Cours d'introduction à Matlab

S. Correia*
Laboratoire de Physique Théorique
3 rue de l'Université
F-67084 Strasbourg Cedex

29 janvier 2001

Table des matières

*correia@lpt1.u-strasbg.fr

Le CURRI propose des cours d'introduction sur MATLAB à partir du 9 mars 2001. Ce cours s'inspire du cours de Mme Picot du CURRI, Strasbourg.

1 Introduction

MATLAB signifie **Mat**rix **Lab**oratory : il est basé sur un langage matriciel. MATLAB est

- un environnement constitué d'un noyau de calcul et de visualisation graphique et de nombreux modules applicatifs,
- un logiciel de calcul numérique, non de calcul formel comme MAPLE,
- utile pour le traitement du signal, l'analyse de données, les maths appliquées ...

Avantages de MATLAB :

- intuitif, convivial,
- interface graphique maniable : accès à chaque élément d'un graphique à l'aide d'un éditeur graphique,
- contient des centaines de fonctions dont le code source est accessible,
- débogueur de programmes,
- outil pour l'optimisation : donne le temps passé dans chaque sous-programme, le nombre d'appel des fonctions, ...
- peut intégrer d'anciens programmes en C ou Fortran,
- peut être appelé depuis le C ou le Fortran.

Inconvénient : MATLAB est un logiciel payant.

Autres logiciels (*libres*) du même type :

- OCTAVE : `http://www.che.wisc.edu/octave/octave.html`,
- SCILAB : `http://www-rocq.inria.fr/scilab/`.

2 Présentation générale

MATLAB contient plusieurs modules :
- `matlab` : module de base,
- `images` : traitement d'images,
- `local` : fichiers de configuration,
- `map` : analyse et visualisation de données graphiques,
- `optim` : optimisation,
- `sb2sl` : system built to simulink translator,
- `signal` : traitement du signal,
- `simulink` : simulation non linéaire,
- `splines` : fonctions splines,
- `stateflow` : modélisation et simulation de systèmes réactifs complexes,
- `wavelet` : ondelettes,
- ...

Composition du module de base Matlab

Élément de l'environnement de travail:
- `general` : commandes d'ordre général.

Éléments du langage matriciel :
- `ops` : opérateurs et caractères spéciaux,
- `lang` : construction du langage de programmation,
- `strfun` : chaînes de caractères,
- `iofun` : fonctions d'entrées-sorties,
- `timefun` : heures et dates,
- `datatypes` : types de données et structures.

Éléments pour les graphiques :
- `graph2d` : graphiques bidimensionnels,
- `graph3d` : graphiques tridimensionnels,
- `specgraph` : graphiques spécialisés,
- `uitools` : interfaces graphiques utilisateurs.

Bibliothèque de fonctions mathématiques :
- `elmat` : matrices élémentaires et manipulation de matrices,

- `elfun` : fonctions mathématiques élémentaires,
- `specfun` : fonctions mathématiques spécialisées,
- `matfun` : fonctions matricielles, algèbre numérique linéaire,
- `datafun` : analyse de données et transformées de Fourier,
- `polyfun` : interpolation et polynômes,
- `funfun` : fonctions de fonctions et résolution d'ODE,
- `sparfun` : matrices creuses.

Interface de programmes d'applications : bibliothèque permettant les échanges entre MATLAB et les programmes C ou Fortran.

3 Utilisation basique de Matlab

3.1 Démarrage et arrêt de Matlab

Dans une console de terminal (Unix ou Linux), taper la commande `matlab`.

L'environnement MATLAB est indiqué par la présence du symbole `>>` en début de ligne.

Pour sortir de MATLAB, taper

```
>> quit
```

3.2 Obtenir de l'aide

Il existe plusieurs façons d'obtenir de l'aide :

- `>> help` suivi du nom de la commande sur laquelle on veut de l'information (sans argument, `help` donne la liste des répertoires de MATLAB),
- `>> lookfor` cherche des informations à partir d'un mot-clé,
- `>> helpwin` ouvre une fenêtre séparée dans laquelle sont listées les commandes,
- `>> helpdesk` ouvre un browser (ex. netscape) où l'on peut rechercher des informations (guide, index, moteur de recherche ...)
- site internet : `www.mathworks.com`.

