Inversion d'une matrice

On veut calculer, si c'est possible, l'inverse d'une matrice carrée a avec l'algorithme de Gauss-Jordan.

I) Mise en œuvre "sans se poser de questions"

1) Rappel de l'algorithme de Gauss-Jordan

On inverse si possible une matrice a carrée nxn . Pour faciliter l'écriture de l'algorithme 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.

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

```
b = identite(n)
pour j variant entre 0 et n-1:
   chercher s'il existe le plus petit entier k ∈ ⟦j, n-1⟧ tel que a[k][j] ≠ 0
   si on trouve cet entier k
      si k ≠ j alors:
         permutation(a, k, j)
         permutation(b, k, j)
      x = a[j][j]
      dilatation(a, j, 1/x)                    # On normalise le pivot à 1
      dilatation(b, j, 1/x)
      pour i variant de 0 à j-1 et de j+1 à n-1:   # réduction et échelonnement de a
         x = -a[i][j]
         transvection(a, i, j, x)
         transvection(b, i, j, x)
   sinon la matrice a n'est pas inversible: c'est terminé
b est la matrice inverse de a
```

2) La recherche de l'entier k

Q1) Ecrire une fonction cherche_V(a, j) (avec V comme vertical) qui retourne s'il existe le plus petit entier k ∈ ⟦j, n-1⟧ tel que a[k][j]≠ 0. Si cet entier k n'existe pas, cherche_V(a, j) retourne -1.

Tester votre fonction avec diverses matrices.

3) Une première version du calcul de l'inverse

Q2) En suivant "à la lettre" l'algorithme décrit au 1), écrire une fonction inverse_matrice(a) retournant si elle existe l'inverse d'une matrice carrée a. Justifier que la complexité de cette fonction est $O(n^3)$.

4) Quelques tests et commentaires

Q3) Inverser les matrices a=[[1,2], [3,4]] ; b=[[3,4], [1,2]] ; c=[[1,2,3], [4,5,6], [7,8,9]] , d=[[9,8,7], [6,5,4], [3,2,2]], e=[[3,5],[1,5/3]].

Q4) Faire d'autres essais (vérifier les résultats avec votre calculatrice ou encore vérifier que $m \times m^{-1} = I$ après avoir défini la fonction produit(a, b) de deux matrices).

Les calculs d'inverses semblent se dérouler plutôt bien (sauf pour e qui n'est pas inversible et qui est pourtant inversée), avec quelques désagréments: des erreurs d'arrondis qui rendent le résultat peu lisible (pour b et d par exemple).

Q5) Inverser la matrice m = [[5, 4, 3, 2, 1], [4, 3, 2, 1, 2], [3, 2, 1, 2, 3], [2, 1, 2, 3, 4], [1, 2, 3, 4, 5]] et constater qu'il y a de grosses erreurs dans $p = m^{-1}$. La bonne matrice p est

p = [[0.08333333, 0, 0.0, 0.5, -0.41666667], [0, 0, 0.5, -1.0, 0.5], [0.0, 0.5, -1.0, 0.5, 0.0], [0.5, -1.0, 0.5, 0.0, 0.0], [-0.41666667, 0.5, 0.0, 0.0, 0.08333333]]

II) Tentative d'amélioration

Le calcul flottant produit immanquablement des erreurs d'arrondis. La plupart sont bénignes (trouver 1.2999999999999 au lieu de 1.3) mais les erreurs d'éliminations peuvent être catastrophiques ici.

> On rappelle qu'une erreur d'élimination se produit lorsqu'on soustrait deux flottants a et b très proches l'un de l'autre. Au pire, le résultat $c = a - b$ peut être non nul alors qu'il devrait être nul. Par exemple, 5/3 - 1/3 * 5 $= 2^{-52} \approx 2.2 * 10^{-16} \neq 0$.

Ce sont des erreurs d'éliminations qui dans le cas des matrices e et m faussent tout à fait le résultat. Par exemple l'échelonnement de la matrice e n'est pas fait correctement: $\begin{pmatrix} 3 & 5 \\ 1 & 5/3 \end{pmatrix} \sim \begin{pmatrix} 3 & 5 \\ 0 & 2.2 * 10^{-16} \end{pmatrix}$ au lieu de $\begin{pmatrix} 3 & 5 \\ 0 & 0 \end{pmatrix}$: la matrice e semble inversible alors qu'elle ne l'est pas.

Dans l'algorithme de calcul de l'inverse, le seul endroit où peut se produire une erreur d'élimination est la transvection, dans l'instruction `a[i][k] = a[i][k] + x * a[j][k]`.

On réécrit une fonction addition(x, y) calculant la somme x+y et évitant (pas dans tous les cas) l'erreur décrite ci-dessus.

```
def addition(x, y):
    """ Pour éliminer certaines erreurs d'élimination dans le calcul de x+y"""
    m = max(abs(x), abs(y))
    if abs(x + y) < 2**(-40) * m:
        return 0
    return x + y
```

Pourquoi 2^{-40} et pas 2^{-51} ou 2^{-52} comme on aurait pu l'imaginer (voir cours "Représentation des nombres flottants") ? Parce que les erreurs d'éliminations peuvent se produire, dans l'échelonnement de la matrice, après un certain nombre d'erreurs (inévitables) d'arrondi. On prend donc de la marge.

