Vecteurs algébriques et matrices

© Pierre Lantagne
Enseignant retraité du Collège de Maisonneuve

Ce doument est une révision de celui produit en 2001 et 2004. L'objectif principal de ce document est de montrer au lecteur comment transposer à l'ordinateur le calcul matriciel. Cette transposition sera réalisée à l'aide des macro-commandes `Matrix` et `Vector` de la bibliothèque de base pour la saisie de matrices et de vecteurs. Le lecteur prendra connaissance également des raccourcis de saisies de vecteurs lignes <|> et colonnes <,>.

Bonne lecture à tous !

* Ce document Maple est exécutable avec la version 2020.2

Initialisation

Note: Nous n'allons pas d'entrée de jeu rendre disponibles les notations abrégées des macro-commandes de la bibliothèque `LinearAlgebra`. La raison est la suivante: c'est pour mettre l'emphase sur le fait que les vecteurs et les matrices sont créés avec les macro-commandes Matrix et Vector de la bibliothèque de base. En tant qu'objets créés par appel à la procédure rtable (Voir sur l'onglet *Autres documents* de mon site Internet le document *L'indexation liste, Array, table, rtable*) les opérations d'addition, de multiplication et de puissance d'expressions de type `Vector` et `Matrix` sont directement évaluées par Maple.

```
> restart;
  with(Physics[Vectors]):
```

Les vecteurs algébriques (Vector)

Les macro-commandes Matrix et Vector de la bibliothèque de base sont à considérer pour la création de matrices et de vecteurs. Il ne faudra pas utiliser ni la macro-commande `Matrix` ni la macro-commande `Vector`. Ces macro-commandes obsolètes créent bien sûr des matrices et des vecteurs mais leur mode de stockage (*hash tabl*e) rend incompatible leur utilisation avec les macro-commandes de la bibliothèque `LinearAlgebra` ainsi qu'avec les macro-commandes de plusieurs autres bibliothèques impliquant des calculs avec des vecteurs et des matrices.

Soit le vecteur ligne $\boldsymbol{v} = \langle -1, 2, -3, 12 \rangle$.

Avec la macro-commande `Vector` nous avons:

- une saisie *explicite* où il faut d'abord déclarer la dimension du vecteur suivi de la *liste* d'expressions composant le vecteur.

```
> U_:=Vector(4,[-1,3,-5,7]);
```

$$\vec{U} := \begin{bmatrix} -1 \\ 3 \\ -5 \\ 7 \end{bmatrix} \tag{2.1}$$

- une saisie *implicite* où il suffit de donner directement la *liste* d'expressions composant le vecteur. Dans ce cas, la dimension du vecteur correspond au nombre d'éléments de la *liste*.

```
> U_:=Vector([1,3,-5,7]);
```

$$\vec{U} := \begin{bmatrix} 1 \\ 3 \\ -5 \\ 7 \end{bmatrix} \tag{2.2}$$

On aura facilement remarqué que l'orientation par défaut d'un vecteur ainsi créé est en colonne. L'orientation ligne du vecteur doit être explicitement spécifié avec l'option `row`.

```
> V_:=Vector[row]([1,3,-5,7]);
```

$$\vec{V} := \begin{bmatrix} 1 & 3 & -5 & 7 \end{bmatrix} \tag{2.3}$$

Les opérations d'addition, de soustraction, de multiplication par un scalaire et de la multiplication scalaire de deux vecteurs de même orientation sont effectuées respectivements par le $(+)$, le $(-)$, l'étoile (*) et le point $(.)$. (Voir Algebra with Matrices, Vectors, and Arrays)

```
> U_:=Vector[row](4,symbol=u);
  V_:=Vector[row](4,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \tag{2.4}$$

```
> 'U_+V_'=U_%+V_;
  ``=U_+V_;
```

$$\vec{U} + \vec{V} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} + \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$= \begin{bmatrix} u_1 + v_1 & u_2 + v_2 & u_3 + v_3 & u_4 + v_4 \end{bmatrix} \tag{2.5}$$

```
> 'U_-V_'=U_%-V_;
  ``=U_-V_;
```

$$\vec{U} - \vec{V} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} - \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$= \begin{bmatrix} u_1 - v_1 & u_2 - v_2 & u_3 - v_3 & u_4 - v_4 \end{bmatrix} \tag{2.6}$$

```
> '5*V_'=5%*V_;
  ``=5*V_;
```

$$5\vec{V} = 5 * \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$= \begin{bmatrix} 5v_1 & 5v_2 & 5v_3 & 5v_4 \end{bmatrix} \tag{2.7}$$

```
> 'U_.V_'=U_%.V_;
  ``=U_.V_;
```

$$\vec{U} \cdot \vec{V} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

$$= \overline{v_1} u_1 + \overline{v_2} u_2 + \overline{v_3} u_3 + \overline{v_4} u_4 \tag{2.8}$$

La barre horizontale sur les composantes v_i du vecteur V correspond au conjuguée complexe de v_i. L'aide de

l'opérateur dot `.` donne les explications au résultat précédent. En fait, l'opérateur multiplication scalaire `dot` fait appel à la macro-commande `DotProduct` de la bibliothèque `LinearAlgebra` (`LinearAlgebra[DotProduct]`). Et pour cette macro-commande, les justifications de ce résultat sont très claires mais la compréhension de ces explications nécessite la connaissance de définitions qui débordent le contenu de notre cours.

Puisque la saisie respective des vecteurs U et V ont créé implicitement des variables mathématiques complexes pouvant être imaginaires, il nous faut donc explicitement signifier à Maple, au moment du calcul demandé, que ce sont des variables réelles. Ainsi, nous retrouverons un développement habituel du produit scalaire.

```
> 'U_.V_'=U_%.V_;
  ``=U_.V_ assuming real;
```

$$\vec{U} \cdot \vec{V} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} = v_1 u_1 + v_2 u_2 + v_3 u_3 + v_4 u_4 \quad \textbf{(2.9)}$$

```
> 'V_.U_'=V_%.U_;
  ``=V_.U_ assuming real;
```

$$\vec{V} \cdot \vec{U} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \cdot \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} = v_1 u_1 + v_2 u_2 + v_3 u_3 + v_4 u_4 \quad \textbf{(2.10)}$$

L'addition $(+)$ et la soustraction $(-)$ de deux vecteurs ne s'effectuent qu'à la condition que les deux vecteurs aient le **même nombre de composantes** et la **même orientation**.

Soit le vecteur $\mathrm{U} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$

```
> U_:=Vector(4,symbol=u);
```

$$\vec{U} := \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \quad \textbf{(2.11)}$$

```
> U_+V_;
Error, (in Physics:-Vectors:-+) a row and column Vector cannot be added together
```

La multiplication scalaire d'un vecteur ligne par un vecteur colonne est définie pourvue que ces deux vecteurs aient le **même nombre de composantes**.

```
> U_:=Vector[row]([1,3,-5,7]);
```

```
V_:=Vector([2,-4,6,8]);
```

$$\vec{U} := \begin{bmatrix} 1 & 3 & -5 & 7 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} 2 \\ -4 \\ 6 \\ 8 \end{bmatrix} \tag{2.12}$$

```
> 'U_.V_'=U_%.V_;
  ``=U_.V_;
```

$$\vec{U} \cdot \vec{V} = \begin{bmatrix} 1 & 3 & -5 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -4 \\ 6 \\ 8 \end{bmatrix}$$

$$= 16 \tag{2.13}$$

Remarque importante sur la multiplication scalaire opérée avec la notation point (.)

Pour Maple, l'opérateur de multiplication point (.) est un opérateur non commutatif possédant la propriété d'associativité à gauche. Son utilisation semble donc bizarre ici pour opérer la multiplication scalaire de vecteurs algébriques. D'une part, chacun sait que la multiplication scalaire de vecteurs est une opération commutative et, d'autre part, évoquer l'associativité de la multiplication scalaire est une ineptie...

Or, l'opérateur de multiplication points (.) est aussi utilisé pour la multiplication matricielle et nous savons que la multiplication matricielle ne possède pas la propriété de commutativité.

Voyons comment cela se passe si nous multiplions U par V.

```
> 'V_.U_'=V_%.U_;
  ``=V_.U_;
```

$$\vec{V} \cdot \vec{U} = \begin{bmatrix} 2 \\ -4 \\ 6 \\ 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & -5 & 7 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 6 & -10 & 14 \\ -4 & -12 & 20 & -28 \\ 6 & 18 & -30 & 42 \\ 8 & 24 & -40 & 56 \end{bmatrix} \tag{2.14}$$

Difficile d'expliquer ce dernier résultat sans évoquer la multiplication matricielle. Or, nous n'avons pas encore présenter la transposition en Maple d'une matrice. De plus, pour expliquer correctement ce dernier résultat, il faudrait aussi tenir compte du type d'objet Maple que sont les vecteurs (`Vector`). Les lecteurs intéressés à trouver des explications pourront consulter, sur mon site Internet, le document *L'indexation liste, Array, table, rtable* dans la section *Complément Maple.*

Quoiqu'il en soit, l'opérateur de multiplication points $(.)$ opère correctement la multiplication scalaire de deux vecteurs à la condition que les deux vecteurs aient la même orientation et le même nombre de composantes réelles.

