TP Maple 5 | Algèbre linéaire, révisions de MPSI

Après quelques rappels sur l'algèbre linéaire de première année au moyen des librairies **LinearAlgebra** *et* **Linalg**, *nous exposerons la théorie et la pratique de la décomposition LU.*

1. La librairie LinearAlgebra

Maple met à disposition de l'utilisateur deux bibliothèques d'Algèbre linéaire : LINALG et LINEARALGEBRA. Cette dernière a été introduite depuis la version 7 du logiciel et vise à supplanter la précédente. Nous ne détaillerons ici que les commandes offertes par LINEARALGEBRA.

```
> with(LinearAlgebra);
```

[‘& x‘, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation, CrossProduct, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, LA Main, LUDecomposition, LeastSquares, LinearSolve, Map, Map2, MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, QRDecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip]

2. Calcul matriciel

Nous commencerons par le cas des vecteurs colonne ou ligne.

2.1. Calcul vectoriel

Le logiciel est capable d'effectuer du calcul vectoriel (en lignes ou en colonnes).

Définitions possibles d'un vecteur

Vecteurs colonnes : V := **Vector(**$[v_1,\ldots,v_n]$**)** Vecteurs lignes : V := $<v_1|v_2|\ldots|v_n>$

On pourra alors utiliser les commandes suivantes aussi bien sur des vecteurs colonnes que sur des vecteurs lignes (mais pas les deux à la fois !) :

```
> W1:=Vector([1,-2,1]): W2:=Vector([1,1,-2]): W3:=Vector([-2,1,1]):
  Basis([W1,W2,W3]);
```

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

```
> Dimension(W1);
```

$$3$$

```
> W1+W2, 3*W3;
```

$$\begin{bmatrix} 2 \\ -1 \\ -1 \end{bmatrix}, \begin{bmatrix} -6 \\ 3 \\ 3 \end{bmatrix}$$

Manipulation des vecteurs

- **Dimension** : nombre de composante(s) d'un vecteur.
- $W_1 + W_2$: somme des vecteurs W_1 et W_2.
- $\lambda * W$: action d'un scalaire sur un vecteur.
- **Basis(**$[W_1, W_2, \ldots, W_n]$**)** : retourne une base de vect$(W_1,\ldots,W_n)$.

2.2. Matrices de taille quelconque

Les matrices sont définies par la donnée de leurs lignes.

Définition d'une matrice ou d'un vecteur

► *Définition par la donnée des lignes :*

M := **Matrix(*L*)**

où L est la liste des lignes (données sous forme de listes) de la matrice M.

► *Définition par la donnée des coefficients :*

M := **Matrix(n,p,f)**

où f est une fonction de deux variables $(i,j) \mapsto f(i,j)$, n et p les dimensions de la matrice M de coefficients $f(i,j)$.

On trouvera ci-dessous cette syntaxe ainsi qu'un mode de définition plus ancien mais toujours valable.

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

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

```
> f:=(i,j)->i+j: Matrix(4,4,f);
```

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

On pourra aussi définir des matrices au moyen de vecteur-colonnes ou de vecteur-lignes [1].

```
> V:= Vector([1,20,1]);
```

$$\begin{bmatrix} 1 \\ 20 \\ 1 \end{bmatrix}$$

L'utilisateur a accès en lecture et en enregistrement aux coefficients de la matrice.

1. Ce point est important car il assure la validité des opérations décrites ci-dessous.

```
> A[1,2]:= 9: A[1,2],A;
```

$$9,\ \begin{bmatrix} 0 & 9 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

2.3. Opérations sur les matrices

Voici un petit panorama des opérations matricielles sous Maple. Les commandes essentielles et leur syntaxe sont consignées dans un tableau en fin de paragraphe.

```
> A:=Matrix([[0,1,2],[1,1,1]]): B:=Matrix([[5,4,0],[2,5,-1]]):
  C:=Matrix([[1,-1,2],[1,0,1],[1,1,2]]): V:= Vector([1,20,1]):
> A,B,C,V;
```

