TP14

Matrices

Dans ce TP, nous allons voir comment définir et manipuler les matrices sous Maple. Pour ce faire, nous chargerons le package `linalg` présentant les commandes indispensables au calcul matriciel:

> `with(linalg)`

BlockDiagonal, GramSchmidt, JordanBlock, (...) wronskian

1 Matrices : définitions et opérations courantes

1.1 Définition d'une matrice

La définition d'une matrice se fait à l'aide de la fonction `matrix` :

> *nom*`:=matrix(`n`,`m`,[`v_1`,..,`v_{nm}` ]);` > *nom*`:=matrix([[`$v_{1,1}$`,..,`$v_{1,m}$`],..,[`$v_{n,1}$`,..,`$v_{n,m}$`]]);`

où la première instruction crée une matrice à n lignes et m colonnes dont les coefficients sont $v_1, \ldots, v_{nm}$ et la deuxième crée une matrice qui contient les valeurs $v_{1,1}, \ldots, v_{n,m}$. Les matrices sont du type `array`. En particulier, il y a équivalence entre les matrices et les tableaux sous Maple. Par exemple,

> `A:=matrix(2,3,[0,0,0,0,0,0]);B:=matrix([[1,2,3],[4,5,6],[7,8,9]]);`

$$A := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Remarquons que Maple n'affiche pas spontanément la matrice sous la forme "habituelle" et lisible. Il faut pour obtenir cela utiliser la fonction `evalm`:

> `A,evalm(A);`

$$A, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

On vérifie que Maple reconnaît ces variables comme des matrices :

> `whattype(A),whattype(B);`

symbol,symbol

Maple reconnaît A et B comme des chaînes de caractères... En fait, il faut évaluer la matrice pour que Maple la reconnaisse comme telle :

> `whattype(evalm(A));`

array

On peut alors modifier les valeurs des coefficients d'une matrice comme suit :

> `A[2,1]:=1`

$$A_{2,1} = 1$$

On vérifie que la matrice A a effectivement été modifiée :

> `evalm(A);`

$$A := \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Pour définir la matrice identité ou une matrice diagonale, on pourra préférer utiliser les commandes suivantes:

> `array(identity,1..3,1..3),diag(1,5,6);`

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

À l'aide de la fonction `blockmatrix`, on peut construire une matrice par blocs à partir de matrices déjà définies:

> `Z:=matrix(1,3,[2,2,2]): A:=blockmatrix(2,1,[A,Z]);`

$$A := \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 2 & 2 \end{bmatrix}$$

Réciproquement, il est possible d'extraire une sous-matrice d'une matrice:

> `submatrix(B,1..3,2..3]`

$$\begin{bmatrix} 2 & 3 \\ 5 & 6 \\ 8 & 9 \end{bmatrix}$$

Remarquons la nécessité de charger préalablement le package `linalg` pour pouvoir utiliser les fonctions `diag`, `blockmatrix` et `submatrix`;

1.2 Opérations courantes sur les matrices

On récapitule dans le tableau ci-contre les principaux opérateurs du calcul matriciel, qui ne nécessite pas d'avoir chargé le package `linalg`. Par contre, pour des opérations plus complexes, comme celles présentées dans la section suivante, il faudra le charger au préalable.

Opérateur	**Notation**
Somme	$+$
Différence	$-$
Produit par un scalaire	$*$
Produit matriciel	. ou &*
Puissance	$\wedge$ ou **

Ilustrons ces opérations sur les matrices A et B définies précédemment (en utilisant `evalm` pour voir le résultat):

> `evalm(A+B),evalm(A-B),evalm(2*A),evalm(A&*B),evalm(A^2);`

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 5 & 6 \\ 9 & 10 & 11 \end{bmatrix}, \begin{bmatrix} -1 & -2 & -3 \\ -3 & -5 & -6 \\ -5 & -6 & -7 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 4 & 4 & 4 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 3 \\ 24 & 30 & 36 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 4 & 4 \end{bmatrix}$$

Il est aussi possible d'appliquer une fonction à une matrice :

> `evalm(sin(A));`

$$A := \begin{bmatrix} 0 & 0 & 0 \\ sin(1) & 0 & 0 \\ sin(2) & sin(2) & sin(2) \end{bmatrix}$$

Avant de revenir sur les fonctions du package `linalg`, intéressons nous un instant aux conversions de structures possibles avec les matrices.