3.3 Manipulation en ligne

Les commandes sont tapées directement dans l'environnement MATLAB.

3.3.1 Création de matrices

Pas de déclaration de type ni de dimension. Taper directement *entrée* à la fin de la ligne de commande (pas de , ; ou autre symbole)

```
>> A=[1 2 3; 4 5 6]
```

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

Nom de variables :
- 31 caractères max, dont 19 significatifs (lettres, chiffres ou _),
- doit commencer par une lettre,
- les majuscules et les minuscules sont différenciées.

Pour créer une matrice,
- ouvrir par [et fermer par],
- séparer les éléments d'une ligne par un espace ou une virgule,
- séparer les lignes par *entrée* ou un ;

Vecteur ligne :

```
>> u=[1 2 9]
```

$$u = \begin{matrix} 1 & 2 & 9 \end{matrix}$$

Vecteur colonne :

```
>> v=[1;2;3;4]
```

$$v =$$
$$\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Variable scalaire (pas besoin de crochets):

```
>> x=6
```

$$x =$$
$$6$$

Matrice de nombres aléatoires :

```
>> w=rand(2,3)
```

$$w =$$
$$\begin{matrix} 0.4467 & 0.6551 & 0.1553 \\ 0.2354 & 0.2159 & 0.7895 \end{matrix}$$

3.3.2 Concaténation de matrices

horizontale

```
>> D=[A w]
```

$$D =$$
$$\begin{matrix} 1 & 2 & 3 & 0.4467 & 0.6551 & 0.1553 \\ 4 & 5 & 6 & 0.2354 & 0.2159 & 0.7895 \end{matrix}$$

verticale

```
>> E=[A;u]
```

$$E =$$
$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 9 \end{matrix}$$

3.3.3 Données sur les matrices

Affichage d'une matrice :

```
>> E
```

$$E =$$
$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 9 \end{matrix}$$

Taille d'une matrice:

```
>> size(D)
```

$$ans =$$
$$\begin{matrix} 2 & 6 \end{matrix}$$

ans est une variable qui peut être utilisée dans la ligne suivante. On peut aussi définir une nouvelle variable qui contient la taille de la matrice D.

```
>> T=size(D)
```

Élément de matrice :

```
>> E(1,2)
```

$$ans =$$
$$2$$

```
>> E(8)
```

$$ans =$$
$$6$$

affiche le 8^{e} élément de la matrice E.

3.3.4 Opérateur deux points (:)

Opérateur très utilisé dans MATLAB. Il permet de créer simplement des vecteurs ou matrices.

Syntaxe : `x=val_début:pas:val_fin`

```
>> x=1:.5:3
```

$$x = 1 \quad 1.5 \quad 2 \quad 2.5 \quad 3$$

Par défaut, le pas vaut 1.

```
>> x=1:4
```

$$x = 1 \quad 2 \quad 3 \quad 4$$

L'opérateur : sert aussi à extraire une partie de matrice.

```
>> D(1:2;2:4)
```

$$ans = \begin{matrix} 2 & 3 & 0.4467 \\ 5 & 6 & 0.2354 \end{matrix}$$

extrait les lignes de 1 à 2 et les colonnes de 2 à 4.

```
>> D(:;1:2)
```

$$D = \begin{matrix} 1 & 2 \\ 4 & 5 \end{matrix}$$

extrait *toutes* les lignes et les colonnes de 1 à 2.

Il est possible de supprimer des lignes ou des colonnes en leur affectant une matrice vide.

```
>> D(:,4:6)=[]
```

$$D = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

3.3.5 Opérations sur les matrices

transposée :

```
>> D'
```
donne la transposée conjuguée de D.

addition :

```
>> D+D'
```
ajoute élément par élément.

```
>> D+10
```
ajoute 10 à chaque élément.

Multiplication matricielle :

```
>> E*w
```
donne le produit matriciel de E par w (les dimensions des matrices doivent être compatibles sinon MATLAB renvoie une erreur).

Division matricielle :

Division à droite :

```
>> A/B
```
signifie $A \times B^{-1}$.

Division à gauche :

```
>> A\B
```
signifie $A^{-1} \times B$.

Divers

```
>> sum(D)
```
renvoie un vecteur ligne dont les éléments sont la somme de chaque colonne de la matrice D.

```
>> diag(E)
```
renvoie les éléments de la diagonale de E sous la forme d'un vecteur colonne.

