IV. Les fonctions

1. Introduction

Les fonctions sont les pièces constituant un programme. On leur donne des paramètres en entrée et elles exécutent une action ou retournent une valeur. A la différence des scripts, ce qui se passe en interne d'une fonction est (presque) indépendant du workspace. On parle de portée des variables. Les variables sont de 3 types :

- Les variables d'entrée de la fonction.
- Les variables de sortie de la fonction, auxquelles vous devez attribuer une valeur.
- Les variables locales, qui sont temporaires et qui disparaissent quand on quitte la fonction. Elles ne sont donc pas accessibles à l'extérieur de la fonction.

Habituellement, on utilise les fonctions afin de [1] :

- Programmer des opérations répétitives → réutilisation, recyclage
- Limiter le nombre de variables dans l'invite Matlab → encapsulation
- Diviser le programme (problème) de manière claire

2. Principes des fonctions

Pour rappel, toutes les variables utilisées dans un script sont disponibles à l'invite MATLAB [1]. C'est-à-dire que, après l'exécution du script suivant :

```
clear all
a = 2;
b = a^3;
```

les variables a et b sont disponibles dans le workspace de Matlab.

Le principe d'une fonction est d'effectuer des opérations sur une ou plusieurs variables d'entrée (arguments) pour restituer un résultat appelé sortie [1]. Les variables internes à la fonction ne font pas partie du workspace global, et ne sont pas accessibles par une autre fonction. Elles existent uniquement dans la fonction et sont appelées variables locales.

Une fonction est contenue dans un fichier .m avec le même nom que la fonction. Le fichier qui définit une fonction doit commencer par [3]:

```
function [output arguments] = nom_fonction(input arguments)
```

Exemple de fonction (myfonction.m) :

```
function returnval = myfunction(var1, var2)

newvar = var1^2;
returnval = newvar*var2;
```

On l'utilise de la manière suivante :

```
>> x = myfunction(2, 3)
x =
    12
```

2.1. Les variables locales

Comme précisé précédemment, il n'y a aucun lien entre les variables du workspace global et celles de la fonction, définies de cette manière. Ceci implique :

1. Que les variables d'entrée et de sortie peuvent avoir un nom différent. Par exemple, nous pouvons utiliser notre fonction précédente de la manière suivante:

```
>> a = 12;
>> b = 2;
>> x = myfunction(a, b)

x =
   288
```

`a` et `b` n'ayant pas le même nom que `var1` et `var2`. Ce qui signifie qu'il faut être prudent de passer les arguments dans le bon ordre, car une inversion ne sera pas identifiée.

2. Qu'il n'est pas possible de récupérer la valeur d'une variable définie dans la fonction, autre que celles de sortie. Dans notre cas, nous ne pouvons pas connaître la valeur de `newvar`.

3. Qu'il n'est pas possible d'utiliser une variable du workspace depuis une fonction.

2.2. Les variables globales

Après ces grands absolus, voyons l'exception… Les variables globales sont définies pour être partagées par plusieurs fonctions ou scripts. En étant globales, elles peuvent être utilisées n'importe où avec le même nom qui leur a été attribué la première fois. Mais pour être visible et donc utilisable, il est nécessaire de :

1. La définir en tant que variable globale avant même de l'utiliser la première fois :

```
global myvar;
myvar = 24;
```

2. L'appeler en tant que variable globale avant toute utilisation dans chaque fonction :

```
global myvar;
x = myvar * 3;
```

Il convient de limiter l'utilisation de ces variables globales. En effet, il est difficile de réutiliser des fonctions qui contiennent des variables globales, parce qu'elles ne fonctionneront pas d'office avec d'autres fonctions.

2.3. Sorties multiples

Notre fonction précédente ne retourne qu'une valeur. Mais il est possible de créer une fonction qui retourne plusieurs valeurs. Celle-ci doit alors lister les variables de sortie entre crochet :

```
function [retval1, retval2, retval3] = myfunction(var1, var2)

retval1 = var1^2;
retval2 = var1*var2;
retval3 = var2^2;
```

Elle est ensuite utilisée de la sorte (éviter l'affichage des sorties à l'aide du point-virgule) :

```
>> a = 12;
>> b = 2;
>> [x, y, z] = myfunction(a, b)

x =
   144
y =
    24
z =
     4
```

Si une valeur de sortie est omise, celle-ci est perdue, mais l'exécution de la fonction se passe correctement :

```
>> [x, y] = myfunction(a, b)

x =
   144
y =
    24
```

