Determinant

April 2, 2023

1 Déterminant, cofacteurs, etc.

Marc Lorenzi

2 avril 2023

[1]:
```
import random
import time
compteur = 0
```

Il est bien connu que l'algorithme de calcul d'un déterminant par un développement selon une ligne ou une colonne est hautement inefficace. Cela n'empêche, c'est un excellent exercice de programmation. Et puis sait-on jamais ?

1.1 1. La formule de Lagrange

1.1.1 1.1 Introduction

Soit $A \in \mathcal{M}_n(\mathbb{K})$. Pour tous $i, j \in [|0, n-1|]$, notons $A[i|j]$ la matrice obtenue en supprimant la ligne i et la colonne j de A. Remarquons que $A[i|j] \in \mathcal{M}_{n-1}(\mathbb{K})$.

- Le mineur de A ligne i, colonne j est

$$\mathrm{Min}(A, i, j) = \det A[i|j]$$

- Le cofacteur de A ligne i, colonne j est

$$\mathrm{Cof}(A, i, j) = (-1)^{i+j}\mathrm{Min}(A, i, j)$$

La *formule de Lagrange* nous dit que pour tout $i \in [|0, n-1|]$,

$$\det A = \sum_{j=0}^{n-1} A_{ij}\mathrm{Cof}(A, i, j)$$

On a aussi pour tout $j \in [|0, n-1|]$,

$$\det A = \sum_{i=0}^{n-1} A_{ij} \mathrm{Cof}(A, i, j)$$

Ces égalités s'appellent aussi les formules de *développement* de $\det A$ par rapport à la ligne i ou la colonne j.

1.1.2 1.2 Afficher une matrice

La fonction `print_mat` affiche de façon agréable une matrice.

[2]:
```
def print_mat(A):
    p = len(A)
    q = len(A[0])
    s = q * '+-----' + '+'
    for i in range(p):
        print(s)
        for j in range(q):
            print('|%4d ' % A[i][j], end='')
        print('|')
    print(s)
```

[3]:
```
print_mat([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
```

```
+-----+-----+-----+
|   1 |   2 |   3 |
+-----+-----+-----+
|   4 |   5 |   6 |
+-----+-----+-----+
|   7 |   8 |   9 |
+-----+-----+-----+
```

1.1.3 1.3 Mineurs, cofacteurs, comatrice

La fonction `mineur` prend en paramètres une matrice carrée A et deux entiers i et j. Elle renvoie $\mathrm{Min}(A, i, j)$.

[4]:
```
def mineur(A, i, j):
    A1 = supprimer_ligne_colonne(A, i, j)
    return determinant(A1)
```

Le cofacteur de A ligne i, colonne j est égal à $(-1)^{i+j}$ fois le mineur correspondant.

[5]:
```
def cofacteur(A, i, j):
    m = mineur(A, i, j)
    if (i + j) % 2 == 0: return m
    else: return -m
```

Tant que nous y sommes, la comatrice de A est la matrice de ses cofacteurs.

[6]:
```
def comatrice(A):
    n = len(A)
    return [[cofacteur(A, i, j) for i in range(n)] for j in range(n)]
```

Enfin, la transcomatrice de A, est la transposée de sa comatrice.

[7]:
```
def transposee(A):
    m = len(A)
    n = len(A[0])
    return [[A[j][i] for j in range(m)] for i in range(n)]
```

[8]:
```
print_mat(transposee([[1] , [2], [3]]))
```

```
+-----+-----+-----+
|   1 |   2 |   3 |
+-----+-----+-----+
```

[9]:
```
print_mat(transposee([[1, 2, 3] , [4, 5, 6], [7, 8, 9]]))
```

```
+-----+-----+-----+
|   1 |   4 |   7 |
+-----+-----+-----+
|   2 |   5 |   8 |
+-----+-----+-----+
|   3 |   6 |   9 |
+-----+-----+-----+
```

[10]:
```
def tilde(A):
    return transposee(comatrice(A))
```

Mais ça veut dire quoi, tilde ? Un peu de culture !