3.3.6 Environnement de travail

Format d'affichage

MATLAB travaille toujours en double précision, mais il est possible de choisir le format d'affichage avec la commande `format`.

Pour les flottants, le choix est le suivant.

`format short`	(par défaut) 5 chiffres significatifs 1.0000
`format short e`	scientifique exponentiel 1.0000e+00
`format short g`	court adapté 1.4323 2.3411e+02
`format long`	15 chiffres significatifs
`format long e`	notation scientifique
`format long g`	adapté
`format rat`	approximation par fractions de petits entiers

Commandes utiles

Rappel des commandes précédentes avec les flèches du clavier ↑, ↓.

`>> A=[1`↑ permet de rappeler uniquement les commandes précédentes commençant par A=[1.

`ctrl`→ déplace le curseur d'un mot à droite.

`esc` efface la ligne courante.

`>> more on` permet d'afficher page par page.

`>> more off` l'écran arrête de défiler seulement lorsque le résultat de la commande est terminé.

`>> clear nom_variable` efface la variable `nom_variable`.

`>> who` liste les variables définie durant la session.

`>> whos` liste détaillée.

Les commandes du système d'exploitation sont accessibles à l'aide du caractère `!`.

`>> !nedit prog.m &` ouvre le fichier prog.m avec l'éditeur nedit.

`>> what` liste les fonctions de MATLAB (fichiers .m).

Enregistrement de la session :

```
>> diary journal.out
```
crée le fichier journal.out et y écrit toutes les commandes qui sont frappées jusqu'à la commande

```
>> diary off.
```

4 Manipulation de fonctions et de tableaux

4.1 Nombres

Dans `3e5`, la lettre `e` signifie 3×10^5.

Les nombres complexes sont notés avec `i` ou `j` : `3+2i`. **Attention !** si `i` ou `j` est utilisé comme variable, il ne peut plus représenter le i imaginaire. La commande `clear` permet de réinitialiser toutes les variables.

```
>> clear i
```
réinitialise la variable i.

Tous les nombres sont stockés en interne, sur 64 bits, suivant le format long spécifié par le standard IEEE pour l'arithmétique flottante. Les nombres flottants ont une précision d'environ 16 chiffres significatifs et une grandeur finie de l'ordre de 10^{-308} à 10^{+308}.

4.2 Fonctions spéciales

`pi`	3.14159 ...
`eps`	précision relative flottante 2^{-52}
`realmin`	nombre réel flottant le plus petit 2^{-1022}
`realmax`	nombre réel flottant le plus grand $(2-\epsilon)2^{1023}$
`Inf`	infini
`NaN`	Not a Number (Ex. : 0/0)

4.3 Caractères spéciaux

`%`	commentaire
;	supprime l'affichage du résultat à l'écran
...	permet de continuer une commande sur la ligne suivante

4.4 Tableaux

Les matrices peuvent être manipulées comme de simples tableaux numériques, sans les lois de l'algèbre linéaire.

4.4.1 Opérations arithmétiques

`.*`	multiplication élément par élément
`./`	division à droite élément par élément
`.\`	division à gauche élément par élément
`.^`	puissance élément par élément
`.'`	transposée non conjuguée

4.4.2 Opérateurs relationnels

`>`	supérieur
`<`	inférieur
`<=`	inférieur ou égal
`>=`	supérieur ou égal
`==`	égal
`~=`	différent

Exemple :

`>> A=rand(4,6)` crée une matrice aléatoire 4×6

`>> B=A>.5` le résultat est une matrice de booléens avec des 0 là où les éléments sont inférieurs ou égaux à 0.5 et des 1 ailleurs.

4.4.3 Opérateurs logiques

`&`	ET logique
`|`	OU logique
~	complément logique (NOT)
`xor`	OU exclusif

Exemple : Extraire les éléments de la matrice A compris entre .2 et .7.

```
>> ((A>=.2) & (A<=.7)).*A
```

4.4.4 Fonctions mathématiques

`sin, asin,sinh, asinh, cos,tan, ...`	fonctions trigonométriques
`exp, log, log10, sqrt`	fonctions exponentielles
`abs,angle,conj,imag,real`	fonctions complexes
`fix,floor,ceil,round,rem,sign,mod`	fonctions numériques

Les fonctions mathématiques s'appliquent aux éléments de matrice. `>> sin(G)` donne une matrice contenant le sinus de chaque élément de la matrice `G`.

