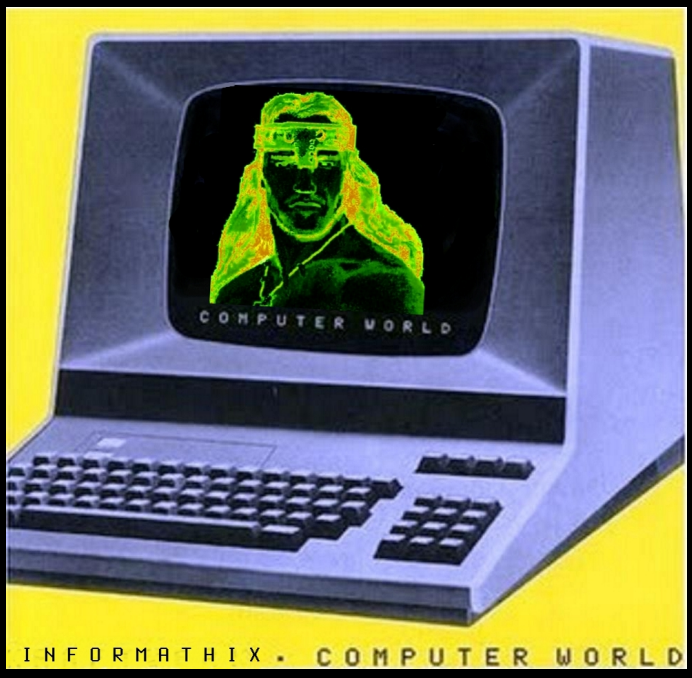
Mathématiques

2011-2012

IUT d'informatique de Nantes 2e année



Guillaume CONNAN

TABLE DES MATIÈRES

CHAPITRE 2

Matrices, déterminants et applications

1 Calcul matriciel avec Python

1.1 Les boucles for

Lorsque l'on souhaite répéter un bloc d'instructions un nombre déterminé de fois, on peut utiliser un *compteur actif*, c'est-à-dire une variable qui compte le nombre de répétitions et conditionne la sortie de la boucle **while**. C'est le cas de l'exemple suivant où l'on définit une fonction qui prend en argument un entier **n** et affiche **n** fois le même message.

```
def f(n):
    i = 0 # on initialise le compteur i
    while i < n:
        print('Je me répète {} fois.'
              ' (i={})'.format(n, i))
        i += 1 # on incrémente i
f(5)
```

```
Je me répète 5 fois. (i=0)
Je me répète 5 fois. (i=1)
Je me répète 5 fois. (i=2)
Je me répète 5 fois. (i=3)
Je me répète 5 fois. (i=4)
```

Pour effectuer une telle répétition, on dispose d'une structure de répétition nous économisant d'une part l'initialisation du compteur (`i = 0`), et d'autre part son incrémentation (`i += 1`) : c'est la structure introduite par le mot-clé `for` :

```
def f(n):
    for i in range(n):
        print('Je me répète {} fois.'
              ' (i={})'.format(n, i))
f(5)
```

```
Je me répète 5 fois. (i=0)
Je me répète 5 fois. (i=1)
Je me répète 5 fois. (i=2)
Je me répète 5 fois. (i=3)
Je me répète 5 fois. (i=4)
```

Nous voyons apparaître ici pour la première fois la fonction `range`. Cette fonction crée un *itérateur*, c'est-à-dire en quelque sorte un distributeur d'entiers consécutifs. Au lieu de créer et garder en mémoire une liste d'entiers, cette fonction génère les entiers au fur et à mesure des besoins, toujours dans un souci d'optimisation de la mémoire.

Remarque 1

Plus précisément, un itérateur est un objet possédant une méthode **`__iter()__`** qui renvoie les éléments d'une collection un par un et qui provoque une exception du type **`StopIteration`** lorsqu'il n'y a plus d'éléments.