___Étymol. et Hist. 1834 orth. esp. (Boiste). Mot esp. att. dep. 1433 (E. de Villena ds Cor.-Pasc.), issu du lat. titulus (titre*) qui signifiait propr. « ce qui désigne, signale ». Cf. a. fr. title, fr. mod. ti(l)tre « signe abréviatif » (v. titre, étymol. 1 b; FEW t. 13, 1, p. 360a).___

Ah, maintenant on comprend mieux.

1.1.4 1.4 Déterminant

Le déterminant de A peut être calculé par un développement par rapport à la première ligne, le cas d'un déterminant vide étant évident. Remarquons que nous allons écrire une fonction `determinant` qui se prépare à appeler `cofacteur` qui appelle `mineur` qui appelle … `determinant`. Nous avons là un exemple de fonctions mutuellement récursives. Mais comme ces fonctions s'appellent sur des matrices de tailles strictement décroissantes et que l'on sait traiter le cas de la matrice « vide », nous sommes assurés de la terminaison de l'algorithme.

La variable `compteur` qui apparaît dans le code ci-dessous est une variable globale, qui compte le nombre d'additions et de multiplications effectuées par l'algorithme. C'est très mal de faire cela mais c'est pédagogique. Nous aurons ainsi une idée de la complexité du calcul d'un déterminant par cette mauvaise (?) méthode.

[11]:
```
def determinant(A):
    global compteur
    n = len(A)
    if n == 0: return 1
    else:
        s = 0
        for j in range(n):
            s = s + A[0][j] * cofacteur(A, 0, j)
            compteur = compteur + 2 # 1 addition, 1 multiplication
    return s
```

Il reste à écrire la fonction supprimant une ligne et une colonne de la matrice A. Une petite fonction auxiliaire `phi` nous évite d'avoir à considérer 4 cas dans la fonction principale.

[12]:
```
def phi(p, i):
    if p < i: return p
    else: return p + 1
```

[13]:
```
def supprimer_ligne_colonne(A, i, j):
    n = len(A)
    rg = range(n - 1)
    return [[A[phi(p, i)][phi(q, j)] for p in rg] for q in rg]
```

Testons.

[14]:
```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print_mat(A)
```

```
+-----+-----+-----+
|   1 |   2 |   3 |
+-----+-----+-----+
|   4 |   5 |   6 |
+-----+-----+-----+
|   7 |   8 |   9 |
+-----+-----+-----+
```

[15]:
```
B = supprimer_ligne_colonne(A, 1, 0)
print_mat(B)
```

```
+-----+-----+
|   2 |   8 |
+-----+-----+
|   3 |   9 |
+-----+-----+
```

[16]:
```
print(determinant(A))
```

```
0
```

[17]:
```
print_mat(tilde(A))
```

```
+-----+-----+-----+
|  -3 |   6 |  -3 |
+-----+-----+-----+
|   6 | -12 |   6 |
+-----+-----+-----+
|  -3 |   6 |  -3 |
+-----+-----+-----+
```

1.1.5 1.4 Matrices « aléatoires »

Pour des tests un peu plus conséquents, définissons une fonction prenant en paramètre un entier n et renvoyant une matrice $n \times n$ dont les coefficients sont des entiers aléatoires entre -9 et 9.

[18]:
```
def random_matrix(n):
    rg = range(n)
    return [[random.randint(-9, 9) for i in rg] for j in rg]
```

Pour une matrice 3×3, le calcul du déterminant demande 30 opérations.

[19]:
```
A = random_matrix(3)
print_mat(A)
compteur = 0
print('Déterminant : ', determinant(A))
print("Nombre d'opérations : ", compteur)
```

```
+-----+-----+-----+
|   1 |   7 |   7 |
+-----+-----+-----+
|   0 |  -2 |  -5 |
+-----+-----+-----+
|  -8 |   7 |   9 |
+-----+-----+-----+
Déterminant :  185
Nombre d'opérations :  30
```

Et pour une matrice 8×8 ?

