

Algebre lineaire

┌ L'ensemble des fonctions traitant de l'algebre lineaire est regroupé dans un package Maple qui s'appelle linalg et qu'il est utile de charger dès le début

```
> with(linalg):  
Warning, new definition for norm  
Warning, new definition for trace
```

┌ Espaces vectoriels

┌ Description, base et dimension d'un espace vectoriel

┌ Les espaces vectoriels considérés par Maple sont toujours des sous espaces de C^n , ce qui n'est pas une limitation bien gênante. Un tel sous espace vectoriel est entré à l'aide d'une famille génératrice qui peut être un vecteur, une liste ou un ensemble de vecteurs, ou une matrice (attention à la distinction faite par Maple entre listes et vecteurs, il faut convertir chaque liste en vecteur avant de pouvoir travailler):

```
> V:=map(vector,[[7, -15, 5, 13, -10],[7, -8, 3, 5, -9],[3, -8, 4, 10, -14], [4, 0, -1, -5, 5], [-7, 1, -1, 3, 8],[3, 3, -9, -6, 3] ]);  
V := [[7, -15, 5, 13, -10], [7, -8, 3, 5, -9], [3, -8, 4, 10, -14], [4, 0, -1, -5, 5], [-7, 1, -1, 3, 8], [3, 3, -9, -6, 3]]
```

┌ On en extrait une base à l'aide de la fonction basis

```
> basis(V);  
[[7, -15, 5, 13, -10], [7, -8, 3, 5, -9], [3, -8, 4, 10, -14], [3, 3, -9, -6, 3]]
```

┌ Maple ne dispose pas d'une fonction donnant la dimension d'un sous-espace vectoriel. Mais il est facile d'en écrire une:

```
> dimension:=X->nops(basis(X)):  
dimension(V);
```

4

┌ Somme et intersection

┌ Maple dispose de deux fonctions qui calculent (une base de) la somme et (une base de) l'intersection de deux sous espaces vectoriels de C^n , les fonctions sumbasis et intbasis.

```
> V:=map(vector,[[7, -15, 5, 13, -10],[7, -8, 3, 5, -9],[3, -8, 4, 10, -14]]);  
W:=map(vector,[[4, 0, -1, -5, 5], [-7, 1, -1, 3, 8],[3, 3, -9, -6, 3] ]);  
V := [[7, -15, 5, 13, -10], [7, -8, 3, 5, -9], [3, -8, 4, 10, -14]]  
W := [[4, 0, -1, -5, 5], [-7, 1, -1, 3, 8], [3, 3, -9, -6, 3]]
```

```
> intbasis(V,W);  
{[7, -1, 1, -3, -8], [4, 0, -1, -5, 5]}
```

```
> sumbasis(V,W);  
[[7, -15, 5, 13, -10], [7, -8, 3, 5, -9], [3, -8, 4, 10, -14], [3, 3, -9, -6, 3]]
```

┌ On vérifie au passage que la somme des dimensions des deux sous espaces est égale à la somme de la dimension de leur intersection et de la

dimension de leur somme. Il est alors facile de tester l'inclusion d'un sous espace vectoriel dans un autre en comparant la dimension de la somme des deux sous espaces à celle du plus grand

```
> est_inclus:=(X,Y)->evalb(nops(sumbasis(X,Y))=nops(basis(Y))):
V1:=map(vector,[[7,-1,1,-3,-8],[4,0,-1,-5,5]]):
est_inclus(V1,V);
est_inclus(W,V1);
```

true

false

De meme on testera facilement si deux sous espaces sont en somme directe

```
> est_somme_directe:= (X,Y) -> evalb(nops(intbasis(X,Y))=0):
est_somme_directe(V,W);
```

false

Les matrices

La représentation des matrices

Les matrices ne sont malheureusement pas représentées par Maple comme des listes de listes, mais comme des tables (une structure particulière de Maple). Heureusement la fonction matrix prend en argument une liste de liste et la transforme en matrice:

```
> A:=matrix([[1,2,3],[4,5,6]]);
```

$$A := \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

Comme toutes les tables en Maple, les matrices sont sujettes à une évaluation particulière. L'évaluation naturelle d'une matrice n'est autre que son nom

```
> A;
```

A

Si l'on veut accéder vraiment à la matrice mathématique, il faut forcer l'évaluation à l'aide de la fonction eval

```
> eval(A);
```

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

```
> A;
```

A

L'avantage de cette représentation est que les éléments d'une matrice sont directement modifiables par affectation.

```
> A[1,1]:=123456; eval(A);
```

$$A_{1,1} := 123456$$

$$\begin{array}{ccc} 123456 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

Attention : après une affectation qui travaille sur les noms des matrices, la matrice à gauche du symbole := d'affectation devient physiquement égale à celle de droite. Toute modification de l'une se répercute sur l'autre:

```
> B:=A: B[1,1]:=1997: eval(A);
```

$$\begin{matrix} 1997 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

Si vous voulez éviter ce phénomène, il faut écrire $B := \text{copy}(A)$

```
> B:=copy(A); B[1,2]:=2000: eval(A),eval(B);
```

$$B := \begin{matrix} 1997 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

$$\begin{matrix} 1997 & 2 & 3 & 1997 & 2000 & 3 \\ 4 & 5 & 6 & 4 & 5 & 6 \end{matrix}$$