Avec un argument, **`range(n)`** renvoie un itérateur parcourant l'intervalle $[\![0, n-1]\!]$; avec deux arguments, **`range(n, p)`** parcourt l'intervalle $[\![n, p-1]\!]$; enfin employée avec trois arguments, **`range(n, p, k)`** parcourt l'intervalle $[\![n, p-1]\!]$ avec un pas égal à k.
Signalons que la fonction **`range`** peut servir à créer des listes d'entiers, moyennant une conversion de type opérée par la fonction **`list`**. Examinons quelques exemples.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0, 10, 3))
[0, 3, 6, 9]
>>> list(range(0, -6, -1))
[0, -1, -2, -3, -4, -5]
```

```
>>> list(range(0)), list(range(1, 0))
[] []
```

```
>>> list(range(9, 0, -1))
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Autre point important, on peut aussi utiliser comme itérateur dans une boucle **`for`** une

liste ou une chaîne de caractère.

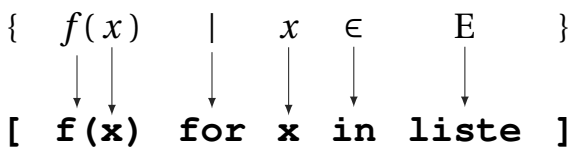
Remarque 2

Plus généralement, le mot-clé **for** peut être utilisé avec n'importe quel objet *itérable* ; un objet itérable étant un objet possédant soit une méthode **__iter__()** soit une méthode **__getitem__()**.

```
for i in [10, 'a', 1.414]:
    print(i, end=' -> ')
for i in 'mot':
    print(i, end=' -> ')
```

```
10 -> a -> 1.414 -> m -> o -> t ->
```

Pour créer des listes, **Python** fournit une facilité syntaxique particulièrement agréable, à savoir les listes (définies) par compréhension (en anglais, *list-comprehensions*). Elles permettent de générer des listes d'une manière très concise, sans avoir à utiliser de boucles. La syntaxe pour définir une liste par compréhension est proche de celle utilisée en mathématiques pour définir un ensemble par compréhension :



Voici quelques exemples :

```
>>> liste = [2, 4, 6, 8, 10]
>>> [3*x for x in liste]
[6, 12, 18, 24, 30]
>>> [[x, x**3] for x in liste]
[[2, 8], [4, 64], [6, 216], [8, 512], [10, 1000]]
>>> [3*x for x in liste if x > 5] # on filtre avec une condition
[18, 24, 30]
>>> [3*x for x in liste if x**2 < 50] # idem
[6, 12, 18]
>>> liste2 = list(range(3))
>>> [x*y for x in liste for y in liste2]
[0, 2, 4, 0, 4, 8, 0, 6, 12, 0, 8, 16, 0, 10, 20]
```

Voici par exemple, un moyen efficace d'obtenir la liste des années bissextiles dans un intervalle donné :

```
>>> bissextile = [b for b in range(2000, 2100)
...               if (b % 4 ==0 and b % 100 != 0) or (b % 400 == 0)]
```

Pour concaténer les éléments d'une liste de listes, on peut imbriquer deux boucles **for** dans la définition d'une liste par compréhension :

```
>>> xll = [[1, 2, 3], [4, 5], [6, 7, 8]]
>>> [x for xl in xll for x in xl]
[1, 2, 3, 4, 5, 6, 7, 8]
```

Pour obtenir les diviseurs d'un entier n, on peut également utiliser une liste par compréhension :

```
>>> n = 100; [d for d in range(1, n+1) if n % d == 0]  # les diviseurs de 100
[1, 2, 4, 5, 10, 20, 25, 50, 100]
```

1 2 Fabriquons nos outils

Dans cette section, nous créerons nos matrices comme une liste de listes représentant les lignes.

Par exemple, $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ sera creée avec :

```
>>> M=[[1,2],[3,4],[5,6]]
```

Pour obtenir le terme $M_{i,j}$ situé sur la ligne i et la colonne j, on entrera `M[i][j]`.