[20]:
```
A = random_matrix(8)
print_mat(A)
compteur = 0
print('Déterminant : ', determinant(A))
print("Nombre d'opérations : ", compteur)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|   6 |  -3 |  -6 |  -6 |   5 |   4 |  -3 |   0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   8 |   1 |  -9 |   5 |   1 |   0 |   5 |   9 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -8 |   2 |  -6 |  -9 |  -3 |   7 |  -7 |   3 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -6 |   3 |  -3 |  -2 |   4 |   7 |  -1 |  -8 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   3 |  -5 |   7 |   4 |   9 |   9 |   0 |  -2 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   1 |   8 |  -2 |   7 |  -2 |  -4 |   8 |   5 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   9 |   6 |  -2 |  -4 |   1 |   0 |  -3 |  -5 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   5 |   8 |   2 |  -2 |   2 |   8 |   8 |  -1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
Déterminant :  106461371
Nombre d'opérations :  219200
```

Il nous faut 219200 opérations. C'est beaucoup.

1.2 2. Factorielle

1.2.1 2.1 Complexité du calcul du déterminant

Faisons un petit calcul. Calculer un déterminant 0×0 est immédiat et demande 0 opération. Pour calculer un déterminant de taille n, il faut calculer n déterminants de taille $n-1$, les multiplier par des nombres, puis additionner. Appelons C_n le nombre d'opérations nécessaires au calcul d'un détermant de taille n. Nous avons $C_0 = 0$ et, pour tout entier $n > 0$, $C_n = nC_{n-1} + 2n$.

[21]:
```
def C(n):
    if n == 0: return 0
    else: return n * C(n - 1) + 2 * n
```

[22]:
```
for k in range(10):
    print(k, C(k))
```

```
0 0
1 2
2 8
3 30
4 128
5 650
6 3912
7 27398
8 219200
9 1972818
```

On est tout bons, $C_8 = 219200$. Et C_n, il vaut combien ? Le lecteur consciencieux montrera par récurrence sur n que

$$C_n = 2n! \sum_{k=0}^{n-1} \frac{1}{k!}$$

Juste pour le plaisir d'écrire du Python ….

[23]:
```
def fact(n):
    p = 1
    for k in range(1, n + 1): p *= k
    return p
```

[24]:
```
[fact(n) for n in range(11)]
```

[24]:
```
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
```

[25]:
```
def complexite(n):
    p = 2 * fact(n)
    s = 0
    for k in range(n ):
        s = s + 1.0 / fact(k)
    return p * s
```

[26]:
```
complexite(8)
```

[26]:
```
219200.0
```

Il est bien connu que

$$\sum_{k=0}^{n-1} \frac{1}{k!} \underset{n\to\infty}{\longrightarrow} e$$

Ainsi, un équivalent de la complexité du calcul d'un déterminant de taille n est

$$C_n \sim 2en!$$

Cela signifie que le temps de calcul d'un temps déterminant est $T_n \sim 2Ken!$ où la constante K dépend évidemment de l'ordinateur utilisé. Que vaut K pour ma machine ?

1.2.2 2.2 Tests

[27]:
```
A = random_matrix(8)
print_mat(A)
compteur = 0
t1 = time.time()
```

```
d = determinant(A)
t2 = time.time()
t_mat8 = t2 - t1
print(d)
print(compteur)
print('%5.3fs' % t_mat8)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -1 |  -6 |   3 |   5 |  -6 |   2 |   1 |  -8 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   2 |  -7 |  -9 |  -4 |  -5 |  -6 |  -8 |   6 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -4 |  -2 |   6 |  -4 |  -8 |   0 |  -3 |   9 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   6 |  -8 |  -7 |  -7 |  -5 |   6 |  -4 |  -7 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   2 |   7 |   6 |  -9 |  -7 |   4 |   6 |   1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -5 |   5 |  -6 |  -9 |  -2 |  -1 |   7 |   0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  -4 |  -1 |  -5 |  -9 |  -3 |  -5 |   0 |  -7 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   3 |   7 |   4 |  -4 |   5 |   9 |   7 |  -4 |
+-----+-----+-----+-----+-----+-----+-----+-----+
-28463421
219200
0.440s
```

Python me dit que pour calculer un déterminant de taille 8 il lui faut environ 0.5 seconde. Ainsi, $2ke8! \simeq 0.5$, d'où la valeur de K pour ma machine :

[28]:
```
K = t_mat8 / (2 * fact(8) * 2.71828)
print(K)
```

```
2.008853072543202e-06
```

[29]:
```
C = 2 * K * 2.718281828
```

Pour calculer un déterminant de taille n, ma machine met environ $1.24 \times 10^{-5} n!$ secondes. Voici donc le temps prévisionnel pour un déterminant de taille 10.

