Système linéaire

On veut résoudre le système linéaire (a|b) avec l'algorithme de Gauss-Jordan.

I) Mise en œuvre

1) Rappel de l'algorithme de Gauss-Jordan

On utilise les fonctions écrites dans le TP sur les matrices: affiche_matrice(a), permutation(a, i, j), dilatation(a, i, x) et transvection(a, i, j, x)

On résout le le système linéaire (a|b) avec a matrice à n lignes et p colonnes et b une liste de longueur n . Pour faciliter son implémentation, on indexe à partir de 0 et on note a[i][j] et a[i] les termes ou les lignes de a, comme en informatique.

```
ligne = 0
# Echelonnement et réduction de a
pour j variant entre 0 et p - 1:
   chercher s'il existe le plus petit entier k ∈ ⟦ligne, n - 1⟧ tel que a[k][j] ≠ 0
   si on trouve cet entier k
      si ligne ≠ k alors:
         permutation(a, k, ligne)
         échanger b[k] et b[ligne]
      x = a[ligne][j]
      dilatation(a, ligne, 1/x)                    # On normalise le pivot à 1
      b[ligne] = 1/x * b[ligne]
      pour i variant de 0 à ligne - 1 et de ligne + 1 à n - 1:
         x = -a[i][j]
         transvection(a, i, ligne, x)
         b[i] = b[i] + x * b[ligne]
      ligne = ligne + 1
# Vérification de la compatibilité du système (a|b), on a rang = ligne
pour i variant de ligne à n - 1:
   si b[i] ≠ 0:
      afficher "Le système n'a pas de solutions"
      terminer
# Le système est compatible; on affiche les solutions
pour i variant de 0 à ligne - 1:
   trouver le plus petit entier j tel que a[i][j] ≠ 0 # On aura a[i][j] = 1
# x[j] est une inconnue principale du système
                                  p-1
   afficher " x[j] = b[i] - ∑_{k=j+1}  a[i][k] x[k]"
# x[j] est exprimée en fonction des éventuelles inconnues secondaires x[k]
# Dans la somme ∑ on affiche que les a[i][k] x[k] avec a[i][k] ≠ 0
```

2) Zoom sur les points délicats

a) La recherche de l'entier k

Q1) Ecrire une fonction cherche_colonne(a, i, j) qui retourne s'il existe le plus petit entier k ∈ ⟦i, n-1⟧ tel que a[k][j]≠ 0. Si cet entier k n'existe pas, cherche_colonne(a, i, j) retourne -1. Tester votre fonction avec diverses matrices.

b) La recherche de l'entier j

Q2) Ecrire une fonction cherche_ligne(a, i) qui retourne s'il existe le plus petit entier j ∈ ⟦0, p-1⟧ tel que a[i][j]≠ 0. Si cet entier j n'existe pas, cherche_ligne(a, i) retourne -1. Tester votre fonction avec diverses matrices.

c) Eviter les erreurs d'élimination # Facultatif dans un premier temps

Comme dans le TP d'inversion de matrice, il est conseillé d'utiliser la fonction addition(x, y) proposée dans l'opération de

transvection, ainsi que dans l'instruction "b[i] = b[i] + x * b[ligne]"

d) L'affichage des solutions

La chaîne de caractères res = " x[j] = b[i] - $\sum_{k=j+1}^{p-1}$ a[i][k] x[k]" doit être construite par concaténations successives. Il est conseillé d'utiliser la méthode format() pour insérer lisiblement et aux bons endroits les valeurs de j, b[i], a[i][k], k. Par exemple (on peut améliorer la gestion des signes):

```
res = "x[{0}] = {1} ".format(j, b[i])
for k in range(j + 1, p):
    if a[i][k] != 0:
        res = res + "- {0}*x[{1}] ".format(a[i][k], k)
```

3) Ecriture de la fonction de résolution du système linéaire (a|b)

Q3) Ecrire une fonction resoudre_systeme(a, b) résolvant le système linéaire (a|b) en suivant à la lettre l'algorithme du 1).
On pourra arrondir les valeurs des coefficients de a et b à 10^{-8} près avecla fonction round() après l'échelonnement et la réduction pour plus de lisibilité.

CONSEIL: Décomposer la fonction resoudre_systeme(a, b) en trois fonctions indépendantes pour pouvoir tester correctement et au fur et à mesure vos algorithmes sur des systèmes très simples.

- Une première fonction echelonnement(a, b) qui échelonne et réduit le système (a|b).
- Une seconde fonction vérification(a, b) qui teste si le système échelonné réduit (a|b) est compatible.
- Une troisième fonction affichage(a, b) qui affiche les solutions du système compatible échelonné réduit (a|b).

Vérifier par exemple:

```
a, b = [[1, 2], [3, 4]], [5, 6]
resoudre_systeme(a, b) ⟶ x[0] = -4.0  ; x[1] = 4.5
```

```
a, b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]], [1, 2, 2]
resoudre_systeme(a, b) ⟶ Le système n'a pas de solutions
```

```
a, b = [[1, 2, 3], [4, 5, 6], [7, 8, 9]], [1, 2, 3]
resoudre_systeme(a, b) ⟶
x[0] = -0.33333333 + 1.0*x[2] ; x[1] = 0.66666667 - 2.0*x[2]
```

```
a, b = [[1 / (i + j + 1) for i in range(4)] for j in range(3)], [1] * 4
resoudre_systeme(a, b) ⟶
x[0] = 3.0 - 0.05*x[3] ; x[1] = -24.0 + 0.6*x[3] ; x[2] = 30.0 - 1.5*x[3]
```

```
a, b = [[(i + j)**2 for i in range(5)] for j in range(5)], [1] * 5
resoudre_systeme(a, b) ⟶ x[0] = 0.5 - 1.0*x[3] - 3.0*x[4] ;
x[1] = -1.0 + 3.0*x[3] + 8.0*x[4] ; x[2] = 0.5 - 3.0*x[3] - 6.0*x[4]
```

II) Pour aller plus loin: du calcul exact sur les fractions

Voir maths-algo.fr

Q4) En adaptant légèrement la fonction resoudre_systeme(a, b), écrire une fonction resoudre_systeme_fractions(a, b) résolvant le système linéaire (alb) de fractions.

On adaptera les fonctions de conversion et d'affichage de fractions données dans le TP d'inversion d'une matrice. Par exemple:

```
a, b = [["1/" + str(i**2 + j + 1) for i in range(6)] for j in range(4)], [1] * 6
resoudre_systeme_fractions(a, b) ⟶
Système linéaire à résoudre

1         1/2       1/5       1/10      1/17      1/26      |   1
1/2       1/3       1/6       1/11      1/18      1/27      |   1
1/3       1/4       1/7       1/12      1/19      1/28      |   1
1/4       1/5       1/8       1/13      1/20      1/29      |   1

Solution
x[0] = -2/3 + 7/969*x[4] + 32/3393*x[5]
x[1] = 5 - 56/969*x[4] - 250/3393*x[5]
x[2] = -28 + 392/969*x[4] + 1600/3393*x[5]
x[3] = 143/3 - 1144/969*x[4] - 275/261*x[5]
```