V. Ripoll,

*vripoll@dma.ens.fr*

# MK1 "Calcul formel" Maple
# TP4 : Algèbre linéaire

**http://www.dma.ens.fr/~vripoll/enseignement.html**

**But du TP4 :**
Nous allons utiliser Maple pour faire de l'algèbre linéaire dans $R^n$ : vecteurs, bases, matrices, systèmes linéaires,...

La plupart des commandes de Maple qui permettent de faire de l'algèbre linéaire se trouvent dans la bibliothèque ***linalg*** :

```
> restart: with(linalg);
```
```
Warning, the protected names norm and trace have been redefined and unprotected
```
(1)
```
[BlockDiagonal, GramSchmidt, JordanBlock, LUdecomp, QRdecomp, Wronskian,
   addcol, addrow, adj, adjoint, angle, augment, backsub, band, basis, bezout,
   blockmatrix, charmat, charpoly, cholesky, col, coldim, colspace, colspan,
   companion, concat, cond, copyinto, crossprod, curl, definite, delcols, delrows, det,
   diag, diverge, dotprod, eigenvals, eigenvalues, eigenvectors, eigenvects,
   entermatrix, equal, exponential, extend, ffgausselim, fibonacci, forwardsub,
   frobenius, gausselim, gaussjord, geneqns, genmatrix, grad, hadamard, hermite,
   hessian, hilbert, htranspose, ihermite, indexfunc, innerprod, intbasis, inverse,
   ismith, issimilar, iszero, jacobian, jordan, kernel, laplacian, leastsqrs, linsolve,
   matadd, matrix, minor, minpoly, mulcol, mulrow, multiply, norm, normalize,
   nullspace, orthog, permanent, pivot, potential, randmatrix, randvector, rank,
   ratform, row, rowdim, rowspace, rowspan, rref, scalarmul, singularvals, smith,
   stackmatrix, submatrix, subvector, sumbasis, swapcol, swaprow, sylvester,
   toeplitz, trace, transpose, vandermonde, vecpotent, vectdim, vector, wronskian]
```
Pensez à recharger la librairie après chaque *restart* !

## 1. Les vecteurs

### 1.1 Construction d'un vecteur

La commande *vector* permet de créer un vecteur. Voici quelques exemples :

```
> vector([1,4,6]);
  whattype(%);
```
$$[1 \quad 4 \quad 6]$$
$$array \tag{2}$$
```
> vector(5,i->i^2);
```
$$[1 \quad 4 \quad 9 \quad 16 \quad 25] \tag{3}$$
```
> randvector(4); # "rand" pour "random" (hasard)
```
$$[6 \quad 74 \quad 72 \quad 37] \tag{4}$$

Afficher le contenu d'un vecteur :

```
> v:=vector([7,8,9,10]);
```
$$v := [7 \quad 8 \quad 9 \quad 10] \tag{5}$$
```
> v;
```
$$v \tag{6}$$

Pour afficher à nouveau le vecteur contenu dans la variable *v*, on utilise la commande *evalm* (m pour *matrix*):

```
> evalm(v);
```
$$[7 \quad 8 \quad 9 \quad 10] \tag{7}$$

Extraire un coefficient d'un vecteur (comme pour les séquences et les listes)

```
> v[2]; # deuxième coefficient
```
$$8 \tag{8}$$

Modifier le contenu d'un vecteur (ce qu'on ne peut pas faire avec les séquences et les listes !) :
Par affectation des coefficients :

```
> v[1]:=0;
  evalm(v);
```
$$v_1 := 0$$
$$[0 \quad 8 \quad 9 \quad 10] \tag{9}$$

Copier un vecteur
De subtiles règles d'évaluation dans Maple amènent le problème suivant lorsqu'on souhaite copier un vecteur dans une autre variable :

```
> w:=v;
  v[1]:=2;
  evalm(v);evalm(w);
```
$$w := v$$
$$v_1 := 2$$
$$[2 \quad 8 \quad 9 \quad 10]$$
$$[2 \quad 8 \quad 9 \quad 10] \tag{10}$$

