

CH 7 : Les structures de contrôle

b) *Les structure itératives*

I. Structures de contrôle

Reprenons la description du chapitre précédent. En informatique, on appelle **structure de contrôle** une commande qui contrôle l'ordre dans lequel les différentes instructions d'un programme sont exécutées. En **Scilab**, on peut distinguer plusieurs types de structures de contrôle :

- a) les **structures séquentielles** : présentées dans le « CH 2 : Premiers pas vers la programmation ». Les différentes commandes sont exécutées les unes à la suite des autres *i.e.* dans un ordre séquentiel.
- b) les **structures conditionnelles** : qui permettent d'introduire des branchements conditionnels dans le programme. Un programme peut comporter plusieurs branches. Chacune d'elle est associée à une condition. La branche exécutée est la première dont la condition est réalisée.
- c) les **structures itératives** : qui permettent d'effectuer la répétition (*i.e.* l'itération) de commandes. Ces répétitions peuvent s'effectuer un nombre de fois fixé explicitement par le programmeur ou peuvent avoir lieu tant qu'une condition n'est pas réalisée.

Dans ce chapitre, nous nous intéressons aux structures itératives.

II. Les boucles **for**

II.1. Syntaxe de la commande

II.1.a) Syntaxe générale

Une boucle **for** permet de réaliser la répétition d'un bloc d'instructions. La syntaxe de cette commande est la suivante.

```
for var = vect
    instruction
end
```

Détaillons les éléments de cette syntaxe.

- *vect* : est un vecteur de valeurs.
- *var* : est une variable qui prend successivement chacune des valeurs contenues dans *vect*.
- *instruction* : désigne un bloc d'instructions qui va être exécuté, de manière successive, pour toutes les valeurs de *var*.

II.1.b) Syntaxe du cas classique d'utilisation

Généralement, le vecteur *vect* sera présenté sous la forme *deb : pas : fin*. Ainsi, la variable *var* prendra successivement les valeurs de *deb* à *fin* avec une distance de *pas*. On rappelle que si la quantité *pas* n'est pas précisée, elle prendra par défaut la valeur 1.

La syntaxe classique d'utilisation est la suivante.

```
for var = deb : pas : fin
    instruction
end
```

II.2. Ordre d'exécution

Afin de détailler la séquence d'exécution on considère l'exemple consistant à calculer les m premiers éléments d'une suite (u_n) où m est une quantité choisie par l'utilisateur. Considérons ici la suite de terme général $u_n = \frac{1}{n}$.

```
// Calcul des m premiers éléments de la suite u_n=1/n
m = input("Entrez la valeur de m : ");
u = zeros(1,m);
for i = 1:m
    u(i) = 1/i;
end
```

La boucle **for** s'exécute comme suit :

- 1) initialement, la variable i est affectée à la valeur 1.
 \hookrightarrow l'instruction $u(1)=1/1$ est alors exécutée.
- 2) la variable i est ensuite affectée à la valeur 2 ($1 +$ le pas de 1).
 \hookrightarrow l'instruction $u(2)=1/2$ est alors exécutée.
- 3) la variable i est ensuite affectée à la valeur 3 ($2 + 1$).
 \hookrightarrow l'instruction $u(3)=1/3$ est alors exécutée.
- 4) ...
- $m-1$) la variable i est ensuite affectée à la valeur $m-1$.
 \hookrightarrow l'instruction $u(m-1)=1/(m-1)$ est alors exécutée.
- m) la variable i est ensuite affectée à la valeur m .
 \hookrightarrow l'instruction $u(m)=1/m$ est alors exécutée.

En sortie de boucle, le vecteur u contient les m premières valeurs de la suite de terme général $u_n = \frac{1}{n}$. La variable i , quant à elle, contient alors la valeur m .

III. Les boucles **while**

III.1. Syntaxe de la commande

Une boucle **while** permet de réaliser la répétition d'un bloc d'instructions sous réserve qu'une condition est vérifiée. La syntaxe de cette commande est la suivante.

```
while condition
    instruction
end
```

Détaillons les éléments de cette syntaxe.

