CH 5 : Manipulation de matrices dans **Scilab**

I. Les matrices en mathématiques

En préambule nous présentons une partie du vocabulaire et des opérations mathématiques sur les matrices

I.1. Définition

- Une matrice est un tableau rectangulaire (éventuellement carré) de nombres. Un tel tableau est caractérisé par :
  - × son nombre n de lignes,
  - × son nombre p de colonnes.
- Une matrice à n lignes et p colonnes est dite de **taille** $n \times p$.
- Une matrice ne possédant qu'une ligne (*i.e.* de taille $1 \times p$) est appelée **matrice ligne** ou **vecteur ligne**.
- Une matrice ne possédant qu'une colonne (*i.e.* de taille $n \times 1$) est appelée **matrice colonne** ou **vecteur colonne**.

Considérons, par exemple, les trois matrices d'entiers U, V et W suivantes.

$$U = \begin{pmatrix} 1 & 0 & -1 \\ 0 & -2 & 7 \\ 5 & 1 & 0 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 3 & 0 \\ 4 & -1 & -2 \end{pmatrix} \qquad W = \begin{pmatrix} 3 & -4 \\ 2 & 1 \\ 1 & 0 \\ 9 & 8 \end{pmatrix}$$

U est une matrice 3×3 (3 lignes et 3 colonnes), V est une matrice 2×3 (2 lignes et 3 colonnes) et W est une matrice 4×2 (4 lignes et 2 colonnes).

I.2. Opérations sur les matrices

I.2.a) Somme de matrices

- La somme de deux matrices A et B est une opération définie uniquement pour des matrices de même taille (même nombre de lignes et même nombre de colonnes). On note $A + B$ la somme de A et B.
- La somme de deux matrices est la somme terme à terme des cœfficients des matrices.

Si l'on reprend l'exemple précédent, nous ne pouvons réaliser la somme de deux matrices différentes car elles sont de tailles différentes deux à deux. Voici un exemple (en couleur le calcul du cœfficient $(2,3)$: $-2+7=5$) :

$$\begin{pmatrix} 1 & 3 & 0 \\ 4 & -1 & -2 \end{pmatrix} + \begin{pmatrix} -5 & 4 & 2 \\ 2 & 1 & 7 \end{pmatrix} = \begin{pmatrix} -4 & 7 & 2 \\ 6 & 0 & 5 \end{pmatrix}$$

I.2.b) Produit d'un réel et d'une matrice

- Pour tout réel λ et toute matrice A on peut définir le produit de λ et A. On note $\lambda \cdot A$ (et pas $A \cdot \lambda$) cette opération.
- Le résultat est obtenu en multipliant tous les cœfficients de A par λ.

En reprenant l'exemple précédent :

$$5 \cdot V = 5 \cdot \begin{pmatrix} 1 & 3 & 0 \\ 4 & -1 & -2 \end{pmatrix} = \begin{pmatrix} 5 & 15 & 0 \\ 20 & -5 & -10 \end{pmatrix}$$

I.2.c) Produit de matrices

- Le produit de deux matrices A et B est défini à la seule condition que le nombre de colonnes de A soit égal au nombre de lignes de B.
  On note alors $A \times B$.
- Autrement dit, si A est de taille $n \times p$, B doit obligatoirement être de taille $p \times q$ (avec n, p et q quelconques). La matrice obtenue est alors de taille $n \times q$.

Si l'on reprend les matrices exemples précédentes :

× les produits $U \times V$, $V \times W$, $U \times W$, et $W \times U$ ne sont pas possibles,

× les produits $V \times U$ et $W \times V$ sont réalisables.

Nous donnerons la définition formelle de produit en cours de mathématiques. Nous la détaillons ici à l'aide de l'exemple du produit $T = V \times U$.