```
> U_:=Vector([1,3,-5,7]);
  V_:=Vector([2,-4,6,8]);
  'U_.V_'=U_.V_;
  'V_.U_'=V_.U_;
```

$$\vec{U} := \begin{bmatrix} 1 \\ 3 \\ -5 \\ 7 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} 2 \\ -4 \\ 6 \\ 8 \end{bmatrix}$$

$$\vec{U} \cdot \vec{V} = 16$$

$$\vec{V} \cdot \vec{U} = 16 \qquad \textbf{(2.15)}$$

```
> U_:=Vector[row]([1,3,-5,7]);
  V_:=Vector[row]([2,-4,6,8]);
  'U_.V_'=U_.V_;
  'V_.U_'=V_.U_;
```

$$\vec{U} := \begin{bmatrix} 1 & 3 & -5 & 7 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} 2 & -4 & 6 & 8 \end{bmatrix}$$

$$\vec{U} \cdot \vec{V} = 16$$

$$\vec{V} \cdot \vec{U} = 16 \qquad \textbf{(2.16)}$$

La macro-commande `Vector` possède de nombreuses options facultatives. Par exemple, l'option permettant l'utilisation d'une fonction d'indexation pour générer ses éléments. Voyons deux exemples.

```
> f := j-> 2^(j-1);
  Vector(8,f);
```

$$f := j \mapsto 2^{j-1} \qquad \textbf{(2.17)}$$

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 8 \\ 16 \\ 32 \\ 64 \\ 128 \end{bmatrix} \tag{2.17}$$

```
> f := j-> (-1)^(j+1)*v[j];
  Vector[row](4,f);
```

$$f := j \mapsto (-1)^{j+1} \cdot v_j$$

$$\begin{bmatrix} v_1 & -v_2 & v_3 & -v_4 \end{bmatrix} \tag{2.18}$$

Terminons cette section en présentant la syntaxe raccourcie pour la saisie d'un vecteur algébrique (raccourcie très utile):

- le cas d'un vecteur ligne < | >
- le cas d'un vecteur colonne < , >

Soit les vecteurs $U = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$ et $V = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$.

```
> u_:=<1|2|3|4>;
  v_:=`<|>`(5,6,7,8);
```

$$\vec{u} := \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\vec{v} := \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix} \tag{2.19}$$

```
> u_:=<8,7,6,5>;
  v_:=`<,>`(8,7,6,5);
```

$$\vec{u} := \begin{bmatrix} 8 \\ 7 \\ 6 \\ 5 \end{bmatrix}$$

(2.20)

$$\vec{v} := \begin{bmatrix} 8 \\ 7 \\ 6 \\ 5 \end{bmatrix} \quad \textbf{(2.20)}$$

Comme nous le verrons dans la prochaine section, ces raccourcis s'avéreront très commodes pour la saisie de matrices.

```
> unassign('U_,V_,u_,v_'):
```

Les matrices (Matrix)

La macro-commande Matrix de la bibliothèque de base est à considérer pour la création de matrices.

Soit la matrice $A = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix}$.

Il y a trois syntaxes pour la saisie d'une matrice. Deux syntaxes avec la macro-commande `Matrix` et une autre utilisant les raccourcis systèmes < | > et < , > pour la création d'objets de type rtable.

Notons immédiatement, par la nature même des objets de type `rtable`, qu'il y a affichage automatique de leur contenu au moment de leur création.

Alors, avec la macro-commande `Matrix` nous avons:

– une saisie *explicite* où il faut d'abord déclarer le format de la matrice suivi
  – de la liste d'expressions composant les lignes de la matrice.
  – de la liste des listes d'expressions composant chaque ligne de la matrice.

```
> A:=Matrix(3,3,[1,1,2,2,4,-3,3,6,-5]);
  A:=Matrix(3,3,[[1,1,2],[2,4,-3],[3,6,-5]]);
```

$$A := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix}$$

$$A := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} \quad \textbf{(3.1)}$$

– une saisie *implicite* où il suffit de donner la liste des listes d'expressions composant chaque ligne de la matrice. Dans ce cas, le nombre de lignes de la matrice correspond au nombre d'éléments de la liste des listes et le nombre de colonnes correspondra au même nombre commun d'éléments de chaque liste de la liste.

```
> A:=Matrix([[1,1,2],[2,4,-3],[3,6,-5]]);
```

(3.2)

$$A := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} \quad \textbf{(3.2)}$$

Avec cette dernière saisie, il n'est donc pas obligatoire de déclarer explicitement le format de la matrice. Encore mieux, puisque Maple ignore complètement les espaces dans la zone de requêtes, nous suggérons la disposition suivante qui facilitera la vérification et la correction durant la saisie des éléments d'une matrice.

```
> A:=Matrix([
      [1,1, 2],
      [2,4,-3],
      [3,6,-5]]);
```

$$A := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & -3 \\ 3 & 6 & -5 \end{bmatrix} \quad \textbf{(3.3)}$$

Remarque: si le nombre d'éléments saisis ne correspond pas au format de la matrice, il y aura automatiquement remplissage des entrées manquantes par 0, remplissage ajusté au vecteur ayant la plus grande dimension.

```
> B:=Matrix([[1,1,2,12],
             [2,4,-3,3,-15],
             [3,6,-5]]);
```

$$B := \begin{bmatrix} 1 & 1 & 2 & 12 & 0 \\ 2 & 4 & -3 & 3 & -15 \\ 3 & 6 & -5 & 0 & 0 \end{bmatrix} \quad \textbf{(3.4)}$$

La troisième syntaxe de saisie d'une matrice est celle utilisant les raccourcis système ligne < | > et colonne < , >.

Ces raccourcis permettent de saisir une matrice :

– en tant que matrice ligne de vecteurs colonnes: A= $\left[C_1\, C_2\, C_3 \ldots C_n \right] = \left\langle C_1 , C_2 , C_3, \ldots , C_n \right\rangle$

– en tant que matrice colonne de vecteurs lignes: A= $\begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ \ldots \\ L_m \end{bmatrix} = \left\langle L_1 \middle| L_2 \middle| L_3 \middle| \ldots \middle| L_m \right\rangle$

Commençons par la saisie des vecteurs colonnes composant la matrice ligne A avec le raccourci colonne < , >.

```
> C1:=<1,2,3,4>;
  C2:=<5,6,7,8>;
  C3:=<9,10,11,12>;
```

$$C1 := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$C2 := \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

$$C3 := \begin{bmatrix} 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} \quad \textbf{(3.5)}$$

Créons ensuite la matrice A comme matrice ligne de vecteurs colonnes avec le raccourci ligne < | >.

```
> A:=<C1|C2|C3>;
```

$$A := \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \quad \textbf{(3.6)}$$

Nous pouvons très bien saisir directement la matrice A comme matrice ligne de vecteurs colonnes directement en utilisant simultanément les deux raccourcis.

```
> A:=< <1,2,3,4>|<5,6,7,8>|<9,10,11,12> >;
```

$$A := \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \quad \textbf{(3.7)}$$

Saisissons maintenant la matrice A comme matrice colonne de vecteurs lignes.

Saissons d'abord les vecteurs lignes composant les lignes de la matrice A avec le raccourci ligne < | >

```
> L1:=<1|5|9>;
  L2:=<2|6|10>;
  L3:=<3|7|11>;
  L4:=<4|8|12>;
```

$$L1 := \begin{bmatrix} 1 & 5 & 9 \end{bmatrix}$$

$$L2 := \begin{bmatrix} 2 & 6 & 10 \end{bmatrix}$$

$$L3 := \begin{bmatrix} 3 & 7 & 11 \end{bmatrix}$$

$$L4 := \begin{bmatrix} 4 & 8 & 12 \end{bmatrix} \quad \textbf{(3.8)}$$

Créons ensuite la matrice A comme matrice colonne de vecteurs lignes avec le raccourci colonne < , >.

```
> A:=<L1,L2,L3,L4>;
```

$$A := \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \quad \textbf{(3.9)}$$

Nous pouvons très bien également saisir la matrice A comme matrice colonne de vecteurs lignes directement en utilisant à la fois les deux raccourcis.

```
> A:=< <1|1|2>,
       <2|4|6>,
       <3|6|-5>,
       <4|8|12> >;
```

$$A := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 4 & 6 \\ 3 & 6 & -5 \\ 4 & 8 & 12 \end{bmatrix} \quad \textbf{(3.10)}$$

```
> unassign('A,C1,C2,C3,L1,L2,L3,L4'):
```

Les opérations d'addition, de soustraction, de multiplication par un scalaire, de la multiplication matricielle, de puissance et de transposition sont effectuées respectivements par le $(+)$, le $(-)$, l'étoile (*) , le point $(.)$, le (^) et le (^(%T). (Voir Algebra with Matrices, Vectors, and Arrays)

Soit les matices $A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix}, B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \end{bmatrix}$ et $C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \\ c_{4,1} & c_{4,2} & c_{4,3} \end{bmatrix}$

```
> A:=Matrix(3,4,symbol=a);
  B:=Matrix(3,4,symbol=b);
  C:=Matrix(4,3,symbol=c);
```

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix}$$

$$B := \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \end{bmatrix}$$

$$C := \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \\ c_{4,1} & c_{4,2} & c_{4,3} \end{bmatrix} \tag{3.11}$$

```
> 'A+B'=A%+B;
  ``=A+B;
```

$$A + B = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \end{bmatrix}$$

$$= \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & a_{1,3} + b_{1,3} & a_{1,4} + b_{1,4} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & a_{2,3} + b_{2,3} & a_{2,4} + b_{2,4} \\ a_{3,1} + b_{3,1} & a_{3,2} + b_{3,2} & a_{3,3} + b_{3,3} & a_{3,4} + b_{3,4} \end{bmatrix} \tag{3.12}$$

