TP9_2024_sujet

March 15, 2024

1 TP9 – Retour sur la logique – Matrices avec numpy ou sympy

1.1 Partie Logique

Les valeurs Booléennes qui représentent le vrai et le faux sont True et False. Les opérations logiques utilisées sont :
- AND & : a & b - OR | : a | b - NOT ~ : ~a - => : a >> b

1) Représentez la formule: $F = ((p \rightarrow q) \vee (r \wedge \neg p)) \wedge \neg(q \wedge r)$

où p,q,r sont des variables propositionnelles donc des symboles

[1]:
```
import sympy as sy
# à compléter
```

```
¬(q  r)  ((r  ¬p)  (p → q))
```

2) Que vaut F quand p est vrai, q aussi et r faux? Et quand p est vrai, q et r sont faux?

[2]:
```
# à compléter
```

```
pour  vrai vrai faux  True
pour  vrai faux faux  False
```

3) On peut aussi mélanger valuation et variables: Que sera le résultat de f si on substitue FAUX à la variable q ?

[3]:
```
# à compléter
```

[3]:
$(r \wedge \neg p) \vee \neg p$

4) Faire la table de vérité de F (on écrira une boucle)

[4]:
```
# à compléter
```

```
pour   (False, False, False) on obtient  True
pour   (False, False, True) on obtient  True
pour   (False, True, False) on obtient  True
pour   (False, True, True) on obtient  False
pour   (True, False, False) on obtient  False
pour   (True, False, True) on obtient  False
```

```
pour   (True, True, False) on obtient  True
pour   (True, True, True) on obtient  False
```

5. Mise sous forme normale: toute expression peut être mise sous forme normale

- d'une conjonction de disjonctions (FN conjonctive)
- d'une disjonction de conjonctions (FN disjonctive)

Pour trouver la FN conjonctive (resp. disjonctive) de F, on fait appel à la fonction to_cnf (resp. to_dnf) de sympy

Donner les FNC et FND de F

[5]:
```
# à compléter
```

```
Forme conjonctive : (q | ~p) & (~q | ~r) & (q | r | ~p)
Forme disjonctive : (q & ~q) | (q & ~r) | (~p & ~q) | (~p & ~r) | (r & ~p & ~q)
| (r & ~p & ~r)
```

6) Simplifier F

[6]:
```
# à compléter
```

```
Forme simplifiée : (q & ~r) | (~p & ~q)
```

7. On considère la formule F donnée par sa table de vérité dont la dernière colonne est dans l'ortdre usuel

10101100

Donner deux formules qui ont cette table de vérité (une conjonctive et une disjonctive)

[7]:
```
# à compléter
```

$(r \wedge \neg q) \vee (\neg p \wedge \neg r)$

$(r \vee \neg p) \wedge (\neg q \vee \neg r)$

1.2 Calcul matriciel

Ce TP présente l'objet spécifique matrix, défini dans NumPy et SymPy pour supporter les matrices et les opérations associées. On y voit également en quoi ces modules sont adaptés à l'algèbre linéaire, en facilitant la résolution de systèmes d'équations linéaires, par exemple, grâce aux opérations efficaces sur les matrices.

[8]:
```
import numpy as np
import sympy as sp
import warnings
warnings.filterwarnings('ignore')
```

1.3 Création de matrices

Une matrice se représente en NumPy à l'aide d'un objet matrix. Il s'agit en fait d'une sous-classe de ndarray, c'est-à-dire que tout ce que l'on a vu sur les atableaux (arrays) s'applique également

aux objets matrix. La manière la plus immédiate de créer une matrice consiste à utiliser la fonction **np.matrix**. Une autre façon de créer une matrice consiste à transformer un tableau multidimensionnel à l'aide de la fonction **np.mat**

Le résultat attendu est un objet matrix, ou plus précisément un objet numpy.matrixlib.defmatrix.matrix, qui est également une instance de la classe ndarray.