- Comme V est de taille 2×3 et U est de taille 3×3, la matrice produit, notée T, est de taille 2×3.
- De manière générale, le cœfficient (i, j) de la matrice produit T est obtenue en combinant la $i^{\text{ème}}$ ligne de V et la $j^{\text{ème}}$ colonne de U.
- Par exemple, le cœfficient $T(2, 1)$ (en vert) est obtenue en combinant la $2^{\text{ème}}$ ligne de V (en bleu) et la $1^{\text{ère}}$ colonne de U (en jaune).

Le produit des matrices $V \times U$ est représenté comme suit :

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & -2 & 7 \\ 5 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 3 & 0 \\ 4 & -1 & -2 \end{pmatrix} \quad \begin{pmatrix} 1 & -6 & 20 \\ -6 & 0 & -11 \end{pmatrix}$$

× le cœfficient $T(2, 1)$ est donné par la formule :
$T(2, 1) = (4 \times 1) + (-1 \times 0) + (-2 \times 5) = 4 + 0 - 10 = -6$

× le cœfficient $T(1, 1)$ est donné par la formule :
$T(1, 1) = (1 \times 1) + (3 \times 0) + (0 \times 5) = 1 + 0 = 1$

× le cœfficient $T(1, 2)$ est donné par la formule :
$T(1, 2) = (4 \times 0) + (-1 \times (-2)) + (-2 \times 1) = 0 + 2 - 2 = 0$

× ...

II. Les matrices dans Scilab

II.1. Créer une matrice

II.1.a) Syntaxe de base

Il existe de multiples manières de définir une matrice en **Scilab**. La manière la plus basique consiste à écrire un par un l'ensemble des cœfficients. Une matrice se définit alors à l'aide d'une paire de crochets `[ ]` et ses cœfficients sont écrits ligne par ligne. Les éléments d'une même ligne sont séparés par des virgules « , ». Chaque ligne est séparée de la suivante par un point-virgule « ; ».

À titre d'illustration, définissons en **Scilab** les matrices précédentes.

```
--> U = [1,0,-1; 0,-2,7; 5,1,0]
   U =
       1.    0. - 1.
       0. - 2.    7.
       5.    1.    0.

--> V = [1,3,0; 4,-1,-2]; W = [3,-4; 2,1; 1,0; 9,8];
```

II.1.b) Création de matrices par blocs

La méthode précédente consiste à construire une matrice ligne par ligne : on écrit d'abord le premier vecteur ligne (`[1,0,-1]` ici), puis le suivant (`[0,-2,7]`) et ainsi de suite jusqu'au dernier (`[5,1,0]`). On peut d'ailleurs mettre en avant cet aspect en faisant apparaître chaque ligne comme un vecteur, à l'aide d'une paire de crochets (`[ ]`).

```
--> U = [[1,0,-1]; [0,-2,7]; [5,1,0]]
   U =
       1.    0. - 1.
       0. - 2.    7.
       5.    1.    0.
```

On construit ainsi une matrice en juxtaposant d'autres matrices. L'objet obtenu est appelé **matrice par blocs**.

On peut construire une matrice colonne par colonne.

```
--> U = [[1;0;5], [0;-2;1], [-1;7;0]]
    U =
        1.    0. - 1.
        0. - 2.    7.
        5.    1.    0.
```

Les différents blocs constituant la matrice résultat ne sont pas obligatoirement des vecteurs. Voici quelques autres exemples de construction par blocs.

```
--> U = [[1],[0,-1]; [[0;5],[-2,7;1,0]]]
    U =
        1.    0. - 1.
        0. - 2.    7.
        5.    1.    0.

--> U = [[1;0;5], [[0,-1]; [-2;1], [7;0]]]
    U =
        1.    0. - 1.
        0. - 2.    7.
        5.    1.    0.

--> U = [[[1;0], [0,-1;2,7]]; [5,1,0]]
    U =
        1.    0. - 1.
        0. - 2.    7.
        5.    1.    0.
```

On conseille aux lecteurs d'écrire au brouillon les constructions par blocs correspondantes. Par exemple, l'un des appels ci-dessus correspond à la matrice $\begin{pmatrix} 1 & 0 & -1 \\ 0 & -2 & 7 \\ 5 & 1 & 0 \end{pmatrix}$ où l'on a groupé les blocs par couleurs.

