Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

وزارة التعليم العالي و البحث العلمي
جامعة باجي مختار- عنابه

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

Faculté des Sciences

COURS DE PROGRAMMATION EN MATLAB POUR LES MATHEMATIQUES

Spécialité : Mathématique et Informatique

LMD 1ére année

Présenté

Par

Dr HAFIDI Mohamed

Année universitaire 2020-2021

Sommaire

Chapitre 4 : Programmer en Matlab

Chapitre 4 : Programmer en 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.

4.1. Généralités

4.1.1 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
```

4.1.2 Lecture des données dans un programme

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')

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
>> 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
```

4.1.3 Ecriture des données dans un programme

Pour afficher la valeur d'une variable, 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.

4.2. Operateurs logiques et de relation

Les operateurs suivants servent a comparer différentes variables :

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

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 :

Commande	**signification**	**exécution**
>> **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 ?	ans = 1 0 0 1
>> **isequal(A,B)**	Tester si effectivement A et B sont égaux	ans =0
>> **C=[] ;**	Créer la matrice vide C	
>> **isempty(C)**	Tester si C est vide	ans =1
>> **isempty(A)**	Tester si A est vide	ans =0

4.3 Structures de contrôle

Les structures de contrôle de flux sont des instructions permettant de définir et de manipuler l'ordre d'exécution des tâches 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.

4.3.1 Sélection - if . . . end et if . . . else . . . end

– if *(expression booléenne)* / *script* / end
– if *(expression booléenne)* / *script si vrai* / else / *script si faux* / end

Le symbole / remplace l'un des symboles séparateur : virgule (,), point-virgule (;) ou saut de ligne. L'usage du point-virgule est vivement conseillé pour éviter les affichages souvent redondants.

Exemple :

```
>> m = -1;
>> if (m<0), a =-m, end
```

Lorsqu'il y a plus de deux alternatives, on peut utiliser la structure suivante :

```
if (exp1)
script1 (évalué si exp 1 est vraie)
elseif (exp2)
script2 (évalué si exp 2 est vraie)
elseif (exp3)
. . .
else
script (évalué si aucune des expressions exp1, exp2, . . . n'est vraie)
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
```

4.3.2. Construction switch . . . case

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 (sélecteur)
case valeur 1, . . . / script 1 /
case Valeur 2, . . . / script 2 /
. . .
otherwise / script
end
```

Comme dans les définitions précédentes, le symbole / remplace un séparateur : virgule (,), point-virgule (;) ou saut de ligne. *sélecteur* désigne une expression dont les valeurs peuvent correspondre aux valeurs associées aux différents case. Lorsque la valeur du sélecteur correspond à une valeur de case, une fois le script correspondant exécuté, l'exécution se continue immédiatement après le end contrairement à ce qui se passe pour certains langages. Ceci explique l'absence de break après chaque script.

Exemple :

```
>> n =17
>> switch rem(n,3) % reste de la division de n par 3
case 0, disp('Multiple de 3')
case 1, disp('1 modulo 3')
case 2, disp('2 modulo 3')
otherwise, disp('Nombre négatif')
end
```

4.3.3 Itération conditionnelle - while . . . end

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 (expression booléenne) / script / end
```

Le symbole / représente comme dans les définitions précédents, un séparateur : virgule (,), point virgule (;) ou saut de ligne. D'autre part, il faut éviter l'utilisation des variables *i* et *j* comme indices puisqu'elles sont des variables prédéfinies dont la valeur est sqrt(−1).

Exemple :

```
>> n = 1 ;
>> while (2^n <= 100)
n = n + 1;
end
>> disp(n-1)
```

4.3.4. Répétition - for . . . end

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

```
for (k = liste) / script / end
```

Exemple 1:

```
>> x = [ ];
>> for (k = 1 : 5), x = [x, sqrt(k)], end
```

Exemple2:

```
>> for (l = 1 : 3)
for ( k = 1 : 3)
A(l,k) = l^2 + k^2 ;
end
end
>> disp(A)
```

Il est possible de sortir directement d'une boucle for ou while en utilisant la commande break :