À retenir

Attention ! **Python** commence à compter à partir de 0. Ainsi le premier élément d'une liste est indexé par 0.

```
>>> M[0]
[1, 2]
>>> M[0][1]
2
```

La commande **len** (comme *length*) renvoie la longueur d'une liste. On obtient donc le nombre de lignes de la matrice avec **len**`(M)` et son nombre de colonnes avec **len**`(M[0])`. Comme nous utiliserons souvent ces nombres, on va créer deux fonctions pour rendre la lecture des programmes plus évidente :

```
def lignes(M):
    return range(len(M))

def cols(M):
    return range(len(M[0]))
```

Exercice 2 - 1 Transposée d'une matrice

Nous vous rappelons que la transposée d'une matrice $(M_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant j\leqslant m}}$ est égale à la matrice $(M_{ji})_{\substack{1\leqslant i\leqslant n\\1\leqslant j\leqslant m}}$.

Définissez une fonction `transpose(M)` qui renvoie la transposée de la matrice M.

Attention aux indices et aux nombre de lignes et de colonnes !

Exercice 2 - 2 Somme de matrices

On voudrait créer une procédure `msom(A,B)` qui prend comme arguments deux matrices et renvoie leur somme. Il faudra vérifier que les matrices à additionner sont de bonnes tailles.

À retenir

Pour la gestion des erreurs (de taille par exemple), on va créer une *classe d'exception*. On la crée la plus générale possible pour l'utiliser dans d'autres domaines.

```
class ExceptionMatrice(Exception):
    def __init__(self,raison):
        self.raison = raison

    def __str__(self):
        return self.raison
```

On l'invoquera avec la commande **raise**. Ici :

```
if len(A[0])!=len(B):
    raise ExceptionMatrice("Problème de taille")
```

Exercice 2 - 3 Produit par un scalaire

On voudrait obtenir la matrice $k \cdot \mathrm{M}$ à partir d'une matrice M et d'un scalaire k. Créer une fonction `mprod_scal(M,k)` qui agira ainsi :

```
>>> A=[[2,4],[6,8],[10,12]]
>>> mprod_scal(A,0.5)
[[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]]
```

Je ne vous fais pas l'injure de vous rappeler comment effectuer le produit de deux matrices...Bon, seulement la formule :
Soit $\mathrm{A} = (a_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant i\leqslant m}}$ et $\mathrm{B} = (b_{ij})_{\substack{1\leqslant i\leqslant m\\1\leqslant i\leqslant p}}$. Alors $\mathrm{A}\times\mathrm{B} = \mathrm{C}$ avec $\mathrm{C} = (c_{ij})_{\substack{1\leqslant i\leqslant n\\1\leqslant i\leqslant p}}$ et

$$\forall (i,j) \in \mathbb{N}_n^* \times \mathbb{N}_p^*, \quad c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

Exercice 2 - 4 Produit de matrices

Créer une fonction `mprod(A,B)` qui renvoie le produit de deux matrices A et B en utilisant trois boucles *pour* imbriquées et sans oublier les tests de taille. Vous testerez par exemple avec $\mathrm{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ et $\mathrm{B} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$. Essayer $\mathrm{A}\times\mathrm{B}$ et $\mathrm{B}\times\mathrm{A}$. Essayer ensuite avec deux matrices carrées.

Variante : créer une fonction `som_liste(L)` qui calcule la somme des éléments d'une liste d'entiers et l'utiliser pour créer une fonction `mprod2(A,B)` sans utiliser de matrice temporaire mais en fabriquant la liste produit en place avec une liste par compréhension.

Variante bis : on peu aussi gagner un peu de temps en créant des variables pour stocker `A[i]` et `C[i]` dans la première version, du style `Ai,Ci = A[i],C[i]`. L'affectation se faisant par pointeurs, les listes A et C vont évoluer avec `Ai` et `Ci`.

Variante ter : On gagnerait encore du temps en faisant de même avec B. Il suffirait d'utiliser la transposée de B : `tB = transpose(B)` et `tBj = tB[j]`.

