Chapitre 4

Éléments de programmation

4.1 Structures de contrôle

Les structures de contrôle sous Matlab sont très proches de celles existant dans d'autres langages de programmation tels que le langage C. Elles sont utiles pour écrire des programmes Matlab.

4.1.1 Structure conditionnelle

• Une structure conditionnelle permet d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est verifiée .

• Différentes formes de structures conditionnelles existent sous Matlab.

1) Forme simple

Syntaxe :

if (expression)

instructions

end

Exemple :

Calculer la racine carrée d'une variable scalaire x seulement si x n'est pas négative.

if (x >= 0)

racine = sqrt(x);

end;

• **expression** est une expression logique dont le résultat peut être **vrai (true)** ou **faux (false)**.

• **instructions** est une suite d'instructions.

• si le résultat de l'évaluation de l'**expression** est vrai, on exécute les instructions, puis l'instruction qui suit le mot clé **end**.

• si le résultat de l'évaluation de l'**expression** est faux on passe directement à l'instruction qui suit le mot clé **end**.

2) Forme alternative

Syntaxe :

if (expression)

instructions1

else

instructions2

end

Exemple :

if ($x \neq 0$)

$y = \frac{1}{x};$

else

disp('Division par 0')

end ;

• si le résultat de l'évaluation de l'**expression** est vrai, on exécute les instructions **instructions**1, puis l'instruction qui suit le mot clé **end**.

• si le résultat de l'évaluation de l'**expression** est faux, on exécute les instructions **instructions**2, puis l'instruction qui suit le mot clé **end**.

3) Forme imbriquée

• Il est possible d'imbriquer des instructions conditionnelles les unes dans les autres.

• Utilisée lorsqu'il y a plus de deux alternatives.

Syntaxe :

if (exp1)

instructions1

elseif (exp2)

instructions2

elseif (exp3)

instructions3

⋮

else

instructions *n*

end

Exemple :

if (moyenne >= 16)

disp('T.Bien') ;

elseif (moyenne >= 14)

disp('Bien');

elseif (moyenne >= 12)

disp('A.Bien');

elseif (moyenne >= 10)

disp('Passable');

else

disp('Non admis');

end

• **exp***i* est une expression logique.

• **instructions***i* est une suite d'instructions.

• si le résultat de l'évaluation de **exp***i* est vrai, on exécute **instructions***i*, puis l'instruction qui suit le mot clé **end**.

• si aucune des expressions exp1, exp2,..., exp$n-1$ n'est vraie, on exécute **instructions***n* (suite d'instructions par défaut), puis l'instruction qui suit le mot clé **end**.

• Il n'est pas nécessaire de prévoir un cas par défaut (bien que cela soit préférable).

• S'il n'y a pas de cas par défaut, et si aucune des expressions exp1, exp2,..., n'est vraie, alors on continue à la première instruction suivant le mot clé end.

4.1.2 Choix multiple instruction-switch

Syntaxe :

switch var

case const1,

instructions1

case const2,

instructions2

⋮

otherwise

instructions *n*

end

Exemple :

On initialise une matrice A en fonction de la valeur d'une variable **var**.

switch var

case 1,

A=ones(3);

case 2,

A=magic(3);

case{3, 4},

A=rand(2);

otherwise

disp('Erreur, numéro d'exemple non prévu')

end;

• **var** est une variable numérique ou une variable chaîne de caractères.

• **const***i* est une constante de même type que **var**.

• **instructions***i* est une suite d'instructions.

• si la variable **var** est égale à la constante **const***i*, on exécute la suite d'instructions correspondante (c'est à dire **instructions***i*, puis l'instruction qui suit le mot clé **end**).

• si **var** n'est pas égale à aucune des constantes const1, const2,..., on exécute **instructions***n* (suite d'instructions par défaut), puis l'instruction qui suit le mot clé **end**.

• Il n'est pas nécessaire de prèvoir un cas par défaut (bien que cela soit préférable).

