MATLAB

Chapitre III : Introduction à la programmation avec Matlab

Nous avons vu jusqu'à présent comment utiliser Matlab pour effectuer des commandes ou pour évaluer des expressions en les écrivant dans la ligne de commande (Après le prompt >>), par conséquent les commandes utilisées s'écrivent généralement sous forme d'une seule instruction (éventuellement sur une seule ligne).
Cependant, il existe des problèmes dont la description de leurs solutions nécessite plusieurs instructions, ce qui réclame l'utilisation de plusieurs lignes. Comme par exemple la recherche des racines d'une équation de second degré (avec prise en compte de tous les cas possibles).
Une collection d'instructions bien structurées visant à résoudre un problème donnée s'appelle un programme. Dans cette partie du cours, on va présenter les mécanismes d'écriture et d'exécution des programmes en Matlab.

1. Généralités :

1.1 Les commentaires :

Les commentaires sont des phrases explicatives ignorées par Matlab et destinées pour l'utilisateur afin de l'aider à comprendre la partie du code commentée.
En Matlab un commentaire commence par le symbole % et occupe le reste de la ligne.
Par exemple :

```
>> A=B+C ;            % Donner à A la valeur de B+C
```

1.2 Écriture des expressions longues :

Si l'écriture d'une expression longue ne peut pas être enclavée dans une seule ligne, il est possible de la diviser en plusieurs lignes en mettant à la fin de chaque ligne au moins trois points.
Exemple :

```
>> (sin(pi/3)^2/cos(pi/3)^2)-(1-2*(5+sqrt(x)^5/(-2*x^3-x^2)^1+3*x)) ;
```

Cette expression peut être réécrite de la façon suivante :

```
>> (sin(pi/3)^2/cos(pi/3)^2)- ...  ↵
>> (1-2*(5+sqrt(x)^5 .....              ↵
>> /(-2*x^3-x^2)^1+3*x)) ;              ↵
```

1.3 Lecture des données dans un programme (Les entrées) :

Pour lire une valeur donnée par l'utilisateur, il est possible d'utiliser la commande ***input***, qui a la syntaxe suivante :

```
variable = input ('une phrase indicative')
```

- `variable` : La valeur déposée par l'utilisateur sera mise dans cette variable
- `'une phrase indicative'` : Une phrase aidant l'utilisateur à savoir quoi entrer

Quand Matlab exécute une telle instruction, La phrase indicative sera affichée à l'utilisateur en attendant que ce dernier entre une valeur.

Par exemple :

```
>> A = input ('Entrez un nombre entier : ')          ↵
   Entrez un nombre entier : 5                       ↵
   A =
      5
>>

>> A = input ('Entrez un nombre entier : ');         ↵
   Entrez un nombre entier : 5                       ↵
>>

>> B = input ('Entrez un vecteur ligne : ')          ↵
   Entrez un vecteur ligne : [1:2:8,3:-1:0]          ↵
   B =
      1     3     5     7     3     2     1     0
```

1.4 Ecriture des données dans un programme (Les sorties) :

On a déjà vu que Matlab peut afficher la valeur d'une variable en tapant seulement le nom de cette dernière. Par exemple :

```
>> A = 5 ;
>> A                  % Demander à Matlab d'afficher la valeur de A
   A =
       5
```

Avec cette méthode, Matlab écrit le nom de la variable (A) puis le signe (=) suivie de la valeur désirée. Cependant, il existe des cas où on désire afficher uniquement la valeur de la variable (sans le nom et sans le signe =).

Pour cela, on peut utiliser la fonction ***disp***, et qui a la syntaxe suivante : ***disp (objet)***

La valeur de l'objet peut être un nombre, un vecteur, une matrice, une chaine de caractères ou une expression.

On signale qu'avec un vecteur ou une matrice vide, ***disp*** n'affiche rien.

