MÉMO MATLAB

RÉSUMÉ : Matlab ou « *Matrix Laboratory* » est un logiciel de calcul numérique basé sur le calcul matriciel utilisé dans de nombreux domaines d'application. Le but de ce document est de fournir aux étudiants débutants en Matlab, un aide-mémoire qu'ils pourront conserver tout au long de leur cursus. Ce document est rédigé pour l'usage de Matlab R2013a.

1. ACCÉDER À MATLAB

Plusieurs méthodes sont à votre disposition pour ouvrir le logiciel sur Windows et Linux : vous pouvez saisir la commande "matlab" dans un terminal ou chercher l'application dans les menus (généralement dans le menu mathématiques). Une fois le logiciel lancé, la fenêtre de la FIGURE 1 devrait s'afficher devant vous. Vous pouvez y distinguer différentes parties, notamment :

— En en-tête : un ensemble de vignettes et d'outils dans une barre des tâches avec en dessous le chemin du dossier dans lequel vous vous trouvez ;
— En haut à gauche (« *Current folder* ») : le contenu du dossier courant ;
— En bas à gauche (« *Details* ») : des détails sur le contenu du fichier sélectionné, généralement des lignes de commentaires ;
— Au centre : un terminal qui est la zone dans laquelle vous saisirez vos commandes. Le chevron » indique que Matlab attend vos instructions ;
— En haut à droite (« *Workspace* ») : les variables d'environnement avec leurs noms et leur valeur ;
— En bas à droite (« *Command history* ») : l'historique des commandes saisies.

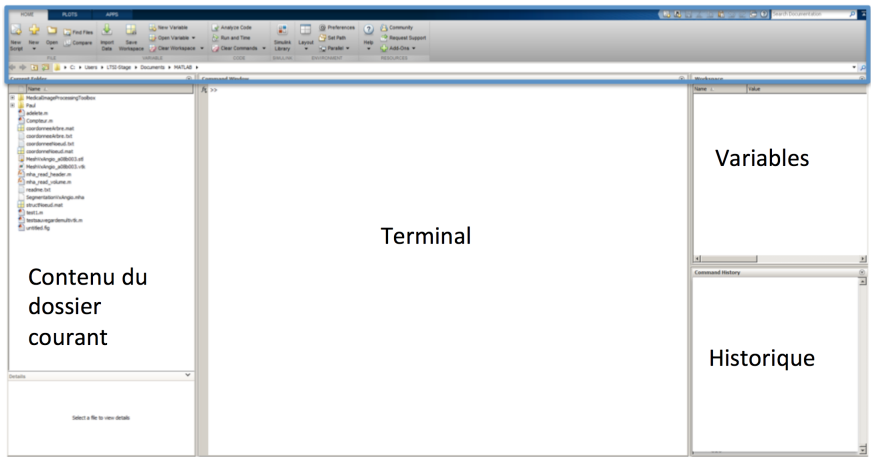

FIGURE 1 – Fenêtre principale de Matlab

Si vous ne parvenez pas à trouver l'un des éléments suivants, vous pouvez les retrouver à partir de l'outil « *Layout* » présent dans l'en-tête. Faites donc le tour de votre nouvel environnement de travail en saisissant les fonctions ci-dessous dans le terminal Matlab.

intro	lance l'introduction à Matlab
demo	lance une démonstration sur les fonctionnalités de Matlab
doc	information sur la boite à outils disponibles

2. L'aide dans Matlab

Mieux vaut apprendre à se repérer tout seul que de demander en permanence à son voisin comment faire. Les fonctions ci-dessous vous aiderons plus d'une fois dans l'élaboration de vos scripts :

help	produit une liste de toutes les commandes par thèmes
help *nom*	décrit la fonction *nom*
lookfor *nom*	recherche une instruction à partir du mot clé *nom*