II.1.c) Création de matrices par opérateurs

Créer un vecteur avec l'opérateur « : »

L'opérateur « : » permet de créer un vecteur ligne composé de valeurs régulièment espacées. On peut l'utiliser avec l'une des deux syntaxes suivantes.

- `a:b` renvoie le vecteur ligne construit en énumérant les valeurs comprises entre `a` (compris) et `b` (au maximum), avec un pas de 1.
- `a:pas:b` renvoie le vecteur ligne construit en énumérant les valeurs comprises entre `a` (compris) et `b` (au maximum), avec un pas de valeur `pas`.

Le deuxième argument est optionnel. Si on ne le précise pas, il est remplacé par 1, sa valeur par défaut.

Illustrons cette méthode de construction de vecteurs.

```
--> 3:8
    ans  =
        3.    4.    5.    6.    7.    8.

--> 3:1:8
    ans  =
        3.    4.    5.    6.    7.    8.
```

Si `a` est une valeur strictement plus grande que `b`, l'appel `a:b` renvoie le vecteur vide noté `[ ]`.

```
--> 8:3
    ans  =
        [ ]
```

L'élément `b` n'est pas forcément atteint.

```
--> 3:2:8
    ans  =
        3.    5.    7.
```

```
--> 3:6:8
   ans  =
       3.
```

L'élément `pas` est une valeur réelle ou entière. Il peut être positif ou négatif.

```
--> 3:0.7:8
   ans  =
       3.   3.7.   4.4   5.1   5.8   6.5   7.2   7.9
--> 8:-1:3
   ans  =
       8.   7.   6.   5.   4.   3.
```

Créer un vecteur avec l'opérateur `linspace`

L'aide **Scilab** (`help linspace` ou aide en ligne) stipule que l'opérateur `linspace` permet de créer un vecteur ligne de valeurs régulièrement espacées.

- `linspace(a, b)` renvoie un vecteur ligne comprenant 100 valeurs régulièrement espacées entre `a` et `b`.
- `linspace(a, b, n)` renvoie un vecteur ligne comprenant `n` valeurs régulièrement espacées entre `a` et `b`.

Le troisième argument de la fonction `linspace` est donc un argument optionnel dont la valeur par défaut est 100.

Illustrons cette méthode de construction de vecteurs.

```
--> linspace(3,8)
        column 1 to 6
3.   3.05051   3.10101   3.15152   3.20202   3.25253
        column 7 to 11
3.30303   3.35354   3.40404   3.45455   3.50505
```

Ce vecteur ayant 100 éléments, il serait trop long de présenter tous ses cœfficients. On note toutefois que le $100^{\text{ème}}$ élément est bien `8`.

On peut faire deux remarques sur cette commande :

1) calculer le pas utilisé est simple. Il suffit de remarquer qu'il est solution de l'équation $3 + 99x = 8$. Ainsi, le pas est ici de $(8-3)/99 \simeq 0.50501$.

> `linspace(a, b)` crée un vecteur dont les valeurs sont espacées de $\dfrac{\texttt{b}-\texttt{a}}{99}$

2) les éléments du vecteur précédent s'écrivent donc sous la forme $3+k\,\dfrac{8-3}{99}$ où $k \in [\![0, 99]\!]$. Il est donc aisé de savoir si un élément fait partie du vecteur créé. Par exemple, 4 n'appartient pas à ce vecteur. En effet :

$$3 + k\,\frac{8-3}{99} = 4 \;\Leftrightarrow\; k = \frac{99}{5}$$

ce qui est impossible car k est une valeur entière.

