Kit de survie sur MATLAB/Octave

1 Interpréteur, éditeur et commandes

1.1 Syntaxe de fin d'instruction

Une instruction MATLAB doit se terminer par l'un de ces trois éléments.

- un saut de ligne (dans l'éditeur pour des scripts ou des fonctions) ou l'appui de la touche entrée dans l'interpréteur (donc attention, en MATLAB, les sauts de ligne font partie de la syntaxe!!) ;
- un point-virgule (`;`) et dans ce cas le résultat n'est pas affiché lors de l'exécution.
- une virgule (`,`) équivalente à un saut de ligne dans l'éditeur et qui permet dans l'interpréteur d'enchaîner plusieurs commandes avant de les exécuter à la suite.

Exemple 1. Ces instructions sont tapées dans l'interpréteur puis suivies de la touche entrée.

`» exp(2)`	`» exp(2);`	`» exp(2), exp(3)`	`» exp(2); exp(3)`
`ans = 7.3891`	(pas d'affichage de sortie [1])	`ans = 7.3891` `ans = 20.086`	`ans = 20.086`

On peut modifier le format d'affichage dans l'interpréteur avec la commande `format` suivie d'arguments. Par exemple pour éviter les nombreux sauts de ligne en sortie, on peut entrer la commande `format compact` et pour demander d'afficher plus de chiffres dans les résultats, `format long`.

Pour connaître toutes les possibilités, il suffit d'entrer `help format`.

La commande `help` est la plus importante de toutes! Elle donne un mode d'emploi pour toutes les fonctions prédéfinies (et mêmes pour celles que vous écrivez si vous commentez votre code).

1.2 Scripts

Dans le dossier courant, créer un fichier texte avec extension `.m`, par exemple `mon_script.m`, ou taper `edit mon_script` dans l'interpréteur.

Exemple 2. On entre et sauvegarde le code suivant dans l'éditeur intégré (ou dans votre éditeur préféré) sous le nom mon_script.m

```
exp(1), log(3)
2+3; 2^3
```

En tapant `» mon_script` dans l'interpréteur, toutes les instructions (il y en a 4) écrites dans le fichier `mon_script.m` seront exécutées dans l'ordre dans lequel elles sont écrites. Pour ce script, MATLAB fera 4 calculs et il y aura 3 affichages.

Les commentaires dans un fichier `.m` sont précédés d'un symbole `%`.

```
% Ceci est un commentaire.
>> exp(1)
% Ceci est un autre commentaire.
>> exp(8) % Encore un commentaire.
```

1. mais la valeur est bien calculée et stockée dans une variable nommée `ans`.

2 Array et classes

Les types de base en MATLAB sont appelés *classes*. Pour simplifier notre propos nous nous limitons aux types suivants :

`double` Nombre à virgule flottante à double précision. *C'est la classe numérique par défaut dans* MATLAB. Si vous tapez le nombre `5`, MATLAB le considérera par défaut comme un `double` (et pas comme un entier). Vu l'objectif de MATLAB, qui est le calcul numérique, c'est un choix très compréhensible [2].

`char` Caractère : lettre minuscule ou majuscule, chiffre, espace, ... Par exemple `'a'`, `'b'`, `' '`, `'1'`, ...

`logical` Booléen : `0` ou `false` et `1` ou `true`.

En MATLAB, la classe seule ne suffit en général pas à décrire un objet. Par défaut, tous les objets sont des *array*. Un array est un tableau *dynamique* (sa taille peut varier) ayant *au moins deux dimensions*. Par soucis de simplicité, nous nous limitons aux array de dimension 2 que nous appellons souvent matrices.

Les restrictions suivantes s'appliquent aux array :

- Tous les éléments ont la même *classe*.
- Toutes les lignes ont le même nombre d'éléments.

Pour entrer une matrice, il suffit d'appliquer les règles suivantes :

- Pour écrire une matrice ligne, c'est-à-dire une matrice de taille $(1, n)$, on ouvre un crochet (`[`), puis les éléments de chaque ligne sont séparés par une virgule (`,`) ou une espace et on referme le crochet (`]`).
- Pour écrire une « vraie » matrice (de taille (m, n) avec $m > 1$), on ouvre un crochet (`[`), puis on entre chaque ligne en la séparant de la suivante par un point-virgule (`;`) et enfin on referme le crochet (`]`).

