INFO-F101 – Programmation
Projet 4 : *Résolution de systèmes linéaires*

Année académique 2015–2016

Introduction

La puissance actuelle des ordinateurs permet de résoudre la plupart des problèmes mathématiques par une simple implémentation d'un de leur algorithme de résolution. Au séances d'exercices nous avons par exemple déjà vu comment représenter et effectuer des opérations sur des polynômes et résoudre des équations du second degré. Des solveurs comme MatLab ou R sont des programmes qui permettent de manipuler des expressions algébriques et effectuer des opérations dessus.

Nous vous demandons pour ce projet d'implémenter un programme qui sera capable de résoudre des systèmes d'équations linéaires carrés à coefficients entiers encodés par l'utilisateur.

Résolution d'un système linéaire

Les systèmes d'équations évoqués dans ce document sont des systèmes linéaires carrés de taille n (n équations et n inconnues) à coefficients entiers où les membres de droites sont égaux à 0. Un exemple de tel système est donné à la figure 1.

$$\begin{cases} 4x + 3y - 2z + 2 = 0 \\ -2x + 2y + 3z + 6 = 0 \\ 5y - z + 10 = 0 \end{cases}$$

FIGURE 1 – Système 3×3

Les systèmes d'équations doivent être résolus par des combinaisons linéaires. Cette méthode consiste à éliminer les inconnues des équations en effectuant des combinaisons linéaires de celles-ci.
Rappel : une combinaison linéaire (combili) de u et v est une expression de la forme $\alpha u + \beta v$ où $\alpha, \beta \in \mathbb{R}$.
Un exemple de combili des lignes L_1 et L_2 du système de la figure 1 visant à éliminer la variable x de L_2 est donné à la figure 2.

$$\begin{cases} 4x+3y-2z+2=0 \\ -2x+2y+3z+6=0 \\ 5y-z+10=0 \end{cases} \Leftrightarrow \begin{cases} 4x+3y-2z+2=0 \\ 7y+4z+14=0 \quad L_1+2L_2 \to L_2' \\ 5y-z+10=0 \end{cases}$$

FIGURE 2 – Combinaison linéaire

La premiere étape d'une résolution par combili consiste à transformer le système de départ en un système triangulaire par une succession de combinaisons linéaires. Un système triangulaire est un système dans lequel il y a une équation avec (au plus) une inconnue, une équation avec (au plus) deux inconnues, ..., une équation avec (au plus) n inconnues. La manière de transformer un système en sytème triangulaire est décrite dans la prochaine section.
Le système de la figure 3 est un système triangulaire équivalent à celui de la figure 1.

$$\begin{cases} 4x + 3y - 2z + 2 = 0 & \\ 7y + 4z + 14 = 0 & L_1+2L_2 \to L_2' \\ 27y + 54 = 0 & L_2'+4L_3 \to L_3' \end{cases}$$

FIGURE 3 – Système triangulaire 3×3

La deuxième étape consiste à substituer les valeurs des inconnues trouvées pour trouver la solution du système. Dans un système triangulaire, s'il y a une équation sans inconnue, le système est indéterminé ou impossible. Sinon il doit y avoir une équation avec une seule inconnue i dont il est possible de trouver la valeur. Cette valeur peut ensuite être substituée dans les autres équations du système.
Le nouveau système contient une équation qui donne la valeur de i et toutes les autres équations ne contiennent plus l'inconnue i. Dans ce système il y a à nouveau soit une équation sans inconnue soit une équation avec une seule inconnue. La même procédure peut être répétée jusqu'à ce que la valeur de toutes les inconnues soit trouvée ou que le système soit jugé comme indéterminé ou impossible.
Une exemple complet de résolution par combili du système de la figure 1 est donné dans les annexes.

Implémentation

Une équation sera modélisée par un dictionnaire. Les clés du dictionnaire seront les noms des inconnues et les valeurs seront les coefficients des inconnues. Les inconnues doivent être des caractères entre a et z. Chaque dictionnaire aura également une clé "ti" qui contiendra la valeur du terme indépendant de l'équation (rappel : le terme indépendant doit être dans le membre de

gauche !).
Un système sera modélisé par un objet de type *list* contenant des équations.
Toutes les clés des équations d'un système doivent être identiques. Si une équation ne contient pas une inconnue, son coefficient vaut 0.

Les fonctions suivantes sont à implémenter :

1. `compute_combili(equation1 ,equation2, inc_eliminer)` qui devra retourner une combili des équations `equation1` et `equation2` en éliminant l'inconnue `inc_eliminer`.

   Pour éliminer une inconnue x à partir de deux équations qui contiennent x via une combinaison linéaire :
   — Calculer le ppcm des valeurs absolues des coefficients de x dans les deux équations ;
   — $c_1 = \frac{ppcm}{\text{coef de } x \text{ dans équation 1}}$; $c_2 = \frac{ppcm}{\text{coef de } x \text{ dans équation 2}}$
   Ces coefficients vont multiplier chacune des équations pour que x ait le même coefficient dans les deux équations ;
   — Renvoyer la différence des deux équations.

