V. Lecture et écriture de fichiers

1. Introduction

La lecture et l'écriture de données dans des fichiers sont des étapes indispensables d'un programme, tant pour charger des données que pour sauver des résultats. Un format de fichier répandu est le fichier texte (ou ascii), qui est fréquemment utilisé pour stocker des données de toutes sortes. Nous allons apprendre à créer un tel fichier, ainsi qu'à en extraire les données.

Un nouveau type de variable présenté est la « structure ». Celle-ci permet de mieux organiser les variables dans le workspace et de garder une certaine rigueur.

Une grande partie du contenu de ce cours provient de l'aide de Matlab.

2. Introduction d'un nouveau type : les structures

Matlab supporte des données plus complexes que les matrices : les structures. Comme son nom l'indique, ce nouveau type permet de structurer les données dans une même variable. La variable contient alors en elle-même des sous-variables de différents types, permettant ainsi de regrouper de manière pertinente des informations.

2.1. Structures simples

Les sous-variables sont appelées des champs (fields) et peuvent contenir n'importe quel type de donnée [1] :

```
>> student.firstname = 'Jean-Charles';
>> student.lastname = 'André';
>> student.birthyear = 1986;
>> student.eval = [5, 4.5, 6, 3.5];

student =
    firstname: 'Jean-Charles'
     lastname: 'André'
    birthyear: 1986
         eval: [5 4.5000 6 3.5000]
```

Les champs peuvent être imbriqués sur plusieurs niveaux. Il n'y a pas de limite.

```
>> student.eval.math.algebra = [2, 3, 4.5];
>> student.eval.math.geometry = [5, 4.5, 6, 3.5];
>> student.eval.enviro.risk = [6, 5, 6, 4.5, 4];
>> student.eval.enviro.geo = [5, 6, 6];
```

```
student =
    firstname: 'Jean-Charles'
     lastname: 'André'
    birthyear: 1986
         eval: [1x1 struct]

>> student.eval.math

ans =
     algebra: [2 3 4.5000]
    geometry: [5 4.5000 6 3.5000]
```

2.2. Structures multi-dimensionnelles

L'approche vectorielle est également implémentée dans les structures. C'est-à-dire que nous pouvons regrouper plusieurs objets similaires dans une même structure. Dans ce cas, il convient de veiller à ce que la structure soit cohérente pour tous les objets, car les champs sont partagés. Si un champ n'est pas utilisé par un des objets, celui-ci reste vide. Les règles sont les suivantes [1] :

- Toutes les structures dans le tableau ont le même nombre de champs.
- Tous les champs ont le même nom.

```
>> student(1).firstname = 'Jean-Charles';
>> student(1).lastname = 'André';
>> student(1).birthyear = 1986;
>> student(2).firstname = 'Hamoul';
>> student(2).lastname = 'Saakhi';
>> student(2).birthyear = 1985;

student =
1x2 struct array with fields:
    firstname
    lastname
    birthyear
```

Les structures sont très utiles pour garder une cohérence dans les variables (p.ex. plusieurs caractéristiques d'un même objet) et en limiter leur nombre. Elles peuvent être utilisées par exemple pour garder les données d'entête d'un MNT avec les données elles-mêmes, pour rassembler tous les paramètres d'un modèle dans une même variable, etc.

2.3. Commandes sur les structures

Il existe quelques commandes sur les structures.

2.3.1. Supprimer un champ

La première permet d'éliminer un champ d'une structure existante :

```
>> newstudent = rmfield(student, 'birthyear')

newstudent =
1x2 struct array with fields:
    firstname
    lastname)
```

2.3.2. Déclarer une structure

Il est également possible de déclarer une structure à l'aide de la fonction `struct`. L'exemple suivant crée une structure avec différents niveaux [1].

```
A = struct( 'data', {[3 4 7; 8 0 1], [9 3 2; 7 6 5]}, ...
            'nest', {...
                    struct( 'testnum', 'Test 1', ...
                            'xdata', [4 2 8], 'ydata', [7 1 6]),…
                    struct( 'testnum', 'Test 2', ...
                            'xdata', [3 4 2], 'ydata', [5 0 9])});
```

La structure définie préalablement est illustrée dans la figure suivante :