3. Commandes générales

3.1. Gestion des fichiers.

Ces quelques fonctions peuvent vous aider à naviguer dans les arborescences de fichiers, créer et supprimer des fichiers,... Vous pouvez aussi utiliser les éléments de l'en-tête (*Browse for folder*,...) :

pwd	affiche le nom du répertoire courant
cd *rep*	change le répertoire courant
dir ou **ls**	fournit le catalogue d'un répertoire
delete	efface des fichiers ou des objets graphiques
mkdir *dirname*	crée le nouveau répertoire *dirname*
what	donne la liste des fichiers .m et .mat présents dans le répertoire courant
path	donne l'ensemble des dossier dans lequel Matlab va chercher ses fonctions
addpath *path*	rajoute le dossier dans les *path* consultables par Matlab
savepath	sauvegarde les *path* mis à jour

3.2. Calculs élémentaires.

Matlab peut être vu comme une grosse calculatrice, il contient donc les opérations élémentaires. En fonction du résultat souhaité, vous avez plusieurs manières de procéder :

5+8	affiche le résultat sans l'enregistrer dans une variable
a = 5+8	affiche le résultat et l'enregistre dans une variable
a = 5+8 ;	enregistre le résultat dans une variable sans l'afficher

NB : Pensez à donner un nom significatif à vos variables, il s'agit d'un élément essentiel pour créer un code clair et compréhensible par tous.

3.3. Constantes prédéfinies.

pi	3.1415
eps	2.2204e-016
Inf	nombre infini
NaN	n'est pas un nombre ou non défini mathématiquement

3.4. Navigation dans l'historique.

Il vous est possible de récupérer des opérations précédemment saisies ou naviguer dans les lignes de commandes à l'aide des flèches sur le clavier. Il s'agit du même principe que dans les terminaux Linux ou Windows.

3.5. Variables d'environnement.

Les variables déclarées dans Matlab restent en mémoire jusqu'à ce que vous demandiez qu'elles soient effacées ou que vous fermiez le logiciel. Elles sont visibles, consultables et modifiables dans la fenêtre *Workspace* en haut à droite de la fenêtre. Nous pouvons aussi le faire à partir de ligne de commande :

who	donne la liste des variables de l'espace de travail
whos	donne la liste des variables avec leurs caractéristiques
clear var_1 ... var_n	efface les variables var_1 ... var_n de l'espace de travail
clear	efface toutes les variables crées dans l'espace de travail
exist	vérifie l'existence d'une variable, d'un répertoire, ...

4. Les types de données

Matlab ne traite qu'un seul type de données : les matrices[1] ! Les scalaires sont des matrices 1×1, les vecteurs lignes des matrices $1 \times n$, les vecteurs colonnes des matrices $n \times 1$.

4.1. Construction explicite.

On peut former des vecteurs et des matrices en saisissant directement leurs valeurs.

. Scalaires

» s = 30

. Vecteurs numériques

» x = [1 ; 2 ; 3] : Création d'un vecteur colonne

» x = [1 2 3] = [1,2,3] : Création d'un vecteur ligne

» x.' : Transposée (Attention, x' représente la transposée de la matrice conjuguée.)

» y = [x, x, x] : création d'un vecteur à partir d'un autre vecteur

» z = [x x x]

1. Il existe 16 classes fondamentales en Matlab, toutes peuvent être organisées sous forme de matrice ou de vecteurs.

. Matrices
  » M = [11 12 13 14 ; 21 22 23 24]
  On peut construire des matrices à partir de vecteurs tant que les dimensions sont respectées :
  » y = [15 ;25]
  » mat1 = [M y]

. Vecteurs de chaines de caractères
La chaine de caractères est un vecteur ligne. Pour le créer, on entre les caractères en commençant et en terminant par '
  » ch='matlab'

. Nombres complexes
Dans Matlab, un nombre complexe est de la forme : $z = ai + b$
  » c = 2+i

. Polynômes
Vous avez la possibilité de créer des polynômes sous Matlab, en rangeant les coefficients dans un vecteur dans l'ordre des puissances décroissantes, ainsi $P(x) = X^2 - 6x + 9$ est représenté par :
  » P = [1 -6 9]

. Tenseurs
Pour certaines applications, il vous sera demandé de créer des matrices de dimension 3 aussi appelé « tenseur » :
  » T(:, :,1) = M1
  » T(:, :,2) = M2

4.2. Création rapide.

Il vous est possible de pouvoir créer plus rapidement certains vecteurs précis :