Parmi les array, MATLAB fait la distinction entre les sous-catégories suivantes :

`matrix :` array à deux dimensions (on se limitera à ce cas) ;

`vector :` matrice dont l'une des dimensions vaut 1 (ligne ou colonne) ;

`scalar :` matrice de dimension 1x1 (ce sont donc aussi des vecteurs et des matrices) ;

`empty :` array dont l'une des dimensions est nulle.

Tout ce qui a été dit précédemment (et bien plus encore) est illustré dans le tableau suivant que vous devez étudier consciencieusement.

Entrée	`[2]`	`2`	`[10, -2, 3.5]`	`['a' 'b' 'c']`	`[3 ; -1 ; 10]`
Interprétation	2	2	$\begin{bmatrix}10 & -2 & 3.5\end{bmatrix}$	« abc »	$\begin{bmatrix}3\\-1\\10\end{bmatrix}$
Classe	`double`	`double`	`double`	`char`	`double`
Dimensions	1×1	1×1	1×3	1×3	3×1
Sous-catégorie	`scalar`	`scalar`	`vector`	`vector`	`vector`

2. Les types entiers existent bien sûr, nous les verrons si nous en avons besoin.

`[[1, 2];[30, 40];[.5, .6]]`	`[[1, 2, 3];[4,5]]`	`ones(2,3)`	`'salut'`	`ones(2,0)`
$\begin{bmatrix} 1 & 2 \\ 30 & 40 \\ 0,5 & 0,6 \end{bmatrix}$	`erreur`	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	« salut »	[ ]
`double`	s. o.	`double`	`char`	`double`
3×2	s. o.	2×3	1×5	2×0
`matrix`	s. o.	`matrix`	`vector`	`empty`

`eye(2)`	`['salut';'bob']`	`(1 > 0)`	`(2==3)`	`[-1, 1] > 0`	`[-1 ; 3] <= [0 ; 1]`
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	`erreur`	`true` ou `1`	`false` ou `0`	$\begin{bmatrix} 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
`double`	s. o.	`logical`	`logical`	`logical`	`logical`
2×2	s. o.	1×1	1×1	1×2	2×1
`matrix`	s. o.	`scalar`	`scalar`	`vector`	`vector`

3 Variables, fonctions prédéfinies et opérateurs

3.1 Variables

Il n'y a pas de déclaration des variables en MATLAB. Pour affecter des objets à des variables, il suffit d'écrire, par exemple l'instruction `a = 3` . Ceci a pour effet de :

1. Créer la variable `a` si elle n'existe pas déjà.
2. Mettre la valeur `3` dans `a`.
3. Maintenant `a` contient un array de type `double` et de dimensions 1×1 (c'est un scalaire).

Si l'on tape maintenant l'instruction `b = a`, MATLAB

1. crée la variable `b` si elle n'existait pas déjà,
2. lit le contenu de `a`, ici un scalaire de type `double` égal à `3`,
3. affecte à `b` la valeur lue, c'est-à-dire `3`.

La variable que l'on affecte doit toujours être *à gauche du signe* `=` tandis que l'expression qui est à droite est évaluée et est la valeur qui sera ensuite affectée à cette variable. Continuons notre exemple.

L'instruction `a='salut !'` :

1. ne provoque aucune erreur ! Les variables dans MATLAB ne sont pas typées, on peut tout à fait affecter une chaîne de caractère à `a` après lui avoir affecté un `double`.
2. Après cette instruction, `a` contient un array de classe `char` et de dimensions 1×7 (c'est donc un vecteur).
3. Le contenu de `b` n'a lui pas changé ! Il a toujours la même classe, la même valeur (`3`) et les mêmes dimensions.
4. Pour afficher sa valeur, il suffit d'entrer `b` dans l'interpréteur.

Et... c'est à peu près tout ce qu'il y a à savoir sur les variables en MATLAB ! On ne parlera pas de pointeur, de référence, de gestion de la mémoire, etc. MATLAB est un langage fait pour être très simple et rapide à utiliser.

3.2 Opérateurs

Soient A et B deux matrices de classe `double`, de dimensions respectives $m \times n$ et $p \times q$. Soit t un scalaire et n un entier. Nous rappelons les opérateurs principaux dans le tableau suivant (certaines généralisations évidentes sont omises).