4.4.5 Fonctions de tests

Pour tester les valeurs des éléments des tableaux.

`all`	=1 si tous les éléments du vecteur $\neq 0$
`any`	=1 si l'un des éléments du vecteur $\neq 0$
`find`	retourne les indices des éléments $\neq 0$
`finite`	=1 pour tout élément de valeur finie
`isinf`	=1 pour tout élément = `Inf`
`isnan`	=1 pour tout élément = `NaN`
`isprime`	=1 pour tout élément = nombre premier

4.4.6 Matrices particulières

`zeros(n,m)`	matrice $n \times m$ remplie de 0
`ones(n,m)`	matrice $n \times m$ remplie de 1
`rand(n,m)`	matrice $n \times m$ de nombres aléatoires (distribution uniforme entre 0 et 1)
`randn(n,m)`	matrice $n \times m$ de nombres aléatoires (distribution normalisée)
`eye(n)`	matrice identité $n \times n$
`magic(n)`	carré magique $n \times n$
`pascal(n)`	matrice de Pascal
`toeplitz(n)`	matrice de Toeplitz
`hankel(n)`	matrice de Hankel
`hadamard(n)`	matrice de Hadamard
`hilb(n)`	matrice de Hilbert

4.4.7 Fonctions de transformations

`triu`	partie triangulaire supérieure
`tril`	partie triangulaire inférieure
`diag`	diagonale
`rot90`	rotation de 90 degrés
`flipud`	symétrie axiale horizontale
`fliplr`	symétrie axiale verticale

Exemple :

```
>> F=[1 2 ; 3 4 ; 5 6];

>> flipud(F)
```

$$ans = \begin{matrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \end{matrix}$$

5 Algèbre linéaire

5.1 Résolution d'équations linéaires

Résolution de AX=B, où X est le vecteur d'inconnues.

```
>> X=A\B
```

Résolution de XA=B :

```
>> X=B/A
```

Il est aussi possible d'utiliser la fonction `inv`,

```
>> X=B*inv(A)
```

mais effectuer la division est plus rapide que calculer l'inverse puis multiplier.

5.2 Fonctions matricielles

`^`	puissance matricielle
`expm`	exponentielle matricielle
`logm`	logarithme matriciel
`sqrtm`	racine carrée matricielle
`funm`	fonctions matricielles générales

Utilisation de `funm` :

```
>> funm(A,'fonction')
```

où `fonction` est une fonction mathématique (`sin`,`cos`, ...).

6 Autres structures de données

Il existe 6 types (6 classes) de données, organisées en tableaux (array) multidimensionnels.

`char`	tableau de caractères
`double`	tableau numérique multidimensionnel
`sparse`	matrice creuse
`uint8`	tableau d'entiers sur 8 bits
`cell`	tableau de cellules
`struct`	tableau de structures

6.1 Tableau de caractères

```
>> A=['salut', ''; 'ca', 'va?']
```

$$A = \begin{matrix} salut \\ ca \qquad va? \end{matrix}$$

`>> char(n,m)` crée un tableau de caractères $n \times m$. Donner une valeur à un élément de ce tableau se fait comme suit.

```
>> char(1,1)='texte'
```

6.2 Tableau multidimensionnel

Un tableau A a autant d'indices que de dimensions. Exemple de tableau à 3 dimensions :

```
>> A=[1 2 ; 3 4 ; 5 6];
>> A(:,:,2)=[0 0 ; 1 7 ; 9 8]
```

$$A(:,:,1) = \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix}$$

$$A(:,:,2) = \begin{matrix} 0 & 0 \\ 1 & 7 \\ 9 & 8 \end{matrix}$$

```
>> size(A)
```

$$ans = \begin{matrix} 3 & 2 & 2 \end{matrix}$$

6.3 Tableau de cellules

Une cellule peut contenir n'importe quel type de données (matrice, tableau, caractère, autre cellule ...). Les tableaux de cellules sont définis par

des accolades :

```
>> C={A sum(A,3) [1 2 3] ; 'phrase' [] .013}
```

$$C = \begin{matrix} [3\times 2\times 2 \quad double] & [3\times 2 \quad double] & [1\times 3 \quad double] \\ 'phrase' & [] & 0.013 \end{matrix}$$

Extraire un élément de la cellule :

```
>> C{1,2}
```

Les tableaux de cellules contiennent des copies des tableaux et non pas des pointeurs sur ces tableaux. Ce qui fait si A est modifié, le tableau de cellules contenant A n'est pas modifié.