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix},\ \begin{bmatrix} 5 & 4 & 0 \\ 2 & 5 & -1 \end{bmatrix},\ \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix},\ \begin{bmatrix} 1 \\ 20 \\ 1 \end{bmatrix}$$

```
> Dimension(A), RowDimension(A), ColumnDimension(A);
```

$$2,3,2,3$$

```
> A+B, A.C, A.V, Transpose(A), C^2, C^(-1);
```

$$\begin{bmatrix} 5 & 5 & 2 \\ 3 & 6 & 0 \end{bmatrix},\ \begin{bmatrix} 3 & 2 & 5 \\ 3 & 0 & 3 \end{bmatrix},\ \begin{bmatrix} 22 \\ 22 \end{bmatrix},\ \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix},\ \begin{bmatrix} 2 & 1 & 5 \\ 2 & 0 & 4 \\ 4 & 1 & 7 \end{bmatrix},\ \begin{bmatrix} -1/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 \\ 1/2 & -1 & 1/2 \end{bmatrix}$$

Opérations sur les matrices

- **A+B** : calcule $A+B$.
- **A.B** : calcule AB.
- **Transpose(A)** : calcule ${}^tA$.
- **A^n** : calcule A^n ($n \in \mathbb{Z}$).
- **Trace** : calcule la trace.
- **Determinant** : calcule le déterminant.
- **Equal(A,B)** : teste si $A = B$.
- **A := Matrix(L)** : crée une matrice A dont la liste des lignes est L.
- **A[i,j]** : calcule le coefficient $A_{i,j}$ de la matrice A.
- **HermitianTranspose** : calcule la transconjuguée d'une matrice.

Opérations sur les matrices (suite)

- **DiagonalMatrix(d_1,...,d_n)** : matrice diagonale de coefficients $d_1,\dots,d_n$.
- **Dimension** : calcule les dimensions d'une matrice (lignes, colonnes).
- **RowDimension** : calcule le nombre de ligne(s) d'une matrice.
- **ColumnDimension** : calcule le nombre de colonne(s) d'une matrice.
- **DeleteColumn** : supprime des colonnes.
- **DeleteRow** : supprime des lignes.
- **SubMatrix(A,R,C)** : sous-matrice de A obtenue en extrayant les colonnes C, les lignes R.

Il est également important de savoir *former* l'ensemble des coefficients d'une matrice donnée. C'est la commande **convert** qui permet ce petit miracle. On s'en souviendra lors de la résolution d'équations d'inconnue matricielle telle que les calculs de commutants (et même certaines équations non-linéaires).

L'ensemble des coefficients d'une matrice

convert(A,set) renvoie l'ensemble dont les éléments sont les coefficients de la matrice A.

2.4. Systèmes linéaires

Deux pistes sont envisageables pour la résolution des systèmes linéaires : l'utilisation de **solve** (qui ne fait pas partie de la bibliothèque LINEARALGEBRA ou de la commande **LinearSolve** de LINEARALGEBRA. On recommande l'utilisation de cette dernière pour ces nombreuses options (voir l'aide en ligne).

La commande **solve** prend en arguments les lignes du système :

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

$$x=1, y=0, z=-1$$

La commande **LinearSolve** prend en arguments la matrice et le second membre du système :

```
> A:=Matrix([[1,1,1],[2,3,1],[1,1,-1]]):b:=Vector([0,1,2]):
  LinearSolve(A,b);
```

$$\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

Dans le cas d'un système de Cramer $AX = b$, on peut aussi calculer $A^{-1}.b$ mais cela est souvent maladroit car beaucoup plus coûteux en calculs.

Résolution d'un système linéaire

- **Solve(*lignes du système séparées par des virgules*)** : résout le système linéaire.
- **LinearSolve(A,b)** : résout le système linéaire $AX = b$.

Remarquons que la commande **LinearSolve** accepte des inconnues matricielles.

```
> A:=Matrix([[1,1,1],[2,3,1],[1,1,-1]]):
  b:=Matrix([[0,1,1],[0,0,0],[2,1,1]]): LinearSolve(A,b);
```

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

2.5. Noyau, image et rang

Maple dispose de commandes permettant le calcul du rang, du noyau et de l'image d'une matrice donnée. Rappelons que l'image d'une matrice, notée $\mathrm{Im}(A)$, est par définition l'espace engendré par ses vecteurs-colonnes ; le noyau, noté $\mathrm{Ker}(A)$, est l'espace de vecteurs colonnes X tels que $AX = 0$.

