University Mohamed Boudiaf of M'sila

Faculty of Technology

Department of Electrical Engineering

Date : December 24, 2020

Level : Licence 2

TP5 : Instructions de contrôle (Boucles for et while, Instuctions if et switch)

1 Introduction :

Dans sa forme la plus simple, le déroulement d'un programme est linéaire dans le sens où les instructions qui le composent s'exécutent successivement. Les structures de contrôle sont des mécanismes qui permettent de modifier la séquence d'exécution des instructions. Plus précisément, lors de l'exécution, en fonction des conditions réalisées certaines parties précises du code seront exécutées.

Les instructions de contrôle sous `Matlab` sont très proches de celles existant dans d'autres langages de programmation.

2 Boucle `for` : parcours d'un intervalle:

Une première possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle pour les valeurs d'un indice, incrémenté à chaque itération, variant entre deux bornes données. Ce processus est mis en œuvre par la boucle for.

Syntaxe :

for indice=borne_inf:borne_sup

séquence d'instructions

end

où

indice est une variable appelée l'indice de la boucle;

borne_inf et borne_sup sont deux constantes réelles (appelées paramètres de la boucle);

séquence d'instructions est le traitement à effectuer pour les valeurs d'indices variant entre borne_inf et borne_sup avec un incrément de 1. On parle du corps de la boucle.

Interprétation :

Si borne_inf est plus petit ou égal à borne_sup, le traitement séquence d'instructions est

exécuté borne_sup - borne_inf fois, pour les valeurs de la variable indice égales à borne_inf, borne_inf+1, ..., borne_sup. Si borne_inf est strictement plus grand que borne_sup, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (end).

Remarque : L'indice de boucle ne prend pas nécessairement des valeurs entières. D'autre part il n'est pas nécessaire que l'indice de la boucle apparaisse dans le corps de la boucle; par contre il est interdit de modifier sa valeur s'il apparaît. Il est possible d'imbriquer des boucles mais elles ne doivent pas se recouvrir. On peut utiliser un incrément (pas) autre que 1 (valeur par défaut). La syntaxe est alors borne_inf: pas : borne_sup. Le pas peut être négatif. Attention à bien gérer la borne supérieure!

Voici un exemple d'utilisation d'une boucle pour calculer $n!$ (le lecteur attentif sait calculer $n!$ plus simplement ... par exemple en exécutant `prod([1:n])`).

» n = 4;

» nfac = 1;

» for k = 1:n

nfac = nfac*k;

end

» nfac

nfac =

24

» On peut créer une boucle en utilisant `for ... end`. On peut aussi réaliser des boucles `for` imbriquées.

Exemple

Boucle `for` simple:

```
for i=1:100
wt = 24*i*0.01;
x(i)=12.5*cos(wt+pi/6);
end
```

Deux boucles `for`:

```
for i=1:5
for j=1:20
amp=i*1.2;
wt=j*0.05;
v(i,j)=amp*sin(wt);
end
```

end

3 Boucle `while` : tant que ... faire:

Une seconde possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle tant qu'une condition reste vérifiée. On arrête de boucler dès que cette condition n'est plus satisfaite. Ce processus est mis en œuvre par la boucle while.

Syntaxe :

while expression logique

séquence d'instructions

end

où

expression logique est une expression dont le résultat peut être vrai ou faux;

séquence d'instructions est le traitement à effectuer tant que expression logique est vraie.

Interprétation :

Tant que expression logique est vraie le traitement séquence d'instructionsest exécuté sous forme d'une boucle. Lorsque expression logique devient faux, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (end).

Remarque : expression logique est en général le résultat d'un test (par exemple i<Imax) ou le résultat d'une fonction logique (par exemple all(x)).

Voici comment calculer $n!$ avec une boucle while: » n = 4;

» k = 1; nfac = 1;

» while k <= n

nfac = nfac*k;

k = k+1;

end

» nfac

nfac =

24

»

4 Branchement conditionnel (`if ... elseif ... else`)

Cette structure permet d'exécuter un bloc d'instructions en fonction de la valeur logique d'une expression. Sa syntaxe est :

if expression

instructions ...

end

L'ensemble des instructions instructions est exécuté seulement si expression est vraie.

Plusieurs tests exclusifs peuvent être combinés.

if expression1

instructions1 ...

elseif expression2

instructions2 ...

else

instructions3 ...

end

Plusieurs elseif peuvent être concaténés. Leur bloc est exécuté si l'expression correspondante est vraie et si toutes les conditions précédentes n'ont pas été satisfaites. Le bloc instruction3 associé au else est quant à lui exécuté si aucune des conditions précédentes n'a été réalisées.

Exemple

if x > 0

disp('x est positif');

elseif x == 0

disp('x est nul');

else

x = 1;

end

Bien évidement, la variable x doit être définie auparavant. La fonction disp permet d'afficher une chaîne de caractère spécifiée entre apostrophes. Si x n'est ni positif ni nul, il reçoit la valeur 1.

5 Branchement multiple (`switch ... case`)

Dans cette structure, une expression numérique est comparée successivement à différentes valeurs.

Dès qu'il y a identité, le bloc d'instructions correspondant est exécuté. Sa syntaxe est :

switch expression

case valeur1,

instructions1 ...

case valeur2,

instructions2 ...

case valeur3,

instructions3 ...

.....

otherwise

instructions ...

end

L'expression testée, expression, doit être un scalaire ou une chaîne de caractère. Une fois qu'un bloc instructionsi est exécuté, le flux d'exécution sort de la structure et reprend après le end. Si aucun case vérifie l'égalité, le bloc qui suit otherwise est exécuté.

Exemple

switch x

case 0,

resultat = a + b;

case

resultat = a * b;

case 2,

resultat = a/b;

case 3,

resultat = a^b;

otherwise

resultat = 0;

end

En fonction de la valeur de x une opération particulière est effectuée. Par défaut, resultat prend la valeur 0.