1.3 Conversion des matrices

On peut passer d'une liste de listes à une matrice et réciproquement. Rappelons que les matrices que l'on considère sont sous la forme de tableaux et donc de type `array`. Définissons une liste de triplets :

> `L:=[[0,5,5],[9,4,2],[2,3,7]];`

L:=[[0,5,5],[9,4,2],[2,3,7]]

Ensuite, on peut transformer cette liste de listes en matrice :

> `C:=convert(L,array);`

$$\text{C:=} \begin{bmatrix} 0 & 5 & 5 \\ 9 & 4 & 2 \\ 2 & 3 & 7 \end{bmatrix}$$

> `type(L,listlist),whattype(evalm(C));`

true,array

L'opération inverse est bien sûr possible :

> `convert(C,listlist);`

[[0,5,5],[9,4,2],[2,3,7]]

Une matrice peut aussi être convertie en ensemble[1] comme le montre l'exemple suivant :

> `convert(C,set);`

$$\{0, 2, 3, 4, 5, 7, 9\}$$

L'opération inverse n'est en revanche pas possible.

[1] Une matrice ne peut pas être convertie en liste, mais seulement en liste de listes et en ensemble.

2 Fonctions de package `linalg`

Comme nous l'avons déjà dit, le package `linalg` est composé de très nombreuses fonctions qui permettent des manipulations plus évoluées sur les matrices, comme on va le voir tout au long de cette section.

Le package `linalg` permet de réaliser, avec d'autres fonctions que celles données précédement, la somme (`matadd`), le produit par un scalaire (`scalarmul`) ou encore le produit de deux matrices (`multiply`). Ces fonctions ne présentent pas d'intérêt particulier. Elles peuvent cependant être utiles dans le cas où on a oublié les commandes données précédement car celles-ci s'affichent lors du chargement du package `with(linalg)` et peuvent donc être facilement retrouvées.

2.1 Opérations sur les matrices

Le tableau suivant regroupe les différentes commandes pour calculer la dimension, la trace, le déterminant, la transposé, l'inverse, le rang d'une matrice.

`>rowdim(A)`	Donne le nombre de lignes de A.
`>coldim(A)`	Donne le nombre de colonnes de A.
`>trace(A)`	Donne la trace de la matrice A.
`>det(A)`	Donne le déterminant de la matrice A.
`>transpose(A)`	Donne la transposé de la matrice A.
`>inverse(A)`	Calcule l'inverse de A si elle est inversible.
`>rank(A)`	Donne le rang de la matrice A.

Remarquons que lorsque la matrice n'est pas inversible, Maple renvoie un message d'erreur :

> `inverse(A);`

```
Error, (in linalg:-inverse_Main:-MatrixInverse) singular matrix
```

2.2 Opérations sur les lignes et les colonnes d'une matrice

On s'intéresse ici plus particulièrement aux différentes opérations possibles sur les lignes et les colonnes d'une matrice. Les fonctions `mulrow`, `swaprow` et `addrow` (respectivement `mulcol`, `swapcol` et `addcol`) permettent d'éffectuer des opérations élémentaires sur les lignes (respectivement sur les colonnes) d'une matrice. Intéressons-nous aux opérations sur les lignes (on pourra bien sûr adapter ce qui suit pour les colonnes).