```
> 'A-B'=A%-B;
  ``=A-B;
```

$$A - B = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix} - \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \end{bmatrix}$$

$$= \begin{bmatrix} a_{1,1} - b_{1,1} & a_{1,2} - b_{1,2} & a_{1,3} - b_{1,3} & a_{1,4} - b_{1,4} \\ a_{2,1} - b_{2,1} & a_{2,2} - b_{2,2} & a_{2,3} - b_{2,3} & a_{2,4} - b_{2,4} \\ a_{3,1} - b_{3,1} & a_{3,2} - b_{3,2} & a_{3,3} - b_{3,3} & a_{3,4} - b_{3,4} \end{bmatrix} \tag{3.13}$$

```
> 'k*A'=k%*A;
  ``=k*A;
```

$$kA = k* \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix}$$

$$= \begin{bmatrix} ka_{1,1} & ka_{1,2} & ka_{1,3} & ka_{1,4} \\ ka_{2,1} & ka_{2,2} & ka_{2,3} & ka_{2,4} \\ ka_{3,1} & ka_{3,2} & ka_{3,3} & ka_{3,4} \end{bmatrix} \tag{3.14}$$

```
> 'A.C'=A%.C;
```

```
``=A.C;
ArrayDims(A.C);
```

$$A \cdot C = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix} \cdot \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \\ c_{4,1} & c_{4,2} & c_{4,3} \end{bmatrix}$$

$$= \left[\left[a_{1,1} c_{1,1} + a_{1,2} c_{2,1} + a_{1,3} c_{3,1} + a_{1,4} c_{4,1},\ a_{1,1} c_{1,2} + a_{1,2} c_{2,2} + a_{1,3} c_{3,2} + a_{1,4} c_{4,2},\ a_{1,1} c_{1,3} + a_{1,2} c_{2,3} + a_{1,3} c_{3,3} + a_{1,4} c_{4,3} \right], \left[a_{2,1} c_{1,1} + a_{2,2} c_{2,1} + a_{2,3} c_{3,1} + a_{2,4} c_{4,1},\ a_{2,1} c_{1,2} + a_{2,2} c_{2,2} + a_{2,3} c_{3,2} + a_{2,4} c_{4,2},\ a_{2,1} c_{1,3} + a_{2,2} c_{2,3} + a_{2,3} c_{3,3} + a_{2,4} c_{4,3} \right], \left[a_{3,1} c_{1,1} + a_{3,2} c_{2,1} + a_{3,3} c_{3,1} + a_{3,4} c_{4,1},\ a_{3,1} c_{1,2} + a_{3,2} c_{2,2} + a_{3,3} c_{3,2} + a_{3,4} c_{4,2},\ a_{3,1} c_{1,3} + a_{3,2} c_{2,3} + a_{3,3} c_{3,3} + a_{3,4} c_{4,3} \right] \right]$$

$$1..3, 1..3 \qquad \textbf{(3.15)}$$

```
> 'C.A'=C%.A;
``=C.A;
ArrayDims(C.A);;
```

$$C \cdot A = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \\ c_{4,1} & c_{4,2} & c_{4,3} \end{bmatrix} \cdot \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{bmatrix}$$

$$= \left[\left[c_{1,1} a_{1,1} + c_{1,2} a_{2,1} + c_{1,3} a_{3,1},\ c_{1,1} a_{1,2} + c_{1,2} a_{2,2} + c_{1,3} a_{3,2},\ c_{1,1} a_{1,3} + c_{1,2} a_{2,3} + c_{1,3} a_{3,3},\ c_{1,1} a_{1,4} + c_{1,2} a_{2,4} + c_{1,3} a_{3,4} \right], \left[c_{2,1} a_{1,1} + c_{2,2} a_{2,1} + c_{2,3} a_{3,1},\ c_{2,1} a_{1,2} + c_{2,2} a_{2,2} + c_{2,3} a_{3,2},\ c_{2,1} a_{1,3} + c_{2,2} a_{2,3} + c_{2,3} a_{3,3},\ c_{2,1} a_{1,4} + c_{2,2} a_{2,4} + c_{2,3} a_{3,4} \right], \left[c_{3,1} a_{1,1} + c_{3,2} a_{2,1} + c_{3,3} a_{3,1},\ c_{3,1} a_{1,2} + c_{3,2} a_{2,2} + c_{3,3} a_{3,2},\ c_{3,1} a_{1,3} + c_{3,2} a_{2,3} + c_{3,3} a_{3,3},\ c_{3,1} a_{1,4} + c_{3,2} a_{2,4} + c_{3,3} a_{3,4} \right], \left[c_{4,1} a_{1,1} + c_{4,2} a_{2,1} + c_{4,3} a_{3,1},\ c_{4,1} a_{1,2} + c_{4,2} a_{2,2} + c_{4,3} a_{3,2},\ c_{4,1} a_{1,3} + c_{4,2} a_{2,3} + c_{4,3} a_{3,3},\ c_{4,1} a_{1,4} + c_{4,2} a_{2,4} + c_{4,3} a_{3,4} \right] \right]$$

$$1..4, 1..4 \qquad \textbf{(3.16)}$$

Voyons voir...même si on connait la réponse

```
> A^2;
Error, (in rtable/Power) exponentiation operation not defined
for non-square Matrices
```

```
> A:=Matrix(3,3,symbol=a);
```

```
A^2;
```

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

$$\begin{bmatrix} \begin{bmatrix} a_{1,1}^2 + a_{1,2}a_{2,1} + a_{1,3}a_{3,1}, a_{1,1}a_{1,2} + a_{1,2}a_{2,2} + a_{1,3}a_{3,2}, a_{1,1}a_{1,3} + a_{1,2}a_{2,3} + a_{1,3}a_{3,3} \end{bmatrix}, \\ \begin{bmatrix} a_{2,1}a_{1,1} + a_{2,2}a_{2,1} + a_{2,3}a_{3,1}, a_{1,2}a_{2,1} + a_{2,2}^2 + a_{2,3}a_{3,2}, a_{2,1}a_{1,3} + a_{2,2}a_{2,3} + a_{2,3}a_{3,3} \end{bmatrix}, \\ \begin{bmatrix} a_{3,1}a_{1,1} + a_{3,2}a_{2,1} + a_{3,3}a_{3,1}, a_{3,1}a_{1,2} + a_{3,2}a_{2,2} + a_{3,3}a_{3,2}, a_{1,3}a_{3,1} + a_{2,3}a_{3,2} + a_{3,3}^2 \end{bmatrix} \end{bmatrix} \quad \textbf{(3.17)}$$

```
> 'A^(%T)'=A%^(%T);
  ``=A^(%T)
```

$$A^T = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}^T$$

$$= \begin{bmatrix} a_{1,1} & a_{2,1} & a_{3,1} \\ a_{1,2} & a_{2,2} & a_{3,2} \\ a_{1,3} & a_{2,3} & a_{3,3} \end{bmatrix} \quad \textbf{(3.18)}$$

```
> unassign('A','B','C');
```

La macro-commande `Matrix` possède de nombreuses options facultatives. Par exemple, celle permettant l'utilisation d'une fonction d'indexation pour créer des matrices particulières.

La macro-commande `Matrix` en tant que créatrice d'objets de type `rtable`, initialise par défaut la valeur 0 à toutes les entrées de la matrice lorsqu'on ne précise aucune valeur initiale de la matrice.

```
> Nulle_4:=Matrix(4);
  Nulle_2x4:=Matrix(2,4);
```

$$Nulle\_4 := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Nulle\_2x4 := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \textbf{(3.19)}$$

L'assignation d'expressions aux emplacements d'une matrice peut être faite au moment de sa création par l'intermédiaire d'une fonction de deux variables. On dit de cette fonction qu'elle est une fonction d'indexation. Son domaine sera le produit cartésien des deux ensembles d'indices ($1..m$ et $1..n$). Dans ce cas, il faut évidemment déclarer explicitement le format de la matrice.

Comme premier exemple, créons une matrice A de format 5×5 dont le terme général est $c_{ij} = i - j$.

```
> f:=(i,j)->i-j;
  A[`5x5`]:=Matrix(5,5,f);
```

$$f := (i,j) \mapsto i - j$$

$$A_{5x5} := \begin{bmatrix} 0 & -1 & -2 & -3 & -4 \\ 1 & 0 & -1 & -2 & -3 \\ 2 & 1 & 0 & -1 & -2 \\ 3 & 2 & 1 & 0 & -1 \\ 4 & 3 & 2 & 1 & 0 \end{bmatrix} \quad \textbf{(3.20)}$$

Obtenons une matrice B ayant le même terme général que la matrice A mais de format 12 x 12.

```
> B[`12x12`]:=Matrix(12,12,f);
```

$$B_{12x12} := \begin{bmatrix} 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 & \dots \\ 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & \dots \\ 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & \dots \\ 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & \dots \\ 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & \dots \\ 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & \dots \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & \dots \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & \dots \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & \dots \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix} \quad \textbf{(3.21)}$$

12 × 12 Matrix

La matrice **B** a bel et bien été créée mais son contenu n'est pas affiché. En effet, dans le cas de très grosses matrices il est fort probable qu'il ne soit pas pertinent de faire afficher leur contenu dans la zone des résultats.

Le format des matrices (objet de type `Matrix`) dont le contenu sera affiché est contrôlé par la variable d'environnement `rtablesize`. La valeur par défaut de cette variable est 10 et cette valeur peut être changée au cours d'une session de travail.