1. Créer les matrices suivantes (en utilisant np.mat et en utilisant matrix):

$A =

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & -2 & 1 \\ 0 & -3 & 3 \end{bmatrix}$$

;;;
B =

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & -2 & 2 \\ 1 & -1 & 1 \end{bmatrix}$$

;;; C =

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & -1 & 1 \end{bmatrix}$$

;;;et;;;; D =

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

$

[9]:
```
# à compléter
```

```
A =
 [[ 1  0  0]
 [ 2 -2  1]
 [ 0 -3  3]]
Type de A  <class 'numpy.matrix'>
B =
 [[ 1  2  0]
 [ 0 -2  2]
 [ 1 -1  1]]
Type de B  <class 'numpy.matrix'>
C =
 [[ 1  2  3]
 [ 1  1 -1]]
Type de C  <class 'numpy.matrix'>
```

[10]:
```
# à compléter
```

```
A = [[ 1  0  0]
 [ 2 -2  1]
 [ 0 -3  3]]
Type de A  <class 'numpy.matrix'>
B = [[ 1  2  0]
 [ 0 -2  2]
 [ 1 -1  1]]
Type de B  <class 'numpy.matrix'>
C = [[ 1  2  3]
 [ 1  1 -1]]
Type de C  <class 'numpy.matrix'>
```

On peut créer facilement des matrices particulières comme les matrices identité ou des matrices nulles. Créer les matrices identité de taille 3 et la matrice nulle à 3 lignes et 4 colonnes. On affichera le type obtenu.

[11]:
```
# à compléter
```

```
matrix([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

```
type  <class 'numpy.matrix'>
```

```
matrix([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])
```

```
type  <class 'numpy.matrix'>
```

1.4 Propriétés:

Taille

- Afficher la taille de la matrice A (sa dimension) et vérifier que la matrice A est bien une matrice carrée

[12]:
```
# à compléter
```

```
Matrice A :
[[ 1  0  0]
 [ 2 -2  1]
 [ 0 -3  3]]
Taille de A : (3, 3)
A est-elle carrée ? :  True
```

Diagonale principale

- Afficher la diagonale principale de la matrice A (c'est-à-dire les éléments (a)ij tels que i=j).
- Afficher ensuite les autres diagonales au-dessus et en-dessous de la diagonale principale

[13]:
```
# à compléter
```

```
Diagnonale principale :  [ 1 -2  3]
Diagnonales au dessus :  [0] [0 1]
Diagnonales en dessous :  [ 2 -3] [0]
```

1.4.1 Trace

- Calculer la trace de la matrice carrée A (avec **np.trace**) et vérifier qu'elle est égale à la somme des éléments de sa diagonale principale.

[14]:
```
# à compléter
```

```
Trace de A : 2
Vérification :  True
```

1.4.2 Rang

- Le rang d'une matrice s'obtient avec la fonction **np.linalg.matrix_rank** qui représente notamment le nombre de lignes linéairement indépendantes d'une matrice. Calculer le rang des trois matrices B,C et D :

[15]:
```
# à compléter
```

```
Rang de A : 3
Rang de B : 3
Rang de C : 2
Rang de D : 1
```

1.4.3 Déterminant

- Le calcul du déterminant, uniquement défini sur les matrices carrées, intervient notamment dans la résolution de systèmes d'équations linéaires. On l'obtient avec la fonction **np.linalg.det**.

Calculer les déterminants des matrices A et B:

[16]:
```
# à compléter
```

```
Déterminant de A: -3.0
Déterminant de B: 4.0
Déterminant de D: 0.0
```

1.4.4 Transposée

- Calculer la transposée d'une matrice revient à construire une nouvelle matrice dont les lignes sont les colonnes de la matrice originale, et vice-versa. Calculer la transposée de la matrice A :

[17]:
```
# à compléter
```

```
Transposée de A:
[[ 1  2  0]
 [ 0 -2 -3]
 [ 0  1  3]]
```

1.4.5 Produit

Tout comme pour les vecteurs, différentes possibilités existent pour le produit de matrices. Ces différentes opérations sont évidemment supportées par Numpy.

Produit matriciel

- Étant donné deux matrices compatibles, c'est-à-dire que le nombre de colonnes de la première est égal au nombre de lignes de la seconde, on peut effectuer un produit matriciel entre ces deux matrices.

Attention à ne pas confondre A*B pour les tableaux et np.dot(A,B)

[18]:
```
# à compléter
```