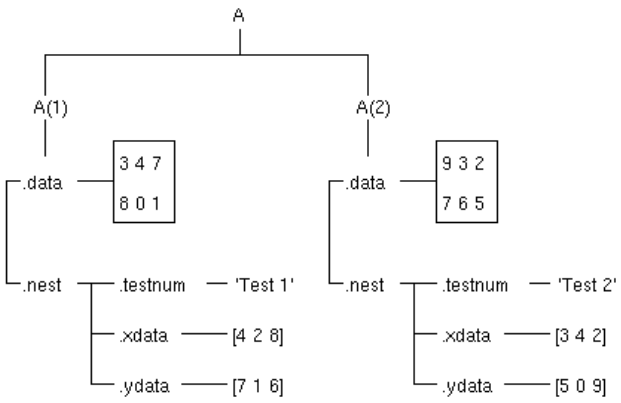

Fig. 1. Illustration de la structure précédemment définie (source : [1])

2.3.3. Tester l'existence d'un champ

La function `isfield` test l'existence d'un champ dans la structure. Elle retourne 1 si le champ existe, 0 autrement :

```
isfield(student,'firstname')
```

3. Importer et exporter des fichiers .mat

3.1. Exporter

Matlab fournit un format de fichier très pratique pour enregistrer et charger des données : les fichiers « .mat ». Les avantages de ces fichiers sont la rapidité de chargement et la petite taille qu'ils occupent.

Il est possible de sauver tout le workspace :

```
save mesdonnees
```

Toutes les variables du workspace seront enregistrées dans le fichier « mesdonnees.mat » dans le dossier courant.

Il est également possible de spécifier un chemin (Attention, le dossier doit exister avant. Il ne sera pas automatiquement créé. Il peut également être créé par Matlab à l'aide de la commande `mkdir('mondossier')`) :

```
save mondossier\mesdonnees
save D:\chemin\vers\mondossier\mesdonnees
```

Pour sauver uniquement certaines variables, il faut les lister à la suite du nom du fichier :

```
save mesvars var1 var2 var3
save('mesvars','var1','var2','var3')
```

3.2. Importer

L'importation des données est effectuée par la commande `load` :

```
load mesvars;
```

Toutes les variables préalablement sauvées sont restaurées avec leur nom d'origine. Afin de ne pas écraser d'autres variables présentes dans le workspace, les variables chargées peuvent être attribuées à une nouvelle variable, sous forme de structure.

```
>> mesnouvvars = load('mesvars')

mesnouvvars =
    var1: 100
    var2: 435
    var3: 54
```

4. Importer et exporter des fichiers textes (ascii)

4.1. Fonction save

Une première méthode est d'utiliser la même commande que pour les fichiers .mat, mais en précisant le type de fichiers `-ascii` après les variables. Il est également possible d'ajouter l'option `-tabs` pour que les valeurs soient séparées par des tabulations.

```
save('mesvars.txt','var1','var2','var3','-ascii')
save('mesvars.txt','var1','var2','var3','-ascii','-tabs')
```

Toutefois, cette méthode a beaucoup de limitations. Premièrement, la mise en page n'est pas contrôlable par l'utilisateur, ce qui fait que toutes les valeurs sont mises à la suite. Deuxièmement, il n'est possible d'enregistrer que des valeurs numériques. Aucun caractère, structure, cell, etc, n'est autorisé. Les noms des variables sont perdus. Finalement, le fichier doit avoir le même nombre de colonne pour chaque ligne (afin de pouvoir le recharger).

4.2. Plus de contrôle sur les fichiers ascii

Il existe une autre méthode pour mieux gérer les fichiers textes. Les étapes sont les suivantes :

4.2.1. Créer un fichier

La première étape est la création du fichier : `fopen`. Les paramètres de la fonction sont le nom du fichier (év. avec le chemin d'accès) et la permission. Les permissions sont les suivantes [1] :

`'r'`	Ouvre le fichier en lecture seule (défaut).
`'w'`	Ouvre le fichier en écriture, ou en crée un nouveau ; l'ancien contenu est écrasé.
`'a'`	Ouvre le fichier en écriture, ou en crée un nouveau ; le nouveau contenu est ajouté à la fin du fichier.
`'r+'`	Ouvre le fichier pour lire et écrire.
`'w+'`	Ouvre le fichier en lecture et écriture, ou en crée un nouveau ; l'ancien contenu est écrasé.
`'a+'`	Ouvre le fichier en lecture et écriture, ou en crée un nouveau ; le nouveau contenu est ajouté à la fin du fichier.