Il est aussi possible de préciser le nombre de valeurs souhaitées.

```
--> linspace(3,8,5)
   ans  =
       3.   4.25   5.5   6.75   8.
```

Et aussi d'obtenir une énumération dans le sens décroissant.

```
--> linspace(8,3,5)
   ans  =
       8.   6.75   5.5   4.25   3.
```

> `linspace(a, b, n)` crée un vecteurs dont les valeurs sont espacées de $\dfrac{\texttt{b}-\texttt{a}}{n-1}$

Remarque

Les opérateurs `linspace` et « : » représentent deux réponses différentes au problème de créer des vecteurs lignes de valeurs régulièrement espacées.

- L'opérateur `linspace` met l'accent sur le nombre d'éléments du vecteur. Le pas utilisé s'en déduit.
- L'opérateur « : » met quant à lui l'accent sur le pas utilisé. Les éléments s'en déduisent.

D'autres opérateurs pour créer des matrices

Il existe d'autres opérateurs **Scilab** permettant de créer des matrices. Nous listons ici les principaux.

- `zeros(n,p)` renvoie la matrice de taille `n` × `p` dont les cœfficients sont tous égaux à 0.

```
--> Z=zeros(2,3)
   Z  =
       0.   0.   0.
       0.   0.   0.
```

- `ones(n,p)` renvoie la matrice de taille `n` × `p` dont les cœfficients sont tous égaux à 1.

```
--> O=ones(2,3)
   O  =
       1.   1.   1.
       1.   1.   1.
```

- `eye(n,p)` renvoie la matrice de taille `n` × `p` dont :
  - × les cœfficients diagonaux (ceux qui ont le même indice de ligne et de colonne) valent 1,
  - × les autres cœfficients valent 0.

```
--> E=eyes(2,3)
   E  =
       1.   0.   0.
       0.   1.   0.
```

- `rand(n,p)` renvoie la matrice de taille `n` × `p` dont les cœfficients sont des réels choisis de manière aléatoire (uniforme) dans $[0, 1]$.

```
--> R=rand(2,3)
   R  =
       0.84974   0.87821   0.56084
       0.68573   0.06837   0.66235
```

II.1.d) Résumé des opérateurs de création

Voici un tableau permettant de résumer les principaux opérateurs de création de matrices.

`a:b`	vecteur ligne de valeurs régulièrement espacées entre `a` et `b` avec pas de 1
`a:pas:b`	vecteur ligne de valeurs régulièrement espacées entre `a` et `b` avec un pas de `pas`
`linspace(a, b)`	vecteur ligne de 100 valeurs régulièrement espacées entre `a` et `b` (d'où un pas de `(b-a)/99`)
`linspace(a, b, n)`	vecteur ligne de `n` valeurs régulièrement espacées entre `a` et `b` (d'où un pas de `(b-a)/(n-1)`)
`zeros(n,p)`	matrice de taille `n` × `p` de cœfficients 0
`ones(n,p)`	matrice de taille `n` × `p` de cœfficients 1
`eye(n,p)`	renvoie la matrice de taille `n` × `p` contenant 1 sur la diagonale et 0 ailleurs
`rand(n,p)`	matrice de taille `n` × `p` de cœfficients choisis de manière aléatoire dans $[0, 1]$

II.2. Accéder aux éléments d'une matrice

II.2.a) Accéder à un élément

Numérotation classique des coefficients d'une matrice

La numérotation dite classique est celle donnée en cours de mathématiques : la position d'un cœfficient d'une matrice est fournie par le couple `(i,j)` où `i` désigne le numéro de ligne et `j` désigne le numéro de colonne. Cette numérotation permet l'accès aux éléments de la matrice.

Commençons par rappeler le contenu de la variable `V`.

```
--> V
       1.   3.   0.
       4. - 1. - 2.
```

```
--> V(1,3), V(2,1)
    ans  =
        0.
    ans  =
        4.
```

Lorsqu'on essaie d'accéder à un élément dont la place se situe à l'extérieur d'une matrice, une erreur est levée.

```
--> V(2,5)
  !--error 21
Index invalide.
```

Pour éviter ces erreurs, il suffit de connaître la taille de la matrice considérée. La fonction `size` prend en paramètre une matrice et renvoie sa taille. Le résultat est obtenu sous la forme d'une matrice `1`×`2`.

```
--> size(V), size(W)
    ans  =
        2.   3.
    ans  =
        4.   2.
```

Numérotation linéaire des cœfficients

Dans la numérotation linéaire, chaque cœfficient est identifié non plus par un couple, mais par un seul nombre. Cette numérotation des cœfficients s'effectue colonne par colonne en commençant en haut à gauche de la matrice pour finir en bas à droite.

