\ x_2 + 8\ x_3 \leq 4500 \\ 40\ x_1 + 30\ x_2 + 20\ x_3 \leq 36000 \\ 3\ x_1 + 2\ x_2 + 4\ x_3 \leq 2700 \\ x_1 \geq 0 \end{cases}$

```
import numpy as np
from scipy.optimize import linprog
from numpy.linalg import solve

A_eq = np.array([[1,1,1]])
b_eq = np.array([999])

A_ub = np.array([
[1, 4, 8],
[40,30,20],
[3,2,4]])

b_ub = np.array([4500, 36000,2700])

c = np.array([70, 80, 85])

res = linprog(c, A_eq=A_eq, b_eq=b_eq, A_ub=A_ub, b_ub=b_ub,
bounds=(0, None))

print('Optimal value:', res.fun, '\nX:', res.x)

Optimization terminated successfully.
Current function value: -22.000000
Iterations: 1
('Optimal value:', 73725.0, '\nX:', array([ 636.,  330.,   33.]))
```

Le package `optlang` permet également de résoudre des programmes linéaires via SciPy. Par exemple, en résolvant le

programme linéaire :
$$\begin{cases} \max 10\,x_1 + 6\,x_2 + 4\,x_3 \\ s.t. \\ x_1 + x_2 + x_3 \leq 100 \\ 10\,x_1 + 4\,x_2 + 5\,x_3 \leq 600 \\ 40\,x_1 + 30\,x_2 + 6\,x_3 \leq 300 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}$$

```
from optlang import Model, Variable, Constraint, Objective

# All the (symbolic) variables are declared, with a name and
optionally a lower and/or upper bound.

x1 = Variable('x1', lb=0)
x2 = Variable('x2', lb=0)
x3 = Variable('x3', lb=0)

# A constraint is constructed from an expression of variables and a
lower and/or upper bound (lb and ub).

c1 = Constraint(x1 + x2 + x3, ub=100)
c2 = Constraint(10 * x1 + 4 * x2 + 5 * x3, ub=600)
c3 = Constraint(2 * x1 + 2 * x2 + 6 * x3, ub=300)

# An objective can be formulated

obj = Objective(10 * x1 + 6 * x2 + 4 * x3, direction='max')

# Variables, constraints and objective are combined in a Model
object, which can subsequently be optimized.

model = Model(name='Simple model')
model.objective = obj
model.add([c1, c2, c3])
status = model.optimize()
print("status:", model.status)
print("objective value:", model.objective.value)
print("----------")

for var_name, var in model.variables.items():
print(var_name, "=", var.primal)
```

On obtient comme résultat :

```
('status:', 'optimal')
('objective value:', 733.3333333333333)
----------
('x1', '=', 33.33333333333333)
('x2', '=', 66.66666666666667)
('x3', '=', 0.0)
```

Introduction to Linear programming with Python

Linear programming is the technique used to maximize or minimize a function. The idea is to optimize a complex function by best representing them with linear relationships. In simpler terms, we try to optimize (to maximize or minimize) a function denoted in linear terms and bounded by linear constraints.

Use case - Miracle worker

Let's try to formalize an use-case and carry it forward throughout the article. Suppose you are a magical healer and you goal is to heal anyone who asks for help. The more you are able to heal someone, the better. Your secret behind the healing is 2 medicines, each of which uses special herbs. To create one unit of medicine 1, you need 3 units of herb A and 2 units of herb B. Similarly, to create one unit of medicine 2, you need 4 and 1 units of herb A and B respectively. Now medicine 1 can heal a person by 25 unit of health (whatever it is) and medicine 2 by 20 units. To complicate things further, you only have 25 and 10 units of herb A and B at your disposal. Now the question is, how many of each medicine will you create to maximize the health of the next person who walks in ?

Modeling the problem

First let's try to identify the objective (what we want to do and how) and constraint (the bounding functions) of the stated problem.