Exemple :

```
>> disp(A)                    % Afficher la valeur de A sans 'A = '
     5
>> disp(A);                   % Le point virgule n'a pas d'effet
     5
>> B                          % Afficher le vecteur B par la méthode classique
   B =
      1     3     5     7     3     2     1     0
>> disp(B)                    % Afficher le vecteur B sans 'B = '
      1     3     5     7     3     2     1     0
>> C = 3 :1 :0                % Création d'un vecteur C vide
   C =
   Empty matrix: 1-by-0
>> disp(C)                    % disp n'affiche rien si le vecteur est vide
>>
```

2. Les expressions logiques :

2.1 Les opérations logiques :

L'opération de comparaison	*Sa signification*
==	l'égalité
~=	l'inégalité
>	supérieur à
<	inferieur à
>=	supérieur ou égale à
<=	inferieur ou égale à
L'opération logique	***Sa signification***
&	le **et** logique
|	le **ou** logique
~	la **négation** logique

En Matlab une variable logique peut prendre les valeurs **1**(vrai) ou **0**(faux) avec une petite règle qui admette que :

1) Toute valeur égale à 0 sera considérée comme fausse **(= 0 ⇒ Faux)**
2) Toute valeur différente de 0 sera considérée comme vrai **(≠ 0 ⇒ Vrai)**.

Le tableau suivant résume le fonctionnement des opérations logiques :

a	***b***	***a & b***	***a | b***	***~a***
1 *(vrai)*	1 *(vrai)*	1	1	0
1 *(vrai)*	0 *(faux)*	0	1	0
0 *(faux)*	1 *(vrai)*	0	1	1
0 *(faux)*	0 *(faux)*	0	0	1

Par exemple :

```
>> x=10;
>> y=20;
>> x < y                          % affiche 1 (vrai)
      ans =
            1
>> x <= 10                        % affiche 1 (vrai)
      ans =
            1
>> x == y                         % affiche 0 (faux)
      ans =
            0
>> (0 < x) & (y < 30)             % affiche 1 (vrai)
      ans =
            1
>> (x > 10) | (y > 100)           % affiche 0 (faux)
      ans =
            0
>> ~(x > 10)                      % affiche 1 (vrai)
      ans =
            1
>> 10 & 1                         % 10 est considéré comme vrai donc 1 & 1 = 1
      ans =
            1
>> 10 & 0                         % 1 & 0 = 1
      ans =
            0
```

2.2 Comparaison des matrices :

La comparaison des vecteurs et des matrices diffère quelque peu des scalaires, d'où l'utilité des deux fonctions '*isequal*' et '*isempty*' (qui permettent de donner une réponse concise pour la comparaison).

La fonction	*Description*
isequal	teste si deux (ou plusieurs) matrices sont égales (ayant les mêmes éléments partout). Elle renvoie 1 si c'est le cas, et 0 sinon.
isempty	teste si une matrice est vide (ne contient aucun élément). Elle renvoie 1 si c'est le cas, et 0 sinon.

Pour mieux percevoir l'impact de ces fonctions suivons l'exemple suivant :

```
>> A = [5,2;-1,3]          % Créer la matrice A
      A =
             5     2
            -1     3
>> B = [5,1;0,3]           % Créer la matrice B
      B =
            5     1
            0     3
```

```
>> A==B                    % Tester si A=B ? (1 ou 0 selon la position)
      ans =
            1     0
            0     1
>> isequal(A,B)            % Tester si effectivement A et B sont égales (les mêmes)
      ans =
            0
>> C=[] ;                  % Créer la matrice vide C
>> isempty(C)              % Tester si C est vide (affiche vrai = 1)
      ans =
            1
>> isempty(A)              % Tester si A est vide (affiche faux = 0)
      ans =
            0
```

3. Structures de contrôle de flux

Les structures de contrôle de flux sont des instructions permettant de définir et de manipuler l'ordre d'exécution des taches dans un programme. Elles offrent la possibilité de réaliser des traitements différents selon l'état des données du programme, ou de réaliser des boucles répétitives pour un processus donnée.
Matlab compte huit structures de control de flux à savoir :

- **if**
- **switch**
- **for**
- **while**
- continue
- break
- try - catch
- return

Nous exposons les quatre premières : (**if, switch, for** et **while**)

3.1 L'instruction `if` :

L'instruction **if** est la plus simple et la plus utilisée des structures de contrôle de flux. Elle permette d'orienter l'exécution du programme en fonction de la valeur logique d'une condition. Sa syntaxe générale est la suivante :

```
if (condition)
      instruction_1
      instruction_2
                . . .
      Instruction_N
end
```

ou bien

```
if (condition)
      ensemble d'instructions 1
else
      ensemble d'instructions 2
end
```