Toute modification de *v* entraine une modification de sa copie *w* ! Pour y remédier, on utilise la commande *copy* :

```
> w:=copy(v);
  v[1]:=3;
  evalm(v);evalm(w);
```
$$w := [2 \quad 8 \quad 9 \quad 10]$$
$$v_1 := 3$$
$$[3 \quad 8 \quad 9 \quad 10]$$
$$[2 \quad 8 \quad 9 \quad 10] \tag{11}$$

### 1.2 Quelques opérations sur les vecteurs

Norme d'un vecteur :

```
> norm(v,2);norm(w,2);
```
$$\sqrt{254}$$
$$\sqrt{249} \tag{12}$$

Somme de deux vecteurs :

```
> u:=vector([-7,4,7,3]);
```
$$u := [-7 \quad 4 \quad 7 \quad 3] \tag{13}$$

```
> matadd(u,v);
```
$$[-4 \quad 12 \quad 16 \quad 13] \tag{14}$$

ou bien :

```
> evalm(u+v);
```
$$[-4 \quad 12 \quad 16 \quad 13] \tag{15}$$

Multiplication d'un vecteur par un scalaire (réel) :

```
> scalarmul(v,Pi);
```
$$[3\pi \quad 8\pi \quad 9\pi \quad 10\pi] \tag{16}$$

ou bien :

```
> evalm(Pi*v);
```
$$[3\pi \quad 8\pi \quad 9\pi \quad 10\pi] \tag{17}$$

Taille d'un vecteur (nombre de composantes) :

```
> vectdim(u);
```
$$4 \tag{18}$$

Tester une égalité de deux vecteurs :

```
> equal(u,v);
```
$$\mathit{false} \tag{19}$$

Produit scalaire de deux vecteurs :

```
> dotprod(vector([1,0]),vector([0,1]));
```
$$0 \tag{20}$$

Produit vectoriel de deux vecteurs :

```
> crossprod(vector([1,1,0]),vector([1,0,1]));
```
$$[1 \quad -1 \quad -1] \tag{21}$$

### 1.3 Opérations avancées

Trouver une base du sous-espace vectoriel engendré par un *ensemble* de vecteurs :

```
> u:=vector([1,0,0]);
  v:=vector([1,1,0]);
  w:=vector([1,0,1]);
  t:=vector([0,0,2]);
```
$$u := [1 \quad 0 \quad 0]$$
$$v := [1 \quad 1 \quad 0]$$
$$w := [1 \quad 0 \quad 1]$$
$$t := [0 \quad 0 \quad 2] \tag{22}$$

```
> basis( { u,v,w,t } );
```
$$\{u, t, v\} \tag{23}$$

## 2. Les matrices

### 2.1 Construction d'une matrice

La commande *matrix* permet de créer une matrice. Voici quelques exemples :

```
> matrix([[1,2,3],[4,5,6],[7,8,9]]);
  whattype(%);
```
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
$$\mathit{array} \tag{24}$$

```
> matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{25}$$

```
> matrix(5,3,(i,j)->i+j);
```
$$\begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \\ 5 & 6 & 7 \\ 6 & 7 & 8 \end{bmatrix} \tag{26}$$

```
> A:=randmatrix(2,3);
```
$$A := \begin{bmatrix} -23 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix} \tag{27}$$

Il existe beaucoup d'autres fonctions permettant de créer des matrices. Consultez les commandes fournies dans *linalg* pour en savoir plus : par exemple *diag, band, blockmatrix, augment*,...

Afficher le contenu d'une matrice

```
> A;
```
$$A \tag{28}$$

Pour afficher la matrice contenue dans la variable *A*, on utilise la commande *evalm*, comme pour les vecteurs :

```
> evalm(A);
```
$$\begin{bmatrix} -23 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix} \tag{29}$$

Extraire un coefficient d'une matrice

```
> A[2,3]; # deuxième ligne, troisième colonne
```
$$-23 \tag{30}$$

Modifier une matrice

Par affectation des coefficients :

```
> A[1,1]:=Pi;
  evalm(A);
```
$$A_{1,1} := \pi$$
$$\begin{bmatrix} \pi & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix} \tag{31}$$

Copier une matrice

On a le meme problème d'évaluation qu'avec les vecteurs : voir le problème suivant lorsqu'on souhaite copier une matrice dans une autre variable :