À l'aide de la fonction `mulrow`, on peut multiplier une ligne par un scalaire donné. Par exemple, on peut multiplier par 5 la dernière ligne de A:

> `mulrow(A,3,5);`

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 10 & 10 & 10 \end{bmatrix}$$

Pour échanger deux lignes, on utilise la fonction `swaprow`:

> `swaprow(A,2,3);`

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

Enfin, on peut ajouter à une ligne une autre ligne multiplier par une constante avec la fonction `addrow`. Par exemple, si on veut soustraire à la 3ème ligne de A 2 fois la 2ème ligne:

> `addrow(A,2,3,-2);`

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

Pour terminer, la fonction `pivot` effectue le pivot sur une matrice à partir du coefficient donné, et la fonction `gausselim` permet de réaliser le pivot de Gauss sur une matrice donnée:

> `pivot(B,2,1),gausselim(B);`

$$\begin{bmatrix} 0 & \frac{3}{4} & \frac{3}{2} \\ 4 & 5 & 6 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \end{bmatrix}, \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}$$

2.3 Tests sur les matrices

Le package `linalg` offre quelques fonctions qui permettent d'effectuer des tests sur les matrices. On peut notamment vérifier si une matrice est orthogonale (à l'aide de la fonction `orthog`) ou si deux matrices sont semblables (à l'aide de la fonction `issimilar`):

```
> orthog(A),issimilar(A,B);
```

false,false

Exercice 1 Soit f l'endomorphisme de $\mathbb{R}^3$ canoniquement associé à la matrice A définie par:

$$A := \frac{1}{10}\begin{bmatrix} 9 & 0 & -3 \\ 0 & 10 & 0 \\ -3 & 0 & 1 \end{bmatrix}$$

1. Montrer que f est un projecteur. Calculer le rang de f. En déduire à quelle matrice A doit-elle être semblable. Vérifier le résultat à l'aide de la fonction `issimilar`.
2. Calculer le noyau et l'image de f (on pourra consulter l'aide sur les fonctions `kernel` et `colspace`).

Exercice 2 On cherche à résoudre pour les matrices suivantes le problème `A&*X-X&*A=B`:

```
A:=matrix(2,2,[1,0,1,0]); B:=matrix(2,2,[-1,1,0,1]); X:=matrix(2,2);
```

La fonction `solve` ne marche pas. Il faut résoudre le système composé des quatre équations données par les quatre coefficients de la matrice. On utilise les instructions suivantes:

```
> equats:={};
  M:=A&*X-X&*A-B;
  for j from 1 to coldim(A) do
  for i from 1 to rowdim(A) do
  equats:=equats union { evalm(M)[i,j]=0 };
  od;
  od;
  solve(equats);
```

On vérifie réciproquement que les solutions trouvées sont effectivement solutions du problème:

```
> X[1,2]:=1; X[2,1]:=X[1,1]-X[2,2]; evalm(A&*X-X&*A-B);
```

1. Vérifier les calculs précédents.
2. Déterminer l'ensemble des matrices qui commutent avec la matrice A définie précédemment.
3. Résoudre dans $\mathcal{M}_2(\mathbb{R})$ l'équation matricielle: $X^2 - \frac{3}{2}X = A$.

Exercice 3 En utilisant les commandes `mulrow`, `swaprow`, `addrow`, `mulcol`, `swapcol` et `addcol`, appliquer l'algorithme du pivot de Gauss à la matrice:

$$\begin{bmatrix} 1 & 8 & -1 & 0 & 2 \\ 0 & 2 & -1 & 4 & 2 \\ -1 & 0 & 0 & 1 & 1 \\ 1 & 2 & 2 & 2 & 0 \end{bmatrix}$$

Vérifier vos calculs à l'aide de la commande `gausselim`.

Exercice 4 On dit qu'une matrice $M \in \mathcal{M}_n(\mathbb{R})$ est magique si les sommes des termes d'une même ligne ou d'une même colonne ont la même valeur.

1. Écrire une procédure `estmagique` qui teste si une matrice est magique (et qui renvoie *true* ou *false*).
2. Tester cette procédure sur les matrices suivantes:

$$\begin{bmatrix} 10 & 3 & 1 \\ 0 & 8 & 6 \\ 4 & 3 & 7 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \begin{bmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{bmatrix}, \quad \begin{bmatrix} 1 & 5 & 6 \\ 4 & 4 & 4 \\ 7 & 2 & 3 \end{bmatrix}$$