I. Introduction à MATLAB

1. Introduction

Le nom MATLAB est la contraction du terme anglais *matrix laboratory* [1]. Ce logiciel est spécialement conçu pour le calcul scientifique et la manipulation de vecteurs et de matrices. Le langage ne nécessite pas de compilation, puisqu'il est interprété. Il est donc traduit en langage machine à l'exécution [2].

MATLAB est plus qu'un langage de programmation, puisqu'il s'agit d'une console d'exécution. Il permet d'exécuter des fonctions, d'attribuer des valeurs à des variables, etc. Il permet d'effectuer des opérations mathématiques, de manipuler des matrices, de faire des graphiques, etc. [3].

2. Environnement de développement

Eléments principaux :

- Command Window
- Workspace
- Current Folder
- Help → extrêmement utile !

Accéder aux commandes précédentes : ↑

Aide sur une commande :

```
help nom_commande
```

3. Les variables

Toutes les variables sont des matrices [2]. La logique de l'indexation est (**li**gne x **col**onne).

Séparateurs :

- de ligne : point virgule ou return
- de colonne : virgule ou espace blanc

Les variables sont déclarées lorsqu'elles sont initialisées :

```
x = 8                                   variable scalaire (1x1)
y = 1.6524981                           variable scalaire (1x1)
vl = [7, 2, 45, 67, 3]                  vecteur ligne (1x5)
vc = [6; 45; 67; 2]                     vecteur colonne (4x1)
z = 'une chaine de caractères'          vecteur ligne (1x25)
```

3.1. La variable « ans »

La variable ans contient la valeur du dernier calcul si celui-ci n'a pas été attribué à une autre variable (ne jamais l'utiliser !) :

```
8 * 3                                   → ans = 24
a = 1 * 2                               → a = 2, ans = 24
```

3.2. Types numériques

Dans MATLAB, la déclaration de types est facultative. Les types sont déterminés lorsque les variables sont initialisées. Ainsi, par défaut, toute variable numérique est à virgule flottante de double précision et les textes sont des chaînes de caractères.

La double précision occupe passablement de place dans la mémoire. Pour de grandes variables, il est donc conseillé de changer le type. Les types existants sont les suivants [4].

Types de données	Taille (bytes)	Gammes de valeurs	Fonctions de conversion
Signed 8-bit integer	1	-2^7 à 2^7-1	`int8()`
Signed 16-bit integer	2	-2^{15} à $2^{15}-1$	`int16()`
Signed 32-bit integer	4	-2^{31} à $2^{31}-1$	`int32()`
Signed 64-bit integer	8	-2^{63} à $2^{63}-1$	`int64()`
Unsigned 8-bit integer	1	0 à 2^8-1	`uint8()`
Unsigned 16-bit integer	2	0 à $2^{16}-1$	`uint16()`
Unsigned 32-bit integer	4	0 à $2^{32}-1$	`uint32()`
Unsigned 64-bit integer	8	0 à $2^{64}-1$	`uint64()`
Double-Precision Floating Point (64 bits)	8	$-1.79769e^{+308}$ à $-2.22507e^{-308}$ et $2.22507e^{-308}$ à $1.79769e^{+308}$	`double()`
Single-Precision Floating Point (32 bits)	4	$-3.40282e^{+038}$ à $-1.17549e^{-038}$ et $1.17549e^{-038}$ à $3.40282e^{+038}$	`single()`
Char	2	0 à 65535	`char()`
Logical (8 bit)	1	0 à 1	`logical()`

3.2.1. Affichage

Attention, on ne voit que 4 chiffres après la virgule lorsque la valeur d'une variable est affichée. Mais il y en a en réalité 16 par défaut [2]. Pour changer cet affichage, on utilise la commande `format`. Exemples avec π [2]:

```
format long                    → 3.14159265358979
format short                   → 3.1416
format long e                  → 3.141592653589793e+00
format short e                 → 3.1416e+00
```

3.3. Les vecteurs

Pour accéder aux éléments d'un vecteur, il faut indiquer l'index de leur emplacement entre parenthèses (en base 1) :

```
a = [5, 23, 57, 89, 111, 4, 23];
a(2) = 0;                      → a = [5, 0, 57, 89, 111, 4, 23]
a(3) = a(4) + a(5);            → a = [5, 0, 200, 89, 111, 4, 23]
```

Si nous désirons accéder à plusieurs éléments pour travailler sur un sous-vecteur, les deux points (`:`) sont utilisés de cette manière :

```
b = a(1:3);                    → b = [5, 0, 200]
b(1:2) = 1;                    → b = [1, 1, 200]
```

La fonction `size()` permet de connaître la taille d'un vecteur ou d'une matrice, selon les deux dimensions :

```
vl = [7, 2, 45, 67, 3]
vc = [6; 45; 67; 2]
size(vl);                      → [1, 5]
size(vc);                      → [4, 1]
```

Puisque les vecteurs n'ont qu'une dimension, la fonction `length()` est plus simple, car elle retourne la plus longue dimension :

```
length(vl);                    → 5
length(vc);                    → 4
```

3.4. Les chaînes de caractères

Il est possible de concaténer des mots à l'aide des parenthèses carrées [3] :

```
mot1 = 'bonjour';
mot2 = 'monde';
phrase = [mot1, ' ', mot2]     → phrase = 'bonjour monde'
```

Il n'est pas possible de mélanger directement les variables numériques et les variables textes. Pour afficher une valeur numérique dans un texte, il est nécessaire de la transformer en texte (string) à l'aide de la fonction `num2str()` :

```
a = 52;
b = a/2;
str = ['La moitié de ', num2str(a), ' est ', num2str(b)];
```

4. Le Workspace

Quelques commandes générales sur les variables [2]:

`who`	affiche toutes les variables.
`whos`	affiche toutes les variables avec taille et type.
`size(a)`	affiche les dimensions de la matrice a.
`clear` *var*	efface la variable var.
`clear all`	efface toutes les variables.

5. Le langage

5.1. Fin de commande

Lorsqu'on élabore un script ou une fonction, on ajoute un point-virgule après chaque commande. Si le point-virgule est oublié, le script fonctionne, mais les résultats de la commande sont affichés dans la fenêtre de commandes :

L'instruction suivante :

```
>> a = pi * 2
```

affichera :

```
a =
      6.283185307179586
```

Alors que :

```
>> a = pi * 2;
```

n'affiche rien. Ceci rend les programmes plus rapides et évite de remplir inutilement la fenêtre de commande.

Pour afficher une valeur de manière propre, on préférera la commande `disp()`.

```
disp(a);                    → 6.283185307179586
```

5.2. Les commentaires

Lorsque nous désirons écrire un commentaire dans notre code, le signe de pourcentage (`%`) permet d'écrire du texte que MATLAB ignorera à l'exécution.

```
% ceci est un commentaire
x = 3 ; % encore un commentaire
```

5.3. Opérateurs basiques

Les opérateurs basiques sont instinctifs. Ils s'utilisent autant sur des variables scalaires que sur des matrices.

```
z = x + y;        addition ou somme matricielle
z = x * y;        multiplication ou produit matriciel
z = x / y;        division ou division matricielle
z = x^a;          puissance ou puissance matricielle (x doit être une matrice carrée)
```

Lors d'une opération matricielle, les matrices doivent être de dimensions cohérentes ! Les opérations éléments par éléments seront étudiées plus tard. Pour transposer une matrice ou un vecteur, l'opérateur (`'`) est utilisé :

```
a = [4, 2, 6];
b = a';                                  → b = [4; 2; 6]
```

Quelques exemples d'opérations sur les vecteurs :

```
a = [34, 56, 5, 8];
b = [23, 7, 34, 7];
c = a * 2;                               → c = [68, 112, 10, 16]
d = a / 4;                               → d = [8.5, 14, 1.25, 2]
e = a + 3;                               → e = [37, 59, 8, 11]
f = a + b;                               → f = [57, 63, 39, 15]
g = a * b;                               → Inner matrix dimensions must agree.
b = b';                                  → b = [23; 7; 34; 7]
g = a * b;                               → g = 1400
```

5.4. Opérateur d'incrémentation

Le double point (`:`) est l'opérateur d'incrémentation dans MATLAB [3].

```
a = [0:1:5];                             → a = [0, 1, 2, 3, 4, 5]
b = [0:0.2:1];                           → b = [0, 0.2, 0.4, 0.6, 0.8, 1]
```

Si le pas d'incrémentation n'est pas précisé, il est de 1 par défaut :

```
a = [0:5];                               → a = [0, 1, 2, 3, 4, 5]
```

6. Les scripts

MATLAB permet d'écrire des programmes, que ce soit des scripts ou des fonctions [2]. Nous nous limiterons ici aux scripts. Un fichier script (script file) est une suite de commandes MATLAB. Ces fichiers texte ont l'extension .m.

Pour créer un script : File > New > M-File.

Nous pouvons écrire dans un script toutes les commandes que nous désirons exécuter consécutivement, comme nous l'aurions fait dans l'invite de commandes.

Pour exécuter un script existant, il faut tout d'abord changer le répertoire courant en sélectionnant le bon répertoire, puis taper le nom du script (sans extension) dans la fenêtre de commande. Ou appuyer sur le bouton Run de l'Editeur.

Il est possible de déboguer un programme en ajoutant des points d'arrêt à l'endroit qui nous intéresse.

Le script modifie les valeurs des variables du workspace. Il ne prend pas d'argument en entrée et ne retourne rien. Ces deux points le différentient des fonctions.

Puisqu'un script partage les mêmes variables que l'invite de commande, il est d'usage de commencer par faire le ménage : `clear all`.

Habituellement, on utilise les fichiers scripts afin de [3] :

- Initialiser le système (fonctions clear)
- Déclarer les variables
- Effectuer les opérations algébriques
- Appeler les fonctions
- Tracer les figures

7. Références

[1] Wikipedia.org

[2] Dellis Jean-Luc. *Introduction à MATLAB*. Université de Picardie.

[3] Hudon Nicolas, 2004. *Initiation à MATLAB*. URCPC, Ecole Polytechnique de Montréal.

[4] MATLAB Help

8. Auteur(s)

Pascal Horton (2009, 2015)