Si la condition est évaluée à vari, les instructions entre le **if** et le **end** seront exécutées, sinon elles ne seront pas (ou si un **else** existe les instructions entre le **else** et le **end** seront exécutées). S'il est nécessaire de vérifier plusieurs conditions au lieu d'une seule, on peut utiliser des clauses **elseif** pour chaque nouvelle condition, et à la fin on peut mettre un **else** dans le cas ou aucune condition n'a été évaluée à vrai. Voici donc la syntaxe générale :

```
if (expression_1)
       Ensemble d'instructions 1
elseif (expression_2)
       Ensemble d'instructions 2
              ....
elseif (expression_n)
       Ensemble d'instructions n
else
       Ensemble d'instructions si toutes les expressions étaient fausses
end
```

Par exemple, le programme suivant vous définit selon votre âge :

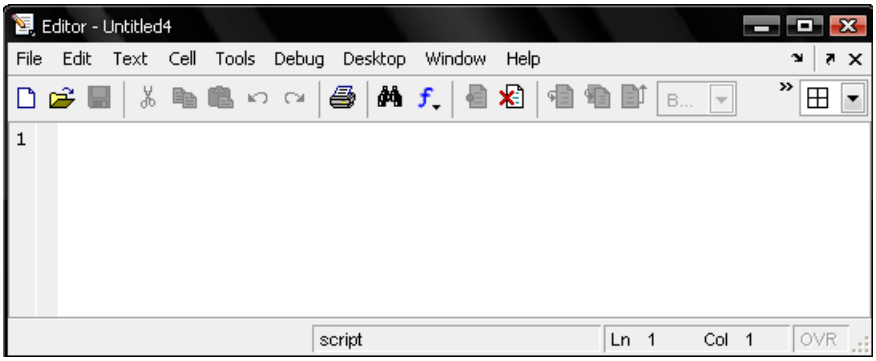
```
>> age = input('Entrez votre âge : '); ...
   if (age < 2)
      disp('Vous êtes un bébé')
   elseif (age < 13)
      disp('Vous êtes un enfant')
   elseif (age < 18)
      disp ('Vous êtes un adolescent')
   elseif (age < 60)
      disp ('Vous êtes un adulte)
   else
      disp ('Vous êtes un vieillard')
   end
```

Comme vous pouvez le constatez, l'écriture d'un programme Matlab directement après l'invité de commande (le prompt >>) est un peu déplaisant et ennuyeux.

Une méthode plus pratique consiste à écrire le programme dans un fichier séparé, et d'appeler ce programme (au besoin) en tapant le nom du fichier dans l'invité de commande. Cette approche est définie en Matlab par les M-Files, qui sont des fichiers pouvant contenir les données, les programmes (scripts) ou les fonctions que nous développons.

Pour créer un M-Files il suffit de taper la commande ***edit***, ou tout simplement aller dans le menu : File → New → M-Files (ou cliquer sur l'icône ).

Dans tous les cas une fenêtre d'édition comme celui ci va apparaitre :



Tout ce qui vous reste à faire et d'écrire votre programme dans cette fenêtre, puis l'enregistrer avec un nom (par exemple : 'Premier_Programme.m'). On signale que l'extension des fichiers M-Files est toujours '.m'.

Maintenant, si nous voulons exécuter notre programme, il suffit d'aller à l'invité de commande habituel (>>) puis taper le nom de notre fichier (sans le '.m') comme ceci :

```
>> Premier_Programme        ↵
```

Et l'exécution du programme va démarrer immédiatement.

Pour retourner à la fenêtre d'édition (après l'avoir fermer) il suffit de saisir la commande :

```
>> edit Premier_Programme        ↵
```

Exemple :

Créons un programme qui trouve les racines d'une équation de second degré désigné par : $ax^2+bx+c=0$. Voici le M-File qui contient le programme (il est enregistré avec le nom 'Equation2deg.m')

```
% Programme de résolution de l'équation a*x^2+b*x+c=0

a = input ('Entrez la valeur de a : ');            % lire a
b = input ('Entrez la valeur de b : ');            % lire b
c = input ('Entrez la valeur de c : ');            % lire c

delta = b^2-4*a*c ;                          % Calculer delta
if delta<0
   disp('Pas de solution')                   % Pas de solution
elseif delta==0
   disp('Solution double : ')                % Solution double
   x=-b/(2*a)
else
   disp('Deux solutions distinctes: ')       % Deux solutions
   x1=(-b+sqrt(delta))/(2*a)
   x2=(-b-sqrt(delta))/(2*a)
end
```

Si nous voulons exécuter le programme, il suffit de taper le nom du programme :

```
>> Equation2deg                          ↵
       Entrez la valeur de a : -2        ↵
       Entrez la valeur de b : 1         ↵
       Entrez la valeur de c : 3         ↵
       Deux solutions :
       x1 =
             -1
       x2 =
             1.5000
```

Ainsi, le programme va être exécuté en suivant les instructions écrites dans son M-File. Si une instruction est terminée par un point virgule, alors la valeur de la variable concernée ne sera pas affichée, par contre si elle termine par une virgule ou un saut à la ligne, alors les résultats seront affichés.