```
> interface(rtablesize=15);
```

$$[10, 10] \quad \textbf{(3.22)}$$

Malgré l'affichage de la valeur 10, la variable d'environnement `rtablesize` a bien été initialisée à la valeur 15.

```
> B[`12x12`];
```

$$\begin{bmatrix} 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 & -10 & -11 \\ 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 & -10 \\ 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 & -9 \\ 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 \\ 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 & -7 \\ 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 & -6 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 & -5 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 & -4 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 & -3 \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 & -2 \\ 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 & -1 \\ 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix} \quad \textbf{(3.23)}$$

Il est possible de toujours avoir l'affichage de ces matrices quelque soit son format en précisant `rtablesize=infinity`. Revenons à la valeur par défaut de l'affichage.

```
> interface(rtablesize=10);
```

$$15 \quad \textbf{(3.24)}$$

Comme second exemple, créons la matrice de format 4 × 6 suivante: $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ à l'aide d'une fonction d'indexation.

```
> f:=(i,j)->if i=j then 1 else 0 fi:
  M:=Matrix(4,6,f);
```

$$M := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \textbf{(3.25)}$$

Il y a plusieurs fonctions d'indexation pré-définies (*built-in*) permettant de créer certaines matrices particulières. En voici quelques-unes:

scalar[j,n]	diagonal
identity	triangular[upper]
scalar[n]	triangular[lower]
zero	triangular[upper,unit]
constant[n]	triangular[lower,unit]

À titre d'exemples, créons une matrice diagonale d'ordre 4, une matrice scalaire d'ordre 6 et une matrice triangulaire inférieure d'ordre 4.

```
> M:=Matrix(1..4,1..4,<-1,3,12,sqrt(2)>,shape=diagonal);
```

$$M := \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 12 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{bmatrix} \tag{3.26}$$

```
> M:=Matrix(1..6,1..6,shape=scalar[-sqrt(3)]);
```

$$M := \begin{bmatrix} -\sqrt{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & -\sqrt{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\sqrt{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & -\sqrt{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & -\sqrt{3} \end{bmatrix} \tag{3.27}$$

```
> f:=(i,j)->(-1)^(2*i+j):
  M:=Matrix(4,f,shape=triangular[lower]);
```

$$M := \begin{bmatrix} -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix} \tag{3.28}$$

Les matrices nulle et identité peuvent être créées comme matrices scalaires mais également avec leur fonction d'indexation pré-définie respectives.

```
> Id:=Matrix(5,5,shape=identity);
  `0`:=Matrix(5,5,shape=zero);
```

$$Id := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$0 := \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.29}$$

```
> unassign('M','A','B','C','Nulle_4','Nulle_2x4','Id','`0`');
```

La bibliothèque LinearAlgebra

Après le lancement de Maple, un nombre minimum de macro-commandes est accessible à l'usager: ce sont les macro-commandes de la bibliothèque de base. Pour enrichir la session Maple d'un nombre accru de macro-commandes spécialisées au calcul en Algèbre linéaire, la bibliothèque LinearAlgebra possède plusieurs macro-commandes facilitant le calcul dans le domaine de l'algèbre linéaire. Rendons disponible la notation abrégée des macro-commandes de cette bibliothèque en l'initialisant avec la macro-commande with.

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

[*&x*, *Add*, *Adjoint*, *BackwardSubstitute*, *BandMatrix*, *Basis*, *BezoutMatrix*, *BidiagonalForm*, *BilinearForm*, *CARE*, *CharacteristicMatrix*, *CharacteristicPolynomial*, *Column*, *ColumnDimension*, *ColumnOperation*, *ColumnSpace*, *CompanionMatrix*, *CompressedSparseForm*, *ConditionNumber*, *ConstantMatrix*, *ConstantVector*, *Copy*, *CreatePermutation*, *CrossProduct*, *DARE*, *DeleteColumn*, *DeleteRow*, *Determinant*, *Diagonal*, *DiagonalMatrix*, *Dimension*, *Dimensions*, *DotProduct*, *EigenConditionNumbers*, *Eigenvalues*, *Eigenvectors*, *Equal*, *ForwardSubstitute*, *FrobeniusForm*, *FromCompressedSparseForm*, *FromSplitForm*, *GaussianElimination*, *GenerateEquations*, *GenerateMatrix*, *Generic*, *GetResultDataType*, *GetResultShape*, *GivensRotationMatrix*, *GramSchmidt*, *HankelMatrix*, *HermiteForm*, *HermitianTranspose*, *HessenbergForm*, *HilbertMatrix*, *HouseholderMatrix*, *IdentityMatrix*, *IntersectionBasis*, *IsDefinite*, *IsOrthogonal*, *IsSimilar*, *IsUnitary*, *JordanBlockMatrix*, *JordanForm*, *KroneckerProduct*, *LA_Main*, *LUDecomposition*, *LeastSquares*, *LinearSolve*, *LyapunovSolve*, *Map*, *Map2*, *MatrixAdd*, *MatrixExponential*, *MatrixFunction*, *MatrixInverse*, *MatrixMatrixMultiply*, *MatrixNorm*, *MatrixPower*, *MatrixScalarMultiply*, *MatrixVectorMultiply*, *MinimalPolynomial*, *Minor*, *Modular*, *Multiply*, *NoUserValue*, *Norm*, *Normalize*, *NullSpace*, *OuterProductMatrix*, *Permanent*, *Pivot*, *PopovForm*, *ProjectionMatrix*, *QRDecomposition*, *RandomMatrix*, *RandomVector*, *Rank*, *RationalCanonicalForm*, *ReducedRowEchelonForm*, *Row*, *RowDimension*, *RowOperation*, *RowSpace*, *ScalarMatrix*, *ScalarMultiply*, *ScalarVector*, *SchurForm*, *SingularValues*, *SmithForm*, *SplitForm*, *StronglyConnectedBlocks*, *SubMatrix*, *SubVector*, *SumBasis*, *SylvesterMatrix*, *SylvesterSolve*, *ToeplitzMatrix*, *Trace*, *Transpose*, *TridiagonalForm*, *UnitVector*, *VandermondeMatrix*, *VectorAdd*, *VectorAngle*, *VectorMatrixMultiply*, *VectorNorm*, *VectorScalarMultiply*, *ZeroMatrix*, *ZeroVector*, *Zip*] **(4.1)**

Obtenons le nombre de macro-commandes supplémentaires maintenant disponibles à l'aide de la macro-commande nops.

```
> nops(%);
```

130 **(4.2)**

Bien sûr, nous n'allons pas aborder dans le détail chacune de ces 130 macro-commandes de l'extension `LinearAlgebra`. On se limitera dans ce document-ci à seulement quelques unes d'entre-elles. Plusieurs autres macro-commandes seront abordées au cours des documents RP (*Résolution de problèmes*) sous l'onglet

NYC de mon site Internet.

Calculs sur les vecteurs algébriques et les matrices

Les macro-commandes de la bibliothèque `LinearAlgebra` rendent plus performant les calculs avec les vecteurs et les matrices. Avec ces macro-commandes, Maple n'a pas besoin de déterminer la nature des opérandes ce qui n'est pas le cas lorsque qu'on utilise les opérateurs arithmétiques habituels.

Opérations sur les vecteurs algébriques

Addition de deux vecteurs

Soit les vecteurs (lignes) $\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$ et $\mathbf{V} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \in \mathbb{R}^4$ quelconques.

```
> U_:=Vector[row](4,symbol=u);
  V_:=Vector[row](4,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \quad \textbf{(5.1.1.1)}$$

```
> VectorAdd(U_,V_);
```

$$\begin{bmatrix} u_1 + v_1 & u_2 + v_2 & u_3 + v_3 & u_4 + v_4 \end{bmatrix} \quad \textbf{(5.1.1.2)}$$

Multiplication d'un vecteur par un scalaire

Soit $\alpha \in \mathbb{R}$ quelconque.

Soit le vecteur $\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \in \mathbb{R}^4$ quelconque.

```
> U_:=Vector[row](4,symbol=u);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \quad \textbf{(5.1.2.1)}$$

```
> ScalarMultiply(U_,alpha)
```

$$\begin{bmatrix} \alpha u_1 & \alpha u_2 & \alpha u_3 & \alpha u_4 \end{bmatrix} \quad \textbf{(5.1.2.2)}$$

Multiplication scalaire de deux vecteurs (scalairement)

Soit les vecteurs (lignes) $\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$ et $\mathbf{V} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \in \mathbb{R}^4$ quelconques.

```
> U_:=Vector[row](4,symbol=u);
  V_:=Vector[row](4,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \quad \textbf{(5.1.3.1)}$$

```
> DotProduct(U_,V_,conjugate=false);
```

$$u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4 \qquad \textbf{(5.1.3.2)}$$

Remarque 1: L'option `conjugate=false` évite de déclarer que les variables soient réelles.

Remarque 2: Comme nous l'avons vu précédemment, la multiplication scalaire effectuée avec l'opérateur point « . » requiert que les deux vecteurs aient la même orientation. Or, dans la définition mathématique de cette multiplication, seule la dimension est impliquée. La macro-commande `DotProduct` ne tient pas compte de l'orienration de chaque vecteur dans son calcul.