- *condition* : représente une expression **Scilab** dont l'évaluation est soit **%t** soit **%f**. Autrement dit, un objet de type **boolean**.
- *instruction* : désigne un bloc d'instructions qui va être exécuté, de manière successive, **tant que condition** est vraie. Il est à noter que cette condition peut dépendre de variables qui sont modifiées lors de l'exécution de *instruction*.



Terminaison : dans le cas d'une boucle **while**, l'exécution se termine dès que *condition* n'est plus vérifiée. Si *condition* est toujours vérifiée (comme $0==0$ ou **%t**) le programme admet une exécution infinie. On parle de **boucle infinie**.

III.2. Ordre d'exécution

Afin de bien comprendre l'ordre d'exécution d'une boucle **while**, on va l'illustrer à l'aide d'un exemple consistant à trouver, dans un vecteur, le premier élément vérifiant une condition donnée. Commençons par générer un vecteur de valeurs aléatoires.

```
--> r=rand(1,7)
r =
    0.49    0.18    0.75    0.94    0.21    0.58    0.26
```

On demande alors à l'utilisateur d'entrer un réel t entre 0 et 1 dans le but de déterminer le premier élément du vecteur r plus grand que t .

```
// Le premier élément de r plus grand que le réel t demandé
t = input("Choisissez une valeur pour t : ");
i = 1;
while r(i) < t
    i = i+1;
end
disp("Dans r, le 1er élément >= à t est " + string(r(i)))
```

Si l'utilisateur choisit la valeur 0.5, on obtient le retour console suivant.

```
--> exec('/Info/Exemple_cours/exempleWhile.sce',-1)
Choisissez une valeur pour t : 0.5

Dans r, le 1er élément >= à t est 0.75
```

Dans cet exemple, la variable i est initialement affecté à la valeur 1. La boucle `while` s'exécute alors comme suit :

- 1) on teste si $r(1) < 0.5$.
 \hookrightarrow c'est le cas ($r(1) = 0.49$). Les instructions de la boucle sont donc exécutées. La variable i prend alors la valeur 2 ($1 + 1$).
- 2) on teste si $r(2) < 0.5$.
 \hookrightarrow c'est le cas ($r(2) = 0.18$). Les instructions de la boucle sont donc exécutées. La variable i prend alors la valeur 3 ($2 + 1$).
- 3) on teste si $r(3) < 0.5$.
 \hookrightarrow c'est le cas ($r(3) = 0.75$). Les instructions de la boucle sont donc exécutées. La variable i prend alors la valeur 4 ($3 + 1$).
- 4) on teste si $r(4) < 0.5$.
 \hookrightarrow ce n'est pas le cas ($r(4) = 0.94$). Les instructions de la boucle ne sont pas exécutées et il y a sortie de la boucle.
- 5) L'instruction d'affichage `disp` est alors exécutée.

Si l'utilisateur choisit la valeur 0.95, on obtient le retour console suivant.

```
--> exec('/Info/Exemple_cours/exempleWhile.sce',-1)
Choisissez une valeur pour t : 0.95

Le vecteur r ne possède pas d'éléments >= à t
```

Pour comprendre ce message d'erreur, étudions l'exécution de la boucle `while` correspondante. La variable i est initialement affectée à la valeur i . Puis, on teste si les éléments successifs du vecteur r sont strictement plus petits que 0.95. Revenons sur le dernier test.

- 7) on teste si $r(7) < 0.5$.
 \hookrightarrow c'est le cas ($r(7) = 0.26$). Les instructions de la boucle sont exécutées. La variable i prend alors la valeur 8 ($7 + 1$).
- 8) on teste si $r(8) < 0.5$.
 \hookrightarrow ce test provoque l'affichage d'un message d'erreur. En effet, on tente un accès hors index puisqu'il n'y a que 7 éléments dans le vecteur r .

