TP Maple 11 | Outils de calcul matriciel

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**.

La manipulation des matrices sous Maple pose souvent des problèmes aux néophytes. Il existe en effet plusieurs types permettant de faire du calcul matriciel et certaines fonctions usuelles (cf. `Determinant`, ect) n'acceptent qu'un seul type d'argument. Le logiciel utilise la fonction `rtable` pour construire les types `Vector`, `array` et `Matrix`. Il existe également un type `matrix` (attention, il faut le distinguer de `Matrix`) qu'il faut considérer comme obsolète depuis la version 7 et l'introduction de la bibliothèque **LinearAlgebra** [1].

1. C'est ce type que la bibliothèque **Linalg** utilise.

```
> 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]

Un conseil pour l'utilisation des matrices

Ne tranformez pas votre feuille de calcul en usine à gaz ! N'utilisez que le type `Matrix` de la bibliothèque **LinearAlgebra**.

2. Le type `Vector` de Maple

Le logiciel permet d'effectuer des calculs sur des vecteurs lignes ou colonnes au moyen du type `Vector` dont il existe deux spécifications : $\texttt{Vector}_{\texttt{column}}$ et $\texttt{Vector}_{\texttt{row}}$.

Définitions possibles d'un vecteur

Vecteurs colonnes : V := `Vector(`$[v_1,\ldots,v_n]$`)` Vecteurs lignes : V :=`Vector[row](`$[v_1,\ldots,v_n]$`)`

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 du type `Vector`

- `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`, ..., `W_n`])` : retourne une base de vect$(W_1, \ldots, W_n)$.

3. Le type `Matrix` de LinearAlgebra

Ce type supplante l'ancien type `matrix` depuis la version 7 du logiciel.

3.1. Définition d'une matrice

La commande `Matrix` permet de définir une matrice à partir de ses lignes ou de l'expression de ses coefficients.

Méthodes pour définir une matrice

- *Définition d'une matrice par une liste de lignes :*

```
M := Matrix(L)
```

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

- *Définition d'une matrice par une liste de colonnes :*

```
M := convert(L,Matrix)
```

où L est la liste des vecteurs colonnes de la matrice M (ces vecteurs peuvent être du type `Matrix` ou `Vector`).

- *Définition d'une matrice par l'expression générale de ses 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 $f(i,j)$.

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

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

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

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

La commande `piecewise` est très utile pour définir des matrices particulières. Elle permet de définir une fonction par morceaux. On retiendra la syntaxe suivante pour définir une fonction f valant f_i si la condition cond$_i$ est vérifiée (i variant de 1 à n) et valant f_{sinon} dans les autres cas[2].

La commande `piecewise` de définition « *par morceaux* »

```
f := x -> piecewise(cond1,f1,cond2,f2,...,condn,fn,fsinon)
```

2. La valeur par défaut de f_{sinon} est zéro.

Attention, la commande **piecewise** ne définit qu'une expression ! On retiendra également que les conditions $cond_i$ sont testées successivement dans l'ordre croissant des indices [3].

```
> f:=(i,j)->piecewise(i=j,1,i=1 or j=1,2,0): Matrix(3,3,f);
```

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

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

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

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

3.2. Matrices remarquables

Il existe des raccourcis pour définir certains types de matrices.

Matrices remarquables

► `ZeroMatrix(n,p)` : crée la matrice nulle de taille (n, p) mais le résultat n'est accessible qu'en lecture.

► `Matrix(n,p)` : crée la matrice nulle de taille (n, p), accessible en lecture et en enregistrement contrairement au cas pérédent.

► `IdentityMatrix(n)` : crée la matrice I_n mais le résultat n'est accessible qu'en lecture.

► `DiagonalMatrix([`$d_1,\dots,d_n$`])` : crée la matrice diagonale dont les coefficients diagonaux sont égaux à $d_1,\dots,d_n$. Le résultat est accessible en lecture et en enregistrement.

```
> M:=ZeroMatrix(2): M[1,1]:=1: M;
Error, attempt to assign non-zero to zero Matrix entry
> M:=Matrix(2): M[1,1]:=1: M;
```

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

3. Ce qui permet de définir une fonction par morceaux d'une autre manière qu'en mathématiques (où l'on utilise une partition de l'ensemble de définition pour définir une fonction par morceaux).

3.3. Matrices et blocs matriciels

On peut extraire et modifier des blocs d'une matrice donnée.

Extraction d'une sous-matrice

- La ligne de commande

```
SubMatrix(M,li..lf,ci..cf)
```

crée la matrice extraite de M définie par $li \leqslant i \leqslant lf$ et $ci \leqslant j \leqslant cf$.

- Si M est un objet du type *Matrix*,

```
M[li..lf,ci..cf]
```

désigne le bloc de M définie par $li \leqslant i \leqslant lf$ et $ci \leqslant j \leqslant cf$. Les lignes de commandes

