Analyse Numérique (MATH-H-202)

Eléments de syntaxe Octave

(version 0.0.7)

(disponible en ligne via
http://metronu.ulb.ac.be/MATH-H-202/an_syntaxe_octave.pdf)

Avant de commencer

Octave est un environnement de calcul scientifique qui dispose de multiples fonctions utiles en analyse numérique. Ce document a pour but d'introduire les quelques éléments de syntaxe Octave qui sont utiles pour le cours. Historiquement Octave a été développé comme une alternative libre et gratuite à Matlab®, la syntaxe de ces deux logiciels est donc virtuellement identique.

Les exécutables d'installation pour divers systèmes d'exploitation sont disponibles via

http://www.gnu.org/software/octave/download.html

et, alternativement,

https://www.jdbonjour.ch/cours/matlab-octave/

décrit une autre procédure d'installation, étape par étape. Pour les systèmes d'exploitation de type unix le logiciel est également inclus dans les distributions officielles, et peut donc être installé à l'aide du gestionnaire de programmes ad hoc. En cas de problème avec une des versions d'Octave vous pouvez répéter la procédure d'installation avec une autre version du logiciel ; les versions à partir de 3.2.0 sont typiquement suffisantes pour l'ensemble des travaux pratiques.

L'utilisation d'Octave se fait principalement dans un terminal de commandes. Lorsque certains groupes d'instructions doivent être réutilisés tels quels ou sous forme d'une fonction, ils peuvent aussi être encodés dans un fichier `*.m` .

Avant de commencer, passons en revue les quelques instructions de base.

`help` *commande*

indique ce que fait une commande particulière. Par exemple

```
help plot
```

indique comment utiliser la commande `plot` pour afficher des graphiques en deux dimensions. Plus précisément, `help` ouvre un environnement d'affichage séparé de l'environnement de travail principal : pour se déplacer dans ce dernier, suivez l'indication en bas à gauche

```
-- less -- (f)orward, (b)ack, (q)uit
```

(utilisez les touches `b` et `f` pour se déplacer dans le texte, `q` pour sortir). Pour interrompre l'exécution d'un programme, utilisez la combinaison de touches `Ctrl+c` . Vous pouvez aussi utiliser `Ctrl+c` pour sortir d'Octave, même si l'usage des instructions `exit` ou `quit` est plus conventionnel et recommandé.

Les instructions les plus récentes s'obtiennent avec la touche ↑ du clavier. La commande `history n` permet d'afficher les `n` dernières instructions.

Lors de son lancement, Octave choisit son répertoire de travail (celui dans lequel il cherchera les fichiers `*.m` que vous avez écrits). Pour déterminer ce répertoire, utilisez `pwd` (une abréviation de *print working directory*). Pour afficher son contenu, utilisez `ls` et pour changer de répertoire – `cd` *nouveau-répertoire* .

Le restant du document est organisé comme suit. La Section 1 porte sur les conditions, la Section 2 détaille l'usage des boucles, la Section 3 concerne l'édition des fichiers `*.m` , et la Section 4 explique l'utilisation des fonctions. Finalement, les quelques instructions qui ne sont pas accessibles via la commande `help` sont détaillées dans le *Petit Guide d'Octave* à la fin du document.

1 Conditions

La structure la plus simple qui permet de spécifier une condition est

```
if (condition)
  code pour if
end
```

Les conditions logiques sont données dans la Table 1. Les opérations logiques sont indiquées dans la Table 2.

Une structure plus complète est donnée par

```
if (condition)
  code pour if
else
  code pour else
end
```

ou encore

```
if (condition 1)
  code pour if
elseif (condition 2)
  code pour premier elseif
else
  code pour else
end
```

Plusieurs `elseif` dans une même construction sont également permis. Notons que `elseif` doit être «en un seul morceau», l'instruction `else if` étant interprétée comme un `else` suivi d'un `if` (ce qui n'est pas la même chose). Finalement, toutes ces constructions se terminent par un `end` , lequel délimite donc la portée de la condition.

relation	instruction
$<$ (ou $>$)	`<` (ou `>`)
$\leq$ (ou $\geq$)	`>=` (ou `<=`)
$=$	`==`
$\neq$	`~=`

TABLE 1 – Conditions logiques.

opérations	instruction
and	*cond 1* `&&` *cond 2*
	`and(`*cond 1*`,` *cond 2*`)`
or	*cond 1* `||` *cond 2*
	`or(`*cond 1*`,` *cond 2*`)`
not	`~=`*cond 1*
	`not(`*cond 1*`)`
xor	`xor(`*cond 1*`,` *cond 2*`)`

TABLE 2 – Opérations logiques.