À retenir

On peut comparer les temps d'exécution avec la fonction **clock** du module **time** :

```
from time import clock
A=[[1 for k in range(200)] for k in range(200)]
debut1 = clock()
mprod(A,A)
fin1 = clock()
debut2 = clock()
mprod2(A,A)
fin2 = clock()
debut3 = clock()
mprod1bis(A,A)
fin3 = clock()
print (fin1 - debut1 , fin2 - debut2, fin3-debut3)
```

Exercice 2 - 5 Matrice identité

Créer une fonction qui fabrique la matrice identité d'ordre n selon le modèle utilisé.
Par exemple, I_3 sera obtenue ainsi :

```
>>> unite(3)
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

Exercice 2 - 6 Puissances d'une matrice

Créer une fonction `mpuis(A,n)` qui calcule A^n. Donner une version récursive puis une version itérative.

Par exemple, avec $J = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$:

```
>>> J=[[1,1],[1,1]]
>>> mpuis(J,5)
[[16, 16], [16, 16]]
```

Démontrer alors par récurrence que $J^n = 2^{n-1}J$.

Exercice 2 - 7 Trace d'une matrice

Créer une fonction `trace(M)` qui calcule la trace d'une matrice.
Par exemple, on peut calculer la trace de J^5 :

```
>>> trace(mpuis(J,5))
32
```

1 3 Inverse d'une matrice par la méthode de Gauss-Jordan

Exercice 2 - 8 Opérations élémentaires

Pour effectuer une combinaison linéaire des lignes, on va créer une fonction :

```
add_ligne_scal(k,M,i,j)
```

qui renverra la matrice construite à partir de M en remplaçant la ligne L_j par $k_j \times L_j + k_i \times L_i$.
On crée également une fonction `mult_ligne(k,M,j):` qui renverra la matrice construite à partir de M en remplaçant la ligne L_j par $k \times L_j$.
On pensera à utiliser la copie « en profondeur » `deepcopy` du module `copy`.

Exercice 2 - 9 Calcul de l'inverse étape par étape

On commence par créer une fonction `GJ(M)` qui crée une matrice où se juxtaposent la matrice initiale et la matrice identité.
Par exemple, avec $M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ on obtient $T = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$
Pour avoir une meilleure « vision » de la matrice, on pourra utiliser une fonction `affiche` :

```
def affiche(M):
    for i in lignes(M):
        print M[i]
```

```
>>> M = [[1,0,1],[0,1,1],[1,1,0]]
>>> T = GJ(M)
>>> affiche(T)
[1, 0, 1, 1, 0, 0]
[0, 1, 1, 0, 1, 0]
[1, 1, 0, 0, 0, 1]
```

Effectuer pas à pas les transformations. À la fin, on devrait obtenir :

$$T = \begin{pmatrix} 1 & 0 & 0 & 1/2 & -1/2 & 1/2 \\ 0 & 1 & 0 & -1/2 & 1/2 & 1/2 \\ 0 & 0 & 1 & 1/2 & 1/2 & -1/2 \end{pmatrix}$$

Il ne reste plus qu'à extraire la moitié droite du tableau qui sera l'inverse cherchée à l'aide d'une petite fonction `JG(T)`.
On rappelle la syntaxe de manipulation des listes : `L[1:4],L[1:], L[:]`, etc.
Alors :

```
>>> IM = JG(T)
>>> affiche(IM)
[0.5, -0.5, 0.5]
[-0.5, 0.5, 0.5]
[0.5, 0.5, -0.5]
>>> affiche(mprod(IM,M))
[1.0, 0.0, 0.0]
[0.0, 1.0, 0.0]
[0.0, 0.0, 1.0]
```

1 3 1 Calcul direct de l'inverse par la méthode de Gauss-Jordan

Si la matrice est inversible, alors, en effectuant des opérations élémentaires sur les lignes, on peut se ramener à la matrice identité dans la partie gauche du tableau et on obtient son inverse dans la partie droite.
Pour y arriver, l'idée est de balayer le tableau par colonne.
Étant donné une colonne, on cherche un élément non nul. S'il n'y en a pas, la matrice n'est pas inversible et le système n'admet pas une unique solution ; sinon, on permute éventuellement deux lignes pour placer l'élément non nul de la colonne k sur la ligne k et on divise tous les éléments de la ligne par le nouveau a_{kk} pour obtenir 1.

Il reste ensuite à remplacer chaque ligne (autre que L_k) dont l'élément de la colonne k est non nul par $L_i - a_{ik} \times L_k$.

Exercice 2 - 10

Créer une fonction `inv_gauss(M)` qui renvoie, si elle existe, la matrice inverse de M.
On créera d'abord une fonction `swap(L,i,j)` qui échange deux lignes.
Analyser ensuite le programme suivant :

