

M.Albenque

albenque@liafa.jussieu.fr

MK1 "Calcul formel" Maple

TP4 : Algèbre linéaire

But du TP4 :

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

Et surtout, n'oubliez pas de vous (et de me) poser des questions !

La plupart des commandes de Maple qui permettent de faire de l'algèbre linéaire se trouvent dans la librairie *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, row space, 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 4 6]
array
```

(2)

```
> vector(5,i->i^2);
[1 4 9 16 25]
```

(3)

```
> randvector(4);
[-85 -55 -37 -35]
```

(4)

Afficher le contenu d'un vecteur :

```
> v:=vector([7,8,9,10]);
v:= [7 8 9 10]
```

(5)

```
> v;
v
```

(6)

Pour afficher à nouveau le vecteur contenu dans la variable v , on utilise la commande *evalm* :

```
> evalm(v);
[7 8 9 10]
```

(7)

Extraire un coefficient d'un vecteur :

```
> v[2]; # deuxième coefficient
8
```

(8)

Modifier le contenu d'un vecteur

Par affectation des coefficients :

```
> v[1]:=0;
evalm(v);
v1:=0
[0 8 9 10]
```

(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
v1:=2
[2 8 9 10]
[2 8 9 10]
```

(10)

Toute modification de v entraîne 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  8  9  10]
v1 := 3
[3  8  9  10]
[2  8  9  10]

```

(11)

1.2 Quelques opérations sur les vecteurs

Somme de deux vecteurs :

```

> u:=vector([-7,4,7,3]);
u := [-7  4  7  3]

```

(12)

```

> matadd(u,v);
[0  12  16  13]

```

(13)

ou bien :

```

> evalm(u+v);
[0  12  16  13]

```

(14)

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

```

> scalarmul(v,Pi);
[7 pi  8 pi  9 pi  10 pi]

```

(15)

ou bien :

```

> evalm(Pi*v);
[7 pi  8 pi  9 pi  10 pi]

```

(16)

Taille d'un vecteur (nombre de composantes) :

```

> vectdim(u);
4

```

(17)

Tester une égalité de deux vecteurs :

```

> equal(u,v);
false

```

(18)

Produit scalaire de deux vecteurs :

```

> dotprod(vector([1,0]),vector([0,1]));
0

```

(19)

Produit vectoriel de deux vecteurs :

```

> crossprod(vector([1,1,0]),vector([1,0,1]));
[1  -1  -1]

```

(20)

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  0  0]

```

(21)

```

v := [1 1 0 ]
w := [1 0 1 ]
t := [0 0 2 ]

```

```
> basis( { u,v,w,t } );
```

```
{v, u, t}
```

(22)

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(%);
```

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

(23)

array

```
> matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

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

(24)

```
> matrix(5,3,(i,j)->i+j);
```

$$\begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \\ 5 & 6 & 7 \\ 6 & 7 & 8 \end{bmatrix}$$

(25)

```
> A:=randmatrix(2,3);
```

$$A := \begin{bmatrix} -35 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

(26)

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
```

(27)

Pour afficher à nouveau la matrice contenue dans la variable *A*, on utilise la commande *evalm* :

```
> evalm(A);
```

(28)

$$\begin{bmatrix} -35 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix} \quad (28)$$

Extraire un coefficient d'une matrice

```
> A[2,3]; # deuxième ligne, troisième colonne
49
```

(29)

Modifier d'une matrice

Par affectation des coefficients :

```
> A[1,1]:=Pi;
evalm(A);
```

$$A_{1,1} := \pi$$

$$\begin{bmatrix} \pi & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

(30)

Copier une matrice

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

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

$$B := A$$

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

$$\begin{bmatrix} 1 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

(31)

Toute modification de A entraîne 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);
```

$$B := \begin{bmatrix} 1 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

$$A_{1,1} := 999$$

$$\begin{bmatrix} 999 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 97 & 50 \\ 79 & 56 & 49 \end{bmatrix}$$

(32)

2.2 Quelques opérations sur les matrices

Somme de deux matrices :

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

$$B := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (33)$$

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

$$\begin{bmatrix} 1000 & 98 & 51 \\ 80 & 57 & 50 \end{bmatrix} \quad (34)$$

ou bien :

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

$$\begin{bmatrix} 1000 & 98 & 51 \\ 80 & 57 & 50 \end{bmatrix} \quad (35)$$

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

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

$$\begin{bmatrix} 999x & 97x & 50x \\ 79x & 56x & 49x \end{bmatrix} \quad (36)$$

ou bien :