Définition	Expression	Restrictions	Évaluation
Somme élément par élément	`C = A+B`	Mêmes dimensions : $m = p$ et $n = q$	C a les mêmes dimensions que A et B et $C(i,j) = A(i,j) + B(i,j)$.
Somme d'une matrice et d'un scalaire	`C = A+t`	s. o.	C a la même dimension que A et $C(i,j) = A(i,j) + t$.
Différence élément par élément	`C = A-B`	Mêmes dimensions : $m = p$ et $n = q$	C a les mêmes dimensions que A et B et $C(i,j) = A(i,j) - B(i,j)$.
Produit de matrices	`C = A*B`	$n = p$	C est de dimensions $m \times q$. $C(i,j) = \sum_{k=1}^{n} A(i,k) \times B(k,j)$.
Produit d'une matrice par un scalaire	`C = t*A`	s. o.	C a les mêmes dimensions que A et $C(i,j) = t \times A(i,j)$.
Puissance de matrice	`C = A^n`	A est carrée : $m = n$	$C = \underbrace{A \times A \times ... \times A}_{n \text{ fois}}$
Produit élément par élément	`C = A.*B`	Mêmes dimensions : $m = p$ et $n = q$	C a les mêmes dimensions que A et B et $C(i,j) = A(i,j) \times B(i,j)$.
Puissance élément par élément	`C = A.^n`	s. o.	C a les mêmes dimensions que A et $C(i,j) = A(i,j)^n$.
Division élément par élément	`C = A./B`	Mêmes dimensions : $m = p$ et $n = q$	C a les mêmes dimensions que A et B et $C(i,j) = \frac{A(i,j)}{B(i,j)}$.
Transposition	`C = A'`	s. o.	C est de dimensions $n \times m$ et $C(i,j) = A(j,i)$.
Test d'égalité élément par élément	`C = (A==B)`	Mêmes dimensions : $m = p$ et $n = q$	C est de classe `logical`, a les mêmes dimensions que A et B et $C(i,j)$ vaut `true` ssi $A(i,j)$ et $B(i,j)$ sont égaux.
Test de non-égalité élément par élément	`C = (A~=B)`	Mêmes dimensions : $m = p$ et $n = q$	C est de classe `logical`, a les mêmes dimensions que A et B et $C(i,j)$ vaut `true` ssi $A(i,j)$ et $B(i,j)$ sont différents.
Comparaison élément par élément	`C = (A<B)`	Mêmes dimensions : $m = p$ et $n = q$	C est de classe `logical`, a les mêmes dimensions que A et B et $C(i,j)$ vaut `true` ssi $A(i,j)$ est strictement inférieur à $B(i,j)$.

Si A et B sont des arrays de classe `logical` et de mêmes dimensions, on a les opérateurs suivants :

Définition	Expression	Évaluation
Négation élément par élément	`C = ~A`	$C(i,j)$ vaut `true` ssi $A(i,j)$ vaut `false`
« Ou » logique élément par élément	`C = (A || B)`	$C(i,j)$ vaut `true` ssi *au moins* l'une des deux valeurs $A\,(i,j)$ ou $B\,(i,j)$ vaut `true`
« Et » logique élément par élément	`C = (A && B)`	$C(i,j)$ vaut `true` ssi *les deux* valeurs $A\,(i,j)$ et $B\,(i,j)$ valent `true`

Pour finir cette longue liste, signalons que les raccourcis d'incrémentation, et de modification-affectation que l'on trouve dans d'autres langages (comme C ou C++) existent aussi en MAT-

LAB. Par exemple `++A` est équivalent à `A = A+1` ; `A.^= 2` est équivalent à `A = A.^2` et `A *= B` est équivalent à `A = A*B`[1].

3.3 Fonctions prédéfinies

L'appel d'une fonction MATLAB se fait très simplement. Plutôt qu'un long discours, donnons quelques exemples importants.

- `A=zeros(3)` affecte à `A` une matrice carrée de dimensions 3×3, de classe `double` et dont tous les éléments ont la valeur 0.
- `zeros(4,15)` renvoie une matrice de dimensions 4×15 remplie de 0.
- `ones(10,1000)` renvoie une matrice de dimensions 10×1000 remplie de 1.
- `eye(2)` renvoie la matrice identité $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
- `size(A)` renvoie une matrice 1×2 contenant le nombre de lignes puis le nombre de colonnes de `A`.
- `numel(A)` renvoie un scalaire égal au nombre total d'éléments de `A`.
- `exp(A)` renvoie une matrice de `double` de mêmes dimensions que `A` et dont les éléments sont les $\exp(A(i,j))$.
- Ceci s'applique également aux fonctions usuelles `cos`, `sin`, `tan`, `atan` (pour la fonction arctangente), `log` (pour le logarithme népérien), etc.