L'avantage des tableaux de cellules est de pouvoir stocker des séquences de matrices de tailles différentes, contrairement aux tableaux numériques qui doivent vérifier une concordance entre les dimensions des matrices.

`>> cell(n,m)` crée un tableau de cellules $n \times m$.

6.4 Tableau de structures

Les structures sont des tableaux dont les éléments sont accédés par des noms de champs. Chaque champ peut contenir n'importe quel type de données.

```
>> S.nom='sinus';
>> S.pas=.1;
>> S.result=sin(0:.1:10);
>> S
```

$$S = \begin{matrix} nom : 'sinus' \\ pas : 0.1000 \\ result : [1\times 101 double] \end{matrix}$$

Ceci est une structure de dimension 1×1. On peut lui ajouter des dimensions supplémentaires :

```
>> S(2).nom='cosinus'
```

...

Une construction plus directe est obtenue par la fonction `struct` :

```
>> S=struct('nom','sinus','pas',.1,'result',sin(0:.1:10))
```

6.5 Matrices creuses

Lorsqu'une matrice contient beaucoup de 0, il est intéressant d'utiliser la fonction `sparse` qui stocke seulement les éléments non nuls de la matrice et ses indices. Cela permet de gagner de la place et de réduire les temps de calcul en éliminant les opérations sur les éléments nuls.

```
>> D=[1 0 0 ; 0 0 2]
```

$$D = \begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 2 \end{matrix}$$

```
>> s=sparse(D)
```

$$s = \begin{matrix} (1,1) & 1 \\ (2,3) & 2 \end{matrix}$$

`>> nnz(s)` donne le nombre d'éléments non nuls de `s`.

7 Représentations graphiques en 2 dimensions

7.1 Tracés élémentaires

```
>> t=0:.01:pi;
>> x=sin(t);
>> y=cos(t);
>> figure(1) ouvre une fenêtre pour afficher la figure
>> plot(x) trace x(t) (équivalent à plot(t,x))
>> figure(2) ouvre une nouvelle fenêtre
>> plot(x,y) trace y en fonction de x
```

```
>> title('cercle')
>> xlabel('x')
>> ylabel('y')
```
`>> text(2,cos(2),'\leftarrow cos(2)=-.42')` ajoute le texte entre quotes à l'emplacement x=2, y=cos(2). Utilisation de la syntaxe LaTeX.

La commande `subplot(n,m,l)` permet de tracer plusieurs graphiques dans une seule fenêtre.

`>> subplot(2,1,1)` divise la fenêtre en 2 lignes, 1 colonne et s'apprête à tracer le premier graphique

```
>> plot(x)
```
`>> subplot(2,1,2)` s'apprête à tracer le deuxième graphique

```
>> plot(y)
```

Superposition de graphiques avec la commande `hold on`. Sur une figure, toutes les courbes se superposent jusqu'à la commande `hold off`. Par défaut, lorsque l'on trace 2 courbes à la suite, la 2^{e} efface la 1re.

7.2 Tracés logarithmiques

Les commandes `semilogx, semilogy, loglog` s'utilisent comme la commande `plot` et permettent d'avoir des axes logarithmiques.

La commande `grid` affiche une grille adaptée à la représentation choisie.

7.3 Graphiques en coordonnées polaires

`>> polar(theta,r)` trace r en fonction de theta.