La fonction retourne un identifiant pour ce fichier qu'il nous faut conserver.

```
fid = fopen(filename, permission);
```

Exemple:

```
fid = fopen('d:\chemin\vers\monfichier.txt', 'w');
```

4.2.2. Ecriture des données formatées : fprintf

L'écriture dans les fichiers se fait avec la fonction `fprintf`. Les paramètres sont l'identifiant du fichier, le format, puis les données :

```
count = fprintf(fid, format, A, ...)
```

L'argument du format est un texte contenant des caractères et des spécifications de conversion des données. La sortie est la longueur du texte écrit. Les spécifications de conversion contrôlent la notation, l'alignement, le nombre de chiffres significatifs, la largeur du champ, ainsi que d'autres aspects d'affichage [1]. Les spécifications de conversion commencent avec le caractère `%` et contiennent les éléments suivants [1]:

- Flags (optionnel) : ils contrôlent l'alignement de la valeur à écrire

Caractère	Description	Exemple
Signe moins (-)	L'élément est justifié à gauche	`%-3.4d`
Signe plus (+)	Affiche toujours le signe (+/-)	`%+3.4d`
Espace	Insère un espace devant la valeur	`% 3.4d`
Zéro (0)	Met des 0 au lieu des espaces	`%03.4d`

- Longueur et précision (optionnel) : contrôlent la longueur de l'élément à imprimer et sa précision

Caractère	Description	Exemple
Longueur	Le nombre minimum d'éléments à imprimer	`%5f`
Précision	Un nombre à virgule (.) spécifiant le nombre de chiffres à imprimer à droite du point.	`%5.3f`

- Caractère de conversion (obligatoire) : spécifie la notation du nombre.

Type	Caractère	Description
Entiers signés	**%d** ou %i	Base 10
Entiers non signés	**%u**	Base 10
	%o	Base 8 (octal)
	%x	Base 16 (hexadécimal), lettres a–f minuscules
	%X	Comme %x, lettres A–F majuscules
Nombres à virgule flottante	**%f**	Notation à point fixe
	%e	Notation exponentielle
	%E	Notation exponentielle avec « E » majuscule
	%g	Le plus compact de %e et %f, sans 0 de queue
	%G	Même que %g, mais avec un « E » majuscule
Texte	%c	Caractère simple
	%s	Chaîne de caractères

- Caractères de mise-en-page (optionnels)

Caractère	Description
\b	Efface le dernier caractère
\f	Saut de page
\n	Nouvelle ligne
\r	Retour à la ligne
\t	Tabulation
\\	Backslash
\'' ou ''	Simple apostrophe
%%	Pourcent

L'exemple ci-dessous crée un fichier log.txt contenant une table des logarithmes :

```
x = 1:.1:100;
y = [x; log(x)];
fid = fopen('log.txt', 'wt');
fprintf(fid, '%6.2f %12.8f\n', y);
fclose(fid)
```

Le fichier log.txt contient alors :

```
  1.00   0.00000000
  1.10   0.09531018
  1.20   0.18232156
  1.30   0.26236426
…
 99.90   4.60416969
100.00   4.60517019
```

4.2.3. Lecture d'un fichier : fscanf

La fonction `fscanf` est l'équivalent de la fonction `fprintf` pour la lecture des fichiers [1].

```
A = fscanf(fid, format)
```

La fonction lit tout le fichier et converti les données au format spécifié, puis les retourne à la matrice A.

```
[A,count] = fscanf(fid, format, size)
```

Avec le paramètre « size » en plus, la fonction lit le nombre de données spécifiées par celui-ci. Les options pour « size » sont les suivantes :

- n: Lit au plus n nombres, caractères ou chaînes de caractères.
- inf: Lit jusqu'au bout du fichier
- [m,n] : Lit au plus m*n nombres, caractères ou chaînes de caractères. Rempli une matrice d'au plus m lignes, colonne après colonne. n peut être « inf », mais pas m.

Le format est précisé afin que Matlab recherche les valeurs dans le fichier qui correspondent au format désiré. Si le format correspond, la valeur est conservée dans la matrice de sortie. Les spécifications de conversion correspondent en gros à celles utilisées pour l'écriture des fichiers.