Il y a des fonctions qui retournent un ou plusieurs arguments (parfois de classes et de dimensions différentes), comme par exemple, `min` ou `eig`.

Exemple 3. Si `L` est un vecteur, `min(L)` renvoie comme premier argument l'élément minimal de `L` et comme second argument son indice dans `L`, mais si l'on ne précise rien, seul le premier argument est renvoyé :

```
>> min([-1, 2, -10, 3])
ans = -10
>> [a, b] = min([-1, 2, 10, 3])
a = -10
b = 3
```

Pour demander plusieurs valeurs de retour à une fonction `f` il faut avoir recours à la syntaxe `[val1, val2, val3, ..., valr] = f( arg1, arg2, ..., argr )`

Notons également que le nombre d'arguments passés à la fonction peut lui aussi varier. Par exemple, les fonctions `zeros` et `ones` ont un comportement différent selon qu'on les appelle avec 1 ou 2 arguments.

4 Matrice : création et accès

4.1 L'opérateur deux-points (:)

Exemple 4. `>> [1:5]`
`ans = 1 2 3 4 5`
`>> [1:3:11.5]`

1. bien sûr, cette opération doit être valide.

```
ans = 1 4 7 10
>> [10:-1:6]
ans = 10 9 8 7 6
```

La syntaxe `a:b` et `a:h:b` permet de gagner énormément de temps.
`a:h:b` renvoie la matrice $\begin{bmatrix} a & a+h & a+2h & \dots & a+n\times h \end{bmatrix}$ où

$$n = \max\{n \geq 0 \mid a + n \times h \leq b\}$$

si $h > 0$ et $n = \max\{n \geq 0 \mid a + n \times h \geq b\}$ si $h < 0$.

4.2 Accès par indices

Attention : En Matlab, les indices dans les arrays commencent à 1. Dans d'autres langages, ils commencent à 0.

Soit la matrice $A = \begin{bmatrix} 1 & -1 & 3 & 5 \\ 2 & -3 & 4 & 10 \\ 3 & -5 & 0 & 30 \end{bmatrix}$

- Pour accéder à un élément, il suffit d'entrer `A(num_ligne, num_colonne)`. Par exemple ici `A(2,3)` a pour valeur 4. La lecture de `A(4,1)` provoque une erreur [2].
- Plus généralement, pour accéder à une sous-matrice, on indique les numéros de ligne et de colonne que l'on souhaite conserver. Par exemple, l'entrée `A([1 3],[2 4])` renvoie la sous-matrice $\begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix}$.
- Ainsi pour obtenir la 3ème colonne, il suffit d'entrer `A([1 2 3], 3)` ce qui donne $\begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}$, mais on aurait aussi pu taper `A(1:3, 3)`, ou mieux encore `A(1:end, 3)` ou (encore encore mieux) `A(:,3)`.
  Le mot réservé `end` est le dernier indice de l'array dans la dimension où il est employé (ici les lignes) et le `:` seul est un raccourci pour `1:end`
- Pour obtenir la deuxième ligne, on peut utiliser l'une des trois expressions suivantes : `A(2, [1 2 3 4])` ou `A(2, 1:4)` ou encore `A(2, 1:end)` ou encore mieux `A(2,:)`. L'évaluation de toutes ces expressions est $\begin{bmatrix} 2 & -3 & 4 & 10 \end{bmatrix}$.

4.3 Affectation par indices

Pour modifier un ou plusieurs éléments d'une matrice, la syntaxe ne change pas (mais la matrice est cette fois à gauche du signe `=`).

- `A(2,3)=-2.45` change l'élément de la deuxième ligne et de la troisième colonne en -2.45.
- `A(1,:)=[.1 .2 .3 .4]` change toute la première ligne de la matrice A.
- `A([1 2],[2 4]) = [20 30; 40 50]` change tous les éléments de la sous-matrice.
- Dans ces affectations, les dimensions et les classes doivent être les mêmes à gauche et à droite du signe `=`.
- Seule exception à cette règle, lorsque le terme de droite est un scalaire, par exemple
- `A([1 2],[2 4]) = 3` change tous les éléments de la sous-matrice en les transformant en 3.