On obtient alors la numérotation ci-dessous de la matrice V.

$$V = \begin{pmatrix} 1^{\,1} & 3^{\,3} & 0^{\,5} \\ 4^{\,2} & -1^{\,4} & -2^{\,6} \end{pmatrix}$$

Ce que l'on peut vérifier simplement à l'aide des appels **Scilab** suivants. Pour éviter les appels hors index, on pourra se servir de la fonction `length` qui fournit le nombre de cœfficients d'une matrice.

```
--> V(5), V(6)
    ans  =
        0.
    ans  =
      - 2.

--> V(7)
  !--error 21
Index invalide.

--> length(V)
    ans  =
        6.
```

Cette numérotation linéaire d'un objet à deux dimensions illustre la bijection qui existe entre les ensembles $[\![1,n]\!] \times [\![1,p]\!]$ et $[\![1,np]\!]$:

$$h \;:\; \left|\begin{array}{ccccc} [\![1,n]\!] & \times & [\![1,p]\!] & \to & [\![1,np]\!] \\ (i & , & j) & \mapsto & i+n\times(j-1) \end{array}\right.$$

Ainsi, le cœfficient `(1,3)` de la matrice `V` (de taille `2`×`3`) est repéré par le numéro `1 + 2 (3-1) = 5`.

II.2.b) Accéder à plusieurs éléments

Il est aussi possible d'extraire plusieurs éléments à la fois d'une même matrice. Plus précisément, on choisit d'abord les numéros des lignes que l'on souhaite sélectionner (stockés dans un vecteur `uL`) puis les numéros des colonnes que l'on souhaite sélectionner (stockés dans un vecteur `uC`). L'appel `V(uL,uC)` extrait alors les cœfficients à l'intersection des lignes et des colonnes mentionnées. Illustrons ce mécanisme avec les matrices précédentes.

On peut par exemple sélectionner les cœfficients se trouvant sur la 1ère ligne et sur les colonnes 2 et 3. Il suffit de réaliser l'appel suivant.

```
--> V(1,[2,3])
    ans  =
        3.    0.
```

Il est évidemment possible de sélectionner plusieurs lignes.

```
--> V([1,2],[2,3])
    ans  =
        3.    0.
      - 1.  - 2.
```

L'ordre dans lequel la sélection est demandée est pris en compte.

```
--> V([2,1],[2,3])
    ans  =
      - 1.  - 2.
        3.    0.
```

Il est possible de sélectionner plusieurs fois la même colonne (ou ligne).

```
--> V([2,1],[2,3,2])
    ans  =
      - 1.  - 2.  - 1.
        3.    0.    3.
```

Comme les lignes et colonnes sélectionnées sont stockées dans des vecteurs, il est tout indiqué d'utiliser l'opérateur « : ».

```
--> W(2:4,1:2)
    ans  =
        2.    1.
        1.    0.
        9.    8.
```

Dans l'exemple précédent, on a en fait sélectionné toutes les colonnes de `W`. Lorsque c'est le cas, on peut utiliser une syntaxe simplifiée consistant à écrire simplement « : » au lieu du vecteur de sélection.

```
--> W(2:4,:)
    ans  =
        2.    1.
        1.    0.
        9.    8.
```

Enfin, il est aussi possible de sélectionner les cœfficients d'une matrice un par un, à l'aide de la numérotation linéaire.

```
--> V([2,5,6])
    ans  =
        4.    0.  - 2.
```

II.3. Modifier les éléments d'une matrice

Pour modifier les éléments d'une matrice, il suffit de considérer chacun de ses cœfficients comme une variable que l'on peut affecter à une autre valeur.

```
--> V(1,2)=5
    V  =
        1.    5.    0.
        4.  - 1.  - 2.

--> V(2,3)=1
    V  =
        1.    5.    0.
        4.  - 1.    1.
```

Attention, une nouvelle fois, à ne pas confondre le symbole d'affectation « = » avec le symbole d'égalité « == ».

