III. Principes de programmation

1. Introduction

Un programme est une séquence de commandes [1]. Dans les cas simples, les commandes sont exécutées une après l'autre, dans l'ordre dans lequel elles sont écrites. Pour traiter des problèmes plus complexes, il est important de pouvoir contrôler l'ordre dans lequel les commandes (ou groupes de commandes) sont exécutées, de manière plus élaborée. Il existe ainsi des moyens de contrôler le flux d'exécution dans un programme [1].

2. Opérations logiques

Changer le flux d'exécution dans un programme est effectué à l'aide d'instructions de contrôle. Ceci implique souvent des comparaisons entre variables.

Les opérateurs de comparaison dans Matlab sont [2]:

>	plus grand que	<=	plus petit ou égal à
<	plus petit que	==	égal à
>=	plus grand ou égal à	~=	différent de

Il s'agit d'opérateurs « binaires », lesquels retournent la valeur 0 (= false) quand la relation est fausse et 1 (= true) quand la relation est vraie.

Exemples :

```
>> x = 5;
>> 3 > x

ans =
     0
```

```
>> x = 5;
>> x == 5

ans =
     1
```

La comparaison peut être effectuée à toute une matrice, coefficient par coefficient.

Exemple :

```
>> x = [1 2 3 ; 4 5 6 ; 7 8 9];
>> x >= 5

ans =
     0     0     0
     0     1     1
     1     1     1
```

Les opérateurs logiques sont [2] :

opérateur	signification	fonction équivalente
~	pas	`not(A)`
&	et	`and(A,B)`
|	ou	`or(A,B)`
&&	(short-circuit)* et	
||	(short-circuit)* ou	

*court-circuit signifie que la seconde condition est évaluée seulement si le résultat n'est pas déjà déterminé par la 1ère condition.

Ces opérateurs permettent de combiner différentes conditions. La comparaison logique s'effectue élément par élément et permet de comparer des matrices.

Exemple :

```
x = (b ~= 0) && (b <= pi)
```

Il est possible combiner des matrices logiques (de même taille). Les opérateurs logiques sont alors exécutés coefficient par coefficient, et la matrice résultante est de même taille que les matrices utilisées, avec le résultat de la comparaison affecté à chaque coefficient (cf. 1er exemple ci-dessous). On peut aussi combiner plusieurs comparaisons / opérateurs logiques (cf. 2ème exemple).

Exemples 1 et 2:

```
>> A = [0 1 1 0 1];
>> B = [1 1 0 0 1];
>> A&B

ans =
     0     1     0     0     1
```

```
>> x = 5;
>> y = 10;
>> (x > 0) & (y > 5) & (x < y)

ans =
     1
```

3. Instructions de contrôle : les conditions

Les instructions de contrôle permettent de gérer le flux d'exécution d'un programme, en fonction de certaines conditions. Pour les instructions de conditions, une expression conditionnelle est déclarée : si l'expression est vraie, le groupe de commandes qui suit la condition est exécutée, sinon cette séquence de commande est ignorée. La forme de base est l'instruction `if`.

3.1. Les structures en if (condition logique):

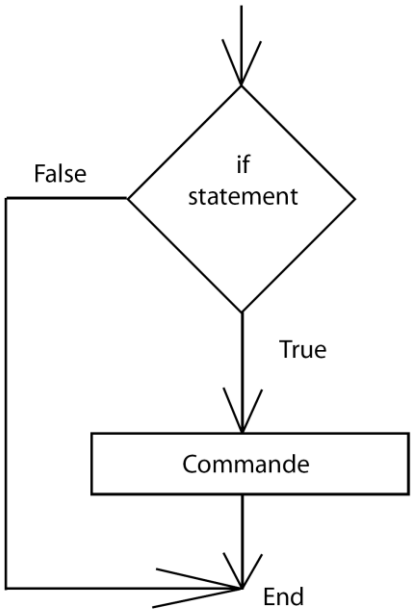
```
if <condition>

    <instructions>

else

    <instructions>

end
```

L'instruction `if`, comme la plupart, se termine toujours par un `end`.