Exemple :

```
>> EPS = 1;
>> for (n = 1 : 100)
EPS = EPS / 2;
If ((EPS + 1) <= 1)
EPS = EPS*2
break
end
end
>> n
```

Le test (EPS + 1) <= 1 provoque la sortie de la boucle for à la 52*ème* itération.
Dans l'exemple suivant, le tableau A est celui de l'exemple 2.

```
>> for (l = 1 : 3)
for (k = 1 : 3)
if (A(l,k) == 10)
[l,k]
break
end
end
end
```

La double boucle se s'est pas arrêté après que le test A(l,k) == 10 ait été validé lorsque l=1 et k=3. En effet break **provoque la sortie de la boucle la plus proche**, ici la boucle for interne. Une version corrigée du test précédent pourrait être la suivante avec deux break pour pouvoir sortir des deux boucles for :

```
>> sortie = 0;
>> for (l=1:3)
if (sortie)
break
end
for (k = 1:3)
if (A(l,k) == 10)
[l,k]
sortie = 1 ;
break
end
end
end
```

4.5. Scripts

Un script est une séquence d'expressions ou de commandes. Un script peut se développer sur une ou plusieurs lignes. Les différentes expressions ou commandes doivent être séparées par une virgule, un point-virgule ou par le symbole de saut de ligne constitué de trois points . . . suivis de <entrer> (le rôle des trois points et d'inhiber le mécanisme d'évaluation lors d'un passage à la ligne). Comme pour une expression unique, la frappe de <entrer> déclenche le processus d'évaluation. Les expressions sont évaluées dans leur ordre d'écriture. Seule la valeur des expressions suivies d'une virgule ou d'un saut de ligne est affichée, celle des expressions suivies d'un point-virgule, ne l'est pas.

Exemple :

```
>> a = .5, 2*a, save a, b = pi; 2*b, c = a*b
>> ans
```

Ecrire un script est assez fastidieux, aussi MATLAB permet d'enregistrer le texte d'un script sous forme de fichier de texte appelés *m-files*, en raison de leur extension.

4.5.1 Création de m-files

Les *m-files* permettent d'enregistrer les scripts sous forme de fichiers-texte et servent en particulier à définir de nouvelles fonctions (une grande partie des fonctions prédéfinies de MATLAB sont stockées sous forme de *m-files* dans la toolbox matlab).

Les *m-files* peuvent être crées par n'importe quel éditeur. Dans les versions récentes de MATLAB il existe un petit éditeur intégré que l'on peut appeler à partir du menu file ou à partir de la barre de menu de la fenêtre de commande.

Dans la fenêtre de l'éditeur tapez les lignes suivantes :

```
% script - essai . m
a = .5;
b = pi;
c = a * b
```

Sauvez le fichier dans le répertoire de travail sous le nom de essai.m.

4.5.2 Exécution d'un m-file

Pour exécuter le script contenu dans un *m-file* et Il suffit de taper le nom de ce m file dans la fenêtre de commande suivi de < *entrer* >

```
>> essai
```

4.6. Fonctions

Nous avons vu un certain nombre de fonctions prédéfinies. Il est possible de définir ses propres fonctions. La première méthode permet de définir des fonctions simples sur une ligne de commande. La seconde, beaucoup plus générale permet de définir des fonctions très évoluées en la définissant dans un fichier.

4.6.1 Fonctions inline

Exemple :
Admettons que je veuille définir une nouvelle fonction que j'appelle sincos définie mathématiquement par: sincos(x) = sin x − x cos x

On écrira :

```
>> sincos = inline('sin(x)-x*cos(x)')

sincos =
Inline function:
sincos(x) = sin(x)-x*cos(x)
```

On peut maintenant utiliser cette nouvelle fonction :

```
>> sincos(pi/12)
ans =
0.0059
```

Essayons maintenant d'appliquer cette fonction à un tableau de valeurs :

```
>> sincos(0:pi/3:pi)

??? Error using ==> inline/subsref
Error in inline expression ==> sin(x)-x*cos(x)
??? Error using ==> *
Inner matrix dimensions must agree.
```