Manipulation des matrices

- **Rank** : calcule le rang de la matrice.
- **ColumnSpace** : calcule une base de l'image de la matrice.
- **NullSpace** : calcule une base du noyau.
- **RowSpace** : calcule une base de l'espace engendré par les lignes d'une matrice.

```
> A:=Matrix([[1,1,1],[2,3,1],[1,2,0]]): NullSpace(A), Rank(A);
```

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

```
> ColumnSpace(A), RowSpace(A);
```

$$\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \ \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}, \ [1,0,2], [0,1,-1]$$

Exercice 1.

Soit $A = \begin{pmatrix} 1 & -3 & 3 & -2 \\ 1 & 1 & 1 & 2 \\ 2 & 4 & 1 & 6 \\ 1 & 1 & 1 & 2 \end{pmatrix}$

1. Déterminer le rang de A ainsi qu'une base de $\mathrm{Im}(A)$.

2. Déterminer le noyau de A. En déduire des relations de liaisons entre les colonnes de A.

Exercice 2.

Trouver l'ensemble des matrices qui commutent avec $A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 0 & 3 \end{pmatrix}$

3. La factorisation LU

La factorisation LU consiste à écrire une matrice inversible A comme le produit de deux autres matrices L et U. L est une matrice triangulaire inférieure (L étant l'initiale de *low*) ayant des 1 sur la diagonale et U une matrice triangulaire supérieure (U pour *up*). Cette décomposition joue un rôle important en Analyse Numérique, ses applications sont innombrables et très utiles (par exemple la résolution des systèmes linéaires avec un moindre coût que la méthode d'élimination de Gauss-Jordan, un calcul efficace du déterminant d'une matrice carrée, etc).

Exercice 3.

Prouver l'unicité de la décomposition LU.

3.1. La théorie

Pour établir l'existence de la décomposition LU, on pourra appliquer la méthode d'élimination de Gauss-Jordan pour échelonner la matrice M de départ.

Proposition 1. (Existence de la décomposition LU)
Une matrice $M \in \mathfrak{M}_n(\mathbb{K})$ inversible admet une décomposition LU si et seulement si tous ses mineurs principaux sont non nuls.

Exercice 4.

Prouver la proposition 1.

3.2. L'algorithme de factorisation

Soit $M \in \mathfrak{M}_n(\mathbb{K})$ inversible admettant une décomposition $M = L \cdot U$. Notons $L = (\ell_{i,j})_{1 \leqslant i,j \leqslant n}$, $M = (m_{i,j})_{1 \leqslant i,j \leqslant n}$ et $U = (u_{i,j})_{1 \leqslant i,j \leqslant n}$. L'égalité $M = L \cdot U$ est équivalente à

$$\forall 1 \leqslant i,j \leqslant n, \quad m_{i,j} = \sum_{k=1}^{n} \ell_{i,k} \cdot u_{k,j}$$

mais puisque $\ell_{i,k} = 0$ pour $k > i$ et $u_{k,j} = 0$ pour $k > j$, on a

$$\forall 1 \leqslant i,j \leqslant n, \quad m_{i,j} = \sum_{k=1}^{\min(i,j)} \ell_{i,k} \cdot u_{k,j}$$

On en déduit que

$$\begin{cases} \text{si } i \leqslant j, & u_{i,j} = m_{i,j} - \displaystyle\sum_{k=1}^{i-1} \ell_{i,k} \cdot u_{k,j} \\ \text{si } i > j, & \ell_{i,j} = \dfrac{1}{u_{j,j}} \cdot \left(m_{i,j} - \displaystyle\sum_{k=1}^{j-1} \ell_{i,k} \cdot u_{k,j} \right) \end{cases}$$

Exercice 5.

En déduire un algorithme de calcul de la factorisation LU d'une matrice carrée M. Ecrire une procédure `LU` prenant en argument M et renvoyant sa factorisation sous la forme d'une liste $[L, U]$.