• S'il n'y a pas de cas par défaut, et si **var** n'est égale à aucune des constantes, alors on continue à la première instruction suivant le mot clé **end**.

• Si la séquence d'instructions à exécuter est la même pour plusieurs cas, il est possible de les regrouper. La syntaxe est alors : case{**const**k, **const**l,...}.

4.1.3 Les boucles

• permettent d'exécuter une séquence d'instructions de manière répétée.

• Le nombre d'itérations (répétitions) est soit connu d'avance (boucle **for**), soit déterminé au cours de l'exécution (boucle **while**).

1) La boucle for

Syntaxe :

for indice = inf : sup

instructions

end

Exemple : calcul du factoriel.

»n = 4;

»fact= 1 ;

»**for** k = 1 : n

fact = fact∗k;

end;

»fact

fact =

24

• **indice** est une variable appelée l'indice de la boucle.

• **inf** (borne inférieure) et **sup** (borne supérieure) sont deux constantes réelles.

• **instructions** est la suite d'instructions à répéter (On parle du corps de la boucle).

• Si inf ≤ sup, **instructions** est exécutée (sup−inf+1) fois, pour les valeurs de la variable **indice** égales à inf , inf+1 . . . ,sup, puis on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle **end**.

• Si inf > sup, on passe directement à l'instruction qui suit immédiatement l'instruction de fin de boucle **end**.

• L'**indice** de boucle ne prend pas nécessairement des valeurs entières.

• On peut naturellement imbriquer des boucles **for** les unes dans les autres.

• Il est possible d'utiliser un incrément (pas) autre que 1 (valeur par défaut). La syntaxe est alors :

for indice=inf : pas : sup

instructions

end

• Le pas peut être negatif.

2) La boucle while

• permet de répéter une suite d'instruction tandis qu'une expression logique est

vraie.

Syntaxe :

while (expression)

instructions

end

Exemple : calcul du factoriel.

»n = 4;

»fact= 1;

» k = 1;

»**while** k <= n

fact = fact*k;

k = k + 1;

end;

»fact

fact =

24

• **expression** est une expression logique.

• **instructions** est une suite d'instructions qui se répète tant que **expression** a la valeur **vrai (true)**.

• Lorsque la valeur de l'**expression** devient **fausse (false)**, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle **end**.

3) Interruption d'une boucle

• L'instruction **break** permet de sortir d'une boucle **for** ou d'une boucle **while**.

• L'exécution se poursuit alors séquentiellement à partir de l'instruction suivant le mot clé **end** fermant la boucle.

• En cas de boucles imbriquées, on interrompt seulement l'exécution de la boucle intérieure contenant l'instruction **break**.

4.2 Fonction

• En plus des fonctions prédéfinies, Matlab permet à l'utilisateur de définir ses propres fonctions.

• Trois méthodes sont possibles :

1) les **m-fonctions** qui sont associées à des m-files.

2) les fonctions **Inline**.

3) les fonctions **anonymes**.

• Les deux derniers types de fonctions peuvent être définis directement dans l'espace de travail où ces fonctions vont être utilisées.

4.2.1 Les m-fonctions

Ces fonctions sont associées à un m-file auxiliaire, ce qui apporte deux avantages :

1) la fonction possède son propre espace de travail.

2) il n'y a pas de limite à la complexité de la séquence de commandes qui définit la fonction.

• À fin d'utiliser une fonction appelée MyFunction, on saisie son code puis on l'enregistre sous le nom MyFunction.m.

• Il ne faut pas oublier de mettre le fichier MyFunction.m dans le répertoire courant.