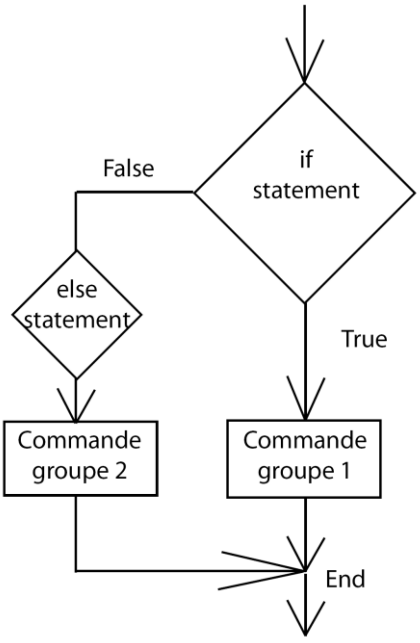
Exemple :

```
if a < b        si a est plus petit que b
if c >= 5       si c est plus grand ou égal à 5
if a == b       si a est égal à b
if a ~= 0       si a est différent de 0
if (d<h) & (x>7)    si … & signifie AND
if (x~=0)| (y<3)    si … | signifie OR
```

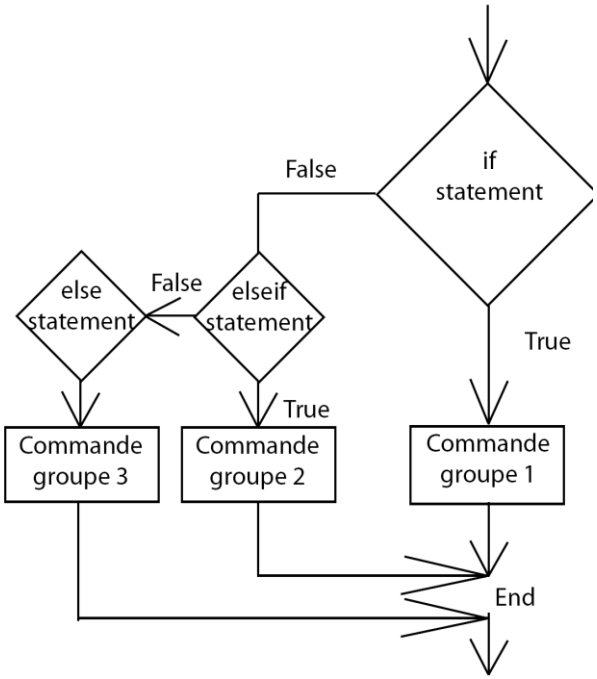
La structure if – end



La structure if – else - end



La structure if – else – elseif - end



Exemple:

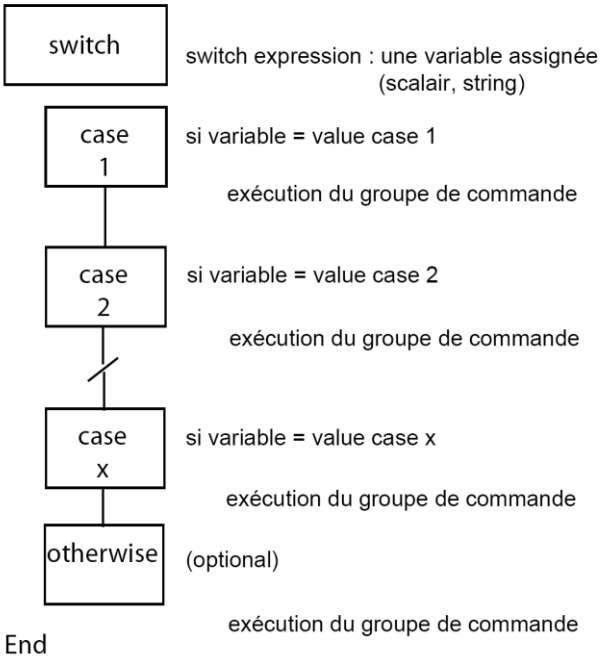
```
if x > 0
    disp('x est positif')
elseif x == 0
    disp('x est égal à zero')
else
    disp('x est négatif')
end
```

Le nombre de conditions `elseif` n'est pas limité.

3.2. La structure en switch – case (jonction conditionnelle) :

La structure `switch - case` permet de choisir un groupe de commandes d'exécution parmi d'autres. Le choix se fait par correspondance de la variable aux différents cas. S'il existe plusieurs correspondances, seulement la 1ère est exécutée. Si aucune correspondance n'existe et que l'expression `otherwise` existe, le groupe de commande entre `otherwise` et `end` est exécuté. Sinon aucun groupe de commandes n'est exécuté.