Remarque : Il existe la fonction ***solve*** prédéfinie en Matlab pour trouver les racines d'une équation (et beaucoup plus). Si nous voulons l'appliquer sur notre exemple, il suffit d'écrire :

```
>> solve('-2*x^2+x+3=0','x')
   ans =
        -1
        3/2
```

3.2 L'instruction `switch` :

L'instruction **switch** exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé a une clause **case** qui définie si ce groupe doit être exécuté ou pas selon l'égalité de la valeur de ce **case** avec le résultat d'évaluation de l'expression de **switch**. Si tous les **case** n'ont pas étés acceptés, il est possible d'ajouter une clause **otherwise** qui sera exécutée seulement si aucun **case** n'est exécuté.

Donc, la forme générale de cette instruction est :

```
switch (expression)
      case valeur_1
            Groupe d'instructions 1
      case valeur_2
            Groupe d'instructions 2
            . . .
      case valeur_n
            Groupe d'instructions n
      otherwise
            Groupe d'instructions si tous les case ont échoué
end
```

Exemple :

```
x = input ('Entrez un nombre : ') ;
switch(x)
      case 0
            disp('x = 0 ')
      case 10
            disp('x = 10 ')
      case 100
            disp('x = 100 ')
      otherwise
            disp('x n''est pas 0 ou 10 ou 100 ')
end
```

L'exécution va donner :

```
Entrez un nombre : 50          ↵
x n'est pas 0 ou 10 ou 100
```

3.3 L'instruction for :

L'instruction **for** répète l'exécution d'un groupe d'instructions un nombre déterminé de fois. Elle a la forme générale suivante :

```
for variable = expression_vecteur
      Groupe d'instructions
end
```

L'expression_vecteur corresponde à la définition d'un vecteur : *début : pas : fin* ou *début : fin*

Le variable va parcourir tous les éléments du vecteur défini par l'expression, et pour chacun il va exécuter le groupe d'instructions.

Exemple :

Dans le tableau suivant, nous avons trois formes de l'instruction **for** avec le résultat Matlab :

L'instruction for	`for i = 1 : 4`
	`  j=i*2 ;`
	`  disp(j)`
	`end`
		`for i = 1 : 2 : 4`
		`  j=i*2 ;`
		`  disp(j)`
		`end`
			`for i = [1,4,7]`
			`  j=i*2 ;`
			`  disp(j)`
			`end`

Le résultat de l'exécution	2 4 6 8	2 6	2 8 14

3.4 L'instruction `while` :

L'instruction **while** répète l'exécution d'un groupe d'instructions un nombre indéterminé de fois selon la valeur d'une condition logique. Elle a la forme générale suivante :

```
while (condition)
      Ensemble d'instructions
end
```

Tant que l'expression de **while** est évaluée à true, l'ensemble d'instructions s'exécutera en boucle.

Exemple :

```
a=1 ;
while (a~=0)
      a = input ('Entrez un nombre (0 pour terminer) : ') ;
end
```

Ce programme demande à l'utilisateur d'entrer un nombre. Si ce nombre n'est pas égal à 0 alors la boucle se répète, sinon (si la valeur donnée est 0) alors le programme s'arrête.

4. Exercice récapitulatif

Il existe des fonctions prédéfinis en Matlab donnée dans le tableau ci-dessous. Essayons de les programmer (pour un vecteur donnée V).