```
A := M[li..lf,ci..cf]  et  M[li..lf,ci..cf] := B
```

ont pour effets respectifs d'enregistrer le bloc dans la variable A et de modifier le bloc de M en l'écrasant par une matrice B donnée dont la taille doit correspondre à celle du bloc sous peine d'un message d'erreur.

Ces outils permettent de définir des matrices par blocs sous Maple à partir de la matrice nulle. Voici comment procéder :

Définir une matrice par blocs

On commence par définir la matrice nulle et puis on la « *remplit* » par blocs :

```
M := Matrix(n,p)  puis  M[l1..l2,c1..c2] := A
```

Le bloc de la matrice M défini par $li \leqslant i \leqslant lf$ et $ci \leqslant j \leqslant cf$ est alors remplacé par A.

La commande `SubMatrix` admet une syntaxe plus générale permettant l'extraction d'un ensemble de lignes et de colonnes n'appartenant pas à un intervalle. Dans ce cas, on donne sous forme de listes les indices des lignes et des colonnes à extraire.

```
> M:=Matrix(4): A:=Matrix([[1,1],[1,2]]):
  M[1..2,1..2]:=A: M[3..4,1..2]:=A: M[3..4,3..4]:=A^(-1): A,M;
```

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

Attention au cas d'une extraction de ligne ou de colonne d'une matrice M donnée : on peut utiliser la ligne de commande `M[1..n,k]` pour extraire la k-ième colonne de M mais le résultat sera un objet du type `Vector`. En revanche, en écrivant `M[1..n,k..k]`, l'objet sera bel est bien du type `Matrix`.

Exercice 1.

Créer une procédure `proc(A,B)` créant la matrice

$$\left(\begin{array}{c|c} A & B \\ \hline B & A \end{array}\right)$$

où A et B sont deux matrices carrées de taille quelconque.

Exercice 2.

Créer une procédure `Mat(A,X,Y)` prenant en arguments deux matrices (de type `Matrix`) colonnes et une matrice carrée A ayant le même nombre de lignes et renvoyant la matrice

$$\begin{pmatrix} A & X \\ {}^tY & 0 \end{pmatrix}$$

4. Opérations sur les matrices (du type `Matrix`)

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 ($n \in \mathbb{Z}$).
- `A[i,j]` : calcule le coefficient $A_{i,j}$ de la matrice A.
- `HermitianTranspose` : calcule la transconjuguée d'une matrice.
- `Trace` : calcule la trace.
- `Determinant` : calcule le déterminant.
- `Equal(A,B)` : teste si $A = B$.

Opérations sur les matrices (suite)

- `DiagonalMatrix([`$d_1, \ldots, d_n$`])` : matrice diagonale de coefficients $d_1, \ldots, d_n$.
- `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.

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.

Exercice 3.

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.

5. 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);
```

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

Exercice 4.

Résoudre le système linéaire suivant,

$$\mathscr{S} \left\{ \begin{array}{rcrcrcl} x & + & 2y & + & 3z & = & 1 \\ -2x & + & 4y & + & z & = & 2 \\ x & + & 18y & + & 17z & = & 9 \end{array} \right.$$

Exercice 5.

Soit

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -2 & 4 & 4 & -2 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

1. Déterminer les matrices $B \in \mathfrak{M}_4(\mathbb{R})$ solutions de l'équation $AB = 0$.

2. Déterminer les matrices $M \in \mathfrak{M}_4(\mathbb{R})$ solutions de l'équation $AMA = A$.

Exercice 6.

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

6. 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 7.

Rang d'une suite de matrice

Pour tout $n \in \mathbb{N}^*$, on note A_n la matrice définie par

$$\forall (i,j) \in \{1,\dots,n\}^2, \quad (A_n)_{i,j} = (i+j)^3$$

1. Ecrire une procédure `A(n)` d'argument n renvoyant la matrice A_n.

2. Utiliser le logiciel pour conjecturer le rang de A_n en fonction de n.

3. Démontrer votre conjecture.

Exercice 8.

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 9.

Soient $n \in \mathbb{N}$, $E_n = \mathbb{R}_n[X]$ et T_n l'endomorphisme de E_n défini par

$$T_n(P) = (nX+1)\cdot P + (1-X^2)\cdot P'$$

1. Ecrire une procédure **MatriceT(n)** d'argument n calculant la matrice $M_n = \mathrm{mat}_{\mathscr{B}_n}(T_n)$ de T_n dans la base canonique $\mathscr{B}_n$ de E_n.

2. Ecrire deux procédures **BaseNoyau(n)** et **BaseImage(n)** renvoyant des bases de $\mathrm{Ker}(T_n)$ et de $\mathrm{Im}(T_n)$ pour un entier n quelconque.

3. Que trouve-t-on dans les cas $n = 3$ et $n = 4$?

7. 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).

7.1. 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.

On admettra provisoirement ce résultat.

7.2. 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 10.

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 $[L, U]$.