```
array([[1, 2],
       [3, 4]])
```

```
'M1*M1= '
```

```
array([[ 1,  4],
       [ 9, 16]])
```

```
'M1@M1'
```

```
array([[ 7, 10],
       [15, 22]])
```

Calculer le produit matriciel entre les différentes matrices si elles sont de tailles compatibles et voir ce qui se produit si les matrices ne sont pas de tailles compatibles.

[19]:
```
# à compléter
#print(A[:,:2]*C)
#print ("si on demande CA qui n'est pas calculable il se produira une erreur  ")
#np.dot(A,C)
```

```
Matrice A :
[[ 1  0  0]
 [ 2 -2  1]
 [ 0 -3  3]]
Matrice B :
[[ 1  2  0]
 [ 0 -2  2]
 [ 1 -1  1]]
Matrice C :
[[ 1  2  3]
 [ 1  1 -1]]
```

```
A * B :
[[ 1  2  0]
 [ 3  7 -3]
 [ 3  3 -3]]
A . B :
[[ 1  2  0]
 [ 3  7 -3]
 [ 3  3 -3]]

produit matriciel entre C et A
 [[  5 -13  11]
 [  3   1  -2]]

produit matriciel entre A (2 premières lignes) et C
 [[ 1  2  3]
 [ 0  2  8]
 [-3 -3  3]]
```

1.4.6 Inverse

- Une autre opération très importante en algèbre linéaire, notamment pour résoudre des systèmes d'équations linéaires, consiste à calculer l'inverse d'une matrice carrée. Si la matrice est inversible, alors on peut obtenir son inverse avec l'attribut I.

Vérifier si la matrice A est inversible, afficher son inverse et vérifier que le produit de A avec son inverse donne la matrice identité

[20]:
```
# à compléter
```

```
Inverse de A :
[[ 1.00000000e+00 -2.22044605e-16 -5.55111512e-17]
 [ 2.00000000e+00 -1.00000000e+00  3.33333333e-01]
 [ 2.00000000e+00 -1.00000000e+00  6.66666667e-01]]
Produit A * A.I :
[[ 1.00000000e+00 -2.22044605e-16 -5.55111512e-17]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00]
 [-4.44089210e-16  2.22044605e-16  1.00000000e+00]]
Vérification  :
[[False False False]
 [ True False  True]
 [False False False]]
```

Pour s'en sortir lorsque l'on fait des calculs en nombres flottants, il ne faut jamais comparer deux valeurs avec l'opérateur d'égalité. Une solution pour comparer deux matrices de float consiste à utiliser la fonction allclose qui permet de faire une égalité approximative :

[21]:
```
print("Vérification avec np.allclose:")
print(np.allclose(A * A.I, np.identity(3)))
```

```
Vérification avec np.allclose:
True
```

Changement de base : Si A est la matrice d'origine et P la matrice de passage on peut calculer B matrice dans la nouvelle base avec $B = P^{-1}AP$

[22]:
```
A=np.matrix([[1,0,0,-2],[2,-2,2,0],[0,-3,3,6],[1,0,0,-2]])
P = np.matrix([[2,0,0,1],[4,1,2,2],[2,1,3,0],[1,0,0,1]])

# à compléter
```

```
matrice A :

matrix([[ 1,  0,  0, -2],
        [ 2, -2,  2,  0],
        [ 0, -3,  3,  6],
        [ 1,  0,  0, -2]])


matrice de passage P :

matrix([[2, 0, 0, 1],
        [4, 1, 2, 2],
        [2, 1, 3, 0],
        [1, 0, 0, 1]])


calcul de P-1 :

matrix([[ 1.,  0.,  0., -1.],
        [-2.,  3., -2., -4.],
        [ 0., -1.,  1.,  2.],
        [-1.,  0.,  0.,  2.]])