» l1 = 1 :10 vecteur contenant les nombres de 1 à 10

» l2 = 1 :1 :10 = l1

» l3 = 10 :1 :1 vecteur contenant les nombres de 10 à 1

» l4 = 1 :0.3 :pi vecteur contenant les nombres de 1 à π en prenant un pas de 0.3

» l5 = linspace(1,5,10) vecteur contenant 10 nombres de 1 à 5

4.3. Résumé sur l'ensemble des opérations sur les matrices/vecteurs.

Vecteurs

n : m	nombres de n à m par pas de 1
n : p : m	nombres de n à m par pas de p
linspace(n,m,p)	p nombres de n à m
length(x)	longueur de x
x(i)	i-ème coordonnées de x
x(i1 : i2)	coordonnées i1 à i2 de x
x(i1 : i2) =[ ]	supprimer les coordonnées i1 à i2 de x
[x,y]	concaténer les vecteurs x et y
x*y'	produit scalaire des vecteurs lignes x et y

Matrices

size(x)	taille de x
size(x,i)	dimension i de x
A(i,j)	coefficient d'ordre i,j de A
A(i1 : i2, :)	ligne i1 à i2 de A
A(i1 : i2, :) = [ ]	supprimer les lignes i1 à i2 de A
A(: , j1 : j2)	colonnes j1 à j2 de A
A(: , j1 : j2) = [ ]	supprimer les colonnes j1 à j2 de A
A(:)	ligne i1 à i2 de A
diag(A)	coefficient diagonaux de A

Matrices particulières

zeros(m,n)	matrice nulle de taille m,n
ones(m,n)	matrice de 1 de taille m,n
eye(n)	matrice identité de taille n
diag(x)	matrice diagonale dont la diagonale est le vecteur x
magic(n)	carré magique de taille n
rand(m,n)	matrice de taille m,n à coefficients i.i.d. de loi uniforme sur [0,1]
randn(m,n)	matrice de taille m,n à coefficients i.i.d. de loi normale $\mathcal{N}(0,1)$

5. Les opérations matricielles et fonctions

5.1. Les opérations matricielles.

A'	transposée de la matrice conjuguée de A
A.'	transposée de la matrice A
rank(A)	rang de A
inv(A)	inverse de A
expm(A)	exponentielle de A
det(A)	déterminant de A
trace(A)	trace de A
poly(A)	polynôme caractéristique de A
eig(A)	valeur propre de A
[U,D]=eig(A)	valeurs et vecteurs propres de A
+ -	addiction, soustraction
* ^	multiplication, puissance (matricielle)
.* .^	multiplication, puissance terme à terme
A\b	solution de Ax = b
b/A	solution de xA = b
./	division terme à termes

5.2. Les fonctions.

Fonctions usuelles

sqrt	**exp**	**log**
sin	**cos**	**tan**
asin	**acos**	**atan**
round	**floor**	**ceil**
abs	**angle**	**conj**

Fonctions vectorielles

max(x)	maximum de x
min(x)	minimum de x
sort(x)	tri par ordre croissant (par défaut) de x
[y, I] = sort(x)	retourne en plus les indices des éléments de x
find(x)	retourne les indices non nuls de x
[y, I] = find(x)	retourne des lignes (dans le vecteur I) et des colonnes (dans le vecteur J) des éléments non nuls du x
sum(x)	somme des éléments de x
cumsum(x)	vecteur contenant la somme cumulée des éléments de x
prod(x)	produit des éléments de x
cumprod(x)	vecteur contenant le produit cumulé des éléments de x
diff(x)	vecteur des différences entre 2 éléments consécutifs de x
mean(x)	moyenne des éléments de x
median(x)	médiane
std(x)	écart-type

6. Opérateurs relationnels et logiques

Il s'agit de composants qui nous permettent de relier logiquement deux matrices.

Opérateurs relationnels	$<$, $<=$, $>=$, $==$ (égalité), $\sim =$ (différence)
Opérateurs logiques	& (et), |(ou), $\sim$ ou not() (non)

Les opérateurs relationnels peuvent être utilisés avec des scalaires ou des matrices. Le résultat d'évaluation d'une expression relationnelle est soit 0(faux), soit 1(vrai). Appliqué à une matrice, l'opération retournera une matrice de même dimension composé de 0 et de 1.