Ça ne marche pas. MATLAB essaye de multiplier x vecteur ligne par cos(x) aussi vecteur ligne au sens de la multiplication de matrice ! Par conséquent il faut bien utiliser une multiplication terme à terme.* dans la définition de la fonction :

```
>> sincos = inline('sin(x)-x.*cos(x)')

sincos =
Inline function:
sincos(x) = sin(x)-x.*cos(x)

>> sincos(0:pi/3:pi)
ans =
0  0.3424  1.9132  3.1416
```

4.6.2 Fonctions définies dans un fichier

C'est la méthode la plus généraliste, et elle permet notamment de réaliser des fonctions ayant plusieurs sorties. Commençons par reprendre l'exemple précédent (sincos). L'ordre des opérations est le suivant :

1. Editer un nouveau fichier appelé sincos.m

2. Taper les lignes suivantes :

```
function s = sincos(x)
s = sin(x)-x.*cos(x);
```

3. Sauvegardez

Le résultat est le même que précédemment :

```
>> sincos(pi/12)
ans =
0.0059
```

Voyons maintenant comment définir une fonction comportant plusieurs sorties. On veut réaliser une fonction appelée cart2pol qui convertit des coordonnées cartésiennes (x, y)
(entrées de la fonction) en coordonnées polaires (r, θ) (sorties de la fonction). Voilà le contenu du fichier cart2pol.m :

```
function [r, theta] = cart2pol (x, y)
r = sqrt(x.^2 + y.^2);
theta = atan (y./x);
end
```

On remarque que les deux variables de sortie sont mises entre crochets, et séparées par une virgule.
Pour utiliser cette fonction, on écrira par exemple :

```
>> [rr,tt] = cart2pol(1,1)
rr =
1.4142
tt =
0.7854
```

On affecte donc simultanément deux variables rr et tt avec les deux sorties de la fonction, en mettant ces deux variables dans des crochets, et séparés par une virgule. Les crochets ici n'ont bien sûr pas le même sens que pour les tableaux.
Il est possible de ne récupérer que la première sortie de la fonction. MATLAB utilise souvent ce principe pour définir des fonctions ayant une sortie «principale» et des sorties «optionnelles».
Ainsi, pour notre fonction, si une seule variable de sortie est spécifiée, seule la valeur du rayon polaire est renvoyée. Si la fonction est appelée sans variable de sortie, c'est ans qui prend la valeur du rayon polaire :

```
>> cart2pol(1,1)
ans =
1.4142
```

A l'intérieur des fonctions comme celle que nous venons d'écrire, vous avez le droit de manipuler trois types de variables :
– Les variables d'entrée de la fonction. Vous ne pouvez pas modifier leurs valeurs.
– Les variables de sortie de la fonction. Vous devez leur affecter une valeur.
– Les variables locales. Ce sont des variables temporaires pour découper des calculs par exemple. Elles n'ont de sens qu''a l'intérieur de la fonction et ne seront pas «vues» de l'extérieur.
Imaginons maintenant que vous écriviez un fichier de commande, et une fonction (dans deux fichiers différents bien sûr), le fichier de commande se servant de la fonction. Si vous voulez passer une valeur du fichier de commandes à la fonction, vous devez normalement passer par une entrée de la fonction. Cependant, on aimerait bien parfois que la variable A du fichier de commandes soit utilisable directement par la fonction. Pour cela on utilise la directive global.

4.7 Exercices d'entraînement

1. Ecrire un programme script Matlab qui permet de créer une matrice carrée d'ordre n telle que chaque élément situé sur et au-dessus de la diagonale principale soit égale au produit du numéro de colonne et du numéro de ligne. Les éléments situés au-dessous de la diagonale sont égale à zéro.

2. Soit une matrice A de dimensions quelconques, écrire une fonction qui retourne comme variables de sortie un vecteur constitué des éléments strictement positifs de A ainsi que le nombre d'éléments non nuls de A.

3. Vous disposez d'une corrélation donnant le **Cp** d'un gaz en fonction de la température :

$$\mathbf{Cp(T) = AT^3 + BT^2 + CT + D}$$

- Réaliser un script Matlab qui fait appel à la fonction Cp ayant 5 entrées : T, A, B, C, D.
- Mais il est plus naturel que cette fonction ait seulement T comme entrée.
  Comment passer A, B, C, D qui sont des constantes ?