Informatique et Algorithmique – Chapitre 3

Fonctions en Scilab

Comme on a pu le voir jusqu'à présent, Scilab dispose de nombreuses commandes et fonctions diverses. On sera vite amené à créer nos propres fonctions afin de simplifier l'écriture d'un programme et de condenser en une seule commande une suite d'instructions.

Ces instructions dépendent éventuellement de variables d'entrées (appelées arguments d'appel ou d'entrée) qui peuvent être des nombres, des vecteurs, des booléens, des chaines de caractères... Après l'exécution de ces instructions, la fonction renvoie généralement une ou plusieurs des variables de sortie (appelées arguments de sortie).

Création d'une fonction Scilab : On écrit le code d'une fonction Scilab dans un script SciNotes. La syntaxe d'une fonction est la suivante :

```
function [y1,...yp]=nomfonction(x1,...,xn)
    instructions
endfunction
```

Les variables `x1,...,xn` sont les arguments d'entrées de cette fonction. Elles vont intervenir dans le bloc d'instructions `instructions`. Les variables `y1,...,yn` sont créées et/ou modifiées par le bloc d'instructions et sont les arguments de sortie. Ici la fonction s'appelle `nomfunction`. Le script contenant le code doit être enregistré dans le répertoire de travail sous le nom `nomfunction.sci` (insistons sur l'extension `.sci` et non `.sce`).

Utilisation d'une fonction Scilab : Pour utiliser la fonction `nomfunction` que nous venons de créer, il faut la charger dans l'environnement de travail Scilab. Pour cela on tape la commande `exec('nomfunction.sci',-1)` dans la console Scilab (en s'assurant au préalable que la fonction est bien enregistrée dans le répertoire de travail). On peut également utiliser le menu `Exécuter` ou les raccourcis Ctrl + E ou Ctrl + Maj + E (selon que l'on souhaite une exécution avec ou sans écho). La fonction que l'on a créée et chargée se rajoute alors aux fonctions Scilab et s'utilise de la même façon.

Remarques :

- Le bloc d'instruction `instructions` s'appelle le corps de la fonction.
- Si la fonction ne renvoie qu'une seule variable en sortie, alors les crochets dans la première ligne de code sont facultatifs.
- ⚠ Avant de l'utiliser, il faut recharger la fonction dès qu'on la modifie.
- ⚠ Ne pas nommer une fonction avec un nom déjà utilisé par une fonction Scilab.
- Les variables utilisées dans le corps d'une fonction sont des variables internes, c'est-à-dire les modifications apportées à un argument d'entrée dans le corps de la fonction ne sont pas visibles à l'extérieur de la fonction (sauf si l'argument d'entrée est aussi un argument de sortie).

  ⚠ Si le script d'une fonction utilise une variable qui n'est pas un argument d'entrée et qui n'est pas définie dans le corps de la fonction, alors
  - ou bien la variable a été définie dans l'espace de travail et Scilab utilisera sa valeur par défaut.
  - ou bien elle n'a pas été définie et Scilab affichera un message d'erreur.

Exemple 1 : Écrivons une fonction Scilab qui prend un vecteur en entrée et renvoie la moyenne et l'écart type de ses coordonnées.

```
function [moy,ect]=MoyEct(X)
    n=length(X); s=0; t=0;
//Calcul de la moyenne :
    for  i=1:n
         s=s+X(i);
    end
    moy=s/n;
//Calcul de l'écart type:
    for  i=1:n
         s=s+(X(i)-moy)^2;
    end
    ect=sqrt(t/n);
endfunction
```

Exécutons cette fonction (appelée `MoyEct.sci`) sans écho dans la console et testons-la.

```
-->exec('C:\Gorny\Scilab\MoyEct.sci', -1)

-->V=[9.5,12,12.5,17,14]; MoyEct(V)
 ans  =
    13.

-->x=MoyEct(V); x
 x  =
    13.

-->[m,e]=MoyEct(V)
 e  =
    2.4779023
 m  =
    13.
```

⚠ Quand une fonction Scilab admet plusieurs arguments de sortie, si on l'applique à des arguments d'entrées, c'est uniquement la première sortie qui est affichée. De même si on stocke les sorties dans une seule variable, c'est uniquement la première sortie qui est stockée. C'est ce que l'on observe ci-dessus. Pour afficher les deux sorties de la fonction, il a fallu stocker les sorties de la fonctions dans un vecteur dont la taille est égale au nombre de sorties.