Par exemple,

```
if (x<=2 && x>=0)
  disp('x est entre 0 et 2 (inclus)')
elseif (x > 0)
  disp('x est supérieur à 2')
else
  disp('x est négatif')
end
```

2 Boucles

La structure d'une boucle qui parcourt un ensemble prédéfini de valeurs est :

```
for i = vect
  code pour for
end
```

Ici, `vect` est un vecteur et `i` prend successivement la valeur de tous ses éléments (du premier au dernier). Pour parcourir les valeurs de `imin` à `imax` (avec `imin` $<$ `imax`) on utilise `imin:imax` à la place de `vect`. Par exemple,

```
for i = 1:10
  disp(i)
end
```

De même, pour parcourir les valeurs de `i1` à `i2` avec un pas de `d`, on utilise

`i1:d:i2` ; en particulier, `imax:-1:imin` permet de parcourir les valeurs de `imax` à `imin` dans le sens décroissant avec un pas de 1.

Une boucle peut être interrompue à tout moment avec l'instruction `break` (en cas de plusieurs boucles imbriquées, seulement une est interrompue ; il s'agit de la boucle la plus interne dans laquelle `break` se trouve).

Notons que des boucles `while` et `do - until` existent également. Un exemple avec `while` serait

```
i=1;
while (i <= 10)
  disp(i); i++;
end
```

et avec `do - until` :

```
i=1;
do
  disp(i); i++;
until (i > 10)
```

3 Edition des fichiers `*.m`

Un fichier `*.m` contient des instructions ou des fonctions Octave sous forme de texte non formaté. Il peut être édité à l'aide d'un programme de traitement de texte standard, tel que `notepad` sous windows ou `TextEdit` sous mac os x. Les éditeurs de code qui reconnaissent l'extension `*.m` (comme `gedit` ou `emacs`, disponibles dans un environnement unix, ou comme `notepad++` sous windows) permettent une édition plus aisée.

Il y a deux types de fichiers `*.m` : les scripts et les fonctions. Si un fichier `*.m` ne contient qu'une séquence d'instructions on parle d'un *script*. Si le script `nom1.m` se trouve dans le répertoire de travail d'Octave, le fait d'exécuter l'instruction `nom1` est équivalent à effectuer un copier-coller du contenu de ce fichier dans le terminal d'Octave.

Des fichiers `*.m` peuvent également contenir des *fonctions*. (voir Section 4 pour plus de détails). Dans ce cas le nom du fichier doit idéalement coïncider avec le nom de la fonction pour qu'Octave sache dans quel fichier il doit chercher sa définition.

Dans les deux cas, l'instruction `help` suivie du nom du fichier (sans l'extension `.m`) permet d'afficher le premier commentaire au sein du fichier. Par exemple, `help nom1` affiche le premier commentaire du fichier `nom1.m` pour autant que ce fichier se trouve dans le répertoire de travail.

4 Fonctions

Les fonctions peuvent être définies de deux manières. Celles qui ont une expression simple et retournent une sortie peuvent être définies via

`nomf1 = @(x1,...,xn)` *expression*

où `x1` , ..., `xn` (qui peuvent être des vecteurs ou des matrices) sont les entrées (arguments, variables) de la fonction et l'*expression* définit la sortie. Notons que la fonction `nomf1` peut être passée comme entrée à une autre fonction.

Alternativement, on peut définir une fonction avec la syntaxe (le `end` n'est pas nécessaire pour indiquer la fin du code de la fonction)

```
function [y1,...,ym] = nomf2 (x1,...,xn)
  code de fonction nomf2
```

et de sauver cette fonction dans un fichier `nomf2.m` . Cette dernière option permet plus de flexibilité. Pour passer cette fonction comme entrée à une autre fonction on utilisera la syntaxe `@nomf2` . Par exemple, la fonction suivante retourne le carré de la valeur de `f` au point `x` (implicitement, `f` doit être une fonction à une entrée et à une sortie) :

```
function f2 = squaref (f,x)
  f2 = f(x)^2;
```

Notez que si la fonction retourne une sortie `f2` , une variable du même nom doit exister au sein de la fonction et recevoir la valeur à retourner. Dans le répertoire où `squaref.m` est défini le code suivant retournera `0.5`

```
f = @(x)(sin(2.*x));
squaref (f,pi/8)
```

Alternativement, on pourrait créer le fichier `maf.m` avec comme contenu

```
function y = maf (x)
  y = sin(2*x);
```

et utiliser alors `squaref (@maf,pi/8)` .