```
> B:=A;
  A[1,1]:=1;
```

```
evalm(A);evalm(B);
```

$$B := A$$
$$A_{1,1} := 1$$
$$\begin{bmatrix} 1 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix} \quad (32)$$

Toute modification de $A$ entraine une modification de sa copie $B$ ! Pour y remédier, on utilise la commande *copy* :

```
> B:=copy(A);
  A[1,1]:=999;
  evalm(A);evalm(B);
```

$$B := \begin{bmatrix} 1 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix}$$
$$A_{1,1} := 999$$
$$\begin{bmatrix} 999 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 87 & 44 \\ 29 & 98 & -23 \end{bmatrix} \quad (33)$$

## 2.2 Quelques opérations sur les matrices

Somme de deux matrices :

```
> B:=matrix(2,3,[1$6]);
```

$$B := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (34)$$

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

$$\begin{bmatrix} 1000 & 88 & 45 \\ 30 & 99 & -22 \end{bmatrix} \quad (35)$$

ou bien :

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

$$\begin{bmatrix} 1000 & 88 & 45 \\ 30 & 99 & -22 \end{bmatrix} \quad (36)$$

Multiplication d'une matrice par un scalaire (réel) :

```
> scalarmul(A,x);
```

$$\begin{bmatrix} 999\,x & 87\,x & 44\,x \\ 29\,x & 98\,x & -23\,x \end{bmatrix} \quad (37)$$

ou bien :

```
> evalm(x*A);
```

$$\begin{bmatrix} 999\,x & 87\,x & 44\,x \\ 29\,x & 98\,x & -23\,x \end{bmatrix} \quad (38)$$

Multiplication de deux matrices (produit matriciel) :

```
> M:=matrix(2,2,[a1,b1,c1,d1]);
  N:=matrix(2,2,[a2,b2,c2,d2]);
```

$$M := \begin{bmatrix} a1 & b1 \\ c1 & d1 \end{bmatrix}$$
$$N := \begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix} \quad (39)$$

```
> multiply(M,N);
```

$$\begin{bmatrix} a1\,a2 + b1\,c2 & a1\,b2 + b1\,d2 \\ c1\,a2 + d1\,c2 & c1\,b2 + d1\,d2 \end{bmatrix} \quad (40)$$

ou bien :

```
> evalm(M&*N);
```

$$\begin{bmatrix} a1\,a2 + b1\,c2 & a1\,b2 + b1\,d2 \\ c1\,a2 + d1\,c2 & c1\,b2 + d1\,d2 \end{bmatrix} \quad (41)$$

Notez bien le **&***, qui désigne la multiplication des matrices ! À différencier de *, qui ne fonctionne pas ici :

```
> evalm(M*N);
Error, (in evalm/evaluate) use the &* operator for matrix/vector multiplication
```

Multiplication d'une matrice par un vecteur :

```
> multiply(A,vector([1,0,0]));
```

$$[999 \quad 29] \quad (42)$$

ou bien avec &* :

```
> evalm(A&*vector([1,0,0]));
```

$$[999 \quad 29] \quad (43)$$

Taille d'une matrice : nombre de lignes, nombres de colonnes

```
> rowdim(A) ; coldim(A);
```

$$2$$
$$3 \quad (44)$$

Tester une égalité de deux matrices :

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

$$false \quad (45)$$

## 2.3 Opérations avancées

Trace d'une matrice (somme des coefficients de la première diagonale) :

```
> trace(M);
```

$$a1 + d1 \quad (46)$$

Transposée d'une matrice :

```
> transpose(A);
```

$$\begin{bmatrix} 999 & 29 \\ 87 & 98 \\ 44 & -23 \end{bmatrix} \quad (47)$$

Calcul de l'inverse d'une matrice (carrée inversible) :