Soit les vecteurs $U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$ et $V = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \in \mathbb{R}^4$ quelconques.

```
> U_:=Vector(4,symbol=u);
  V_:=Vector[row](4,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \qquad \textbf{(5.1.3.3)}$$

```
> DotProduct(U_,V_,conjugate=false);
  DotProduct(V_,U_,conjugate=false);
```

$$u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4$$

$$u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4 \qquad \textbf{(5.1.3.4)}$$

Multiplication vectorielle de deux vecteurs (vectoriellement)

Soit les vecteurs (lignes) $\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$ et $\mathbf{V} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \in \mathbb{R}^3$ quelconques.

```
> U_:=Vector[row](3,symbol=u);
  V_:=Vector[row](3,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \qquad \textbf{(5.1.4.1)}$$

```
> CrossProduct(U_,V_);
  CrossProduct(V_,U_);
```

$$\begin{bmatrix} u_2 v_3 - u_3 v_2 & -u_1 v_3 + u_3 v_1 & u_1 v_2 - u_2 v_1 \end{bmatrix}$$

$$\begin{bmatrix} -u_2 v_3 + u_3 v_2 & u_1 v_3 - u_3 v_1 & -u_1 v_2 + u_2 v_1 \end{bmatrix} \quad \textbf{(5.1.4.2)}$$

```
> U:='U':V:='V':
```

Norme d'un vecteur algébrique

Soit $\mathbf{v} = \begin{bmatrix} a & b & c \end{bmatrix} \in \mathbb{R}^3$ quelconque

```
> v_ := `<|>`(a, b, c);
```

$$\vec{v} := \begin{bmatrix} a & b & c \end{bmatrix} \quad \textbf{(5.1.5.1)}$$

```
> VectorNorm(v_,Euclidean);
  VectorNorm(v_,Euclidean,conjugate=false) ;
```

$$\sqrt{|a|^2 + |b|^2 + |c|^2}$$

$$\sqrt{a^2 + b^2 + c^2} \quad \textbf{(5.1.5.2)}$$

```
> VectorScalarMultiply(v_, 1/VectorNorm(v_,2,conjugate=false))
```

$$\begin{bmatrix} \frac{a}{\sqrt{a^2 + b^2 + c^2}} & \frac{b}{\sqrt{a^2 + b^2 + c^2}} & \frac{c}{\sqrt{a^2 + b^2 + c^2}} \end{bmatrix} \quad \textbf{(5.1.5.3)}$$

```
> VectorNorm((5.1.5.3),2,conjugate=false);
```

$$\sqrt{\frac{a^2}{a^2 + b^2 + c^2} + \frac{b^2}{a^2 + b^2 + c^2} + \frac{c^2}{a^2 + b^2 + c^2}} \quad \textbf{(5.1.5.4)}$$

```
> simplify((5.1.5.4));
```

$$1 \quad \textbf{(5.1.5.5)}$$

```
> v_:='v_':
```

Angle entre deux vecteurs

Soit les vecteurs (lignes) $\mathbf{U} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$ et $\mathbf{V} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \in \mathbb{R}^3$ quelconques. (Les noms de vecteurs sont ici en majuscules. On ne peut pas utiiser la même lettre à la fois pour le nom du vecteur et pour les composantes de celui-ci)

```
> U_:=Vector[row](3,symbol=u);
  V_:=Vector[row](3,symbol=v);
```

$$\vec{U} := \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$$

$$\vec{V} := \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \quad \textbf{(5.1.6.1)}$$

```
> VectorAngle(U_,V_,conjugate=false);
```

$$\arccos\left(\frac{u_1 v_1 + u_2 v_2 + u_3 v_3}{\sqrt{u_1^2 + u_2^2 + u_3^2}\sqrt{v_1^2 + v_2^2 + v_3^2}} \right) \quad \textbf{(5.1.6.2)}$$

```
> U:='U':V:='V':
```

Opérations sur les matrices

Addition de deux matrices

La macro-commande MatrixAdd de la bibliothèque `LinearAlgebra` effectue l'addition de deux matrices mais, il est aussi possible d'additionner plusieurs matrices directement avec l'opérateur « + » sans utiliser `MatrixAdd`.

Comme exemple de saisie de matrices, créons une matrice A = $\begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix}$ comme matrice colonne de trois lignes et une matrice B = $\begin{bmatrix} -1 & -2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & -9 \end{bmatrix}$ comme une matrice ligne de trois colonnes.

```
> A:= < <3|6|4>,
        <1|1|1>,
        <1|-2|1> >;
```

$$A := \begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix} \quad \textbf{(5.2.1.1)}$$

```
> B:= < <-1,4,7>|
        <-2,5,8>|
        <3,-6,9> >;
```

$$B := \begin{bmatrix} -1 & -2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & 9 \end{bmatrix} \quad \textbf{(5.2.1.2)}$$

La somme avec avec `MatrixAdd`:

```
> 'A+B'=A%+B;
  ``=MatrixAdd(A,B);
```

$$A + B = \begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} -1 & -2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 & 7 \\ 5 & 6 & -5 \\ 8 & 6 & 10 \end{bmatrix} \quad \textbf{(5.2.1.3)}$$

Multiplication d'une matrice par un scalaire

La macro-commande ScalarMultiply de la bibliothèque `LinearAlgebra` effectue la multiplication d'une matrice par un scalaire mais, il est aussi possible d'effectuer ce calcul directement avec l'opérateur de multiplication « * » sans utiliser `ScalarMultiply`.

```
> '5*A'=5%*A;
  ``=ScalarMultiply(A,5);
```

$$5A = 5 * \begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 15 & 30 & 20 \\ 5 & 5 & 5 \\ 5 & -10 & 5 \end{bmatrix} \quad \textbf{(5.2.2.1)}$$

Multiplication matricielle

La macro-commande MatrixMatrixMultiply de la bibliothèque `LinearAlgebra` effectue la multiplication matricielle de deux matrices, lorsque, bien sûr, il y a compatibilité des formats mais, il est aussi possible d'utiliser l'opérateur de multiplication non commutative: « . ».

```
> 'A.B'=A%.B;
  ``=MatrixMatrixMultiply(A,B);
  'B.A'=B%.A;
  ``=MatrixMatrixMultiply(B,A);
```

$$A \cdot B = \begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$= \begin{bmatrix} 49 & 56 & 9 \\ 10 & 11 & 6 \\ -2 & -4 & 24 \end{bmatrix}$$

$$B \cdot A = \begin{bmatrix} -1 & -2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 3 & 6 & 4 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -2 & -14 & -3 \\ 11 & 41 & 15 \\ 38 & 32 & 45 \end{bmatrix} \quad \textbf{(5.2.3.1)}$$

Remarque: la multiplication matricielle n'est pas une opération commutative: `MatrixMatrixMultiply(A,B)` est différent de `MatrixMatrixMultiply(B,A)`. Attention donc à l'ordre des opérandes dans la macro-commande MatrixMatrixMultiply.

```
> MatrixMatrixMultiply(A,B)<>MatrixMatrixMultiply(B,A);
```

$$\begin{bmatrix} 49 & 56 & 9 \\ 10 & 11 & 6 \\ -2 & -4 & 24 \end{bmatrix} \neq \begin{bmatrix} -2 & -14 & -3 \\ 11 & 41 & 15 \\ 38 & 32 & 45 \end{bmatrix} \quad \textbf{(5.2.3.2)}$$

Transposée d'une matrice

La transposée d'une matrice est obtenue avec la macro-commande Transpose de la bibliothèque `LinearAlgebra`.

```
> C:= Matrix([
        [1,2,3],
        [4,5,6]] );
```

$$C := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{5.2.4.1}$$

```
> 'C^(%t)'=C^(%t);
  ``=Transpose(C);
```

$$C^{t} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^{t}$$

$$= \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \tag{5.2.4.2}$$

Inverse d'une matrice régulière

La matrice inverse d'une matrice régulière est obtenue avec la macro-commande MatrixInverse de l'extension `LinearAlgebra`.

```
> A:=Matrix([
      [1,3,-5],
      [2,-4,1],
      [1,1,1] ]);
```

$$A := \begin{bmatrix} 1 & 3 & -5 \\ 2 & -4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.2.5.1}$$

```
> 'A^(-1)'=MatrixInverse(A);
```

$$\frac{1}{A} = \begin{bmatrix} \frac{5}{38} & \frac{4}{19} & \frac{17}{38} \\ \frac{1}{38} & -\frac{3}{19} & \frac{11}{38} \\ -\frac{3}{19} & -\frac{1}{19} & \frac{5}{19} \end{bmatrix} \tag{5.2.5.2}$$

Vérification:

```
> A %. MatrixInverse(A) = A.MatrixInverse(A);
```

(5.2.5.3)

$$\begin{bmatrix} 1 & 3 & -5 \\ 2 & -4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{5}{38} & \frac{4}{19} & \frac{17}{38} \\ \frac{1}{38} & -\frac{3}{19} & \frac{11}{38} \\ -\frac{3}{19} & -\frac{1}{19} & \frac{5}{19} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5.2.5.3)}$$

```
> MatrixInverse(A) %. A = MatrixInverse(A).A;
```

$$\begin{bmatrix} \frac{5}{38} & \frac{4}{19} & \frac{17}{38} \\ \frac{1}{38} & -\frac{3}{19} & \frac{11}{38} \\ -\frac{3}{19} & -\frac{1}{19} & \frac{5}{19} \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 & -5 \\ 2 & -4 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5.2.5.4)}$$

Maple reconnaît les expressions $\frac{1}{A}$ et A^{-1} comme la matrice inverse de la matrice A (si elle est régulière).

```
> 1/A;
```

$$\begin{bmatrix} \frac{5}{38} & \frac{4}{19} & \frac{17}{38} \\ \frac{1}{38} & -\frac{3}{19} & \frac{11}{38} \\ -\frac{3}{19} & -\frac{1}{19} & \frac{5}{19} \end{bmatrix} \quad \textbf{(5.2.5.5)}$$

```
> A^(-1);
```

$$\begin{bmatrix} \frac{5}{38} & \frac{4}{19} & \frac{17}{38} \\ \frac{1}{38} & -\frac{3}{19} & \frac{11}{38} \\ -\frac{3}{19} & -\frac{1}{19} & \frac{5}{19} \end{bmatrix} \quad \textbf{(5.2.5.6)}$$

Voyons comment s'affichera le résultat dans le cas de la matrice A ci-dessous qui n'est pas régulière.