2. elle n'en provoque pas en affectation par contre, voir plus loin.

Si on affecte une ou plusieurs valeurs en dehors des dimensions de la matrice, celle-ci sera « agrandie » pour pouvoir accueillir ses nouveaux locataires. Si besoin, les éléments non précisés seront remplacés par des 0 (ou par des caractères vides s'il s'agit d'une matrice de `char`).

Soit une matrice $B = \begin{bmatrix} 3 & 2 \\ 4 & -1 \end{bmatrix}$.

- L'instruction `B(1,4) = 30` affecte à la matrice B la valeur $B = \begin{bmatrix} 3 & 2 & 0 & 30 \\ 4 & -1 & 0 & 0 \end{bmatrix}$.
- Ensuite l'instruction `B(end+1,:) = 1:4` permet d'obtenir la matrice $B = \begin{bmatrix} 3 & 2 & 0 & 30 \\ 4 & -1 & 0 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix}$.
- Enfin l'instruction `B = [[0 ; 1 ; 0], B, [ 0 1 ; 1 1 ; -1 1]]` permet d'obtenir

$$\begin{bmatrix} 0 & 3 & 2 & 0 & 30 & 0 & 1 \\ 1 & 4 & -10 & 0 & 0 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & -1 & 1 \end{bmatrix}.$$

Cette dernière façon de créer des matrices dite *par concaténation* (ici horizontale) est assez délicate, il faut faire très attention aux dimensions.

4.4 Accès par ordre selon les colonnes

Lorsque l'utilisateur entre un seul indice au lieu de deux, MATLAB convertit automatiquement la matrice en colonne, l'ordre se faisant *selon les colonnes*.

Exemple 5. Reprenons la matrice $A = \begin{bmatrix} 1 & -1 & 3 & 5 \\ 2 & -3 & 4 & 10 \\ 3 & -5 & 0 & 30 \end{bmatrix}$. Pour MATLAB, on a (première colonne) $A(1) = 1$, $A(2) = 2$, $A(3) = 3$ puis (deuxième colonne) $A(4) = -1$, $A(5) = -3$, $A(6) = -5$, et (troisième colonne) $A(7) = 3$, ... jusqu'à $A(12) = 30$.

Comme précédemment, on peut extraire une partie de cette colonne, par exemple $A(3:6) = \begin{bmatrix} 3 \\ -1 \\ -3 \\ -5 \end{bmatrix}$. Si l'on veut avoir toute la matrice A sous forme de colonne, il suffit d'entrer l'instruction $B = A(:)$ et alors B est un vecteur colonne (de dimensions 12×1).

Tout ceci permet en fait de créer des matrices très rapidement et *sans utiliser de boucle*. Par exemple, la suite d'instructions `A=zeros(10), A(1:11:100)=1` permet d'obtenir la matrice identité de dimensions 10×10 (mais pour cela on préfère bien sûr `A=eye(10)`). Plus intéressant : `A=zeros(10), A(11:11:end)=1` permet d'obtenir une matrice ayant des zéros partout sauf juste au-dessus de la diagonale où il y a des 1. Si vous entrez ensuite l'instruction `A(2:11:end)=5`, vous rajoutez des 5 en-dessous de la diagonale.

4.5 Accès par indices logiques

5 Structures de contrôle

5.1 If else if else

```
if CONDITION_1
  ...
```

```
else if CONDITION_2
  ...
else if CONDITION_3
  ...
...
else
  ...
end
```

5.2 Boucle for

La boucle `for` parcourt les éléments d'un vecteur ligne

```
for k = VECTEUR                          %%(le plus souvent k = 1:N)
  ...
  ...
end
```

5.3 Boucle while

```
while CONDITION
  ...
  ...
end
```

6 Fonctions

6.1 Fonctions séparées dans un fichier .m

La fonction doit avoir le même nom que le fichier.

```
function [arg_out1, ..., arg_outR] = ma_fonction(arg_in1, ..., arg_inN)
  ...
  ...
  ...
end
```

6.2 Function Handle

```
>> f = @(x) x^2 + cos(x);
```

6.3 Appliquer une fonction aux éléments d'une matrice

`B = arrayfun(f,A)`.

7 Tracer des courbes et des surfaces (`plot`, ...)

8 Simulation (`rand`, `randn`, ...)