```
def inv_gauss(T):
    if lignes(M) != cols(M):
    raise ExceptionMatrice("La matrice doit être carrée")
    S = GJ(T)
    for NoColonne in lignes(T):
        NoLigne = NoColonne
        while S[NoLigne][NoColonne] == 0:
            if NoLigne == len(T) - 1:
                raise ExceptionMatrice("Matrice non inversible")
            NoLigne += 1
        S = swap(S,NoLigne,NoColonne)
        S = mult_ligne(float(1)/float(S[NoColonne][NoColonne]),S,NoColonne)
        E = set([k for k in lignes(T)]) - set([NoColonne])
        for k in E:
            S = comb_ligne(-S[k][NoColonne],1,S,NoColonne,k)
    return JG(S)
```

1 3 2 Réduction sous-diagonale d'une matrice

Exercice 2 - 11

On s'inspire de la fonction précédente, mais au lieu de travailler sur toutes les lignes, on ne transforme que les lignes en-dessous du pivot.
Cela permet également de généraliser l'emploi de la méthode de GAUSS à des matrices non carrées pour la résolution de systèmes par exemple. Il faut donc faire attention maintenant à utiliser le minimum entre le nombre de lignes et le nombre de colonnes de la matrice.
On obtiendra ainsi une matrice triangulaire. L'intérêt est aussi que tous les calculs s'effectuent dans l'anneau de départ car il n'y a plus de divisions.

1 3 3 Base de l'image - Rang d'une matrice

On écrit les vecteurs engendrant l'image de l'endomorphisme en ligne : bref, on transpose la matrice. En effet, ce sera plus simple d'étudier la matrice par lignes car il s'agit des sous-listes de la liste représentant la matrice. Suite à la réduction de `tri_gauss` on ne garde que les lignes non nulles : la famille obtenue est une base de l'image.

Exercice 2 - 12

Déterminer une fonction `Im(M)` qui retourne une base de l'image puis une fonction `rang(M)` qui renvoie le rang de la matrice M.

2 Déterminants

2 1 Propriétés des déterminants

Dans la suite, $A = (a_{ij})$, $B = (b_{ij})$ sont des matrices carrées de taille n, λ est un scalaire. L_i est une ligne quelconque de A.

Propriété 2 - 1

Soient i et j deux entiers compris entre 1 et n.

$$\det A = \sum_{k=1}^{n} a_{1k}\Delta_{1k} = \sum_{k=1}^{n} a_{ik}\Delta_{ik} = \sum_{k=1}^{n} a_{kj}\Delta_{kj}$$

On admettra cette propriété. « Traduisez-la » par une phrase en bon français.
Appliquez-la pour calculer de tête le déterminant de la matrice suivante :

$$M = \begin{bmatrix} 1 & 17 & 32 & 49 \\ 0 & 5 & 0 & 3 \\ 0 & 39 & 2 & -49 \\ 0 & 7 & 0 & 1 \end{bmatrix}$$

Propriété 2 - 2

Le déterminant d'une matrice triangulaire est égal au produit des éléments de sa diagonale principale.

Sans utiliser cette propriété, calculez de tête le déterminant de la matrice suivante :