```
> A:=matrix(3,3,[7/10,3/5,1/10,-3/5,11/5,1/5,-3/10,3/5,11/10]):
  inverse(A);
```

$$\begin{bmatrix} \frac{23}{20} & \frac{-3}{10} & \frac{-1}{20} \\ \frac{3}{10} & \frac{2}{5} & \frac{-1}{10} \\ \frac{3}{20} & \frac{-3}{10} & \frac{19}{20} \end{bmatrix} \quad (48)$$

Calcul des valeurs propres d'une matrice carrée : (en anglais : *eigenvalues*, de l'allemand *eigen*)

```
> eigenvals(A);
```

$$2, 1, 1 \quad (49)$$

Calcul des vecteurs propres d'une matrice carrée : (en anglais : *eigenvectors*)

```
> eigenvects(A);
```

$$[2, 1, \{[1 \quad 2 \quad 1]\}], [1, 2, \{[1 \quad 0 \quad 3], [0 \quad 1 \quad -6]\}] \quad (50)$$

Que signifie la réponse de Maple ? Au besoin, consultez l'aide de Maple sur *linalg [eigenvects]*.

Trouver une base du noyau de l'application linéaire associée à une matrice :

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

$$\{\} \quad (51)$$

```
> B:=matrix(2,2,[1,2,1/2,1]);
```

$$B := \begin{bmatrix} 1 & 2 \\ \frac{1}{2} & 1 \end{bmatrix} \quad (52)$$

```
> kernel(B);
```

$$\{[-2 \quad 1]\} \quad (53)$$

Que signifient les réponses de Maple ?

Trouver une base de l'image de l'application linéaire associée à une matrice :
( = une base du sous-espace vectoriel engendré par les vecteurs colonnes de la matrice)

```
> colspace(B);
```

$$\left\{\begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}\right\} \quad (54)$$

Rang de la matrice (= dimension de l'image)

```
> rank(B);
```

$$1 \quad (55)$$

Déterminant d'une matrice carrée (on rappelle qu'une matrice carrée est inversible si et seulement si son déterminant est non nul) :

```
> det(A);
```

$$2 \quad (56)$$

## 3. Les systèmes linéaires

### 3.1 Systèmes linéaires donnés par des équations

On utilise *solve*. Par exemple, pour le système :

$$\begin{cases} x+y+z=3 \\ x+2y-z=-1 \\ -x+y+2z=0 \end{cases}$$

```
> solve( {x+y+z=3 , x+2*y-z=-1 , -x+y+2*z=0} , {x,y,z} );
```

$$\left\{y=\frac{-6}{7}, z=\frac{11}{7}, x=\frac{16}{7}\right\} \quad (57)$$

### 3.2 Systèmes linéaires donnés par une matrice

On utilise *linsolve*. Par exemple, le système précédent correspond à A.X=B, où A est la matrice :

```
> A:=matrix( [ [1,1,1], [1,2,-1], [-1,1,2] ] );
```

$$A := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ -1 & 1 & 2 \end{bmatrix} \quad (58)$$

et B le vecteur colonne :

```
> B:=vector([3,-1,0]);
```

$$B := [3 \quad -1 \quad 0] \quad (59)$$

On résout le système par :

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

$$\begin{bmatrix} \frac{16}{7} & \frac{-6}{7} & \frac{11}{7} \end{bmatrix} \quad (60)$$

### 3.3 Le pivot de Gauss

Maple peut effectuer le pivot de Gauss sur une matrice avec la commande *gausselim*. Le résultat est une matrice échelonnée mais non réduite. En utilisant cette commande avec *augment*, on peut résoudre des systèmes linéaires.

```
> C:=augment(A,B);
```

$$C := \begin{bmatrix} 1 & 1 & 1 & 3 \\ 1 & 2 & -1 & -1 \\ -1 & 1 & 2 & 0 \end{bmatrix} \quad (61)$$

*augment* crée une nouvelle matrice en mettant le vecteur colonne B à droite de A ( A | B) .

```
> gausselim(C);
```

$$\begin{bmatrix} 1 & 1 & 1 & 3 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & 7 & 11 \end{bmatrix} \quad (62)$$

Enfin, la commande *backsub*, appliquée à la matrice échelonnée, permet de trouver le (s) solution(s) :

```
> backsub(%);
```

$$\begin{bmatrix} \frac{16}{7} & \frac{-6}{7} & \frac{11}{7} \end{bmatrix} \quad (63)$$