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

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad \textbf{(5.2.5.7)}$$

```
> 1/A;
Error, (in rtable/Power) singular matrix
```

Un dernier exemple.

```
> A:=Matrix([
      [cos(theta),-sin(theta)],
      [sin(theta),cos(theta)] ]);
```

$$A := \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{5.2.5.8}$$

```
> 1/A;
```

$$\begin{bmatrix} \frac{\cos(\theta)}{\cos(\theta)^2+\sin(\theta)^2} & \frac{\sin(\theta)}{\cos(\theta)^2+\sin(\theta)^2} \\ -\frac{\sin(\theta)}{\cos(\theta)^2+\sin(\theta)^2} & \frac{\cos(\theta)}{\cos(\theta)^2+\sin(\theta)^2} \end{bmatrix} \tag{5.2.5.9}$$

```
> simplify((5.2.5.9));
```

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{5.2.5.10}$$

Multiplions la matrice A avec son inverse qu'on obtiendra avec la macro-commande `MatrixInverse`.

```
> A%.MatrixInverse(A):=A.MatrixInverse(A);
```

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} \frac{\cos(\theta)}{\cos(\theta)^2+\sin(\theta)^2} & \frac{\sin(\theta)}{\cos(\theta)^2+\sin(\theta)^2} \\ -\frac{\sin(\theta)}{\cos(\theta)^2+\sin(\theta)^2} & \frac{\cos(\theta)}{\cos(\theta)^2+\sin(\theta)^2} \end{bmatrix} := \begin{bmatrix} \left[\frac{\cos(\theta)^2}{\cos(\theta)^2+\sin(\theta)^2} + \frac{\sin(\theta)^2}{\cos(\theta)^2+\sin(\theta)^2}, 0 \right], \\ \left[0, \frac{\cos(\theta)^2}{\cos(\theta)^2+\sin(\theta)^2} + \frac{\sin(\theta)^2}{\cos(\theta)^2+\sin(\theta)^2} \right] \end{bmatrix} \tag{5.2.5.11}$$

Le mécanisme de la simplification automatique n'a pas simplifier les éléments de la matrice produit comme nous aurions aimé qu'il le fasse. Faisons alors une simplification sur demande avec la macro-commande passe-partout `simplify`.

```
> simplify((5.2.5.11));
```

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.2.5.12}$$

Remarque: En multipliant la matrice A par son inverse dans la notation A^{-1}, la simplification passera directement par le mécanisme de la simplification automatique plutôt que par la simplification du calcul lui-même.

```
> 'S'%.'S^(-1)':=S.S^(-1); # La matrice S n'a même jamais été
  créée.
```

$$S \cdot \frac{1}{S} := 1 \qquad \textbf{(5.2.5.13)}$$

Attention: il est de même avec A^0 qui est automatiquement simplifiée par 1 et non pas par la matrice identité d'ordre 2. Il faut donc faire attention aux simplifications dans les matrices.

```
> A%^0=A^0;
```

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}^0 = 1 \qquad \textbf{(5.2.5.14)}$$

Afin d'avoir pour résultat la matrice identité A^0 d'ordre 2, il faut l'obtenir avec la macro-commande `MatrixPower`.

```
> A%^0:=MatrixPower(A,0); #A étant d'ordre 2
```

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}^0 := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \textbf{(5.2.5.15)}$$

Tout de même, avec les opérateurs arithmétiques habituels, le nombre 1 est traité comme une matrice identité dont l'ordre est automatiquement adapté pour que l'opération puisse être faite.

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

$$A := \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \qquad \textbf{(5.2.5.16)}$$

```
> 1-A;
```

$$\begin{bmatrix} 0 & -2 & -2 \\ -2 & 0 & -2 \\ -2 & -2 & 0 \end{bmatrix} \qquad \textbf{(5.2.5.17)}$$

```
> 1*A;
```

$$\begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \qquad \textbf{(5.2.5.18)}$$

Mais

```
> 1.A;
Error, missing operator or `;`
```

Puissance d'une matrice carrée

Les puissances entières de matrices carrées sont reconnues de manière habituelle sauf avec la puissance 0 qui donne le nombre 1 comme résultat et non pas la matrice identité (le mécanisme de la simplification automatique ne contrôle pas la nature de la base lorsque l'exposant est 0).

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

$$A := \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{5.2.6.1}$$

```
> MatrixPower(A,18);
```

$$\begin{bmatrix} 1 & 18 \\ 0 & 1 \end{bmatrix} \tag{5.2.6.2}$$

```
> A^18;
```

$$\begin{bmatrix} 1 & 18 \\ 0 & 1 \end{bmatrix} \tag{5.2.6.3}$$

Dernier exemple.

```
> A:=Matrix([
      [ 1, 2, 3, 4, 5],
      [ 5, 4, 3, 2, 1],
      [ 1, 1, 1, 1, 1],
      [-1,-2,-3,-4,-5],
      [-5,-4,-3,-2,-1] ]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & -2 & -3 & -4 & -5 \\ -5 & -4 & -3 & -2 & -1 \end{bmatrix} \tag{5.2.6.4}$$

```
> Sum(A^i,i=1..25)=sum(A^i,i=1..25);
```

$$\sum_{i=1}^{25} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ -1 & -2 & -3 & -4 & -5 \\ -5 & -4 & -3 & -2 & -1 \end{bmatrix}^i = \begin{bmatrix} -359 & -358 & -357 & -356 & -355 \\ 509 & 508 & 507 & 506 & 505 \\ 25 & 25 & 25 & 25 & 25 \\ 359 & 358 & 357 & 356 & 355 \\ -509 & -508 & -507 & -506 & -505 \end{bmatrix} \tag{5.2.6.5}$$

Matrices aléatoires

La macro-commande RandomMatrix de l'extension `LinearAlgebra` permet de générer aléatoirement les entrées d'une matrice. Cela peut être commode pour tester ou pour illustrer des propriétés de calculs matriciels. Par défaut, les éléments générés de la matrice sont des nombres entiers compris entre -99 et 99 inclusivement. Les nombres entiers sont générés à l'aide du générateur de

nombres aléatoires rand.

Pour générer aléatoirement les entrées d'une matrice, il suffit de préciser le format voulu dans la macro-commande `RandomMatrix`.

```
> restart;
  with(LinearAlgebra):
> A:=RandomMatrix(3,2);
```

$$A := \begin{bmatrix} 99 & 92 \\ 29 & -31 \\ 44 & 67 \end{bmatrix} \quad \textbf{(5.2.7.1)}$$

On peut modifier " l'intervalle " des valeurs permises pour les entrées de la matrice en précisant l'option `generator=a..b`. Générons aléatoirement deux nouvelles matrices triangulaires inférieure et supérieure de format 4×4 de nombres entiers compris entre -10 et 10.

```
> A:=RandomMatrix(4,4,generator=-10..10,outputoptions=[shape=
  triangular[lower]]);
```

$$A := \begin{bmatrix} -10 & 0 & 0 & 0 \\ -5 & 9 & 0 & 0 \\ -6 & 3 & 5 & 0 \\ 8 & -10 & 8 & 9 \end{bmatrix} \quad \textbf{(5.2.7.2)}$$

```
> B:=RandomMatrix(4,4,generator=-10..10,outputoptions=[shape=
  triangular[upper]]);
```

$$B := \begin{bmatrix} -6 & -2 & 10 & 7 \\ 0 & -2 & -2 & -10 \\ 0 & 0 & 4 & -6 \\ 0 & 0 & 0 & 4 \end{bmatrix} \quad \textbf{(5.2.7.3)}$$

La macro-commande `Rand` utilisée par la macro-commande `RandomMatrix` est en fait une procédure de calcul qui implique la congruence d'un produit de deux très grands nombres selon une certaine formule.

Le générateur `rand` génère un nombre " pseudo-aléatoire " de 12 entiers, ce qui est la plupart du temps un nombre beaucoup trop grand pour la plupart de nos besoins. Comme on l'a constaté avec le résultat précédent, il est possible avec la macro-commande `rand` de générer des séquences de nombres entiers plus petits et même des séquences de nombre décimaux compris entre -1 et 1 avec un nombre quelconque de décimales. Mais, on évitera ici, de donner des détails explicatifs sur la manière dont il faut s'y prendre.

Pour générer le premier élément x_1 de la séquence de nombres aléatoires, Maple initialise une valeur x_0 entière et positive. Cette valeur est appelé amorce (seed, semence) du générateur. Nous n'avons pas besoin de connaître cette valeur. Par contre, Maple nous donne la possibilité de spécifier un entier positif de son choix comme amorce pour initier le processus pseudo-aléatoire.

La macro-commande `randomise` permet à l'usager d'imposer au générateur la semence du processus

pseudo-aléatoire.

Si on "rafraîchi" la session Maple avec un `restart`, on observera que la matrice C est identique à la matrice A qui a été obtenue au début de cette section lors du premier appel du générateur `RandomMatrix`.