2.4. Fonctions natives

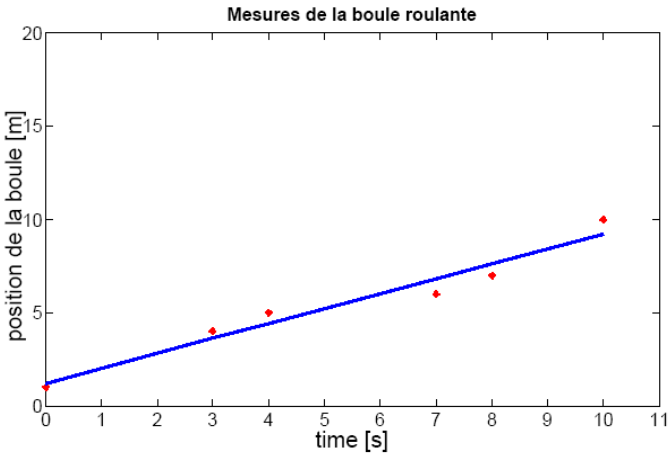
Matlab contient un grand nombre de fonctions disponibles (voir l'aide), qui fonctionnent exactement de cette manière. Quelques exemples :

```
>> x = max(3, 9)
x =
    9
```

```
>> x = sin(1)
x =
    0.8415

>> x = sum([1,2,3,4,5,6,7])
x =
    28

>> x = factorial(7)
x =
    5040

>> x = exp(3)
x =
    20.0855

>> [x,y] = size([1,2,3;4,5,6])
x =
    2
y =
    3
```

Il existe des fonctions plus complexes que ces derniers exemples. Ainsi, Matlab permet par exemple d'effectuer des régressions d'une manière relativement simple. Reprenons l'exercice du cours 2 :

t	y
0	1
3	4
4	5
7	6
8	7
10	10



La solution peut être obtenue directement grâce à la fonction `polyfit` :

```
>> t = [0,3,4,7,8,10]
>> y = [1,4,5,6,7,10]
>> resmodel = polyfit(t,y,1)

resmodel =
    0.8020    1.2228
```

Ce qui signifie : $y^{est} = 0.8020 \cdot t + 1.2228$

3. Evaluer une fonction

Une autre manière d'évaluer (`feval`) une fonction dans Matlab nous permet d'utiliser des fonctions dont les noms eux-mêmes sont variables, ou dont la définition de la fonction est très courte (déclarées par la commande `inline`). A n'utiliser qu'en cas de nécessité (plutôt rare) ! Il faut alors que le nom de la fonction soit sous forme de texte :

```
[y1, y2, ...] = feval('myfunction', x1, ..., xn);
```

Les variables d'entrées sont introduites à la suite du nom de la fonction, et les valeurs de sortie sont récupérées de la même manière qu'auparavant. Dans le cas de fonctions existantes:

```
>> x = feval('max', 3, 9)
x =
    9

>> fctname = 'sum';
>> x = feval(fctname, [1,2,3,4,5,6,7])
x =
    28

>> fctname = 'size';
>> [x,y] = feval(fctname, [1,2,3;4,5,6])
x =
    2
y =
    3
```

Dans le cas où la fonction est déclarée en ligne [3]:

```
>> f = inline('x.^3 + a.*x','x','a');
>> xval = 2; aval = 3;
>> fval = feval(f, xval, aval)
fval =
     14
```

4. Interactions avec un programme

Il est recommandé de ne pas écrire les paramètres d'un script ou d'une fonction directement dans le code, car il y a un risque d'oublier de les changer. Il peut donc être utile d'introduire un moyen interactif de demander ces paramètres à l'utilisateur.

4.1. Entrées dans l'invite Matlab

Une méthode pour demander à l'utilisateur d'entrer une valeur est la commande `input` :

```
user_entry = input('prompt')          → pour les valeurs numériques
user_entry = input('prompt', 's')     → pour les textes
```

La question est écrite en tant qu'entrée de la fonction input et la réponse de l'utilisateur est récupérée en sortie de fonction. L'exécution du code est donc mise en pause jusqu'à ce que l'utilisateur donne une réponse.

Exemple avec valeur numérique :

```
function askforexp()

reply = input('Exponentiel de cette valeur : ');
answer = exp(reply);
disp(['Réponse: ', num2str(answer)])
```

Exemple avec texte :

```
function askfordate()

reply = input('Voulez-vous connaître la date ? Y/N : ', 's');

switch reply
    case 'Y'
        disp(['Nous sommes le ', date ])
    case 'N'
        disp('Tant pis.')
    otherwise
        disp('Réponse non autorisée.')
end
```

4.2. Les boîtes de dialogue

Une autre méthode pour demander des entrées de l'utilisateur est d'utiliser des boîtes de dialogues.

4.2.1. Avec champ à remplir

La première forme contient un champ que l'utilisateur rempli. Elle fonctionne sur le même principe que la fonction `input`. Les arguments 2 à 5 sont facultatifs.

```
answer = inputdlg(prompt, dlg_title, num_lines, defAns, options)
```

où :

`prompt` est la question (string ou cell).
`dlg_title` est le titre de la boîte de dialogue (string)
`num_lines` est le nombre de lignes autorisé dans la réponse (entier)
`defAns` est la réponse par défaut (string ou cell)
`options` permet de spécifier d'autres options d'affichage

La fenêtre peut contenir autant de questions que désiré. Les questions et les valeurs par défaut doivent être placées dans des cells (matrices pouvant contenir des éléments de différentes tailles, {…}) si leur nombre > 1. La réponse est également placée dans une cell.

Une cell ressemble à une matrice. Elle permet de stocker des chaînes de caractères de longueurs différentes dans chaque cellule. Elle est donc utilisée pour les variables de type texte, et requiert quelques manipulations supplémentaires.