As it's clear from the problem, we want to increase the health by as many units as possible. And medicines are the only thing which can help us with it. What we are unsure of, is the amount to each medicines to create. Going by a mathematician's logic, lets say we create x units of medicine 1 and y units of medicine 2. Then the total health restored can be given by,

$$25 \times x + 20 \times y = \text{Health Restored}$$

Where

$x =$ Units of medicine 1 created
$y =$ Units of medicine 2 created

This is the objective function, which we want to maximize. Now both the medicines are dependent on the herbs which we have in finite quantity. Let's understand the constraints. If we create x and y units of medicine 1 and 2,

- We use $3x + 4y$ units of herb A. But we only have 25 units of it, hence the constraint is, our total usage of herb A should not exceed 25, denoted by,
$$3x + 4y \leq 25$$
- We use $2x + 1y$ units of herb B. We have 10 units of it, hence the constraint is, our total usage of herb B should not exceed 10, denoted by,
$$2x + y \leq 10$$
- Also the amount of medicines created cannot be negative (doesn ?t make sense) hence, they should be equal or greater than zero, denoted by,
$$x \geq 0;\ y \geq 0$$

Solution - Graphical representation

One way to solve the problem is by representing it into graph which requires plotting the functions, constraints and finding any point of interest. Let's plot our functions and see what we can infer from it.

The point of intersection, as obvious, from the plot is $(3, 4)$, which says, If we create 3 units of medicine 1 and 4 units of medicine 2, considering the constraints on herbs, we are best equipped to heal the next patient. Intuitively, we wanted to find a solution which satisfy all of our constraints. As the constraint have few variables (only x and y), transforming the problem into graph of lower dimension, we can visualize it as a 2D plot. With constraints on herbs being transformed into lines, our solution now transforms into a point of intersection. To make matter things better, it was on the positive quadrant, satisfying our 3rd constraint.

But what about problems with larger variable count ? or with more constraints ? wouldn't it be difficult to plot and visualize them all ? And do you always want to plot and solve these kinds of problems ? Lets try to leverage modern computing prowess.

Solution - Python Programming

Python has a nice package named PuLP which can be used to solve optimization problems using Linear programming. To start with we have to model the functions as variables and call PuLP ?s solver module to find optimum values. Here it goes,

```
1  # import PuLP
2  from pulp import *
3
4  # Create the 'prob' variable to contain the problem data
5  prob = LpProblem("The Miracle Worker", LpMaximize)
6
7  # Create problem variables
8  x=LpVariable("Medicine_1_units",0,None,LpInteger)
9  y=LpVariable("Medicine_2_units",0, None, LpInteger)
10
11 # The objective function is added to 'prob' first
12 prob += 25*x + 20*y, "Health restored; to be maximized"
13 # The two constraints are entered
14 prob += 3*x + 4*y <= 25, "Herb A constraint"
15 prob += 2*x + y <= 10, "Herb B constraint"
16
17 # The problem data is written to an .lp file
18 prob.writeLP("MiracleWorker.lp")
19
20 # The problem is solved using PuLP's choice of Solver
21 prob.solve()
```

Lets try to understand the code,

- Line 1-2 : We import the PuLP package.
- Line 4-5 : We define our problem by giving a suitable name, also specifying that our aim is to maximize the objective function.
- Line 7-9 : Here, we define LpVariable to hold the variables of objective functions. The next arguments specifies the lower and upper bound of the defined variable. As per 3rd constraint, its non-negative, hence made second argument as 0 and 3rd as None (in-place of infinity). The last argument says if the required output is discrete or continuous.
- Line 11-12 : Denotes the objective function in terms of defined variables, along with a short description.
- Line 13-15 : Entering the herb's constraints in terms of variables.
- Line 17-18 : Output the objective and constraints into a file for portability and reuse. (Next time just load and run)
- Line 20-21 : This calls the solver module and optimizes the functions.

Now lets review the output,

```
# The status of the solution is printed to the screen
print("Status:", LpStatus[prob.status])
# Output=
# Status: Optimal

# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)
# Output=
# Medicine_1_units = 3.0
# Medicine_2_units = 4.0

# The optimised objective function value is printed to the screen
print("Total Health that can be restored = ", value(prob.objective))
# Output=
# Total Health that can be restored =  155.0
```

As evident, the solution was optimum and similar to what we got from graphical representation.

More details about the use of the Pulp package can be found at

`http://benalexkeen.com/linear-programming-with-python-and-pulp/`.