7.4 Affichage

Les fonctions précédentes (`plot,semilog, ...`) acceptent un 3^{e} argument qui permet de choisir le type de tracé et la couleur.

couleurs		traits discontinus		traits continus	
		`+`	signe +		
		`o`	cercle		
`c`	cyan	`*`	astérisque		
`m`	magenta	`.`	point		
`y`	yellow	`x`	croix	`-`	trait plein (défaut)
`r`	red	`square`	carré	`--`	tiret
`g`	green	`diamond`	carreau	`:`	pointillé
`b`	blue	`pentagram`	étoile (5)	`-.`	tiret/pointillé
`w`	white	`hexagram`	étoile (6)	`none`	pas de ligne
`k`	black	`>`	triangle		
		`<`	triangle		
		`v`	triangle		

8 Représentations graphiques en 3 dimensions

8.1 Commandes principales

Tracés graphiques

`plot3`	tracé de lignes et de points
`mesh,surf`	tracé de maillages et de surfaces
`meshc,surfc`	tracé avec projection de contours
`meshz`	tracé de surface avec plan de référence

L'annotation des graphiques se fait comme pour les graphiques à 2 dimensions avec les commandes `title, xlabel, ylabel, zlabel, text`. La commande `colorbar` affiche une table des couleurs.

L'aspect des graphiques peut être modifié avec les commandes

`hidden`	suppression des lignes cachées
`shading`	lissage des couleurs
`colormap`	table des couleurs
`surfl`	surface éclairée
`lighting`	mode d'éclairage

Toutes les propriétés d'un graphique sont modifiables via un éditeur de propriétés associé à chaque figure.

9 Programmation en Matlab

Nous avons vu jusqu'à maintenant comment taper des commandes en ligne. Toutes ces commandes peuvent être tapées dans un fichier puis exécutées en appelant ce fichier. Les fichiers contenant du code en langage MATLAB sont appelés **M-Files** et on leur adjoint le suffixe **.m**.

Il existe 2 types de M-Files : les **scripts** et les **fonctions**.

9.1 Les scripts

Les scripts sont les M-Files les plus simples. Ils sont composés d'une suite de commandes MATLAB comme celles utilisées en mode interactif.

Ils n'acceptent aucun argument et n'en retournent aucun.

Ils opèrent sur les variables de l'espace de travail MATLAB (variables définies au cours d'une session).

L'intérêt d'un script est d'automatiser une série de commandes souvent utilisées.

Les éventuelles nouvelles variables définies dans un script continuent d'exister après l'exécution du script (variables globales).

L'appel d'un script exemple.m se fait en tapant le nom du fichier sans l'extension.

```
>> exemple
```

La commande

```
>> echo on
```

affiche chaque commande du script avant son exécution.

9.2 Les fonctions

Les fonctions acceptent des arguments en entrée et retournent des valeurs en sortie.

9.2.1 Construction d'une fonction

Exemple : Le fichier moy.m contient le code suivant.

```
function y=moy(x)
% MOY : moyenne des éléments d'un vecteur
% MOY(X), où X est un vecteur, est la moyenne des éléments de X

% les arguments en entrée non vectoriels donnent une erreur
[m,n]=size(x);
if (∼ ((m==1)|(n==1)|(m==1 & n==1)))
    error('L entrée doit être un vecteur')
end y=sum(x)/length(x); % calcul de la moyenne
```

Toute fonction doit commencer par le mot-clé `function`. Ensuite vient la liste des variables de sortie (notée [x,y,z] s'il y en a plusieurs). Puis après le signe =, la fonction avec ses arguments d'entrée entre parenthèses.

Remarque : Il est préférable de choisir le même nom pour la fonction et pour le programme. Dans le cas où les 2 noms sont différents, l'appel de la fonction se fait avec le nom du programme.

Le symbole % permet de faire des commentaires. Les commentaires venant aussitôt après la 1re ligne de définition (sans saut de ligne) servent à la description générale de la fonction. En tapant

```
>> help moy
```

les deux lignes de commentaires du fichier moy.m s'affichent. S'il y a un saut de ligne entre la 1re ligne et ces commentaires, la commande `help moy` ne retourne rien.
Les lignes de commentaires suivantes ne s'affichent pas avec la commande `help`.

La commande `lookfor` cherche seulement dans la première ligne de commentaire.

```
>> lookfor moyenne
```

MOY : moyenne des éléments d'un vecteur

9.2.2 Appel d'une fonction

La fonction peut être appelée sur la ligne de commande ou à partir d'une autre fonction. À l'appel de la fonction, MATLAB analyse et interprète le code de la fonction en pseudo-code qu'il stocke en mémoire dans l'espace de travail de MATLAB. La fonction est interprétée durant toute la session. La fonction `clear` permet de libérer de la mémoire.

```
>> clear functions
```
enlève toutes les fonctions de l'espace de travail.