Il faut donc modifier le script du programme de sorte à ne pas réaliser d'accès hors index. Pour ce faire, il suffit de vérifier que i ne dépasse pas la taille du vecteur avant de réaliser le test $r(i) < t$.

```
// Le premier élément de r plus grand que le réel t demandé
t = input("Choisissez une valeur pour t : ");
n = length(r);
i = 1;
while (i <= n) & (r(i) < t)
    i = i+1;
end
if (i > n) then
    disp("Le vecteur r ne possède pas d'éléments >= à t");
else
    disp("Dans r, le 1er élément >= à t est "+string(r(i)));
end
```

On teste ce nouveau script. Si l'utilisateur choisit la valeur 0.95, on obtient maintenant le retour console suivant.

```
--> exec('/Info/Exemple_cours/exempleWhile.sce',-1)
Choisissez une valeur pour t : 0.95

Le vecteur r ne possède pas d'éléments >= à t
```

La modification a bien empêché l'accès hors index. Ceci est notamment dû au mécanisme d'évaluation de l'opérateur `&`. À chaque tour de boucle, la condition `(i <= n) & (r(i) < t)` est testée. L'évaluation de ce test se déroule comme suit.

Tout d'abord, l'interpréteur teste si `(i <= n)`.

- (i) Si c'est le cas, `(r(i) < t)` est alors testé.
 - a) Si c'est le cas, la condition est vérifiée et les instructions de la boucle sont exécutées.
 - b) Sinon, la condition n'est pas réalisée et on sort de la boucle.
- (ii) Sinon, la condition n'est pas réalisée et on sort de la boucle.

IV. Les différences entre les boucles `for` et `while`

IV.1. Simuler une boucle `for` à l'aide d'un `while`

On peut simuler un `for` à l'aide d'un `while` en utilisant un compteur que l'on incrémente à chaque tour de boucle.

Sur l'exemple précédent, ceci se fait de la manière suivante.

```
// Calcul des m premiers éléments de la suite u_n=1/n
m = input("Entrez la valeur de m : ");
u = zeros(1,m);
i = 1;
while i <= m
    u(i) = 1/i;
    i = i+1;
end
```

Dans cet exemple, la variable `i` est initialement affectée à la valeur 1. La boucle `while` s'exécute alors comme suit :

- 1) on teste si `1 <= m`.
 \hookrightarrow c'est le cas. Les instructions de la boucle sont donc exécutées. Ainsi `u(1)` prend la valeur 1 ($1/1$) et `i` prend la valeur 2 ($1 + 1$).
- 2) on teste si `2 <= m`.
 \hookrightarrow c'est le cas. Les instructions de la boucle sont donc exécutées. Ainsi `u(2)` prend la valeur 0.5 ($1/2$) et `i` prend la valeur 3 ($2 + 1$).
- 3) ...
- m) on teste si `m <= m`.
 \hookrightarrow c'est le cas. Les instructions de la boucle sont donc exécutées. Ainsi `u(m)` prend la valeur $1/m$ et `i` prend la valeur $m+1$.
- $m+1$) on teste si `m+1 <= m`.
 \hookrightarrow ce n'est pas le cas. Les instructions de la boucle ne sont pas exécutées et il y a sortie de la boucle.

Au final, on obtient bien le même résultat que le précédent : le vecteur ligne u contient à la fin du script les m premiers éléments de la suite $(u_n) = \frac{1}{n}$.

IV.2. Boucle **for** ou boucle **while** : que choisir ?

Le choix de la structure itérative est fonction du problème que l'on a à résoudre. Plus précisément :

- 1) si l'on connaît a priori le nombre d'itérations à effectuer, on se tournera plutôt vers une boucle **for**.
 - a) Modifier tous les coefficients d'un vecteur ligne / colonne.
 - b) Calculer les n premiers éléments d'une suite.
 - c) Calculer une somme $\sum_{k=1}^n$ ou un produit $\prod_{k=1}^n$.
- 2) si l'on ne connaît pas a priori le nombre d'itérations à effectuer, on se tournera plutôt vers une boucle **while**.
 - a) Trouver l'indice du premier coefficient non nul d'un vecteur.
 - b) Pour une suite donnée, trouver l'indice du premier élément vérifiant une condition donnée.
 - c) Connaissant le chiffre d'affaire \$ d'une entreprise et une estimation de son pourcentage de progression pour les 10 années à venir, calculer en combien de temps \$ sera doublé.