```
> restart;
  with(LinearAlgebra):
> C:=LinearAlgebra[RandomMatrix](3,2);
```

$$C := \begin{bmatrix} 99 & 92 \\ 29 & -31 \\ 44 & 67 \end{bmatrix} \quad \textbf{(5.2.7.4)}$$

Pour s'assurer une plus grande variabilité de matrices initiales qui seront générées avec les différentes équipes d'une classe, chaque équipe devra, avant le premier appel indirecte au générateur `rand` effectué avec la macro-commande `RandomMatrix,` initialiser l'amorce au nombre correspondant à la date de naissance d'un des deux membres de votre équipe dans le format donné ci-dessous.

AnnéeMoisJour

Par exemple:

```
> restart;
  with(LinearAlgebra):
  randomize(840309);  # Personne née le 9 mars 1984
```

$$840309 \quad \textbf{(5.2.7.5)}$$

Ainsi les matrices A de chaque équipe seront *probablement* différentes.

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

$$A := \begin{bmatrix} 43 & -32 \\ -3 & -97 \\ -16 & -32 \end{bmatrix} \quad \textbf{(5.2.7.6)}$$

Déterminant

Il sera présenté ici sommairement la syntaxe Maple pour le calcul des déterminants.

La macro-commande Determinant de l'extension `LinearAlgebra` donne pour résultat la valeur du déterminant d'une matrice.

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

$$A := \begin{bmatrix} 1 & 2 & 1 \\ -1 & 1 & 1 \\ 2 & 3 & 1 \end{bmatrix} \quad \textbf{(6.1)}$$

```
> `|A|`=Determinant(A);
```

(6.2)

$$|A| = -1 \tag{6.2}$$

Le développement suivant montrera le détail formel du calcul d'un déterminant d'une matrice d'ordre *n*, $n = 2..5$.

Soit A une matrice quelconque d'ordre *n* et initalisons *n* successivement avec les entiers 2, 3, 4 et 5.

```
> n:=2:
  A:=Matrix(n,symbol=a);
```

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \tag{6.3}$$

Obtenons le développement du calcul de chacun des déterminants ci-dessous.

```
> `|A|`=Determinant(A);
```

$$|A| = a_{1,1}\,a_{2,2} - a_{1,2}\,a_{2,1} \tag{6.4}$$

Poursuivons les développements avec la différentes valeurs de *n*.

```
> n:=3:
  A:=Matrix(n,symbol=a):
  `|A|`=Determinant(A);
```

$$|A| = a_{1,1}\,a_{2,2}\,a_{3,3} - a_{1,1}\,a_{2,3}\,a_{3,2} - a_{1,2}\,a_{2,1}\,a_{3,3} + a_{1,2}\,a_{2,3}\,a_{3,1} + a_{1,3}\,a_{2,1}\,a_{3,2} - a_{1,3}\,a_{2,2}\,a_{3,1} \tag{6.5}$$

```
> n:=4:
  A:=Matrix(n,symbol=a):
  `|A|`=Determinant(A);
```

$$\begin{aligned} |A| = {} & a_{1,1}\,a_{2,2}\,a_{3,3}\,a_{4,4} - a_{1,1}\,a_{2,2}\,a_{3,4}\,a_{4,3} - a_{1,1}\,a_{2,3}\,a_{3,2}\,a_{4,4} + a_{1,1}\,a_{2,3}\,a_{3,4}\,a_{4,2} \\ & + a_{1,1}\,a_{2,4}\,a_{3,2}\,a_{4,3} - a_{1,1}\,a_{2,4}\,a_{3,3}\,a_{4,2} - a_{1,2}\,a_{2,1}\,a_{3,3}\,a_{4,4} + a_{1,2}\,a_{2,1}\,a_{3,4}\,a_{4,3} \\ & + a_{1,2}\,a_{2,3}\,a_{3,1}\,a_{4,4} - a_{1,2}\,a_{2,3}\,a_{3,4}\,a_{4,1} - a_{1,2}\,a_{2,4}\,a_{3,1}\,a_{4,3} + a_{1,2}\,a_{2,4}\,a_{3,3}\,a_{4,1} \\ & + a_{1,3}\,a_{2,1}\,a_{3,2}\,a_{4,4} - a_{1,3}\,a_{2,1}\,a_{3,4}\,a_{4,2} - a_{1,3}\,a_{2,2}\,a_{3,1}\,a_{4,4} + a_{1,3}\,a_{2,2}\,a_{3,4}\,a_{4,1} \\ & + a_{1,3}\,a_{2,4}\,a_{3,1}\,a_{4,2} - a_{1,3}\,a_{2,4}\,a_{3,2}\,a_{4,1} - a_{1,4}\,a_{2,1}\,a_{3,2}\,a_{4,3} + a_{1,4}\,a_{2,1}\,a_{3,3}\,a_{4,2} \\ & + a_{1,4}\,a_{2,2}\,a_{3,1}\,a_{4,3} - a_{1,4}\,a_{2,2}\,a_{3,3}\,a_{4,1} - a_{1,4}\,a_{2,3}\,a_{3,1}\,a_{4,2} + a_{1,4}\,a_{2,3}\,a_{3,2}\,a_{4,1} \end{aligned} \tag{6.6}$$