La fonction	*Description*	*Le programme qui la simule*

sum (V)	La somme des éléments d'un vecteur V	`n = length(V);`
		`somme = 0 ;`
		`for i = 1 : n`
		`  somme=somme+V(i) ;`
		`end`
		`disp(somme)`

prod (V)	Le produit des éléments d'un vecteur V	`n = length(V);`
		`produit = 1 ;`
		`for i = 1 : n`
		`  produit=produit*V(i) ;`
		`end`
		`disp(produit)`

mean (V)	La moyenne des éléments d'un vecteur V	`n = length(V);`
		`moyenne = 0 ;`
		`for i = 1 : n`
		`  moyenne = moyenne+V(i) ;`
		`end`
		`moyenne = moyenne / n`

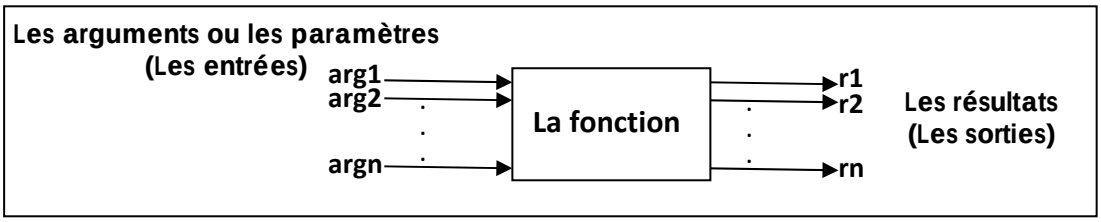
diag (V)	Créer une matrice ayant le vecteur V dans le diagonale, et 0 ailleurs	`n = length(V);`
		`A = zeros(n) ;`
		`for i = 1 : n`
		`  A(i,i)=V(i) ;`
		`end`
		`disp(A)`

sort (V)	Ordonne les éléments du vecteur V par ordre croissant	`n = length(V);`
		`for i = 1 : n-1`
		`  for j = i+1 : n`
		`  if V(i) > V(j)`
		`  tmp = V(i) ;`
		`  V(i) = V(j) ;`
		`  V(j) = tmp ;`
		`  end`
		`  end`
		`end`
		`disp(V)`

5. Les fonctions

Il existe une différence de concept entre les fonctions en informatique ou en mathématique:

1. En informatique, une fonction est une routine (un sous programme) qui accepte des arguments (des paramètres) et qui renvoie un résultat.



2. En mathématique une fonction **f** est une relation qui attribue à chaque valeur x au plus une seule valeur **f(x)**.

5.1 Création d'une fonction dans un M-Files :

Matlab contient un grand nombre de fonctions prédéfinies comme ***sin***, ***cos***, ***sqrt***, ***sum***, …etc. Et il est possible de créer nos propres fonctions en écrivant leurs codes source dans des fichiers M-Files (portant le même nom de fonction) en respectant la syntaxe suivante :

```
function [r1, r2, …, rn] = nom_fonction (arg1, arg2, …, argn)

      % le corps de la fonction
      . . .
      r1 = . . .  % la valeur retournée pour r1
      r2 = . . .  % la valeur retournée pour r2
      . . .
      rn = . . .  % la valeur retournée pour rn
end               % le end est facultatif
```

Ou : $r_1...r_n$ sont les valeurs retournées, et $arg_1...arg_n$ sont les arguments.

Exemple : Ecrire une fonction qui calcule la racine carrée d'un nombre par la méthode de Newton (vue dans le TD).

Solution :

```
>> edit
```

Le fichier racine.m

```
function r = racine(nombre)
    r = nombre/2;
    precision = 6;
    for i = 1:precision
        r = (r + nombre ./ r) / 2;
    end
```

L'exécution :

```
>> x = racine(9)
      x =
           3
>> x = racine(196)
      x =
         14.0000
>> x = racine([16,144,9,5])
      x =
            4.0000   12.0000    3.0000    2.2361
```

Remarque :

Contrairement à un programme (un script), une fonction peut être utilisée dans une expression par exemple : **2*racine(9)-1.**

Comparaison entre un programme est une fonction

Un programme	*Une fonction*

`a = input('Entrez un nombre positif: ');`
`x = a/2;`
`precision = 6;`
`for i = 1:precision`
`  x = (x + a ./ x) / 2;`
`end`
`disp(x)`
	`function r = racine(nombre)`
	`  r = nombre/2;`
	`  precision = 6;`
	`  for i = 1:precision`
	`  r = (r + nombre ./ r) / 2;`
	`  end`

L'exécution :
`>> racine` ↵
`Entrez un nombre positif: 16` ↵
`4`
	L'exécution :
	`>> racine(16)`
	`ans =`
	`4`

on ne peut pas écrire des expressions tel que : **>> 2 * racine + 4** ✗	on peut écrire sans problème des expressions comme : **>> 2 * racine(x) + 4** ✓