C. Armana
armana@math.jussieu.fr

MK1 "Calcul formel" Maple

TP8 : Algèbre linéaire

Rappel : les documents de TP, sujets et corrections de devoir sont disponibles sur :
http://www.math.jussieu.fr/~armana/mk1

But du TP8 :

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
```

[*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(%);
```

$$[1, 4, 6]$$

array

```
> vector(5,i->i^2);
```

$$[1, 4, 9, 16, 25]$$

```
> randvector(4);
```

$$[-85, -55, -37, -35]$$

Afficher le contenu d'un vecteur :

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

$$v := [7, 8, 9, 10]$$

```
> v;
```

$$v$$

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

```
> evalm(v);
```

$$[7, 8, 9, 10]$$

Extraire un coefficient d'un vecteur :

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

$$8$$

Modifier le contenu d'un vecteur

Par affectation des coefficients :

```
> v[1]:=0;
  evalm(v);
```

$$v_1 := 0$$

$$[0, 8, 9, 10]$$

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);
```

$$w := v$$

$$v_1 := 2$$

$$[2, 8, 9, 10]$$

$$[2, 8, 9, 10]$$

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);
```

$$w := [2, 8, 9, 10]$$

$$v_1 := 3$$

$$[3, 8, 9, 10]$$

$$[2, 8, 9, 10]$$

1.2 Quelques opérations sur les vecteurs

Somme de deux vecteurs :

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

$$u := [-7, 4, 7, 3]$$

```
> matadd(u,v);
```

$$[0, 12, 16, 13]$$

ou bien :

```
> evalm(u+v);
```

$$[0, 12, 16, 13]$$

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

```
> scalarmul(v,Pi);
```

$$[7\,\pi, 8\,\pi, 9\,\pi, 10\,\pi]$$

ou bien :

```
> evalm(Pi*v);
```

$$[7\,\pi, 8\,\pi, 9\,\pi, 10\,\pi]$$

Taille d'un vecteur (nombre de composantes) :

```
> vectdim(u);
```

$$4$$

Tester une égalité de deux vecteurs :

```
> equal(u,v);
```

false

Produit scalaire de deux vecteurs :

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

$$0$$

Produit vectoriel de deux vecteurs :

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

$$[1, -1, -1]$$

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]);
```

$$u := [1, 0, 0]$$
$$v := [1, 1, 0]$$
$$w := [1, 0, 1]$$
$$t := [0, 0, 2]$$

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

$$\{v, u, t\}$$

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(%);
```

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

array

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

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

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

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

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

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

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;
```

$$A$$

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

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

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

Extraire un coefficient d'une matrice

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

$$49$$

Modifier d'une matrice

Par affectation des coefficients :

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

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

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

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);
```

$$B := A$$

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

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

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

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);
```

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

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

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

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

2.2 Quelques opérations sur les matrices

Somme de deux matrices :

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

$$B := \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

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

$$\begin{bmatrix} 1000 & 98 & 51 \\ 80 & 57 & 50 \end{bmatrix}$$

ou bien :

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

$$\begin{bmatrix} 1000 & 98 & 51 \\ 80 & 57 & 50 \end{bmatrix}$$

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

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

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

ou bien :

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

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

Multiplication de deux matrices (produit matriciel) :

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

$$M := \begin{bmatrix} a1 & b1 \\ c1 & d1 \end{bmatrix}$$

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

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

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

ou bien :

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

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

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]));
```

$$[999, 79]$$

ou bien avec &* :

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

$$[999, 79]$$

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

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

$$2$$

$$3$$

Tester une égalité de deux matrices :

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

$$false$$

2.3 Opérations avancées

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

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

$$5$$

Transposée d'une matrice :

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

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

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);
```

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

Calcul des valeurs propres d'une matrice carrée :

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

$$2, 1, 1$$

Calcul des vecteurs propres d'une matrice carrée :

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

$$[2, 1, \{[1, 2, 1]\}], [1, 2, \{[0, 1, -6], [1, 0, 3]\}]$$

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);
```

$$\{\ \}$$

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

$$B := \begin{bmatrix} 1 & 2 \\ \frac{1}{2} & 1 \end{bmatrix}$$

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

$$\{[-2, 1]\}$$

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);
```

$$\{\begin{bmatrix} 1, \frac{1}{2} \end{bmatrix}\}$$

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

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

$$1$$

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

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

$$2$$

3. Les systèmes linéaires

3.1 Systèmes linéaires donnés par des équations

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

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

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

$$\{x = \frac{16}{7}, z = \frac{11}{7}, y = \frac{-6}{7}\}$$

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] ] );
```

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

et B le vecteur colonne :

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

$$B := [3, -1, 0]$$

On résout le système par :

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

$$\begin{bmatrix} \frac{16}{7}, \frac{-6}{7}, \frac{11}{7} \end{bmatrix}$$

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);
```

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

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

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

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

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

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

$$\begin{bmatrix} \frac{16}{7}, \frac{-6}{7}, \frac{11}{7} \end{bmatrix}$$