La structure if – else – elseif - end



Exemples:

```
switch x
    case 1
        disp('x = 1')
    case {2,3,4}
        disp('x = 2, 3 ou 4')
    otherwise
        disp('x ~= 1, 2, 3 ou 4')
end


Q1 = input('avez-vous soif ?
                  oui/non ','s');

switch Q1;
    case 'oui'
        disp('alors buvez!')
    case 'non'
        disp('c''est pas grave')
    otherwise
        disp('répondez par oui/non')
end
```

4. Instructions de contrôle : les boucles

Une boucle permet de répéter l'exécution d'une commande/groupe de commandes plusieurs fois consécutivement. Matlab possède deux types de boucle :

La structure for - end

boucle avec un nombre prédéfini de passage :

```
for <i> = <start> : <incr> : <stop>

    <instructions>

end
```

exemple:

```
for k = 1 : 1 : 10

    x = k^2

end
```

la structure while - end

boucle dépendante d'une condition logique.
Le nombre de passage est défini par la condition. La boucle est effectuée jusqu'à ce que la condition soit fausse:

```
while <condition>

    <instructions>

end
```

exemple :

```
n = 1;

while n<=10

    y(n) = 1+n-2*n^2+0.1*n^3
    n = n+1
end
```

4.1. Remarque concernant la boucle for – end vs. opérations matricielles:

Dans certaines situations, le même résultat peut être obtenu en utilisant soit la structure en boucle for – end, soit l'opération matricielle élément par élément (cf. Chapitre 2). Les opérations matricielles sont une des caractéristiques de Matlab, permettant d'effectuer des opérations réalisable avec des boucles. Toutefois, l'opération matricielle est beaucoup plus rapide à exécuter que les structures en boucle [2].

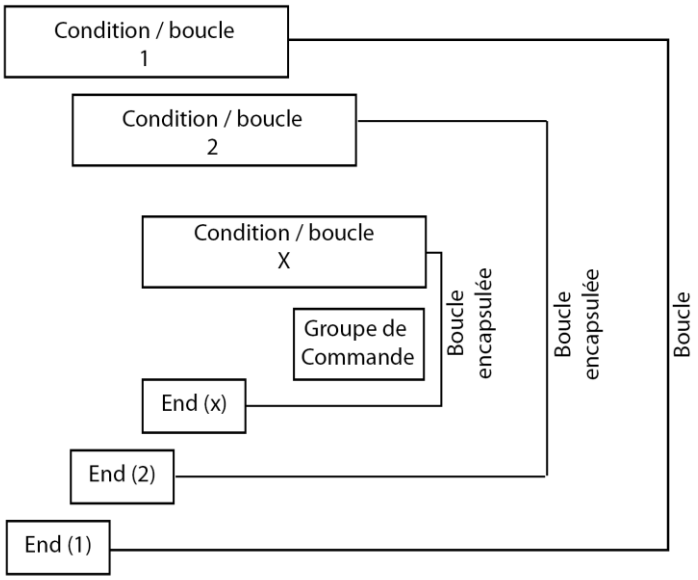
Exemple:

```
x = [2 4 6 8 10];
y = [3 6 9 12 15];

% Matrix calculation (coefficient by coefficient)
z1 = (y./x).^2 + (x + y).^((y - x)./x)

% Using the for - end structure
for n = 1:length(x)
    z2(n) = (y(n)/ x(n))^2 + (x(n) + y(n))^(( y(n) - x(n))/ x(n))
end
```

5. Boucles et conditions encapsulées

Les conditions et boucles peuvent être encapsulées. C'est-à-dire qu'une boucle et/ou une condition peut se situer à l'intérieur d'une autre boucle et/ou une condition (aucune limite de nombre). Chaque fois qu'une encapsulation a lieu, on indente la nouvelle boucle relativement à la précédente (indentation automatique : Ctrl + i).

La structure encapsulée



for i = 1:n
  for j = 1:m
  …
  …
  end
end
	chaque fois que i augmente de 1 (boucle), la boucle encapsulée (j) est exécutée m fois.
	
	Les commandes à l'intérieur des 2 boucles auront été exécutées n x m fois

exemple:

Création d'une matrice de dimension m x n:

```
m = input('nbre de lignes?');
n = input('nbre de colonnes?');

A = zeros(m,n);
for l=1:m
    for c=1:n
        A(l,c) = l+c;
    end
end

disp(A)
```

Création d'une matrice nulle de dimension m x n avec diagonale = 1

```
for l=1:m
    for c=1:n
        A(l,c) = 0;

        if l==c
            A(l,c) = 1
        end
    end
end
```

6. La commande break et continue

A l'intérieur d'une boucle (`for` et `while`), la commande `break` termine l'exécution de la boucle et continue avec la suite des instructions après la boucle en question. A l'intérieur d'une boucle encapsulée, seule cette dernière sera terminée. Cette commande est surtout utilisée quand il s'agit de terminer une boucle (ou le programme) quand une condition est atteinte ou quand une variable devient inconsistante.

La commande `continue` permet dans une boucle (for et while) de sauter un passage seulement.

7. Références

[1] Amos, Gilat, 2007. *Matlab, an introduction with application,* Johne Willey and Sohn, Inc.

[2] MATLAB Help

8. Auteurs

Alexandre Loye (2009)

Pascal Horton (2015)