$$M = \begin{bmatrix} 2 & 32 & 45 & 87 & 987 \\ 0 & -1 & 568 & -542 & 712 \\ 0 & 0 & 5 & 741 & -12 \\ 0 & 0 & 0 & 1 & -789 \\ 0 & 0 & 0 & 0 & -10 \end{bmatrix}$$

Prouvez cette propriété dans le cas général.

Propriété 2 - 3

Si la matrice B résulte de l'échange de deux lignes ou de deux colonnes d'une matrice A alors $\det B = -\det A$

On admettra cette propriété.
Calculez de tête le déterminant de la matrice suivante :

$$M = \begin{bmatrix} 45 & 32 & 2 & 987 & 87 \\ 568 & -1 & 0 & 712 & -542 \\ 5 & 0 & 0 & -12 & 741 \\ 0 & 0 & 0 & -789 & 1 \\ 0 & 0 & 0 & -10 & 0 \end{bmatrix}$$

Propriété 2 - 4

$$\det\left({}^{t}A\right) = \det A$$

Démontrez cette propriété.

Propriété 2 - 5

Le déterminant d'une matrice comportant une ligne (ou une colonne) de 0 est nul.

Démontrez cette propriété.

Propriété 2 - 6

Soit B la matrice obtenue à partir de A en effectuant $L_i \leftarrow \lambda L_i$, alors $\det B = \det A$.

Démontrez cette propriété.

Propriété 2 - 7

$$\det(\lambda A) = \lambda^n \det A$$

Démontrez cette propriété.
Si A est une matrice carrée d'ordre 5 dont le déterminant vaut 4, que vaut $\det(-2A)$?
Démontrez que le déterminant d'une matrice antisymétrique (i.e. ${}^tA = -A$) d'ordre 37 est nul.

Propriété 2 - 8

Le déterminant d'une matrice comportant deux lignes (ou colonnes) identiques est nul.

Démontrez cette propriété en utilisant la propriété 2 - 3 page ci-contre.

Propriété 2 - 9

Le déterminant d'une matrice qui comporte deux lignes (ou colonnes) dont l'une est multiple de l'autre est nul.

Démontrez cette propriété.

Propriété 2 - 10

Soit B la matrice obtenue à partir de A en effectuant $L_i \leftarrow \lambda L_i + \lambda L_j$, alors $\det B = \det A$.

Démontrez cette propriété. Utilisez-la pour calculer le déterminant de la matrice suivante :

$$M = \begin{bmatrix} 1 & 3 & 2 & -1 \\ -1 & -2 & 2 & 6 \\ 2 & 4 & -3 & 2 \\ -2 & -7 & 4 & 1 \end{bmatrix}$$

Estimez le nombre d'opérations nécessaires pour effectuer le calcul du déterminant d'une matrice de taille n en utilisant uniquement la définition.
Faites de même lorsqu'on utilise l'algorithme de GAUSS.
Il y a quelques jours, un ordinateur japonais de la firme Fujitsu a établi un nouveau record de vitesse de calcul de 10 pétaflops, soit 10^{16} opérations mathématiques à virgule flottante par seconde. L'ordinateur utilisé compte quelque 88 000 processeurs enchâssés dans 864 serveurs ; chacun des processeurs dispose de 8 cœurs d'exécution.
Combien de temps lui faudra-t-il environ pour effectuer le calcul d'un déterminant de taille 30 à l'aide de la définition ? Avec l'algorithme de GAUSS ?

Analysez le programme Python suivant :