[30]:
```
C * fact(10)
```

[30]:
```
39.63105959496394
```

On essaye ?

[31]:
```
A = random_matrix(10)
t1 = time.time()
d = determinant(A)
t2 = time.time()
print(d)
print('%5.3fs' % (t2 - t1))
```

```
30156438310
36.084s
```

Et pour un déterminant de taille 17 ? Rappelons qu'il y a 86400 secondes dans une journée et 365 jours par an.

[32]:
```
1.08e-5 * fact(17) / 86400 / 365
```

[32]:
```
121.81076304657533
```

121 ans. Eh bien non, on ne teste pas :-). Alors tout cela ne sert à rien ? Pas si sûr.

1.3 3. Inutile ?

1.3.1 3.1 Calcul formel

Soient $a, b, c \in \mathbb{C}$. La matrice

$$A = \begin{pmatrix} 0 & a & b & c \\ a & 0 & c & b \\ b & c & 0 & a \\ c & b & a & 0 \end{pmatrix}$$

est-elle inversible ?

[33]:
```
from sympy import *
a, b, c = symbols('a b c')
init_printing()
```

[34]:
```
A = [
    [0, a, b, c],
    [a, 0, c, b],
    [b, c, 0, a],
    [c, b, a, 0]
]
Matrix(A)
```

[34]:
$$\begin{bmatrix} 0 & a & b & c \\ a & 0 & c & b \\ b & c & 0 & a \\ c & b & a & 0 \end{bmatrix}$$

[35]:
```
e = determinant(A)
e
```

[35]: $a\left(a^3 - ab^2 - ac^2\right) + b\left(-a^2b + b^3 - bc^2\right) + c\left(-a^2c - b^2c + c^3\right)$

[36]:
```
factor(e)
```

[36]: $(a - b - c)\,(a - b + c)\,(a + b - c)\,(a + b + c)$

La matrice est inversible si et seulement si $a \pm b \pm c \neq 0$.

Tentons l'exo 412 du TD 820. Et ayons une pensée pour ceux qui n'ont toujours pas installé Jupyter.

[37]:
```
alpha, beta, gamma = symbols('alpha beta gamma')
```

[38]:
```
A = [
    [alpha - beta - gamma, 2 * alpha, 2 * alpha],
    [2 * beta, beta - alpha - gamma, 2 * beta],
    [2 * gamma, 2 * gamma, gamma - alpha - beta]
]
Matrix(A)
```

[38]: $\begin{bmatrix} \alpha - \beta - \gamma & 2\alpha & 2\alpha \\ 2\beta & -\alpha + \beta - \gamma & 2\beta \\ 2\gamma & 2\gamma & -\alpha - \beta + \gamma \end{bmatrix}$

[39]:
```
D = determinant(A)
D
```

[39]: $2\alpha\left(4\beta\gamma - 2\beta\left(-\alpha - \beta + \gamma\right)\right) + 2\alpha\left(4\beta\gamma + 2\gamma\left(\alpha - \beta + \gamma\right)\right) + \left(-4\beta\gamma + \left(-\alpha - \beta + \gamma\right)\left(-\alpha + \beta - \gamma\right)\right)\left(\alpha - \beta - \gamma\right)$

[40]:
```
factor(D)
```

[40]: $(\alpha + \beta + \gamma)^3$

1.3.2 3.2 Conclusion

Tout n'est pas sombre en Terre du Milieu. Pour des matrices de taille raisonnable dont certains coefficients contiennent des paramètres, nous pouvons déterminer si oui ou non ces matrices sont inversibles, grâce à notre algorithme de calcul de déterminant.