On peut également créer des matrices formelles à l'aide de la fonction array. L'avantage est que l'on peut alors définir des comportements par défaut des coefficients; on peut par exemple imposer à la matrice d'être symétrique, diagonale, creuse (les coefficients non définis, représentés sinon par un point d'interrogation indexé, valent 0) ou l'identité.

```
> A:=array(1..3,1..3): A[1,2]:=1: A[3,2]:=4: eval(A);
```

$$\begin{matrix} ?_{1,1} & 1 & ?_{1,3} \\ ?_{2,1} & ?_{2,2} & ?_{2,3} \\ ?_{3,1} & 4 & ?_{3,3} \end{matrix}$$

```
> A:=array(1..3,1..3,symmetric): A[1,2]:=1: A[3,2]:=4: eval(A);
```

$$\begin{matrix} ?_{1,1} & 1 & ?_{1,3} \\ 1 & ?_{2,2} & 4 \\ ?_{1,3} & 4 & ?_{3,3} \end{matrix}$$

```
> A:=array(1..3,1..3,diagonal): A[1,2]:=1: A[3,2]:=4: eval(A);
```

Error, cannot assign to the off-diagonal elements of a diagonal array
Error, cannot assign to the off-diagonal elements of a diagonal array

$$\begin{matrix} ?_{1,1} & 0 & 0 \\ 0 & ?_{2,2} & 0 \\ 0 & 0 & ?_{3,3} \end{matrix}$$

```
> A:=array(1..3,1..3,sparse): A[1,2]:=1: A[3,2]:=4: eval(A);
```

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 4 & 0 \end{matrix}$$

```
> A:=array(1..3,1..3,identity): eval(A);
```

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Construction de matrices

Maple peut construire des matrices en empilant des matrices ou des vecteurs, soit horizontalement,

```
> v1:=[1,2,3]: v2:=[4,5,6]: v3:=[7,8,9]:  
v4:=[10,11,12]:  
stack(v1,v2,v3,v4);
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

soit horizontalement

```
> concat(v1,v2,v3,v4);
```

$$\begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

et avec une combinaison des deux

```
> concat(stack(v1,v2),stack(v3,v4));
```

$$\begin{pmatrix} 1 & 2 & 3 & 7 & 8 & 9 \\ 4 & 5 & 6 & 10 & 11 & 12 \end{pmatrix}$$

On peut également construire des matrices par blocs

```
> A:=concat(v1,v2); B:=concat(v3,v4);  
blockmatrix(2,2,A,B,B,A);
```

$$\begin{pmatrix} A & B \\ B & A \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} & \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} \\ \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \end{pmatrix}$$

```

1  4  7 10
2  5  8 11
3  6  9 12
7 10  1  4
8 11  2  5
9 12  3  6

```

[ou encore par blocs diagonaux

```

> A:=matrix([[1,1],[0,1]]); B:=matrix([[3,1],[0,3]]);
  diag(A,B);

```

```

A := 1  1
      0  1

```

```

B := 3  1
      0  3

```

```

1  1  0  0
0  1  0  0
0  0  3  1
0  0  0  3

```

[Si vous désirez des matrices au hasard pour faire des tests, la fonction **randmatrix** permet de construire des matrices aléatoires

```

> A:=randmatrix(3,4,entries=rand(-7..7));

```

```

A := -1 -2  5  1
      -6  1 -2  1
      -6 -3  3 -4

```

[si vous la voulez triangulaire avec des 1 sur la diagonale, ajoutez l'option **unimodular**

```

> A:=randmatrix(3,4,entries=rand(0..12),unimodular);

```

```

A := 1  2  2 12
      0  1  5  1
      0  0  1  1

```

[Si vous désirez une matrice inversible à coefficients entiers dont l'inverse est aussi à coefficients entiers, il suffit de la choisir de déterminant 1; on prendra par exemple le produit d'une matrice triangulaire inférieure avec des 1 sur la diagonale par une matrice triangulaire supérieure avec des 1 sur la diagonale

```

> A:=evalm(transpose(randmatrix(3,3,unimodular))*randmatrix(3,3,unimodular)); inverse(A);

```

```

A := 1  92  43
      45 4141 1873
      -8 -829 5423

```

$$\begin{bmatrix} 24009360 & -534563 & -5747 \\ -259019 & 5767 & 62 \\ -4177 & 93 & 1 \end{bmatrix}$$

Opérations matricielles

Maple peut effectuer les opérations habituelles sur les matrices: addition, soustraction, multiplication par un scalaire, multiplication matricielle (notée $A \&* B$ car le symbole $*$ est réservée à une multiplication commutative), puissance et inversion (noté $A^{(-1)}$). Cependant, contrairement aux opérations algébriques habituelles, ces opérations ne sont pas effectuées directement mais seulement à la demande à l'aide de la fonction **evalm**.