7. Représentation graphique des résultats

7.1. Représentations de points dans le plan.

Il existe plusieurs possibilités pour représenter un ensemble de points $(x(i), y(i))$. Voici quelques exemples les plus utilisés.

plot(x,y,'s')	tracé d'une courbe ou d'un nuage de points
bar(x,y,'s')	tracé sous forme d'un histogramme
stem(x,y,'s')	diagramme en bâtons
stairs(x,y)	tracé en escalier des valeurs discrètes
fplot	représente des fonctions
hist	trace des histogrammes

's' est une paramètre facultatif qui permet de donner des précisions quant à la couleur ou au type de tracé souhaité. Par exemple, l'élément '-r' permet de tracer une ligne continue rouge.

7.2. Gestion de la fenêtre graphique.

hold on	les prochains tracés se superposeront aux tracés déjà effectués
hold off	le contenu de la fenêtre graphique active sera effacé lors du prochain tracé
clf	efface le contenu de la fenêtre graphique active
figure(n)	affiche ou rend active la fenêtre graphique numéro n
close	ferme la fenêtre graphique active
close all	ferme toutes les fenêtres graphiques
subplot(n,m,p)	partage la fenêtre graphique active en m× n espaces graphiques et sélectionne le p-ième

7.3. Axes et légendes.

axis([xmin xmax ymin ymax])	définir les échelles des axes
grid	quadrillage du graphique actif
grid off	enlever le quadrillage du graphique
title('titre')	titre pour le graphique
xlabel('titre')	légende pour l'axe des abscisses
ylabel('titre')	légende pour l'axe des ordonnées
legend('titre1', 'titre2')	légende pour chaque courbe du graphique
text(x,y,'texte')	texte explicatif à la position (x,y)
gtext('texte')	texte positionné à l'aide de la souris

7.4. Sauvegarder ses figures.

Il vous est possible de sauvegarder vos figures en cliquant sur le menu déroulant 'File' de la fenêtre graphique et en sélectionnant 'Save as'. Le format par défaut est le format '.fig' qui est un format qui ne peut s'ouvrir qu'à l'aide de Matlab [2], n'oubliez pas de changer le type de format si vous souhaitez travailler sur votre rapport depuis un poste sans Matlab.

8. Utilisation de fichiers

8.1. Les fichiers de sauvegarde.

La commande 'diary *nomfichier*' crée un fichier de journal de bord intitulé *nomfichier* qui garde la trace de toutes les commandes que vous avez tapées dans la fenêtre de commandes ainsi que les réponses de l'ordinateur. 'diary off' permet d'arrêter l'écriture dans le journal, tandis que 'diary on' permet de le reprendre.

save *nomfichier* var_1 ... var_n	sauvegarde les variables var_1 ...var_n dans *nomfichier.mat*
save *nomfichier*	sauvegarde toutes les variables de l'espace de travail dans *nomfichier.mat*
save -ASCII *nomfichier* var_1 ... var_n	sauvegarde les variables var_1 ...var_n dans *nomfichier.txt*
load *nomfichier*	charge les valeurs de *nomfichier.mat*

2. il existe en réalité d'autres méthodes mais nécessitant un peu plus de temps

8.2. Les fichiers de script (ou fichier d'instructions).

Il s'agit de fichier *.m* que vous pouvez créer pour éviter d'avoir à retaper un ensemble de commandes. Le plus pratique est de créer un fichier de script par TP, afin de pouvoir évaluer votre progression, noter vos observations et commenter des parties non fonctionnelles.

8.3. Les fichiers de fonctions.

Il s'agit de fichier *.m* comme le script, à l'exception qu'il va contenir la signature de la fonction en guise d'en-tête. Leur syntaxe est particulière et s'écrit automatiquement lorsque vous passer par le menu de création : *new -> function*.

Exemple signature : function [output_args] = untitled (input_args)

Comme vous pouvez le constater, la fonction n'attend pas forcément une variable de sortie, mais un vecteur de variables (*[output_args]*). Rien ne vous empêche dons de pouvoir créer une fonction retournant plusieurs arguments. Attention, pour pouvoir utiliser la fonction, il faut que celle-ci soit accessible à Matlab (et donc dans un dossier présent dans le path).