   Un exemple d'exécution de cette fonction sur les deux premières équations du système de la figure 1 dans le but d'éliminer l'inconnue x est donné à la figure 4.

   A cause du problème de la représentation des nombres décimaux, il faudra éviter d'avoir des coefficients qui ont une valeur trop proche de zéro. Lors du calcul de la différence des deux équations, soit deux coefficients d'une même inconnue à soustraire c_1 et c_2. Si $\frac{|c_1-c_2|}{|c_1|+|c_2|} < 10^{-5}$, il faudra assigner 0.0 comme coefficient dans l'équation qui sera renvoyée.

```
eq1 = {'x' : 4,'y' : 3,'z' : -2,'ti' : 2}
eq2 = {'x' : -2,'y' : 2,'z' : 3,'ti' : 6}
combili = compute_combili(eq1,eq2,'x') # ppcm(4,2) = 4; c1 = 1; c2 = -2
print(combili)
> {'x' : 0,'y' : 7,'z' : 4,'ti' : 14}
```

FIGURE 4 – `compute_combili(equation1 ,equation2, inc_eliminer)`

2. `triangulation(systeme)` qui devra retourner une version triangulée du système `systeme` ou `None` si le système est indéterminé ou impossible. Soit `sysTriangulaire` un système vide qui contiendra la version triangulée de `systeme`. Pour trianguler `systeme` vous pouvez parcourir l'ensemble de ses inconnues dans une boucle et pour chaque inconnue x :
   — Chercher l'ensemble E de toutes les équations qui contiennent l'inconnue x dans `systeme`.
   — Si $|E| = 0$, le système est indéterminé ou impossible, la fonction l'imprime à l'écran. Vous ne devez pas faire la distinction entre le cas où le système est indéterminé et celui où il est impossible, vous pouvez utiliser le même message d'erreur pour les deux cas. La fonction renverra dans ce cas `None`.

— Sinon choisir une équation $e \in E$ qui servira de base et éliminer x de toutes les autres équations de E via la méthode `compute_combili`. Chaque combili trouvé devra être remplacée dans `systeme`. A la fin de cette opération `systeme` n'aura plus qu'une seule équation contenant x, l'équation de base e. L'équation e est alors ajoutée à `sysTriangulaire` et retirée de `systeme` avant de recommencer la même procédure sur une autre inconnue y et ce tant que toutes les inconnues n'ont pas été parcourues.

Une fois cette opération effectuée pour chaque inconnue, soit `systeme` est jugé comme indéterminé ou impossible, soit il a été triangulé dans `sysTriangulaire`, la dernière équation n'ayant qu'une seule inconnue.

3. `substitution(sysTriangulaire)` qui prendra en paramètre un système triangulaire `sysTriangulaire` et devra renvoyer la solution du système dans un dictionnaire qui aura comme clé le nom des inconnues et comme valeur leur solution sous forme de *float*. Appelez cette fonction sur le résultat de `triangulation(systeme)` si cette fonction a réussi à trianguler `systeme`, ceci vous évitera de devoir vérifier dans `substitution(sysTriangulaire)` qu'on reçoit bien un système triangulaire en paramètre.

   Pour trouver la valeur de chaque inconnue, effectuer de manière itérative les opérations suivantes jusqu'à ce que toutes les inconnues soient trouvées :
   — Récupérer une équation n'ayant qu'une seule inconnue x
   — Isoler cette inconnue x pour trouver sa valeur
   — Remplacer la valeur de x dans les autres équations en y supprimant x et en mettant à jour la valeur du terme indépendant.

4. `solve_system()` qui devra :
   — Appeler une fonction `encode_system()`, décrite dans la section suivante, qui renverra un système au format décrit ci-dessus.
   — Trouver une triangulaire du système encodé avec la fonction `triangulation(systeme)`.
   — Si un système triangulaire est trouvé, appeler la fonction `substitution(sysTriangulaire)` et imprimer la solution du système.
   — Renvoyer la solution retournée par `substitution(sysTriangulaire)` ou `None` si le système est indéterminé ou impossible.

Le code peut bien sûr être découpé en d'autres fonctions que celles mentionnées ci-dessus pour être plus efficace. Pensez à bien commenter votre code pour que vos fonctions soient compréhensibles.

Vos fonctions devront être implémentées dans un fichier projet4.py. Lorsque le fichier sera lancé en ligne de commande, la fonction `solve_system()` devra être appelée.

Input et output

Les consignes générales d'input et d'output sont décrit dans cette section, veillez à prévoir tous les cas de figure possibles lors de vos tests.

Input

Une fonction `encode_system()` est appelée au début de `solve_system()` pour que l'utilisateur encode les inconnues et les coefficients des équations du système.
L'utilisateur encode d'abord toutes les inconnues. Les inconnues doivent être des caractères entre *a* et *z*, séparées par un espace et encodés sur une même ligne. Il faut au minimum une inconnue et il ne peut pas y avoir deux inconnues identiques.
Une fois les inconnues encodées, le nombre d'équations *n* du système est déterminé. Pour chaque équation, l'utilisateur va encoder les coefficients de toutes les inconnues en terminant par le terme indépendant. Les coefficients doivent être des nombres entiers, séparés par un espace et encodés sur une même ligne dans le même ordre que celui dans lequel les inconnues ont été encodées. Il vous est demandé d'utiliser un mécanisme d'exception lors de la lecture et de la conversion des coefficients encodés en *int*.
Si une mauvaise valeur est encodée pour les inconnues ou les coefficients, l'utilisateur en est directement informé et doit à nouveau encoder des valeurs jusqu'à ce que celles-ci soient correctes.
A la fin de l'exécution de `encode_system()`, le système doit être affiché et est renvoyé au format décrit dans la section précédente.