Q6) Modifier la fonction transvection(a, i, j, x) en utilisant cette fonction addition() donnée ici et constater que l'inversion des matrices e et m se déroule maintenant correctement. Faire éventuellement des essais sur d'autres matrices.

Attaquons nous maintenant au désagréments bénins décrits au début de se paragraphe. La fonction round(x, n) arrondit le flottant x à 10^{-n} près. En prenant n=8, on obtient un compromis acceptable entre ce que l'on veut (transformer 0.499999999999999 en 0.5 par exemple) sans trop risquer de perdre des petits nombres ($2 * 10^{-9}$ serait arrondi à 0) dont la présence est peu probable si la matrice que l'on cherche à inverser est "standard".

Q7) Ecrire une fonction arrondir(a) (sans retour) arrondissant tous les termes de la matrice a à 10^{-8} près et intégrer cette fonction juste avant le retour dans la fonction inverse_matrice(). Tester entre autres sur la matrice m : c'est opérationnel

On ne peut guère faire mieux, mais il ne faut pas se faire d'illusions. En cherchant un peu, on pourrait trouver des matrices (un peu tordues) échappant au premier "patch". Aucune solution ne peut être parfaite si l'on s'en tient à du calcul flottant. Pour faire mieux, il faut faire du calcul exact: c'est l'objet du paragraphe suivant.

Pour ceux qui veulent aller plus loin, la suite est sur maths-algo.fr. On introduit le module fractions de Python qui permet de faire du calcul exact sur les fractions et donc d'inverser exactement une matrice à coefficients rationnels.

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

1) Définir et calculer avec des fractions

```
import fractions as F          # On importe une fois pour toutes le module fractions
f1 = F.Fraction("-7/13")       # On peut définir une fraction comme cela
f2 = F.Fraction(10,6)          # Ou comme ceci
f3 = F.Fraction(1/3)        # Ou encore ainsi, mais c'est une mauvaise idée
print(f1, f2, float(f2))    # float(f2) reconvertit en flottant
print(f3)
print (f1 + f2, f1*f2, f1 - 4*f2, f1/f2)  # On calcule de manière habituelle
# renvoie
-7/13 5/3 1.6666666666666667           # Les fractions sont réduites automatiquement
6004799503160661/18014398509481984     # Tiens donc, que s'est-il passé pour f3 ?
44/39 -35/39 -281/39 -21/65            # Tout va bien...
```

2) Saisie, conversion et affichage d'une matrice a de fractions

Une matrice a de fractions n/d est donnée sous la forme de listes de chaînes "n/d" (On peut mixer avec des entiers). Il faudra les convertir en listes de fractions. Pour voir les matrices, il faut faire l'opération inverse. On donne les fonctions suivantes:

```
def conversion_matrice_fractions(a):
    """ conversion d'une matrice a de fractions "n/d"
      vers une matrice de fractions n/d"""
    return ([[F.Fraction(e) for e in ligne] for ligne in a])
```

```
def affiche_matrice_fractions(a):
    """ Affichage d'une matrice de fractions a"""
    print()
    for i in range(len(a)):
        s = ""
        for e in a[i]:
            s = s + "{0:10}".format(str(e))
        print(s)
    print()
```

Pour comprendre le fonctionnement de ces fonctions, essayer par exemple:

```
a = [["1", "2/3", "3"], ["4/7", "5", "6/5"], ["7/2", "8", "9"]]
b = [[1, "2/5", 3], [4, "5/3", 6], [7, 8, 9]]
a1 = conversion_matrice_fractions(a)
b1 = conversion_matrice_fractions(b)
print(a1)
print(b1)
affiche_matrice_fractions(a1)
affiche_matrice_fractions(b1)
```

3) Inverse d'une matrice de fractions

L'algorithme décrit au I) peut être repris intégralement, sans modifier les fonctions annexes (on peut revenir à la version initiale de la fonction transvection, mais ce n'est même pas indispensable). Il suffit de convertir la matrice a entrée en une matrice de fractions et d'afficher la matrice inverse calculée avec la fonction donnée ci dessus.

Q8) Ecrire une fonction inverse_matrice_fractions(a) calculant l'inverse de la matrice de fractions a. Vérifier par exemple qu'avec la matrice *m*, l'inverse est:

```
1/12      0         0         1/2       -5/12
0         0         1/2       -1        1/2
0         1/2       -1        1/2       0
1/2       -1        1/2       0         0
-5/12     1/2       0         0         1/12
```

Et que:

```
n=4
a = [["1/" + str(i + j + 1) for i in range(n)] for j in range(n)]
inverse_matrice_fractions(a)
1         1/2       1/3       1/4
1/2       1/3       1/4       1/5
1/3       1/4       1/5       1/6
1/4       1/5       1/6       1/7

16        -120      240       -140
-120      1200      -2700     1680
240       -2700     6480      -4200
-140      1680      -4200     2800
```