déterminant de P =  1.0

matrice dans la nouvelle base :

matrix([[ 0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.],
        [ 0.,  0.,  1.,  0.],
        [ 0.,  0.,  0., -1.]])
```

2 Systèmes d'équations linéaires

Résoudre un système d'équations linéaires revient à résoudre une équation de la forme $A \cdot X = B$ où A est la matrice du systeme et X et B deux matrices colonnes. On déduit que la solution est simplement $X= A^{-1} \cdot B$ *$souscertainesconditions*.

- Résoudre le système suivant :

$$\begin{cases} 3x + y - z & = 2 \\ -x - y + z & = 0 \\ 2x + y + z & = 3 \end{cases}$$

Le systeme devient $A \cdot X = B$

[23]:
```
# à compléter
```

```
sans  arrondi  x= 1.0  y= 1.6653345369377346e-16  z= 1.0

avec arrondi : x= 1.0  y= 0.0  z= 1.0
```

[24]:
```
A=np.array([[3,1,-1],[-1,-1,1],[2,1,1]])
B=np.array([[2],[0],[3]])
C=np.array([2,0,3])
# à compléter
```

```
matrice A
 [[ 3  1 -1]
 [-1 -1  1]
 [ 2  1  1]]

matrice B
 [[2]
 [0]
 [3]]

matrice C
 [2 0 3]

solve A,B
  [[1.00000000e+00]
 [1.66533454e-16]
 [1.00000000e+00]]

avec arrondi :
  [[1.]
 [0.]
 [1.]]

solve A, C
 [1.00000000e+00 1.66533454e-16 1.00000000e+00]

avec arrondi
  [1. 0. 1.]
```

Résoudre l'équation suivante : (Résolution approximative)

$$\begin{cases} x + 2y & = 1 \\ 2x + 4y & = 0 \end{cases}$$

[25]:
```
# à compléter
```

```
Determinant de A: 0.0
```

Heureusement, tout n'est pas perdu. On peut, en effet, obtenir une solution approximative, qui va faire en sorte que Ax soit le plus proche possible de b, c'est-à-dire une solution qui va minimiser bAx. Pour cela, il suffit d'utiliser la fonction lstsq du module scipy.linalg, à qui on passe en paramètres les matrices A et b, comme ci-dessous :

[26]:
```
# à compléter
```

```
Solutions proches :
x= 0.04  y= 0.08
```

Résoudre l'équation suivante : x +y =2

On va, en fait, obtenir une des solutions acceptables pour le système :

[27]:
```
# à compléter
```

```
Une solution acceptable :
[[1.]
 [1.]]
```

3 Matrices avec sympy

On peut aussi utiliser sympy pour travailler avec les matrices. Ceci permet en plus d'utiliser des symboles et de calculer avec des symboles.

1. Créer les matices suivantes puis calculer les déterminants. On pourra utiliser factor pour factoriser les déterminants.

$A =
$$\begin{bmatrix} -1-a & 4 & 2 \\ 2 & 1-a & -1 \\ -2 & 0 & 2-a \end{bmatrix}$$
;;; B =
$$\begin{bmatrix} -1 & 3 & a \\ -2 & 4 & a \\ 2 & -2 & -1 \end{bmatrix}$$
;;;et;;;; C =
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a \\ 1 & 1 & a \end{bmatrix}$$
$

[28]:
```
a# à compléter
```

[29]:
```
# à compléter
```

$$\begin{bmatrix} -a-1 & 4 & 2 \\ 2 & 1-a & -1 \\ -2 & 0 & 2-a \end{bmatrix}$$

```
determinant de la matrice  -a**3 + 2*a**2 + 5*a - 6
determinant de la matrice  -(a - 3)*(a - 1)*(a + 2)
```

[30]:
```
B=sp.Matrix([[-1,3,a],[-2,4,a],[2,-2,-1]])
display(B)
da=B.det()
print("determinant de la matrice B ",da)
# à compléter
```

$$\begin{bmatrix} -1 & 3 & a \\ -2 & 4 & a \\ 2 & -2 & -1 \end{bmatrix}$$

```
determinant de la matrice B  -2
```

[31]:
```
# à compléter
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a \\ 1 & 1 & a \end{bmatrix}$$

```
determinant de la matrice  a**2 - 2*a + 1
determinant de la matrice  (a - 1)**2
```

[32]:
```
# à compléter
```

```
inverse de C quand c'est possible
```

$$\begin{bmatrix} \frac{a}{a-1} & -\frac{1}{a-1} & 0 \\ 0 & \frac{1}{a-1} & -\frac{1}{a-1} \\ -\frac{1}{a-1} & 0 & \frac{1}{a-1} \end{bmatrix}$$