La fonction définie via un fichier `*.m` s'arrête après l'exécution de la dernière ligne. On peut aussi arrêter une fonction en utilisant l'instruction `return` (dans ce cas Octave continue l'exécution de la fonction appelante) ou `error (`*message*`)` (dans ce cas Octave affiche le *message* d'erreur et arrête l'exécution de toutes les fonctions appelantes).

Les variables définies dans le code de la fonction ne sont pas accessibles dans l'environnement d'Octave. Il s'agit des variables locales à la fonction et leur contenu est perdu à la fin de l'exécution. La communication entre l'environnement d'Octave et une fonction

```
function [y1,...,ym] = nomf2 (x1,...,xn)
```

s'effectue via les arguments d'entrée `x1,...,xn` et ceux de sortie `y1,...,ym` . De même, les variables de l'environnement d'Octave ne sont pas accessibles dans le code de la fonction.

Petit Guide d'Octave

Ce guide contient des opérations accessibles en Octave et qui ne sont pas documentées via la commande `help` . Pour les commandes accessibles via `help` les informations mises à jour peuvent être obtenues (entre autres) sur la page

http://octave.sourceforge.net/functions_by_alpha.php .

Un guide (sensiblement) plus complet est aussi disponible via

http://metronu.ulb.ac.be/MATH-H-202/refcard-a4.pdf .

`a:b`	si $b \geq a$, résulte en le vecteur ligne `[a a+1` $\cdots$ `a+n],` où n est le plus grand entier tel que $a+n \leq b$;
`a:d:b`	si $b \geq a$ et $d > 0$, crée un vecteur ligne `[a a+d` $\cdots$ `a+nd],` où n est le plus grand entier tel que $a+dn \leq b$; idem si $b \leq a$ et $d < 0$, avec n le plus grand entier satisfaisant $a+dn \geq b$; renvoi un résultat vide dans les autres cas.
`A+B` , `A-B` , `A*B`	si A et B sont deux matrices (vecteurs et scalaires étant des cas particuliers), effectue l'opération `+` , `-` ou `*` matricielle correspondante (bien entendu, si les dimensions sont concordantes) ;
`A\B`	si A une matrice carrée régulière, l'instruction donne la solution X du système $AX = B$, pour autant que les dimensions correspondent ; notez que B peut être un vecteur colonne ou une matrice (dans ce dernier cas $X = A^{-1}B$ est aussi une matrice). Si A est une matrice surdéterminée, $AX = B$ est résolu au sens des moindres carrés ;
`A.*B` , `A./B` , `A.\B`	si $A = (a_{ij})$ et $B = (b_{ij})$ sont deux matrices de mêmes dimensions $m \times n$, le résultat est une matrice $C = (c_{ij})$ de dimensions $m \times n$ telle que $c_{ij} = a_{ij} * b_{ij}$, $c_{ij} = a_{ij}/b_{ij}$ ou $c_{ij} = a_{ij} \backslash b_{ij} = b_{ij}/a_{ij}$; l'opération est donc effectuée élément par élément ;
`A^p` , `A.^p`	la première instruction produit l'exposant p de la matrice $A = (a_{ij})$, c'est-à-dire A^p, alors que la seconde donne la matrice avec les éléments $(a_{ij})^p$;
`A'`	transposition de A (si A est une matrice complexe, transposition et conjugaison complexe) ;
`@(x1,...,xn) expr`	renvoie une fonction qui dépend des variables $x_1, \ldots, x_n$ (dont certaines peuvent être vectorielles ou matricielles) définie par l'expression `expr` (pouvant également être un vecteur ou une matrice) ;
`i` , `e`	voir `help i` , `help e` ; attention, si une variable portant le même nom est créée, la valeur par défaut est automatiquement réécrite avec la nouvelle variable ; essayez `disp(e); e=1; disp(e)` .