NB : N'oubliez pas de commenter votre fonction avec son rôle, ses entrées et sorties pour que vous puissiez vous y retrouver tout comme votre correcteur.

8.4. Dialogue avec l'utilisateur.

disp (var)	affiche le contenu de 'var'
rep=input('texte')	affiche la chaîne de caractère 'texte' et donne la main à l'utilisateur pour qu'il saisisse la valeur de rep
pause(n)	arrête l'exécution d'un programme pendant n secondes ; par défaut (sans n), reprends l'exécution du programme lorsque l'utilisateur appuie sur une touche
keyboard	arrête l'exécution du programme et donne la main à l'uti lisateur jusqu'à ce qu'il tape 'Entrée'
'\n'	ce string signifie que l'on effectuera un retour à la ligne, très utile pour la fonction input

9. Les commandes structurées

9.1. Structures conditionnelles.

if ... else ... end

if *conditions1* (cf 6.)
instructions1
elseif *conditions2* (cf 6.)
instructions2
...
else
instructions3
end

Vous n'êtes pas obligé de mettre l'ensemble du bloc, mais vous devez impérativement trouver l'élément '*end*' à la de votre structure. Il est tout à fait envisageable de mettre seulement un '*if*' pour vérifier qu'une condition est respectée. Les instructions ne seront effectuées que si la condition est vraie (=1) : les conditions se construisent à l'aide des opérateurs relationnels et logiques.

switch ... case ... end

switch *var*
case *value1*
instructions1
case *value2*
instructions2
otherwise
instructions3
end

On compare successivement la variable à la valeur *value1* puis *value2*, ... jusqu'à ce qu'on trouve sa valeur et que l'on exécute les instructions correspondantes. Si on ne trouve pas de correspondance, les instructions du '*default*' seront exécutées. L'élément '*break*' est nécessaire si l'on souhaite arrêter la boucle après la première bonne valeur trouvée.

9.2. Les boucles.

boucle for

for *variable* = *vecteur*
instructions
end

Pour chaque tour, la variable prendra une valeur du vecteur pour exécuter les instructions. On peut s'en servir comme d'une variable d'incrémentation (1, 2, 3,...) ou comme d'un élément de test (essayer telle fonction pour les valeurs 0.5, π, ...).

Attention : Il faut toujours privilégier les opérations vectorielles aux boucles, Matlab est optimisé pour cela.

boucle while

while *conditions* *instructions* **end**

Tant que la condition est vraie, on exécute les instructions.

NB : Vous avez créé une boucle infinie et votre programme ne s'arrête pas ? Appuyez sur 'Ctrl+C' pour forcer Matlab à arrêter l'exécution du programme.

10. Autres fonctionnalités utiles

tic ... toc	calcul du temps d'exécution du programme entre tic et toc
break	termine l'exécution de la boucle la plus proche
return	termine l'exécution d'un fichier script sans aller jusqu'à la fin
error('*message*')	affiche la chaîne de caractère message et interrompt l'exécution du programme en cours

11. Références

[1] M.Mokhtari et A. Mesbah. *Apprendre et maitriser Matlab.* Springer

[2] S.Lemaire. Documents pour la préparation à l'agrégation d'Orsay

[3] B. Laurent. Documents pour les TPs de Matlab de la Maîtrise de Mathématiques ingénierie

[4] ENS - Auteur inconnu. *TP 0 : INTRODUCTION A MATLAB.* (n.d.). Retrieved from http ://www.math.ens.fr/cours-apprentissage/TP/TPIntro.pdf

12. Notions avancées sur Matlab

Cette section est à destination des étudiants souhaitant pousser un peu leurs notions sur le logiciel Matlab, soit parce qu'ils y sont obligés dans le cadre d'un projet, d'un stage ou d'un emploi soit parce qu'ils ont envie d'agrandir leur culture sur les possibilités de ce logiciel.

Gestion des erreurs, calculs parallèles, objets... : Matlab est capable d'exécuter un certain nombre d'opérations que l'on peut trouver dans d'autres langages, certaines fonctionnalités ont été intégrées pour pouvoir coller au mieux avec les demandes. De plus, une communauté s'est constituée autour de la programmation Matlab, mettant à disposition des scripts ou des réponses à différents problèmes que rencontrent les utilisateurs.