Allocation dynamique de la mémoire

Que se passe-t-il si l'on essaie d'affecter une valeur à un cœfficient en dehors de la matrice ? De manière assez surprenante, un appel du type « `V(1,7)=3` » ne provoque pas la levée d'un message d'erreur. Au contraire, le cœfficient `V(1,7)` est bien ajouté à la matrice `V` qui est complétée avec des `0` afin qu'elle conserve sa structure matricielle.

```
--> V(1,7)=3
   V  =
       1.   5.   0.   0.   0.   0.   3.
       4. - 1.   1.   0.   0.   0.   0.
```

On peut d'ailleurs se servir de cette fonctionnalité pour créer des matrices.

```
--> F(2,3)=5
   F  =
       0.   0.   0.
       0.   0.   5.

--> G(4)=-1
   G  =
       0.
       0.
       0.
     - 1.
```

Cette fonctionnalité a des conséquences en terme de gestion de la mémoire en **Scilab**. Détaillons ce point.

- Dans de nombreux langages de programmation, la mémoire nécessaire à une matrice est allouée une fois pour toute lors de la création de cette matrice. Une tentative de modification d'un cœfficient hors index provoquera alors un message d'erreur.
- En **Scilab**, une premier bloc mémoire est réservé lors de la création d'une matrice. Comme on l'a vu, la taille de la matrice peut être modifiée à la volée et on parle alors d'**allocation dynamique** de mémoire. Le bloc initial pourra alors s'avérer ne plus être suffisamment grand. Si c'est le cas, la matrice sera déplacée dans un nouveau bloc mémoire, ce qui génère une augmentation du temps d'exécution.

II.4. Opérations sur les matrices

II.4.a) Les opérations mathématiques

Les opérations matricielles classiques sont prises en charge par **Scilab**. On peut ainsi effectuer la somme de deux matrices, la multiplication d'une matrice par un réel, la multiplication de deux matrices.

```
--> H=E+F
   H  =
       1.   0.   0.
       0.   1.   5.

--> 2*H
   ans  =
       2.   0.   0.
       0.   2.   10.

--> T=H*U
   T  =
       1.   0. - 1.
       25.   3.   7.
```

Ces opérations sont valides sous réserve des contraintes sur les tailles évoquées précédemment. En cas de non respect, une erreur sera levée.

```
--> H+U
  !--error 8
Addition incohérente.
```

En **Scilab**, il est aussi possible d'additionner une matrice et un réel. Le réel est alors sommé à tous les cœfficients de la matrice.

```
--> S=H+2
    S  =
        3.    2.    2.
        2.    3.    7.
```

Deux autres opérations classiques sont prises en charge dans **Scilab** : le calcul de la transposée et le calcul du rang[1] d'une matrice.

```
--> H'
    ans  =
        1.    0.
        0.    1.
        0.    5.

--> rank(H)
    ans  =
        2.
```

II.4.b) Les opérateurs terme à terme

Si deux matrices `A` et `B` sont de même taille, il est possible de les combiner à l'aide d'opérateurs terme à terme. Effectuer une opération terme à terme (*element-wise* en anglais) consiste à appliquer cet opérateur entre les cœfficients de `A` et `B` situés aux mêmes positions. Les principales opérations arithmétiques peuvent être effectuées terme à terme. Notez que l'opérateur `+` est, de base, un opérateur qui agit terme à terme.