3.1 Résolution de systèmes linéaires

- Résoudre le système suivant : (avec une solution)

$$\begin{cases} 2x - 3y + z & = -1 \\ x - y + 2z & = -3 \\ 3x + y - z & = 9 \end{cases}$$

Remarque : Avec l'aide de la méthode sp.Matrix().rref() , nous pouvons mettre une matrice sous forme d'échelon de ligne réduit . Matrix().rref() renvoie un tuple de deux éléments. La première est la forme d'échelon de ligne réduite et la seconde est un tuple d'indices des colonnes pivot.

[ ]:

4 Des exercices d'algèbre 2

4.1 Exercice A

Soit E l'espace vectoriel $\mathbb{R}^3$. On considère l'application linéaire f de E dans E dont la matrice dans la base canonique $\mathcal{E}$ de E est

$$A = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}.$$

1. Soit v le vecteur de E de coordonnées (x, y, z) dans la base $\mathcal{E}$. Calculer $f(v)$. 2. Calculer le rang de A. L'endomorphisme f est-il injectif, surjectif, bijectif? 3. On pose $v_1 = (0, 1, 0)$, $v_2 = (1, 0, -1)$ et $v_3 = (1, 0, 1)$. Montrer que $\mathcal{V} = (v_1, v_2, v_3)$ est une base de E. 4. Calculer $f(v_1)$, $f(v_2)$ et $f(v_3)$ en fonction des éléments de $\mathcal{V}$. 5. Déterminer la matrice B de f dans la base $\mathcal{V}$. 6. Ecrire la matrice de passage P de la base $\mathcal{E}$ 'a la base $\mathcal{V}$. 7. Calculer P^{-1}. 8. Donner la relation qui lie les matrices A, B, P et P^{-1} . Comment peut -on calculer A^n?

4.2 Exercice B

On considère la matrice A suivante, où m désigne un nombre réel :

$$A = \begin{pmatrix} m & (1+m^2) & 2m \\ (2-m^2) & m & 1 \\ 4 & 2m & 4 \end{pmatrix}.$$

1. Montrer que le déterminant de la matrice A est égal à $2m^2 - 4$. Pour quelles valeurs du réel m, la matrice A est-elle inversible?
2. On se place dans le cas où A est inversible. Déterminer la comatrice de A puis la matrice inverse de A.
3. Soit f l'endomorphisme de R^3 dont la matrice associée dans la base canonique de R^3 est A. Pour quelles valeurs de m, l'application f est un isomorphisme de R^3 dans R^3 ?

4.3 Exercice C

Pour $m \in \mathbb{R}$, on note $A_m = \begin{pmatrix} 3 & 1 & -2 \\ 2+m & 2 & -2 \\ 4 & 2+m & -3 \end{pmatrix}$ et $f_m : \mathbb{R}^3 \to \mathbb{R}^3$ l'endomorphisme de $\mathbb{R}^3$ ayant pour matrice A_m dans la base canonique de $\mathbb{R}^3$.

1. Calculer $\det(A_m)$.
   Pour quelles valeurs de m a-t-on f_m bijectif ? Injectif ?
2. Calculer $\ker(f_m)$ en distinguant suivant les valeurs de m.
3. Dans cette question, on s'intéresse à A_1 et f_1 (c'est-à-dire qu'on fixe $m = 1$ dans la matrice A_m).

a. Calculer A_1^{-1}.

b. En déduire, pour $(x, y, z) \in \mathbb{R}^3$, une expression de $f_1^{-1}(x, y, z)$.

4. On fixe maintenant $m = 0$ et on note $f = f_0$.

a. Donner une base de $\mathrm{Im}(f)$.

b. Montrer que f est la projection sur un sous-espace vectoriel F parallèlement à un sous-espace vectoriel G (à préciser).

c. Donner une base $\mathcal{B}$ de $\mathbb{R}^3$ dans laquelle la matrice de f est $D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

4.4 A l'aide de numpy et sympy, obtenez un maximum de resultats avec Python !

[ ]: