CHAPITRE 2 : PROGRAMMATION AVEC MATLAB

Dans le premier chapitre, nous avons vu comment utiliser la *Command Window* pour exécuter des commandes ou pour évaluer des expressions. Les commandes utilisées s'écrivent généralement sous forme d'une seule instructions (voir plusieurs dans une même ligne). Cependant, lorsque la tâche à réaliser devient plus complexe (plusieurs dizaines de ligne de code) ou que l'on souhaite pouvoir la transmettre à quelqu'un d'autre simplement, on utilise la *fenêtre Editor*. On a recours à l'utilisation des structures de programmation. Ces structures se résument en deux outils incontournables de MATLAB : les scripts et les fonctions. Pour faire des scripts complexes il est souvent nécessaire de faire appel aux structures de programmation qui font l'objectif de notre deuxième chapitre.

Pour visualiser et interpréter des résultats, il est souvent recommandé d'utiliser les graphiques et les outils de visualisation 2D et 3D de MATLAB. On détaillera à la fin de ce chapitre certains de ces outils de visualisation. Nous présentons aussi certaines commandes MATLAB pour le traitement d'images.

1) Généralités

1.1. Variables, affectation :

L'instruction d'affectation $nom \leftarrow valeur$ se code : $nom = valeur$. S'il n'existe pas de variable nommée nom, cette instruction crée une variable appelé nom et lui affecte$valeur$. Sinon, sa valeur sera modifiée (remplacée) par la nouvelle $valeur$. Lorsqu'une variable est créée, son nom apparait dans le *Workspace* avec un icone qui indique la nature de son contenu, et la description de son encombrement mémoire.

Les noms de variables valides commencent par une lettre et sont en un seul mot. MATLAB distingue les majuscules et les minuscules.

Il existe cinq types de variables sous MATLAB : les entiers, les réels, les complexes, les chaines de caractères et le type logique.

1.2. Entrées/Sorties :

Sorties :

1. L'absence du ; à la fin d'une instruction provoque l'affichage du résultat de l'instruction.

```
>> y=3+5
y =
    8
```

2. Invoquer le nom d'une variable (qui existe) sans ; permet l'affichage de son nom et de sa valeur :

```
>> x=3+9;
>> x
>>x =
        12
```

3. L'instruction *disp* : l'instruction $disp(nombre)$ provoque l'affichage du nombre qu'elle reçoit en argument.

```
>> disp(2*15)
  30
```

L'instruction $disp(texte)$ provoque l'affichage du texte qu'elle reçoit en argument.

```
>> disp('hello world!!');
hello world!!
```

L'instruction $disp(nom\ de\ variable)$ provoque l'affichage du contenu de la variable qu'elle reçoit en argument.

```
>> x=(6+4)*15;
>> disp(x);
  150
```

4. Message d'erreur : la commande $error('message')$ permet d'afficher en rouge le texte $message$ et arrête l'exécution en indiquant l'endroit de l'erreur.

```
>> error('message')
??? message
```

Entrées :

1. La commande $x = input('saisir\ une\ valeur')$ affiche à l'écran $saisir\ une\ valeur$ et attend qu'une réponse soit saisie (tapée) au clavier et validée par un $entrée$. La réponse sera affectée à x après la validation.

```
>> x=input('saisir une valeur');          % saisir une valeur au clavier
saisir une valeur 5                       % l'utilisateur tape 5
>> x                                      % 5 est affecté a x
x =
   5
```

1.3. Commentaires : les caractères qui suivent % ne sont pas lus par MATLAB. Cette fonctionnalité sert à écrire des commentaires (explications) à l'usage des utilisateurs.

```
>> x=input('saisir une valeur');          %la commande input permet de saisir une valeur au clavier
saisir une valeur 6
```

1.4. Les expressions logiques : le langage MATLAB dispose en standard de deux valeurs logiques : **0** représente **Faux/ False** et **1** qui représente **Vrai/ True**.

Un test ou expression logique est une expression qui retourne une valeur logique. Le **Tableau 2.1** montre les opérateurs de comparaison qui s'appliquent sur des variables numériques réelles et entre caractères.

Opérateur	**Objectif**
Opérateurs arithmétiques	
+	**Addition**
-	**Soustraction**
*	**Multiplication**

/	**Division**
^	**puissance**
Opérateurs relationnels	
==	Egalité
~=	Inégalité
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
Opérateurs logiques	
&	**Et** logique
|	**Ou** logique
~	**Non** (négation) logique

Tableau 2.1 : Opérateurs arithmétiques, relationnels et logiques sous MATLAB

Le **Tableau 2.2** résume le fonctionnement des opérations logiques :

x	y	x & y	x | y	~x
1	**1**	1	1	0
1	**0**	0	1	0
0	**1**	0	1	1
0	**0**	1	0	1

Tableau 2.2: Fonctionnement des opérations logiques

Exemples d'utilisation des opérateurs :

```
>> x=15;
>> y=30;
>> x<y                    %x=15 est inférieur à y=30 affiche 1 vrai
ans =
    1
>> x<=y                   %x=15 est inférieur à y=30 affiche 1 vrai
ans =
    1
>> x==y                   %x est différent de y donc affiche 0 faux
ans =
    0
>> (0<x)&(y<50)           %30 affiche 1 vrai
ans =
    1
>> (x>15)|(y>55)          %affiche 0 faux
ans =
    0
>> ~(x>15)                % affiche 1 vrai- la négation de x>15 qui est faux (0)
ans =
    1
>> 10&1                   % 10 est considéré comme vrai 1. 1&1=1 vrai
ans =
    1
>> 10&0                   %1&0=0 faux
ans =
    0
```

Noté bien :

- Si x et y sont de type caractère (l'affectation est un caractère), alors, les commandes :
  1) $x == y$ retourne **1** si la valeur affecté à la variable x est égale à celle affectée à y et **0** sinon ;
  2) $x < y$ renvoie **1** si la valeur de x précède celle de y dans l'ordre alphabétique conventionnel utilisé par MATLAB, **0** sinon. L'ordre utilisé par MATLAB utilise les conventions suivantes : $'\,' < \cdots <' Z \ldots <' a' <' b' < \cdots <' z' \; et \; '0' <' 1' < \cdots <' 9'$. Les caractères accentués sont plus loin dans la table.
- On peut combiner les expressions logiques à l'aide des trois opérateurs logiques (&, |, ~). Par exemple l'expression $(N > 0)\&(N == round(N))$ retourne **1** (vrai) si la variable réelle N contient un entier strictement positif et **0** (faux) sinon.
- La comparaison des vecteurs et des scalaires diffère des scalaires. Il existe des fonctions prédéfinies qui font ces comparaisons comme : $isequal, isempty$. $isequal$ Teste si deux (voir plusieurs) matrices sont égaux (ayant les mêmes éléments partout, elle renvoie 1 si oui sinon 0). $isempty$ teste si une matrice est vide (ne contient pas d'éléments), elle renvoie 1 si oui sinon 0.

```
>> A=[1 2 3;4 5 6]          %définition de A
A =
   1   2   3
   4   5   6
>> B=[1 2 3;4 5 6]          %définition de A
B =
   1   2   3
   4   5   6
>> A==B                     % tester si A et B sont égaux (leurs éléments sont-ils égaux ou non ?)
ans =
   1   1   1
   1   1   1
>> isequal(A,B)             % tester si A et B sont égaux aussi (équivalente a A==B)
ans =
   1
>> M=[];                    % définition d'une matrice M vide
>> isequal(M,A)
ans =
   0
>> isempty(M)               % tester si la matrice M est vide ?
ans =
   1
```

2) Création de programmes

Un programme MATLAB est un fichier stocké avec le suffixe $.\boldsymbol{m}$ ou $.\boldsymbol{M}$. Un fichier programme est appelé le script de ce programme. Les noms de fichiers doivent être en un seul mot (pas d'espace) et sans accent, ni tiret ni point (sauf le .M). On peut utiliser le caractère de soulignement ' _'. On utilise la fenêtre $Editor$ pour créer un script. Un script est une suite de commande que l'on aurait tout aussi bien pu taper dans la *Command Window*.

Pour créer un nouveau script, suivez les étapes suivantes :

1. Ouvrir Editor-Debugger par : $file \rightarrow new \rightarrow M.file$ ou $cliquer\ sur\ l'icone\ page\ blanche$;
2. Tapez une suite d'instructions dans l'éditeur ;
3. Sauvegarder par l'un des procédés suivants : $file \rightarrow save\ as$ ou $file \rightarrow save$ ou $icone\ disquette$.

Pour exécuter un programme, invoquer son nom (le nom sous lequel il a été enregistré) sans l'extension $.m$ dans la $Command\ Window$ ou le bouton run dans 'éditeur.

Pour modifier un programme suivre le schéma :

1. Ouvrir le fichier : $file \rightarrow open \rightarrow double\ clique\ sur\ le\ fichier\ concern\acute{e}$ ou $icone\ open$
2. Effectuer les corrections ou les modifications voulus, puis sauvegarder.

Noté bien :

- Un programme peut appeler un autre programme ou s'appeler lui-même ;
- Lorsqu'un programme ne s'arrête pas, ($plantage$). On quitte un plantage avec MATLAB en maintenant pressées les deux touches $CTRL$ et C, ou en provoquant une erreur d'entrée (en tapant une lettre accentuée) sans apostrophes lors de l'exécution d'une instruction input, sinon il faut fermer la session ou même éteindre l'ordinateur.

Exemples d'un script :

1. Programme qui affiche Hello World !! à l'écran. Le code MATLAB équivalent est le suivant sauvegardé sous le nom $pgm1.m$

```
str='Hello World !!';      %variable chaine de caractères qui contient hello World !!
str                        % affichage de la variable str
```

pour exécuter aller sur le bouton run, ou invoquer le nom du programme (nom de sauvegarde)

```
>> pgm1
str =
Hello World !!
```

2. écrire un programme qui demande deux nombres a et b, puis calcule et affiche le quotient a/b.

On peut déduire l'algorithme équivalent : $\begin{cases} lire(a,b) \\ c \leftarrow a/b \\ afficher(c) \end{cases}$

Le code MATLAB (à taper dans $Editor - Debugger - Window$) est :

```
disp('donner deux nombres pour calculer le quotient a/b');
A=input('tapez a=  ');
B=input('      b=  ');
C=A/B;
disp('le quotient = ');
```

```
disp(C);
```

On exécute dans la $Command\ Window$ de la même façon que l'exemple 1 :

```
>> pgm2
donner deux nombres pour calculer le quotient a/b
tapez a=  9
    b=  3
le quotient =
   3
```

3) Création d'une fonction

3.1. Définitions

Une fonction c'est un script qui commence par le mot $\boldsymbol{function.}$ Une fonction permet d'étendre les possibilités au-delà des fonctions préprogrammées par les développeurs de MATLAB. Elle reçoit en entrée $zéro, un, deux\ ou\ plus$ ... variables d'entrées appelées aussi $paramètres\ données$. Elle donne en sortie $zéro, ou\ une$ variable appelée aussi $paramètre\ résultat$. Les variables d'entrée et de sortie peuvent être des matrices. Attention, lors de l'appel d'une fonction, c'est l'ordre des variables d'entrée et non pas leur nom qui détermine leur utilisation.

On utilise la syntaxe suivante pour créer une fonction dont :

- le nom est $calcul$;
- qui prend en variables d'entrée $x_1, x_2, \dots, x_n$;
- qui a comme résultat Y.

$$function\ Y = calcul(x_1, x_2, \dots, x_n)$$

$$description\ du\ procédé\ de\ calcul\ de\ Y\ au\ moyen\ de\ x_1, x_2, \dots, x_n$$

Exemples de fonctions :

1. écrivez la fonction fonc1 qui modifie le programme $pgm1$ précédent pour qu'il affiche : $Hello\ Nawel$

```
function [str]=fonc1(prenom)
str=['hello ', prenom];
end
```

2. on sauvegarde la fonction sous le nom $hello.m$. puis on l'appelle depuis la $Command\ Window$ pour son exécution, mais cette fois avec un paramètre en entré de type chaine de caractère:

```
>> hello('Nawel')
ans =
hello Nawel
```

3. Écrivez la fonction équivalente au script 2 :

```
function[c]=qot(a,b)
c=a/b;
end
```

4. en $Command\ Window$ en exécute la fonction en donnant les deux valeurs de a et b :

```
>> pgm2(9,3)
```

```
ans =
    3
```

3.2. Variable formelle et variable effective, variable locale et globale

5. **Variable formelle :** dans la fonction *qot* , les variables a, b, c sont des *variables formelles*, elle n'ont pas d'existence effective. Même après appel de *qot*, en regardant *Workspace*, on voit qu'elles ne sont pas dans la liste des variables reconnues par CW. L'exemple montre ceci :

```
>> pgm2(9,3)
ans =
    3
>> a
??? Undefined function or variable 'a'.
```

Retenir: *les variables formelles d'entrée et de sortie ne servent qu'à indiquer le procédé de calcul. Elles ne sont reconnues que dans le script de la fonction.*

6. **Variable effective :** c'est une variable reconnue par *Command Window* . Dans l'exemple suivant les variables x, y et z sont effectives. Elles sont dans la liste des variables fournie par *Workspace*.

```
>> x=9;
>> y=3;
>> z=pgm2(x,y)
z =
    3
>> x
x =
    9
>> y
y =
    3
```

1) Variable locale : outre les variables formelles et effectives, toutes les autres variables utilisées par une fonction sont classées en deux types : *locales* ou *globales*. Dans MATLAB une variable qui n'est pas explicitement déclarée globale est une variable locale.

On déclare une variable globale comme suit : ***global x***

Lorsqu'une variable est locale à une fonction, elle n'est reconnue que par les actions de cette fonction. Lorsqu'une variable est déclarée globale dans plusieurs scripts, elle est partagée par tous les scripts.

Noté bien :

✍ Le **Tableau 2.3** montre une comparaison entre une fonction et un script MATLAB. Cette comparaison concerne la structure du programme et la façon de l'exécuter ou de l'invoquer dans la *Command Window*.

Un programme (script)	Une fonction
Structure	

x=input('donner un nombre positif: ');
y=input('donner un nombre positif: ');
z=x+y;
disp('la somme = ');
disp(z);
	function z= somme(x,y)
	z=x+y;
	end

Exécution	
>> **pgm2** %le nom du programme suffit donner un nombre positif: 5 donner un nombre positif: 15 la somme = 20	>> **somme(3,7)** % il faut les paramètres d'entrées ans = 10
Utilisation	
>> 2*pgm2+4 %utilisation interdite ??? Attempt to execute SCRIPT pgm2 as a function.	>> 2***somme(5,4)**+5 %utilisation permise ans = 23

Tableau 2.3: Comparaison entre un programme et une fonction MATLAB

4) Les structures de contrôle

4.1. Alternative if :

Les instructions $si\ (test) alors < instruction >$ se traduit comme suit sous MATLAB, mais aucun mot ne traduit $alors$. Il y a trois versions qui traduit cette instruction :

1) **Version courte (conditionnelle simple):** Si test est une expression logique, le code de la structure :

$$\begin{cases} si\ test\ alors \\ instruction \\ fin \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test \\ instruction \\ end \end{cases}$$

Exemple : La fonction suivante affiche $Bonjour$!! si le nom passé en argument ='Nawel', ne fait rien autrement.

```
function nom(x)
if x=='nawel'
disp('Bonjour!!');
end
```

L'exécution dans la $Command\ Window$ donne:

```
>> pgm2('nawel')                %si le nom = nawel le programme affiche Bonjour
Bonjour!!

>> pgm2('laval')        %si le nom est different de nawel, le programme n'affiche rien
```

2) **Version normale (conditionnelle alternative):** Si test est une expression logique, le code de la structure :

$$\begin{cases} si\ test\ alors \\ instruction1 \\ sinon \\ instrcution2 \\ fin\ (si) \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test \\ instruction1 \\ else \\ instruction2 \\ end \end{cases}$$

Exemple : le programme suivant permet d'afficher si un nombre est positif ou négatif

```
x=input('donner nombre');
if x<0
   disp('nombre négatif');
else
   disp('nombre positif');
end
```

En exécutant dans la $Command\ Window$, on aura :

```
>>donner nombre-5
nombre négatif
>>donner nombre6
nombre positif
>>donner nombre0
nombre positif
```

3) Version compliqué (conditionnelle imbriquée): quand on dispose de trois conditions (cases) ou plus, on a recours à l'utilisation des si dans les rubriques sinon pour achever tous les conditions. La structure est la suivante :

$$\begin{cases} si\ test1\ alors \\ instruction1 \\ sinonsi\ test2\ alors \\ instruction2 \\ sinon \\ instrcution3 \\ fin\ (si) \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test1 \\ instruction1 \\ elseif\ test2 \\ instruction2 \\ else \\ instruction3 \\ end \end{cases}$$

Exemple : reprenant le même exemple dans 2 mais on ajoutant le cas où le nombre est nul.

```
x=input('donner nombre');
if x<0
   disp('nombre négatif');
elseif x>0
   disp('nombre positif');
else
   disp('nombre nul');
end
```

En exécutant dans la $Command\ Window$, on aura :

```
>>donner nombre9
nombre positif
>>donner nombre-9
nombre négatif
>>donner nombre0
nombre nul
```

4.2. Instruction de Switch : exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé à une clause **case** qui définit si ce groupe doit être exécutée ou non. Si tous les cas n'ont pas été acceptés, il est possible d'ajouter une clause **otherwise** qui sera exécutée dans ce cas. La structure de cette instruction est :

$$
\begin{array}{|c}
switch\ (expression) \\
case\ valeur\_1 \\
instructions\ 1 \\
case\ valeur\_2 \\
instrcution\ 2 \\
\dots \\
case\ valeur\_n \\
instrcution\ n \\
otherwise \\
autre\ instruction
\end{array}
$$

Exemple : prenant un exemple qui affiche plusieurs cas selon la valeur de x saisie au clavier :

```
x=input('donner nombre');
switch(x)
   case 0
      disp('x = 0');
   case 1000
      disp('x = 1000');
   case 2000
      disp('x = 2000');
   otherwise
      disp('x n''est pas 0 ni 1000 ni 2000');
end
```

L'exécution dans la *Command Window* donne comme résultat:

```
>>donner nombre0
x = 0
>>donner nombre1000
x = 1000
>>donner nombre2000
x = 2000
>>donner nombre5000
x n'est pas 0 ni 1000 ni 2000
```

4.3. Les boucles (répétitions):

Une boucle permet de répéter la même commande un grand nombre de fois en faisant varier un paramètre. On verra deux types de boucles : boucles **while (tantque)** et **for (pour).**

1) La boucle while

Répète l'exécution d'un ensemble d'instructions un nombre indéterminé de fois, selon l'évaluation d'une condition logique. Elle s'écrit syntaxiquement comme suit :

$$
\begin{array}{|c}
\boldsymbol{tant\ que\ test\ faire} \\
\boldsymbol{instructions} \\
\boldsymbol{fin\ (tantque)}
\end{array}
\qquad \text{Devient en MATLAB :} \qquad
\begin{array}{|c}
\boldsymbol{while\ (condition)} \\
\boldsymbol{instructions} \\
\boldsymbol{end}
\end{array}
$$

Tant que la condition du **while** (expression) est correcte (évaluée par un **true**) la partie **instructions** sera exécutée encore.

Exemple : écrire une fonction somme qui prend en entrée un réel $\in]0, 20]$, donne un message d'erreur si $x \notin]0, 20]$, sinon calcule et donne en sortie le plus petit n tel que la somme $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ est supérieure à x.

Analyse :

La structure algorithmique de la fonction est :

$$\left| \begin{array}{c} \boldsymbol{si\ x \leq 0\ ou\ x \geq 20\ alors\ erreur} \\ \boldsymbol{sinon} \begin{cases} \boldsymbol{initialiser\ n\ a\ 0\ et\ S\ } \text{à } \boldsymbol{0\ (somme\ de\ 0\ termes)} \\ \boldsymbol{tantque\ S_n \leq x\ (c'est\ } \text{à } \boldsymbol{dire\ S < x) faire} \\ \begin{cases} \boldsymbol{n \leftarrow n + 1} \\ \boldsymbol{S \leftarrow S + {}^{1}/_{n}} \end{cases} \end{cases} \end{array} \right.$$

L'arrêt de la répétition a lieu pour la première valeur n telle que $S_n > x$.