- Création : le type cell est attribué lorsque nous définissons une variable en utilisant des accolades { } au lieu des crochets [ ]. Les règles lignes-colonnes et virgules / points-virgules sont les mêmes que pour les matrices.

- Extraction d'un élément : il est nécessaire de retransformer l'élément qui nous intéresse du type cell au type char en utilisant la fonction de conversion `char()`, ou au type numérique en utilisant `str2num()` ou `str2double()`.

Exemple :

```
function askfornamebirthday()

prompt = {'Prénom:','Date de naissance:','Nombre d''enfants:'};
dlg_title = 'Questionnaire';
num_lines = 1;
def = {'Jules','01.01.1950','0'};
answer = inputdlg(prompt,dlg_title,num_lines,def);

disp([char(answer(1)), ', né le ', char(answer(2)), ', a ',...
       char(answer(3)), ' enfants'])

x = str2num(answer{3});
```

4.2.2. Avec liste de réponses

La dernière approche consiste à fournir des réponses à choix sous forme de boutons. L'utilisateur doit alors choisir une des options. Il peut y avoir autant de réponses possibles que désiré. Cette fois-ci, il n'y a pas lieu d'utiliser des cells, tout se fait avec des strings. La réponse est également une chaîne de caractères, qui contient le texte de l'option sélectionnée.

```
button = questdlg(qstring, dlg_title, str1, str2, default)
```

où :

`qstring` est la question (string)
`dlg_title` est le titre de la boîte de dialogue (string)
`str1` est la première réponse (string)
`str2` est la deuxième réponse (string)
`default` est la réponse par défaut (string)

Exemple :

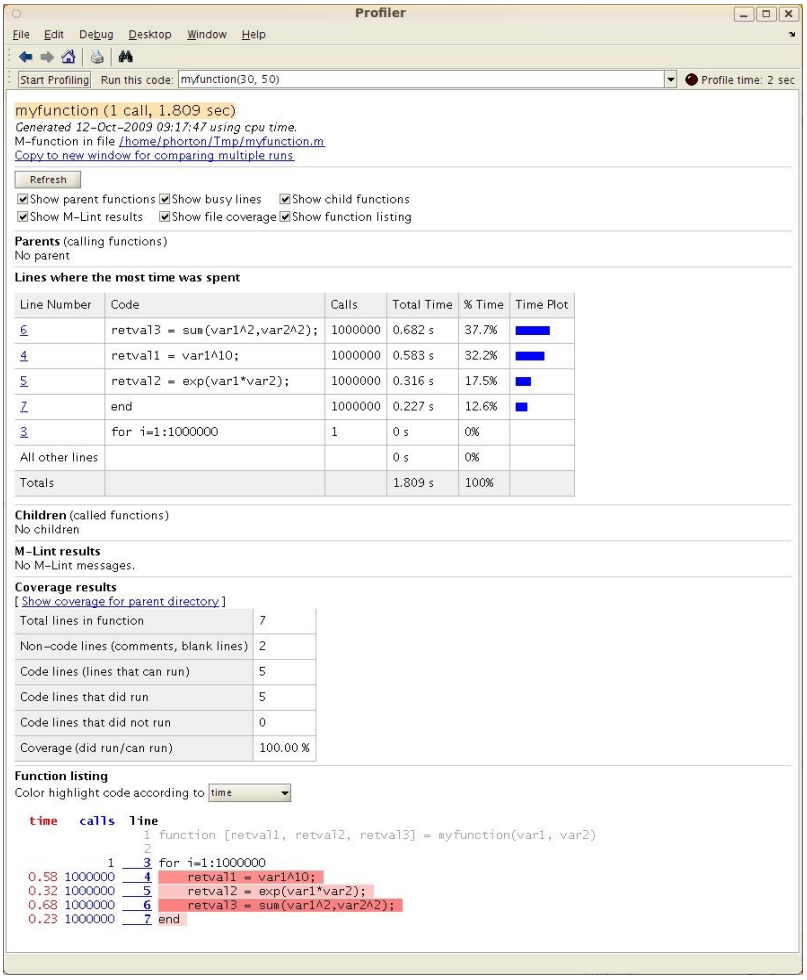
```
>> button = questdlg('Aimez-vous Matlab ?', 'Question', 'oui',
'non', 'pas encore', 'oui')
```

4.2.3. Autres

D'autres boîtes de dialogue existent. A voir: `errordlg`, `helpdlg`, `listdlg`, `msgbox`, `warndlg`, ...

5. Le Profiler

Le profiler est un outil utile lorsque le temps de calcul devient important. Celui-ci vous permet d'optimiser le code développé en vous indiquant le temps passé sur chaque ligne et dans chaque fonction.



6. Références

[1] Hudon Nicolas, 2004. *Initiation à MATLAB*. URCPC, Ecole Polytechnique de Montréal.

[2] MATLAB Help

[3] Dellis Jean-Luc. *Introduction à MATLAB*. Université de Picardie.

7. Auteurs

Pascal Horton (2009, 2015)