Il ne faut pas confondre opérateurs classiques et ceux opérant terme à terme.

```
--> T*T
  !--error 10
Multiplication incohérente.
```

```
--> T.*T
    ans  =
        1.    0.    1.
        625.    9.    49.

--> U*U
    ans  =
      - 4. - 1. - 1.
        35.    11. - 14.
        5. - 2.    2.

--> U.*U
    ans  =
        1.    0.    1.
        0.    4.    49.
        25.    1.    0.

--> T.^T
    ans  =
        1.            0.  - 1.
        8.882D+34    27.    823543.

--> T./T
  !--error 27
Division par zéro...
```

Et veiller à respecter les contraintes de taille.

```
--> T.*U
  !--error 9999
inconsistent element-wise operation
```

[1] La fonction `rank` apparaît dans le programme officiel d'informatique de 1ère année.
Cependant, le rang d'une matrice n'est défini qu'en 2nde année de mathématiques ...

II.4.c) Les opérateurs `sum` et `prod`

Comme leurs noms l'indiquent, les opérateurs `sum` et `prod` permettent de réaliser respectivement la somme et le produit des cœfficients d'une matrice.

```
--> sum(U), sum(S)
    ans  =
        11.
    ans  =
        19.

--> prod(S)
    ans  =
        504.

--> prod(size(U)) == length(U)
    ans  =
        T
```

II.4.d) Résumé des opérations sur les matrices

`+`	opérateur d'addition
`-`	opérateur de soustraction
`*`	opérateur de multiplication
`.*`	opérateur de multiplication terme à terme
`./`	opérateur de division terme à terme
`.^`	opérateur de puissance terme à terme
`A'`	permet d'obtenir la transposée de la matrice `A`
`sum`	somme tous les cœfficients d'une matrice
`prod`	multiplie tous les cœfficients d'une matrice
`rank`	calcule le rang d'une matrice
`find`	prend en paramètre une matrice booléenne et renvoie les positions des cœfficients dont la valeur est `%t`
`size`	renvoie la taille d'une matrice
`length`	renvoie le nombre d'éléments d'une matrice

II.5. Matrices et types de données

II.5.a) Tout est matrice !

Revenons tout d'abord sur la définition de **Scilab**. Nous l'avons présenté comme un outil de calcul numérique et l'avons opposé aux outils de calcul formel qui permettent de faire du calcul symbolique. Nous pouvons maintenant être plus précis : **Scilab** est un outil conçu pour effectuer des calculs numériques sur **des matrices**. Cet outil a donc été pensé pour rendre la manipulation de matrices simple, rapide et efficace. Les matrices sont d'ailleurs les éléments de base du langage, ce qui est souvent présenté sous la formule :

> « En **Scilab**, tout est matrice »

Pour bien comprendre cette affirmation, effectuons quelques appels.

```
--> size(3.8)
    ans  =
        1.   1.
```

Cette évaluation révèle que l'objet `3.8` (de type `constant`) est en fait une matrice de taille 1×1.
Cette remarque se généralise. Par exemple, les objets `%f` (de type `boolean`) et `"Bonjour !"` (de type `string`) sont aussi des matrices de taille 1×1.

```
--> size(%f), size("Bonjour !")
    ans  =
        1.   1.
    ans  =
        1.   1.
```

Ceci doit paraître étonnant au vu des définitions données au « CH 4 : Types de données en **Scilab** ». En fait, la présentation faite lors de ce précédent chapitre est imprécise et nous la corrigeons ici.

Les principaux types de données en Scilab

`constant`	type de données des matrices dont les cœfficients ont pour valeur des réels **Scilab**
`boolean`	type de données des matrices dont les cœfficients ont pour valeur `%f` ou `%t`
`string`	type de données des matrices dont les cœfficients ont pour valeur des chaînes de caractère

Ces définitions peuvent s'illustrer à l'aide de la fonction `typeof`.

```
--> typeof([3.2,5,4;6,7.1,-1]), typeof(3:8)
    ans  =
        constant
    ans  =
        constant

--> typeof([6>3,%t,5==7])
    ans  =
        boolean

--> typeof(["un mot","ou";"bien une","phrase"])
    ans  =
        string
```