Les messages que vous faites apparaître lors de l'encodage d'un système n'ont pas d'importance tant qu'ils sont clairs. Par contre, **respectez scrupuleusement** l'ordre d'encodage. Un exemple d'exécution du programme est donné dans les annexes. Vous pouvez vous inspirer des messages de cet exemple.

Output

Un système `systeme` de taille *n* doit être affiché sur *n* lignes, une ligne par équation (pas d'accolade au début du système), par une fonction `print_systeme(systeme)`.
L'ordre des termes d'une équation n'a pas d'importance, seul le terme indépendant doit se trouver à la dernière position.
Eviter que des termes ayant un coefficient égal à zéro n'apparaissent ainsi que des successions de symboles, comme $2x+(-3y)$ qui doit s'afficher $2x-3y$. Eviter également de faire apparaître des coefficients égaux à 1.
Par exemple :

A éviter !		Ok
$2x+1y+1=0$		$2x+y+1=0$
$0x+-2y+-3=0$	$\longrightarrow$	$-2y-3=0$
$0x+0y+0z+-1=0$		$-1=0$
$2x+3y+2z+0=0$		$2x+3y+2z=0$

Consignes pour la remise du projet

Les consignes pour la remise du projet sont disponibles en ligne sur la page du cours sur l'Université Virtuelle. Ces consignes sont à respecter *scrupuleusement* ; relisez-les attentivement avant la remise !
Le projet est à remettre en version électronique sur l'université virtuelle et version papier au secrétariat du département..
Pour toute question concernant l'énoncé adressez-vous à Jérôme De Boeck (email : jdeboeck@ulb.ac.be ; bureau : 2.N3.207).

Date limite de remise. Le vendredi 4 décembre 2015 à 13h.

Annexes

Résolution par combinaison linéaire

```
equation1 = {'x' : 4,'y' : 3,'z' : -2,'ti' : 2}
equation2 = {'x' : -2,'y' : 2,'z' : 3,'ti' : 6}
equation3 = {'x' : 0,'y' : 5,'z' : -1,'ti' : 10}
system = [equation1,equation2,equation3]
```

$$\begin{cases} 4x + 3y - 2z + 2 = 0 & (L_1) \\ -2x + 2y + 3z + 6 = 0 & (L_2) \\ 5y - z + 10 = 0 & (L_3) \end{cases}$$

$$\Updownarrow$$

Combinaison de L_1 et L_2 pour éliminer x de L_2

$$\begin{cases} 4x + 3y - 2z + 2 = 0 & \\ 7y + 4z + 14 = 0 & (L_1 + 2L_2 \to L2) \\ 5y - z + 10 = 0 & \end{cases}$$

$$\Updownarrow$$

Combinaison de L_2 et L_3 pour éliminer z de L_3

$$\begin{cases} 4x + 3y - 2z + 2 = 0 & \\ 7y + 4z + 14 = 0 & \\ 27y + 54 = 0 & (L_2 + 4L_3 \to L_3) \end{cases}$$

$$\Updownarrow$$

Equation avec une seule inconnue trouvée, y isolé et valeur remplacée dans L_1 et L_2

$$\begin{cases} 4x - 2z - 4 = 0 \\ 4z = 0 \\ y = -2 \end{cases}$$

$$\Updownarrow$$

z isolé et valeur remplacée dans L_1

$$\begin{cases} 4x & - & & & 4 & = & 0 \\ & & z & & & = & 0 \\ & y & & & & = & -2 \end{cases}$$

$$\Updownarrow$$

x isolé

$$\begin{cases} x & & & = & 1 \\ & & z & = & 0 \\ & y & & = & -2 \end{cases}$$

Valeur de retour de solve_system : { "x" : 1,"y" : -2,"z" : 0}

Exemple d'exécution

```
Encoder les inconnues (lettres entre a et z séparées par un espace)
x y x
Erreur : Toutes les inconnues doivent être des caractères entre 'a'
et 'z' et n'apparaître qu'une fois
Encoder les inconnues (lettres entre a et z séparées par un espace)
x y
Equation 1 :
Coefficients de x y et du terme indépendant :
1 a 2
Erreur : Tous les coefficients doivent être des nombres entiers
Coefficients de x y et du terme indépendant :
1 2 -1
Equation encodée :
2y + x - 1 = 0
Equation 2 :
Coefficients de x y et du terme indépendant :
-1 2 0
Equation encodée :
2y - x = 0
Sytème
2y + x - 1 = 0
2y - x = 0
Solution :
x = 0.5
y = 0.25
```