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

$$\begin{bmatrix} 999x & 97x & 50x \\ 79x & 56x & 49x \end{bmatrix} \quad (37)$$

Multiplication de deux matrices (produit matriciel) :

```
> M:=matrix(2,2,[a1,b1,c1,d1]);
```

```
  N:=matrix(2,2,[a2,b2,c2,d2]);
```

$$M := \begin{bmatrix} a1 & b1 \\ c1 & d1 \end{bmatrix} \quad (38)$$

$$N := \begin{bmatrix} a2 & b2 \\ c2 & d2 \end{bmatrix}$$

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

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

ou bien :

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

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

Notez bien le **&***, qui dénote 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 79]
```

(41)

ou bien avec &* :

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

```
[999 79]
```

(42)

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

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

```
2
```

(43)

```
3
```

Tester une égalité de deux matrices :

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

```
false
```

(44)

2.3 Opérations avancées

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

```
> trace(A);
```

```
5
```

(45)

Transposée d'une matrice :

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

```
[ 999  79 ]
 [  97  56 ]
 [  50  49 ]
```

(46)

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);
```

```
[ 23/20  -3/10  -1/20 ]
 [  3/10   2/5   -1/10 ]
 [  3/20  -3/10  19/20 ]
```

(47)

Calcul des valeurs propres d'une matrice carrée :

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

```
2, 1, 1
```

(48)

Calcul des vecteurs propres d'une matrice carrée :

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

$$[2, 1, \{ [1 \ 2 \ 1] \}], [1, 2, \{ [0 \ 1 \ -6], [1 \ 0 \ 3] \}] \quad (49)$$

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 :

$$\begin{aligned} > \text{kernel}(\mathbf{A}); \\ & \{ \} \end{aligned} \quad (50)$$

$$\begin{aligned} > \mathbf{B} := \text{matrix}(2, 2, [1, 2, 1/2, 1]); \\ & \mathbf{B} := \begin{bmatrix} 1 & 2 \\ \frac{1}{2} & 1 \end{bmatrix} \end{aligned} \quad (51)$$

$$\begin{aligned} > \text{kernel}(\mathbf{B}); \\ & \{ [-2 \ 1] \} \end{aligned} \quad (52)$$

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)

$$\begin{aligned} > \text{colspace}(\mathbf{B}); \\ & \left\{ \left[\begin{array}{c} 1 \\ \frac{1}{2} \end{array} \right] \right\} \end{aligned} \quad (53)$$

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

$$\begin{aligned} > \text{rank}(\mathbf{B}); \\ & 1 \end{aligned} \quad (54)$$

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

$$\begin{aligned} > \text{det}(\mathbf{A}); \\ & 2 \end{aligned} \quad (55)$$

3. Les systèmes linéaires

3.1 Systèmes linéaires donnés par des équations

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

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

$$\begin{aligned} > \text{solve}(\{ \mathbf{x} + \mathbf{y} + \mathbf{z} = 3, \mathbf{x} + 2 * \mathbf{y} - \mathbf{z} = -1, -\mathbf{x} + \mathbf{y} + 2 * \mathbf{z} = 0 \}, \{ \mathbf{x}, \mathbf{y}, \mathbf{z} \}); \\ & \left\{ x = \frac{16}{7}, z = \frac{11}{7}, y = \frac{-6}{7} \right\} \end{aligned} \quad (56)$$

3.2 Systèmes linéaires donnés par une matrice

On utilise *linsolve*. Par exemple, le système précédent correspond à $\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$, où \mathbf{A} est la matrice :

$$\begin{aligned} > \mathbf{A} := \text{matrix}(\left[\left[1, 1, 1 \right], \left[1, 2, -1 \right], \left[-1, 1, 2 \right] \right]); \\ & \end{aligned} \quad (57)$$

$$A := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & -1 \\ -1 & 1 & 2 \end{bmatrix} \quad (57)$$

et B le vecteur colonne :

$$\begin{aligned} > \mathbf{B} := \mathbf{vector}([3, -1, 0]); \\ B := [3 \quad -1 \quad 0] \end{aligned} \quad (58)$$

On résout le système par :

$$\begin{aligned} > \mathbf{linsolve}(A, B); \\ \left[\frac{16}{7} \quad \frac{-6}{7} \quad \frac{11}{7} \right] \end{aligned} \quad (59)$$

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.

$$\begin{aligned} > \mathbf{C} := \mathbf{augment}(A, B); \\ C := \begin{bmatrix} 1 & 1 & 1 & 3 \\ 1 & 2 & -1 & -1 \\ -1 & 1 & 2 & 0 \end{bmatrix} \end{aligned} \quad (60)$$

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

$$\begin{aligned} > \mathbf{gausselim}(C); \\ \begin{bmatrix} 1 & 1 & 1 & 3 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & 7 & 11 \end{bmatrix} \end{aligned} \quad (61)$$

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

$$\begin{aligned} > \mathbf{backsub}(\%); \\ \left[\frac{16}{7} \quad \frac{-6}{7} \quad \frac{11}{7} \right] \end{aligned} \quad (62)$$