Le code MATLAB équivalent est :

```
function N=somme(x)
if(x<=0) | (x>=20)
   error('x non conforme');
else
   N=0; S=0;              %initialisation de N et S
   while S<=x
      N=N+1;              %N est incrémenté (augmente) de 1
      S=S+1/N;             % $S_n$ est remplacée par $S_n+1$
   end                   %end while
end                      %end if
```

2) La boucle for

La boucle for répète l'exécution d'un groupe d'instructions un nombre défini de fois (le nombre de répétitions voir itération est connu à l'avance). Une boucle for se définie syntaxiquement comme suit :

$$\left| \begin{array}{c} \boldsymbol{pour\ i = m1 : h : m2\ faire} \\ \boldsymbol{instrcution} \\ \boldsymbol{fin\ (pour)} \end{array} \right. \quad \text{Devient en MATLAB :} \quad \left| \begin{array}{c} \boldsymbol{for\ i = m1 : h : m2} \\ \boldsymbol{instructions} \\ \boldsymbol{end} \end{array} \right.$$

- La boucle effectue *instructions* pour les valeurs i comprises entre $m1$ et $m2$ suivantes : $m1, m1 + h, m1 + 2h$ et ainsi de suite. Par contre si $h > 0\ et\ m1 > m2$ alors *instructions* n'est pas effectuée ; de même si $h < 0\ et\ m1 < m2$.
- Si h=1, on ne peut pas l'écrire, donc les écritures $for\ i = m1 : 1 : m2$ et $for\ i = m1 : m2$ sont équivalentes.

Exemple : écrire une fonction $factorielle(n)$ qui prend comme entrée un entier $n \geq 0$, et donne en sortie n !

Analyse : La structure algorithmique de la fonction est :

$$\begin{array}{|c} \boldsymbol{donner\ n\ comme\ paramètre\ d'entréede\ la\ fonction} \\ \boldsymbol{initialiser\ f\ à\ 1\ (f \leftarrow 1)} \\ \boldsymbol{pour\ i\ allant\ de\ 1\ à\ n\ faire} \\ \boldsymbol{f \leftarrow f * i} \\ \boldsymbol{afficher\ f} \end{array}$$

Le code MATLAB équivalent est :

```
function f=factorielle(n)
f=1;
for i=1:n
   f=f*i;
end
```

3) Sorties continue et break en cours d'instruction

Dans les structures de boucles, un bloc d'instructions est exécuté un certain nombre de fois, et ce nombre de fois peut même être infini dans le cas de boucles while.

Deux instructions permettent d'interrompre l'exécution d'un bloc d'instructions d'une boucle.

1. **Continue:** interrompt l'exécution du bloc d'instructions en cours d'exécution, et passe à l'itération suivante de la boucle for ou while.
2. **Break:** interrompt l'exécution du bloc d'instructions en cours d'exécution, et sort totalement de la boucle for ou while, en ignorant les itérations suivantes

Exemple 1 (continue): le programme MATLAB qui permet d'afficher les nombres divisibles par 7 compris entre 7 et 50. L'instruction continue permet si un nombre non divisible par 7 d'ignorer son affichage avec l'instruction disp et passe au nombre suivant.

```
for n = 1:50
   if mod(n,7)
      continue
   end
   disp(['Divisible by 7: ' num2str(n)])
end
```

L'exécution sur CW donne :

Divisible by 7: 7
Divisible by 7: 14
Divisible by 7: 21
Divisible by 7: 28
Divisible by 7: 35
Divisible by 7: 42
Divisible by 7: 49

Exemple 2 (break):

```
c=input('donner un nombre des nombres')
n=0;
while n<=c
   x=input('siaisr un nombre ');
   if x==0
      break
   end
   disp(['le nombre saisie est',num2str(x)])
   n=n+1;
end
```

L'exécution sur CW donne :

donner un nombre des nombres5
c =5
siaisr un nombre 3
le nombre saisie est3
siaisr un nombre 4
le nombre saisie est4
siaisr un nombre 0