Mode débug : Un souci dans votre script ? Un arrêt subit sur une ligne sans que vous compreniez pourquoi ? La barre gauche de chaque script sert pour poser des "breakpoints" qui vous permettront d'arrêter le script sur ce point. Sitôt que le programme est arrêté, le chevron Matlab devient "K>", et attend que vous saisissiez de nouvelles instructions pour les tester manuellement sans pour autant tout effacer ou arrêter. Vous pouvez naviguer de points en points à l'aide de "suivant".

Comparaison de versions : Il vous est possible de comparer deux fichiers ".m" dans Matlab. Il vous suffit de cliquer sur l'outil "compare" présent dans l'en-tête juste à côté de l'outil "open". Cette fonctionnalité peut s'avérer très utile si vous créez plusieurs versions d'un même script.

Les applications : Nous ne vous présentons qu'un des multiples aspects de Matlab, n'hésitez pas à explorer par vous-mêmes l'ensemble des outils mis à votre disposition. Certains add-on présent dans la section APPS de l'en-tête peuvent vous aider à générer des problèmes et solutions en traitement du signal, mathématiques et électronique. De même, certaines applications peuvent réécrire vos scripts Matlab en langage C++ ou compiler vos applications.

Création d'interfaces graphiques : Matlab contient des éléments vous permettant de créer vos propres interfaces graphiques, ce qui peut être un atout dans le cadre d'un projet où vous souhaitez limiter les actions de l'utilisateur. Si vous êtes intéressés, n'hésitez pas à fouillez sur le site officiel de Matlab ou d'appeler la fonction « *guide* » dans votre invite de commande.

Matlab et autres langages : Matlab peut s'intégrer dans d'autres langage informatique tout comme il est capable d'utiliser des ressources d'autres langage. Notamment, on peut retrouver des exemples d'usage avec du Java, du C/C++, du python et des applications avec des systèmes informatique embarqués comme arduino et raspberry pi.

Pointeurs de fonctions et fonctions anonymes : Certaines fonctionnalités de Matlab vous obligeront à donner une fonction en paramètre d'une autre fonction. Pour ce faire, vous allez avoir besoin des pointeurs (ou *handle* en anglais) : il se déclare à l'aide de l'opérateur '@'. Ainsi, imaginons que je possède la fonction *computeSquare* qui calcule le carré d'un paramètre x, je n'aurais qu'à écrire :

» *func_handle = @computeSquare*

Pour que *func_handle* devienne le pointeur vers la fonction *computeSquare*. Si la fonction est simple et peut tenir en une ligne, nous n'avons pas forcément besoin de créer un fichier de fonction (ou une fonction imbriquée). Nous allons avoir recours à ce que nous appelons des fonctions anonymes :

» *func_handle = @(n) n.^2*

Gestion de la mémoire - allocation et désallocation : Une chose importante quand on utilise cette douce notion qu'est la mémoire, il est important de bien gérer ses variables pour ne pas se retrouver avec le fameux "OUT OF MEMORY". Parmi celle-ci, vous pouvez :

— Supprimer ou réutiliser les variables. Si vous devez exécuter un script sur 15 images : chargez une image, exécutez votre script, sauvegardez le résultat puis chargez la seconde. C'est bien mieux que de charger les 15 images et de lancer le script sur tous ;

— Lorsque vous ouvrez un fichier ('*fid = fopen*') pour lire ce qu'il y a dedans (données d'expérience, tableaux, protocoles,...), n'oubliez pas de le fermer ('*fclose(fid)*'). C'est peut-être innocent dans le cadre des Travaux Pratiques, mais on peut avoir des surprises dans le milieu professionnel ;

— Ignorer des paramètres à la sortie des fonctions : en utilisant tilde, vous pouvez vous passer de déclarer des variables inutiles si des paramètres sortants d'une fonction ne vous intéressent pas :

  Exemple : Pour « *[, ,iB] = union(A,B)* », vous n'obtiendrez que le troisième paramètre de sortie.