- Flags (optionnels)

Caractère	Description	Exemple
`*`	Ignore la valeur	`%*e`
`(nombre)`	Longueur maximale du champ	`%12d`
`(lettre)`	Taille de l'objet recevant la valeur. h pour short ou l pour long	`%hd` `%lg`

- Caractère de conversion (obligatoire) : très semblable au tableau de 4.2.2 mais sans les majuscules (%X, %E, %G)

En reprenant le fichier log.txt:

```
  1.00   0.00000000
  1.10   0.09531018
  1.20   0.18232156
  1.30   0.26236426
…
 99.90   4.60416969
100.00   4.60517019
```

Sa lecture est effectuée ainsi :

```
fid = fopen('log.txt', 'r');
a = fscanf(fid, '%g %g', [2 inf]);   % sur 2 lignes
a = a';                              % transposition en colonnes
fclose(fid)
```

4.2.4. Lecture avec de multiples variables de sortie : textread

La différence de `textread` par rapport à `fscanf` est la capacité de celui-ci à extraire de multiples variables en une commande. La fonction est effectuée jusqu'à ce que le fichier soit entièrement lu. `textread` est utile pour lire des fichiers textes avec un format connu [1]. Il converti des groupes de caractères dans le format désiré. Chaque élément dans le fichier est défini comme une chaîne de caractères sans espace, qui s'étend jusqu'au prochain espace ou caractère de délimitation [1]. Le nombre d'éléments lus est le nombre d'éléments dans le paramètre `format`.

```
[A,B,C,...] = textread('filename','format')
```

Pour contrôler le nombre d'éléments lus, un paramètre supplémentaire, spécifiant le nombre d'extractions, peut être ajouté.

```
[A,B,C,...] = textread('filename','format',N)
```

Il est également possible d'ajouter d'autres paramètres. Le lecteur se référera à l'aide de Matlab.

```
[...] = textread(...,'param','value',...)
```

Comme auparavant, les spécifications de conversion sont les suivantes [1]:

Format	Description	Sortie
literal	Ignore le texte correspondant	Aucune
%d	Entier (signé)	Double array
%u	Entier	Double array
%f	Nombre à virgule flottante	Double array
%s	Chaîne de caractères séparée par un espace ou un délimiteur	Cell array
%q	Chaîne de caractères entre crochets (ignore les crochets)	Cell array
%c	Caractères, y compris les espaces	Char array
%[...]	Lit la plus longue chaîne de caractères contenant les caractères spécifiés dans les crochets	Cell array
%[^...]	Lit la plus longue chaîne de caractères non vide contenant des caractères qui ne sont pas spécifiés dans les crochets	Cell array
%*...	Ignore le caractère correspondant	Aucune

Exemple :

Fichier contact.dat, première ligne :

```
Sally    Level1 12.34 45 Yes
```

Lecture de la première ligne du fichier « contact.dat » en format libre [1] :

```
[names, types, x, y, answer] = textread('mydata.dat', ...
    '%s %s %f %d %s', 1)

names =
    'Sally'
types =
    'Level1'
x =
  12.34000000000000
y =
    45
answer =
    'Yes'
```

Lecture de la première ligne du fichier « contact.dat » en format fixe et en ignorant le nombre à virgule flottante [1] :

```
[names, types, y, answer] = textread('mydata.dat', ...
'%9c %5s %*f %2d %3s', 1)

names =
Sally
types =
    'Level1'
y =
    45
answer =
    'Yes'
```

Lecture de la première ligne du fichier « contact.dat » en utilisant la correspondance littérale [1] :

```
[names, typenum, x, y, answer] = textread('mydata.dat', ...
'%s Level%d %f %d %s', 1)

names =
    'Sally'
typenum =
    1
x =
  12.34000000000000
y =
    45
answer =
    'Yes'
```

4.2.5. Lecture avec sortie en tableau de cells : textscan

La spécificité de cette fonction est de mettre tout le contenu du fichier dans un seul tableau de cells, et de pouvoir lire le fichier depuis un certain point autre que le début. L'utilisateur intéressé peut consulter l'aide de Matlab.

4.2.6. Fermer le fichier

Après avoir écrit ou lu le contenu du fichier, il est nécessaire de le refermer. La fonction `fclose` ferme le fichier correspondant et le rend à nouveau disponible pour d'autres programmes. Le paramètre en entrée est soit l'identifiant du fichier, soit « all », pour tous les fichiers.

```
status = fclose(fid)
status = fclose('all')
```

4.2.7. Autres fonction de traitement de fichiers

Il existe encore quelques autres fonctions pour lire et écrire dans des fichiers ascii. Le lecteur intéressé peut voir : fgets, fgetl, dlmread, dlmwrite, csvread, csvwrite

5. Références

[1] MATLAB Help

6. Auteurs

Pascal Horton (2009, 2015)