```
def mystere(T):
    if lignes(M) != cols(M):
        raise ExceptionMatrice("La matrice doit être carrée")
    S = T
    i = 1
    for NoColonne in lignes(T):
        NoLigne = NoColonne
        while (S[NoLigne][NoColonne]==0) and (NoLigne<len(T)-1) :
            NoLigne+=1
        swap(S,NoLigne,NoColonne)
        i *= -1
        for k in range(NoLigne+1,len(T)):
            S = comb_ligne(-S[k][NoColonne],S[NoColonne][NoColonne],S,NoColonne,k)
            i *= -S[NoColonne][NoColonne]
    d = 1
    for k in lignes(S):
           d *= S[k][k]
    return(d/i)
```

Propriété 2 - 11

$$\det(A \times B) = \det(A) \cdot \det(B)$$

On admettra cette définition.
Soit A une matrice inversible. Calculez $\det(A^{-1})$ en fonction de $\det(A)$.

Définition 2 - 1

Une matrice dont le déterminant est nul est dite **singulière**.

Une matrice singulière peut-elle être inversible ?
La matrice suivante est-elle inversible ?

$$M = \begin{bmatrix} x^2 & xy & xz \\ x+2 & y+2 & z+2 \\ 3 & 3 & 3 \end{bmatrix}$$

2 2 Quelques exercices

Exercice 2 - 1

Les déterminants peuvent servir à calculer certaines surfaces. On peut montrer par exemple que l'aire du triangle dont les sommets sont les points $A(x_1, y_1)$, $B(x_2, y_2)$ et $C(x_3, y_3)$ est égal à la valeur absolue de

$$\frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Quelle est l'aire du triangle dont les sommets ont pour coordonnées (2,3), (1,1) et (5,-1) ? (3,4), (2,1) et (4,7) ? Qu'en concluez-vous dans ce dernier cas ?

Exercice 2 - 2

Soit $A = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$ et $B = \begin{bmatrix} 0 & 0 \\ 3 & 4 \end{bmatrix}$
Calculez $\det(A)$, $\det(B)$ et $\det(A+B)$. Conclusion ?

Exercice 2 - 3

Résolvez les équations suivantes dans $\mathbb{R}$:

1. $\begin{vmatrix} 3 & x \\ x+2 & 1-x \end{vmatrix} = -3$
2. $\begin{vmatrix} x-2 & 3 \\ x & x+1 \end{vmatrix} = 3$

Exercice 2 - 4

Pour quelles valeurs de x les matrices suivantes sont-elles singulières ?

1. $M_1 = \begin{bmatrix} 1 & x-1 \\ x & 3x \end{bmatrix}$
2. $M_2 = \begin{bmatrix} 2x & x+1 \\ x-2 & 3x \end{bmatrix}$
3. $M_3 = \begin{bmatrix} 1 & x & x \\ 1 & 2 & 3 \\ x & x & 1 \end{bmatrix}$
4. $M_4 = \begin{bmatrix} 1 & x & x^2 \\ 1 & 1 & 1 \\ 1 & 3 & 9 \end{bmatrix}$

Exercice 2 - 5

$A_{ij} = \begin{cases} 2 \text{ si } i \leqslant j \\ 0 \text{ si } i > j \end{cases}$. Calculer det(A). Faites de même si $A_{ij} = \begin{cases} i \text{ si } i = j \\ 0 \text{ si } i \neq j \end{cases}$.

Exercice 2 - 6

Soit $M = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$ et det(M) = −4. Calculez le déterminant des matrices suivantes :

$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

$B = \begin{bmatrix} g & a & d \\ h & b & e \\ i & c & f \end{bmatrix}$

$C = \begin{bmatrix} 3a & d & -g \\ 3b & e & -h \\ 3c & f & -i \end{bmatrix}$

$D = \begin{bmatrix} a & d & g \\ 4b+3a & 4e+3d & 4h+3g \\ c & f & i \end{bmatrix}$

Exercice 2 - 7

Soient A, B et P trois matrices de même ordre telles que $B = P^{-1}AP$. Calculez le déterminant de B en fonction de celui de A.

Exercice 2 - 8

Une matrice est dite **orthogonale** si, et seulement si, sa transposée est égale à son inverse.

1. La matrice I_n est-elle orthogonale ?
2. La matrice $A = \begin{bmatrix} \sin(t) & \cos(t) \\ -\cos(t) & \sin(t) \end{bmatrix}$ est-elle orthogonale pour tout $t \in \mathbb{R}$?
3. Montrez que le déterminant d'une matrice orthogonale est 1 ou −1.
4. Déterminez la troisième ligne de la matrice orthogonale : $\begin{bmatrix} 1/3 & 2/3 & 2/3 \\ 2/3 & -2/3 & 1/3 \\ & & \end{bmatrix}$