```
> A:=matrix([[2,3,4],[4,5,0],[7,8,9]]):
B:=array(1..3,1..3,symmetric):
B[1,1]:=0: B[1,2]:=a: B[1,3]:=a^2: B[2,2]:=1: B[2,3]:=a: B[3,3]:=1:
2*A+B;
```

$$2A + B$$

```
> evalm(2*A+B);
```

$$\begin{bmatrix} 4 & 6+a & 8+a^2 \\ 8+a & 11 & a \\ 14+a^2 & 16+a & 19 \end{bmatrix}$$

```
> A&*B;
```

$$A \&* B$$

```
> evalm(A &* B);
```

$$\begin{bmatrix} 3a+4a^2 & 6a+3 & 2a^2+3a+4 \\ 5a & 4a+5 & 4a^2+5a \\ 8a+9a^2 & 16a+8 & 7a^2+8a+9 \end{bmatrix}$$

```
> A^(-1);
```

$$\frac{1}{A}$$

```
> evalm(A^(-1));
```

$$\begin{bmatrix} -\frac{3}{2} & -\frac{1}{6} & \frac{2}{3} \\ \frac{6}{5} & \frac{1}{3} & -\frac{8}{15} \\ \frac{1}{10} & -\frac{1}{6} & \frac{1}{15} \end{bmatrix}$$

Fonctions matricielles classiques

On dispose des fonctions matricielles élémentaires comme la trace et le déterminant

```
> B:=array(1..3,1..3,symmetric):
    B[1,1]:=1: B[1,2]:=a: B[1,3]:=a^2: B[2,2]:=1: B[2,3]:=a: B[3,3]:=1:
    trace(B);
```

3

Le rang est obtenu avec la fonction **rank**, le noyau à l'aide de la fonction **kernel**, l'image à l'aide de la fonction **colspace**

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

2

```
> kernel(A);
```

{ [1, -2, 1] }

```
> colspace(A);
```

{ 1, 0, $\frac{-3}{2}$, 0, 1, $\frac{5}{2}$ }

```
> det(B);
```

$1 - 2a^2 + a^4$

La fonction **adjoint** renvoie la transposée de la comatrice

```
> A:=matrix([[a,b],[c,d]]); adjoint(A);
```

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

$$\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

alors que la fonction **minor** renvoie une matrice de cofacteur obtenue en supprimant une ligne et une colonne

```
> A:=matrix([[a1,b1,c1],[a2,b2,c2],[a3,b3,c3]]):
    minor(A,2,1);
```

$$\begin{pmatrix} b1 & c1 \\ b3 & c3 \end{pmatrix}$$

L'algorithme du pivot

La fonction **pivot** permet d'appliquer pas à pas l'algorithme du pivot. L'appel **pivot(A, i, j)** élimine les termes de la colonne j d'indices différents de i , en se servant de $A_{i,j}$ comme pivot. L'appel **pivot(A, i, j, r .. s)** élimine les termes de la colonne j d'indices variant de r à s différents de i , en se servant de $A_{i,j}$ comme pivot.

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

```
> pivot(",2,2,2..3);
```

$$\begin{pmatrix} 1 & a & a^2 \\ 0 & 1 - a^2 & a - a^3 \\ 0 & a - a^3 & -a^4 + 1 \end{pmatrix}$$

```
> pivot(",3,3,1..2);
```

$$\begin{pmatrix} 1 & a & a^2 \\ 0 & 1 - a^2 & a - a^3 \\ 0 & 0 & 1 - a^2 \end{pmatrix}$$

```
> pivot(",2,2,1..1);
```

$$\begin{pmatrix} 1 & a & 0 \\ 0 & 1 - a^2 & 0 \\ 0 & 0 & 1 - a^2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 - a^2 & 0 \\ 0 & 0 & 1 - a^2 \end{pmatrix}$$

La fonction gausselim permet d'effectuer l'algorithme du pivot partiel sur les lignes, jusqu'à obtenir une matrice triangulaire

```
> B:=array(1..3,1..3,symmetric):
  B[1,1]:=1: B[1,2]:=a: B[1,3]:=a^2: B[2,2]:=1: B[2,3]:=a: B[3,3]:=1:
  gausselim(B);
```

$$\begin{pmatrix} 1 & a & a^2 \\ 0 & 1 - a^2 & a - a^3 \\ 0 & 0 & 1 - a^2 \end{pmatrix}$$

Cours de mathématiques

par Denis Monasse

Ed. Vuibert

Table des matières

- Plan général
- Algèbre générale
- Algèbre linéaire
- Réduction des endomorphismes
- Topologie des espaces métriques
- Espaces vectoriels normés
- Comparaison des fonctions
- Suites et séries numériques
- Fonctions d'une variable réelle
- Intégration
- Suites et séries de fonctions

- Séries entières
- Formes quadratiques
- Formes hermitiennes
- Séries de Fourier
- Calcul différentiel
- Equations différentielles
- Espaces affines
- Courbes
- Surfaces
- Intégrales multiples
- Index