Exemple : La fonction **produit** permettant de calculer le produit de deux valeurs réelles.

function res= **produit**(x,y)

[a1, b1] = size(x) ;

[a2, b2] = size(y) ;

if((*a*1 == 1)&(*b*1 == 1)&(*a*2 == 1)&(*b*2 == 1)

res=x*y ;

else

disp('erreur, les arguments doivent être des scalaires')

end

return

• Cette fonction est appelée depuis la fenêtre de commande comme suit :

»*a* = 3; *b* = 5; *c* = [12; 34];

» **produit**(a,b)

ans =

15

»**produit**(a,c)

erreur, les arguments doivent être des scalaires.

Syntaxe de fonction

• Une fonction est constituée par :

1) un **en-tête** qui a la forme suivante :

function result = **nom-de-fonction**(liste des paramètres séparés par des virgules), si la fonction renvoie une seule valeur.

function[result1, result2, ...] = **nom-de-fonction**(liste des paramètres séparés par des virgules), si la fonction renvoie plusieurs valeurs.

2) une section de commentaires (9 lignes au plus, débutant par le symbole %%).

3) le corps de la fonction défini par un script.

4) le mot **return** qui n'est pas obligatoire, mais il est conseillé.

Règles et propriétés

1) Le nom de la fonction est un **identificateur** construit conformément aux règles définies précédemment.

2) Le nom de la fonction et celui du fichier m-file qui contient sa définition doivent être identiques.

3) Chaque fonction possède son propre espace de travail et toute variable apparaissant dans le corps d'une fonction est locale à celle-ci.

4) L'exécution d'une fonction s'achève :

• lorsque la fin du script définissant la fonction a été atteinte.

• lorsque une commande **return** ou un appel de la fonction error a été rencontré.

• **return** termine immédiatement l'exécution de la fonction sans que la fin du script définissant celle-ci ait été atteinte.

• error ('message') procède de même, mais en plus, affiche le contenu de message.

• Le contrôle est alors renvoye au point d'appel de la fonction, fenêtre de commande ou autre fonction.

5) Le fichier m-file associé à une fonction peut contenir d'autres définitions de fonctions.

• La fonction qui partage son nom avec le fichier ou fonction principale doit apparaitre en premier.

• Les autres fonctions ou fonctions internes peuvent être appelées par la fonction principale, mais pas par d'autres fonctions ou depuis la fenêtre de commande.

4.2.2 Les fonctions Inline

• Lorsque le corps de la fonction se résume à une expression relativement simple, on peut créer la fonction directement dans l'espace de travail courant, sans utiliser un m-file auxiliaire.

• **Syntaxe :** La syntaxe des fonctions Inline est simple :

nom-de-fonction = inline ('expression', 'var1', 'var2',...)

L'expression mathématique qui constitue le corps de la fonction ainsi que les variables sont considérées par Matlab comme des chaînes de caractères et doivent donc être tapées entre apostrophes.

La déclaration des variables peut être optionnelle dans la définition des fonctions Inline.

Matlab effectuant une déclaration implicite de celles-ci.

Cette facilité, source d'ambiguités dans le cas de fonctions de plusieurs variables(n'est pas à recommander dans ce cas).

Exemple1 **:**

» f = inline('x.^2 + $x. * y$',' x',' y')

f = Inline function :

f(x,y)=x.^2 + $x. * y$

» f(1,2)

ans =

3

» f([1 2], [3 4])

ans =

4 12

Exemple2 : (mécanisme de déclaration implicite)

» f = inline ('x.^2')

f =

Inline function :

f(x) = x.^2

est équivalente à :

» f =inline ('x.^2','x')

4.2.3 Les fonctions anonymes

• Ce mode de définition de fonctions utilise comme pour les fonctions Inline l'espace de travail courant.

• **Syntaxe :** nom-de-fonction = @(var1, var2,...)expression

Contrairement aux fonctions Inline l'expression mathématique qui constitue le corps de la fonction ainsi que les variables ne doivent pas être tapées entre apostrophes.

Exemple :

» g =@(x, y)x.^2 + x. * y

g =

@(x, y) x.^2 + x. * y

» g(1, 2)

ans =

3

»g([1 2], [3 4])

ans =

4 12