Remarque : Il existait déjà des fonctions prédéfinies dans Scilab qui calculent la moyenne et l'écart type des éléments d'un vecteur : il s'agit respectivemet de `mean` et `stdev`.

Exemple 2 : Implémentons la fonction $f : x \longmapsto \dfrac{\sin(x)}{1+x^2}$.

```
function y=f(x)
    y=sin(x)/(1+x^2);
endfunction
```

Exécutons cette fonction (appelée `f.sci`) sans écho dans la console et testons-là.

```
-->exec('C:\Gorny\Scilab\f.sci', -1)

-->f(3),f(-1)
 ans  =
    0.0141120
 ans  =
  - 0.4207355

-->f([1,2])
 !
at line      2 of function f
called by :
f([1,2])
```

⚠ Il faut penser à utiliser les opérations `.*`, `./` et `.^` afin de pouvoir prendre en argument des vecteurs ou des matrices.

Modifions la fonction pour permettre cela :

```
function y=f(x)
    y=sin(x)./(1+x.^2);
endfunction
```

N'oublions pas d'exécuter à nouveau la fonction la fonction... et testons-la :

```
-->exec('C:\Gorny\Scilab\f.sci', -1)

-->X=[1,2]; Y=f(X)
 Y  =
    0.4207355   0.1818595
```

 Par contre cette méthode ne fonctionne pas si la fonction présente des structures conditionnelles. Dans ce cas on pourra utiliser une boucle `for`, comme dans l'exemple suivant.

Exemple 3 : Implémentons la fonction $f : x \in \mathbb{R} \longmapsto \begin{cases} x\exp(x) & \text{si} \quad x \in]-\infty, 0] \\ \cos\left(\dfrac{9\pi x^2}{2}\right) & \text{si} \quad x \in]0, 1] \\ \sqrt{\ln(x)} & \text{si} \quad x \in]1, +\infty[\end{cases}$ en langage Scilab.

Le code le plus naturel est le suivant :

```
function y=g(x)
    if   x<=0   then
          y=x.*exp(x);
    elseif   x>1   then
          y=sqrt(log(x));
    else
          y=cos(9*%pi*x.^2/2);
    end
endfunction
```

Exécutons cette fonction (appelée `g.sci`) sans écho dans la console et testons-là.

```
-->exec('C:\Gorny\Scilab\g.sci', -1)

-->x=[-2,0.5,4.5]; g(x)
 ans  =
    1.  - 0.9238795  - 0.9238795
```

```
-->g(-2), g(0.5), g(4.5)
 ans  =
  - 0.2706706
 ans  =
  - 0.9238795
 ans  =
    1.2264083
```

Dans l'exemple ci-dessus, on constate qu'il y a une erreur lorsque l'on évalue la fonction g en un vecteur. En effet les conditions `x<=0` et `x>1` ne sont pas vraies pour les trois coordonnées de `x` donc c'est l'instruction du `else` qui est appliquée pour les trois coordonnées. Modifions le code en utilisant une boucle `for`.

```
function y=g(x)
//On crée un vecteur nul de la même taille que x.
    n=length(x); y=zeros(1:n);
//Pour chaque i, on stocke dans y(i) la valeur de f(x(i))
    for   i=1:n
          if   x(i)<=0   then
                y(i)=x(i)*exp(x(i));
          elseif   x(i)>1   then
                y(i)=sqrt(log(x(i)));
          else
                y(i)=cos(9*%pi*x(i)^2/2);
          end
    end
endfunction
```

N'oublions pas d'exécuter à nouveau la fonction la fonction... et testons-la :

```
-->exec('C:\Gorny\Scilab\g.sci', -1)

-->x=[-2,0.5,4.5]; g(x)
 ans  =
  - 0.2706706 - 0.9238795   1.2264083
```

Une dernière façon (plus condensée) de coder cette fonction est d'utiliser des indicatrices mais celles-ci sont hors-programme :

```
function y=f(x)
    y=x.*exp(x).*(x<=0)+sqrt(log(x)).*(x>1)+cos(9*%pi*x.^2/2).*((x<=1)&(x>0));
endfunction
```