Les M-Files peuvent être interprétés à l'avance grâce à la commande `pcode` qui crée un fichier du même nom avec l'extension .p.

L'avantage de créer un pseudo-code à l'avance est d'éviter une réinterprétation de la fonction à chaque session. C'est un gain de temps.

Un autre avantage est que le pseudo-code n'est pas lisible. On peut donner ainsi un programme sans le fichier source.

9.2.3 Arguments d'une fonction

Certaines fonctions ont un nombre d'arguments d'entrée et de sortie variable. C'est le cas de la fonction `sum`.

```
>> sum(X)
```
calcule la somme des colonnes de X (X étant une matrice).

```
>> sum(X,2)
```
calcule la somme des éléments de X selon la 2^{e} dimension, c.-à-d. la somme des lignes.

Le code de cette fonction utilise pour cela les variables `nargin` et `nargout` qui donnent le nombre d'arguments en entrée et en sortie (voir page `help` en ligne).

La variable `varargin` permet d'avoir un nombre d'arguments en entrée variable.

9.2.4 Variables globales

Par défaut, les variables internes sont locales (n'existent pas en dehors de la fonction). Pour définir une variable globale, il suffit de la déclarer avec la

fonction `global`, puis de lui donner une valeur.

```
>> global(var_glob);
>> var_gob=137;
```

Dans une fonction, il faut déclarer cette variable comme étant globale.

```
function test(a,b,c)
global var_glob
a=2*var_glob
 ...
```

9.2.5 Chemins de recherche des fonctions

Pour que MATLAB trouve les fonctions que l'on a créées, il faut lui indiquer le chemin. Par défaut, MATLAB cherche dans le répertoire courant et dans ses répertoires.

```
>> path
```
donne la liste des chemins de recherche de MATLAB.

```
>> addpath
```
permet d'ajouter un chemin.

9.3 Instructions de contrôle

MATLAB en possède quatre.

9.3.1 if

syntaxe :

```
if    expression logique 1
      instructions
      elseif expression logique 2
        instructions
      else instructions
end
```

9.3.2 switch

syntaxe :

```
switch  expression
        case valeur 1
          instructions
        case valeur 2
          instructions
        ...
        otherwise instructions
end
```

9.3.3 while

syntaxe :

```
while expression
  intructions
end
```

9.3.4 for

syntaxe :

```
for index=debut:pas:fin
  instructions
end
```

10 Optimisation

10.1 vectorisation

Il est important de choisir une vectorisation du code dès que possible, c.-à-d. remplacer les boucles par des opérations vectorielles.

Par exemple, plutôt que

```
>> i=0;
>> for t=0:.001:1
    i=i+1;
    y(i)=sin(t);
```

```
end
```

choisir

```
>> t=0:.001:1;
>> y=sin(t);
```

10.2 gestion du temps

Les commandes `tic` et `toc` mesurent le temps écoulé entre l'appel de ces 2 commandes.

Exemple :

```
>> tic; sum(X); toc
```

évalue le temps pour calculer `sum(X)`.

La commande `profile` est plus puissante et plus complète. Elle indique la durée totale du programme. Elle donne le temps passé dans chaque sous-programme en pourcentage et en secondes. Elle compte le nombre d'appel de chaque sous-programme.

Utilisation :

```
>> profile on
>> sum(X)
>> profile report
```

génère un rapport.

```
>> profile plot
```

génère un tracé du profil de la fonction.

```
>> profile off
```

arrête le profile.

10.3 Préallocation

Lorsque les boucles sont inévitables, il est préférable de préallouer les tableaux dans lesquels seront stockées des données avec la commande

```
>> zeros(n,m)
```

Cette commande crée une matrice $n \times m$ composés de zéros. La mémoire pour cette matrice est réservée à l'avance.

Exemple :

```
>> r=zeros(32,1)
>> for n=1:32
    r(n)=rank(magic(n));
end
```

Sans préallocation, MATLAB augmente la taille du vecteur r à chaque passage dans la boucle.

10.4 Débogueur

Le débogueur utilise les commandes `dbstep, dbclear, dbcont, dbstop, dbtype ...`

`>> edit` ouvre l'éditeur/débogueur.

Pour plus de précision, voir les pages d'aide dans MATLAB.