Il est d'ailleurs intéressant de remarquer qu'écrire une matrice contenant des objets de plusieurs types de données est, de base, proscrit en **Scilab**.

```
--> [42,"bonjour"]
  !--error 144
Opération non définie pour les opérandes donnés.
```

II.5.b) Conséquence sur les opérateurs Scilab

Nous venons de voir que les types de données classiques (`constant`, `boolean`, `string`) regroupent des matrices. En conséquence, les opérateurs définis sur ces types de données ont été codés pour manipuler des matrices. Le chapitre précédent nous a permis d'établir la définition de ces opérateurs sur des matrices de taille 1×1, ce qui ne représente qu'un cas particulier. Définissons maintenant ces opérateurs dans le cas général.

Opérations arithmétiques

- Dans ce chapitre, nous avons défini les opérateurs arithmétiques `+`, $\star$, `-` pour des matrices. Rappelons que ces opérations correspondent aux opérations mathématiques et qu'elles sont définies sous certaines conditions sur les tailles des matrices.
- Nous avons de plus présenté les opérateurs `.*`, `./` et `.^` qui sont des opérateurs terme à terme.

Opérateurs de comparaison

Les opérateurs de comparaison (`>`, `<`, `>=`, `<=`, `==`) sont eux aussi codés pour manipuler des matrices. Plus précisément, si `op` est l'un de ces opérateurs de comparaison, et si `A` et `B` sont des matrice de même taille, l'appel `A op B` effectue la comparaison terme à terme de `A` et de `B`. Le résultat est alors une matrice booléenne `C` de même taille que `A` et `B`. Détaillons ce mécanisme sur des exemples.

On peut comparer des matrices de même taille.

```
--> O>H
    ans  =
       F   T   T
       T   F   F
```

Ceci correspond à la comparaison des cœfficients situés à la même position.

```
--> O>=H
    ans  =
        T   T   T
        T   T   F
--> O==H
    ans  =
        T   F   F
        F   T   F
```

Il faut veiller à comparer des matrices de mêmes tailles.

```
--> U>H
  !--error 60
Dimension erronée de l'argument:Dimensions incompatibles.
```

Il y a deux exceptions à cette contrainte d'égalité des tailles :

- on peut tester l'égalité de deux objets de tailles différentes. Le résultat est alors `%t` ou `%f`.

```
--> U==H
    ans  =
        F
```

- on peut comparer une matrice avec un réel. Le réel est comparé à tous les cœfficients de la matrice et le résultat est une matrice de même taille.

```
--> W<2
    ans  =
        F   T
        F   T
        T   T
        F   F
```

Ce dernier mécanisme est souvent associé à l'opérateur `find` qui prend en paramètre une matrice de type `boolean` et renvoie les positions des cœfficients dont la valeur est `%t`. On utilise généralement cette fonction pour trouver les **indices** des cœfficients qui vérifient une condition donnée. Par exemple, on peut chercher tous les cœfficients dont la valeur est plus petite que 2.

Par défaut, le résultat est donné à l'aide de la numérotation linéaire.

```
--> I=find(W<2)
    I  =
        3.   5.   6.   7.
```

L'appel précédent fournit un vecteur d'indices. On peut évidemment accéder aux cœfficients correspondants (cf II.2.b)).

```
--> W(I)
    ans  =
        1.
        -4.
        1.
        0.
```

Ou tout simplement faire appel à l'opérateur `length` pour connaître le nombre de cœfficients qui vérifient la condition initiale.

```
--> length(I)
    ans  =
        4.
```

Fonctions prédéfinies en Scilab

Les fonctions prédéfinies (`sqrt, log, abs, floor, ceil`) sont elles aussi codées pour pouvoir s'appliquer sur des matrices. La fonction considérée est alors appliquée à tous les cœfficients de la matrice.

```
--> sqrt(3:9)
    ans  =
        1.732   2.   2.236   2.449   2.646   2.828   3.
```