```
> n:=5:
  A:=Matrix(n,symbol=a):
  `|A|`=Determinant(A);
```

$$\begin{aligned} |A| = {} & a_{1,1}\,a_{2,2}\,a_{3,3}\,a_{4,4}\,a_{5,5} - a_{1,1}\,a_{2,2}\,a_{3,3}\,a_{4,5}\,a_{5,4} - a_{1,1}\,a_{2,2}\,a_{3,4}\,a_{4,3}\,a_{5,5} + a_{1,1}\,a_{2,2}\,a_{3,4}\,a_{4,5}\,a_{5,3} \\ & + a_{1,1}\,a_{2,2}\,a_{3,5}\,a_{4,3}\,a_{5,4} - a_{1,1}\,a_{2,2}\,a_{3,5}\,a_{4,4}\,a_{5,3} - a_{1,1}\,a_{2,3}\,a_{3,2}\,a_{4,4}\,a_{5,5} + a_{1,1}\,a_{2,3}\,a_{3,2}\,a_{4,5}\,a_{5,4} \\ & + a_{1,1}\,a_{2,3}\,a_{3,4}\,a_{4,2}\,a_{5,5} - a_{1,1}\,a_{2,3}\,a_{3,4}\,a_{4,5}\,a_{5,2} - a_{1,1}\,a_{2,3}\,a_{3,5}\,a_{4,2}\,a_{5,4} + a_{1,1}\,a_{2,3}\,a_{3,5}\,a_{4,4}\,a_{5,2} \\ & + a_{1,1}\,a_{2,4}\,a_{3,2}\,a_{4,3}\,a_{5,5} - a_{1,1}\,a_{2,4}\,a_{3,2}\,a_{4,5}\,a_{5,3} - a_{1,1}\,a_{2,4}\,a_{3,3}\,a_{4,2}\,a_{5,5} + a_{1,1}\,a_{2,4}\,a_{3,3}\,a_{4,5}\,a_{5,2} \\ & + a_{1,1}\,a_{2,4}\,a_{3,5}\,a_{4,2}\,a_{5,3} - a_{1,1}\,a_{2,4}\,a_{3,5}\,a_{4,3}\,a_{5,2} - a_{1,1}\,a_{2,5}\,a_{3,2}\,a_{4,3}\,a_{5,4} + a_{1,1}\,a_{2,5}\,a_{3,2}\,a_{4,4}\,a_{5,3} \\ & + a_{1,1}\,a_{2,5}\,a_{3,3}\,a_{4,2}\,a_{5,4} - a_{1,1}\,a_{2,5}\,a_{3,3}\,a_{4,4}\,a_{5,2} - a_{1,1}\,a_{2,5}\,a_{3,4}\,a_{4,2}\,a_{5,3} + a_{1,1}\,a_{2,5}\,a_{3,4}\,a_{4,3}\,a_{5,2} \\ & - a_{1,2}\,a_{2,1}\,a_{3,3}\,a_{4,4}\,a_{5,5} + a_{1,2}\,a_{2,1}\,a_{3,3}\,a_{4,5}\,a_{5,4} + a_{1,2}\,a_{2,1}\,a_{3,4}\,a_{4,3}\,a_{5,5} - a_{1,2}\,a_{2,1}\,a_{3,4}\,a_{4,5}\,a_{5,3} \end{aligned} \tag{6.7}$$

$$
\begin{aligned}
&- a_{1,2}a_{2,1}a_{3,5}a_{4,3}a_{5,4} + a_{1,2}a_{2,1}a_{3,5}a_{4,4}a_{5,3} + a_{1,2}a_{2,3}a_{3,1}a_{4,4}a_{5,5} - a_{1,2}a_{2,3}a_{3,1}a_{4,5}a_{5,4} \\
&- a_{1,2}a_{2,3}a_{3,4}a_{4,1}a_{5,5} + a_{1,2}a_{2,3}a_{3,4}a_{4,5}a_{5,1} + a_{1,2}a_{2,3}a_{3,5}a_{4,1}a_{5,4} - a_{1,2}a_{2,3}a_{3,5}a_{4,4}a_{5,1} \\
&- a_{1,2}a_{2,4}a_{3,1}a_{4,3}a_{5,5} + a_{1,2}a_{2,4}a_{3,1}a_{4,5}a_{5,3} + a_{1,2}a_{2,4}a_{3,3}a_{4,1}a_{5,5} - a_{1,2}a_{2,4}a_{3,3}a_{4,5}a_{5,1} \\
&- a_{1,2}a_{2,4}a_{3,5}a_{4,1}a_{5,3} + a_{1,2}a_{2,4}a_{3,5}a_{4,3}a_{5,1} + a_{1,2}a_{2,5}a_{3,1}a_{4,3}a_{5,4} - a_{1,2}a_{2,5}a_{3,1}a_{4,4}a_{5,3} \\
&- a_{1,2}a_{2,5}a_{3,3}a_{4,1}a_{5,4} + a_{1,2}a_{2,5}a_{3,3}a_{4,4}a_{5,1} + a_{1,2}a_{2,5}a_{3,4}a_{4,1}a_{5,3} - a_{1,2}a_{2,5}a_{3,4}a_{4,3}a_{5,1} \\
&+ a_{1,3}a_{2,1}a_{3,2}a_{4,4}a_{5,5} - a_{1,3}a_{2,1}a_{3,2}a_{4,5}a_{5,4} - a_{1,3}a_{2,1}a_{3,4}a_{4,2}a_{5,5} + a_{1,3}a_{2,1}a_{3,4}a_{4,5}a_{5,2} \\
&+ a_{1,3}a_{2,1}a_{3,5}a_{4,2}a_{5,4} - a_{1,3}a_{2,1}a_{3,5}a_{4,4}a_{5,2} - a_{1,3}a_{2,2}a_{3,1}a_{4,4}a_{5,5} + a_{1,3}a_{2,2}a_{3,1}a_{4,5}a_{5,4} \\
&+ a_{1,3}a_{2,2}a_{3,4}a_{4,1}a_{5,5} - a_{1,3}a_{2,2}a_{3,4}a_{4,5}a_{5,1} - a_{1,3}a_{2,2}a_{3,5}a_{4,1}a_{5,4} + a_{1,3}a_{2,2}a_{3,5}a_{4,4}a_{5,1} \\
&+ a_{1,3}a_{2,4}a_{3,1}a_{4,2}a_{5,5} - a_{1,3}a_{2,4}a_{3,1}a_{4,5}a_{5,2} - a_{1,3}a_{2,4}a_{3,2}a_{4,1}a_{5,5} + a_{1,3}a_{2,4}a_{3,2}a_{4,5}a_{5,1} \\
&+ a_{1,3}a_{2,4}a_{3,5}a_{4,1}a_{5,2} - a_{1,3}a_{2,4}a_{3,5}a_{4,2}a_{5,1} - a_{1,3}a_{2,5}a_{3,1}a_{4,2}a_{5,4} + a_{1,3}a_{2,5}a_{3,1}a_{4,4}a_{5,2} \\
&+ a_{1,3}a_{2,5}a_{3,2}a_{4,1}a_{5,4} - a_{1,3}a_{2,5}a_{3,2}a_{4,4}a_{5,1} - a_{1,3}a_{2,5}a_{3,4}a_{4,1}a_{5,2} + a_{1,3}a_{2,5}a_{3,4}a_{4,2}a_{5,1} \\
&- a_{1,4}a_{2,1}a_{3,2}a_{4,3}a_{5,5} + a_{1,4}a_{2,1}a_{3,2}a_{4,5}a_{5,3} + a_{1,4}a_{2,1}a_{3,3}a_{4,2}a_{5,5} - a_{1,4}a_{2,1}a_{3,3}a_{4,5}a_{5,2} \\
&- a_{1,4}a_{2,1}a_{3,5}a_{4,2}a_{5,3} + a_{1,4}a_{2,1}a_{3,5}a_{4,3}a_{5,2} + a_{1,4}a_{2,2}a_{3,1}a_{4,3}a_{5,5} - a_{1,4}a_{2,2}a_{3,1}a_{4,5}a_{5,3} \\
&- a_{1,4}a_{2,2}a_{3,3}a_{4,1}a_{5,5} + a_{1,4}a_{2,2}a_{3,3}a_{4,5}a_{5,1} + a_{1,4}a_{2,2}a_{3,5}a_{4,1}a_{5,3} - a_{1,4}a_{2,2}a_{3,5}a_{4,3}a_{5,1} \\
&- a_{1,4}a_{2,3}a_{3,1}a_{4,2}a_{5,5} + a_{1,4}a_{2,3}a_{3,1}a_{4,5}a_{5,2} + a_{1,4}a_{2,3}a_{3,2}a_{4,1}a_{5,5} - a_{1,4}a_{2,3}a_{3,2}a_{4,5}a_{5,1} \\
&- a_{1,4}a_{2,3}a_{3,5}a_{4,1}a_{5,2} + a_{1,4}a_{2,3}a_{3,5}a_{4,2}a_{5,1} + a_{1,4}a_{2,5}a_{3,1}a_{4,2}a_{5,3} - a_{1,4}a_{2,5}a_{3,1}a_{4,3}a_{5,2} \\
&- a_{1,4}a_{2,5}a_{3,2}a_{4,1}a_{5,3} + a_{1,4}a_{2,5}a_{3,2}a_{4,3}a_{5,1} + a_{1,4}a_{2,5}a_{3,3}a_{4,1}a_{5,2} - a_{1,4}a_{2,5}a_{3,3}a_{4,2}a_{5,1} \\
&+ a_{1,5}a_{2,1}a_{3,2}a_{4,3}a_{5,4} - a_{1,5}a_{2,1}a_{3,2}a_{4,4}a_{5,3} - a_{1,5}a_{2,1}a_{3,3}a_{4,2}a_{5,4} + a_{1,5}a_{2,1}a_{3,3}a_{4,4}a_{5,2} \\
&+ a_{1,5}a_{2,1}a_{3,4}a_{4,2}a_{5,3} - a_{1,5}a_{2,1}a_{3,4}a_{4,3}a_{5,2} - a_{1,5}a_{2,2}a_{3,1}a_{4,3}a_{5,4} + a_{1,5}a_{2,2}a_{3,1}a_{4,4}a_{5,3} \\
&+ a_{1,5}a_{2,2}a_{3,3}a_{4,1}a_{5,4} - a_{1,5}a_{2,2}a_{3,3}a_{4,4}a_{5,1} - a_{1,5}a_{2,2}a_{3,4}a_{4,1}a_{5,3} + a_{1,5}a_{2,2}a_{3,4}a_{4,3}a_{5,1} \\
&+ a_{1,5}a_{2,3}a_{3,1}a_{4,2}a_{5,4} - a_{1,5}a_{2,3}a_{3,1}a_{4,4}a_{5,2} - a_{1,5}a_{2,3}a_{3,2}a_{4,1}a_{5,4} + a_{1,5}a_{2,3}a_{3,2}a_{4,4}a_{5,1} \\
&+ a_{1,5}a_{2,3}a_{3,4}a_{4,1}a_{5,2} - a_{1,5}a_{2,3}a_{3,4}a_{4,2}a_{5,1} - a_{1,5}a_{2,4}a_{3,1}a_{4,2}a_{5,3} + a_{1,5}a_{2,4}a_{3,1}a_{4,3}a_{5,2} \\
&+ a_{1,5}a_{2,4}a_{3,2}a_{4,1}a_{5,3} - a_{1,5}a_{2,4}a_{3,2}a_{4,3}a_{5,1} - a_{1,5}a_{2,4}a_{3,3}a_{4,1}a_{5,2} + a_{1,5}a_{2,4}a_{3,3}a_{4,2}a_{5,1}
\end{aligned}
$$

Pour terminer, mettons en évidence la croissance du nombre de termes composant ces différents déterminants. Réalisons cette tâche avec une boucle itérative `for`.

```
> n:=8;  # Attention, ne pas prendre n plus grand... car un temps de
  calcul de plus en plus interminable
```

$$n := 8 \tag{6.8}$$

```
> L:=[]:
  for k from 2 to n do
   A:=Matrix(k,symbol=a):
   L:=[op(L),nops(Determinant(A))]
  od:
> printf(`            Nombres de termes développées selon `);
```

```
printf(`\n                    l'ordre du déterminant`);
printf(`\n   ======================================================`);
printf(`\n   |   Ordre |    Nombre de termes  |      Ordre !     |`);
printf(`\n   ======================================================
\n`);
for k in [$2..n]
 do
 printf(`   | %5d   |     %8d         |    %8d      |\n`,
 k,
 L[k-1],
 k!);
od;
```

Nombres de termes développées selon
l'ordre du déterminant

Ordre	Nombre de termes	Ordre !
2	2	2
3	6	6
4	24	24
5	120	120
6	720	720
7	5040	5040
8	40320	40320

Il semble qu'il y ait une croissance factorielle dans le nombre de termes pour le développement du déterminant.

```
> seq(n!,n=2..8);
```

$$2, 6, 24, 120, 720, 5040, 40320 \tag{6.9}$$

Ce petit développement nous permet de déduire qu'en développant un déterminant d'une matrice selon la ligne ou la colonne de son choix, nous avons

– $n!$ termes

– $n!-1$ additions

– $(n-1)n!$ multiplications.