MASTER : Traitement de l'Information et Exploitation des Données

Fouad Badran, Cécile Mallet, Carlos Mejia
Charles Sorror, Sylvie Thiria

« Mémo » d'initiation à Octave/Matlab

Sommaire : (+ thèmes et fonctions mentionnées)

Introduction, présentation

Matlab est un logiciel conçu pour réaliser des calculs scientifiques et de la visualisation de données en se basant sur le formalisme matriciel. Il possède un noyau qui contient des fonctions très élaborées comme par exemple la résolution de systèmes d'équations linéaires ou la recherche des valeurs propres et vecteurs propres d'une matrice. Par conséquent, afin de l'utiliser efficacement, il est indispensable de garder toujours présent à l'esprit que **toute variable est systématiquement considérée comme une matrice (ou ses cas particuliers : vecteur ou scalaire)**. Par la suite nous présupposons que le lecteur possède les connaissances d'algèbre matricielle (il doit connaître les notions de base du calcul matriciel) et qu'il a des notions suffisantes de programmation (algorithmique, boucle, fonction, condition, ...).

Ce document n'est pas exhaustif de toutes les possibilités de Matlab, il ne s'agit que d'un document de 1ère initiation, d'un mémo des principales fonctions les plus souvent utilisées.
La syntaxe des instructions Matlab ne sera pas explicitement présentée, elle ne sera vue qu'au travers des exemples présentés. Indiquons toutefois que les éléments du langage sont écris en **ARIAL BLACK**, les variables utilisateurs en **ARIAL GRAS**, et que certaines précisions sont parfois apportées par un commentaire qui commence par le caractère **%**.

On suppose par ailleurs que l'utilisateur dispose d'une session Matlab à partir de laquelle il pourra saisir les lignes de commandes proposées dans ce fascicule.

Matlab founit une aide "en ligne" qui peut être appelée grâce à la commande **help** ou par la barre de menu, ce qui vous donne accès à une table des matières (topics). **help** suivi d'un topic liste les fonctions qui s'y rapportent. **help** suivi du nom d'une fonction vous affiche la documentation de cette fonction ; la commande **type** vous en affichera le contenu (sauf pour les fonctions internes à Matlab dites « built-in functions » ?).

1) Mode de saisie et exécution d'un programme Matlab

Dans Matlab, la saisie des commandes peut se faire en mode immédiat, c'est-à-dire en tapant une instruction à l'invite **>>** (prompt) dans un éditeur en ligne (fenêtre de commande). Mais, Lorsque la suite des instructions à exécuter devient importante, l'éditeur de ligne de la fenêtre de commande n'est pas suffisant. Il est plus commode d'écrire cette suite d'instructions à l'aide d'un éditeur de texte et de la sauvegarder sous la forme d'un fichier de commandes ou script, pour une exécution ultérieure, dans un fichier désigné par '**nom.m**' (l'extension **.m** est obligatoire).
Pour créer ce fichier, passer par la barre de menu en choisissant : **File > New > M-file**. Une fenêtre d'éditeur de texte Matlab s'ouvre.
Saisissez une commande dans l'éditeur de texte, par exemple : **disp('Hello World');**
Sauvegarder ce script en passant par la barre de menu (**File > Save As...**) en renseignant le nom voulu pour ce script (par exemple **hello.m**) et en l'enregistrant. Dès lors, ce script devient **exécutable**, il suffit de taper son nom (sans l'extension **.m**) dans la fenêtre de commande de Matlab, mais il pourra aussi être à son tour, utilisé par d'autres scripts. Exemple :

```
>> hello
Hello World
```

Vous pourrez de façon analogue écrire des fonctions (voir **help function**) avec passage de paramètres en entrée-sortie (ce qui sera bordé ultérieurement dans un paragraphe dédié).
Tous vos fichiers de commandes (et ultérieurement vos fonctions) seront stockés dans un ou plusieurs répertoires dont il faudra indiquer, (lorsqu'ils ne sont pas présent dans le répertoire courant d'exécution de Matlab), les chemins d'accès par la commande **path** ou **addpath**.

Il est impératif de ne jamais désigner une variable par le même nom qu'un programme ou une fonction Matlab.

Pendant toute la durée d'une session Matlab, les touches du clavier flèche haut et flèche bas permettent de parcourir l'historique de toutes les commandes exécutées depuis le début. Cette

possibilité fournit une aide en évitant d'avoir à entrer à nouveau certaines commandes fastidieuses à écrire.

2) Définir une matrice

2.1) Création d'une matrice par une liste explicite des éléments

↳ Vérifier que les instructions suivantes donnent le même résultat pour la variable **x**, (pour ces premiers exemples, rester dans la fenêtre de commande Matlab et écrire après l'invite: **>>**)

```
x = [1 2 3 4 5 6]          % Les crochets [] permettent de définir une variable
x = [1,2,3,4,5,6]          % le pourcentage % permet de saisir un commentaire
x = [1:1:6]
```

La particule ' **:** ' permet de préciser des intervalles sous la forme **[d:p:f]** où **d** marque le début, **p** le pas et **f** la fin.

```
x = [1;2;3;4;5;6]'
```

L'opérateur de transposition hermitienne est le signe 'apostrophe'.

D'un point de vue général et pour éviter toute ambiguïté, nous ferons la convention qu'un vecteur x est toujours représenté par une matrice colonne (donc de dimension (n,1)) et que son transposé x' est donc une matrice ligne de dimension (1,n).

↳ Entrer maintenant la matrice Y :

```
Y = [1 2 3; 4 5 6; 7 8 9]
```

Le séparateur de colonnes est l'espace ou la virgule, tandis que le séparateur de lignes est le point-virgule.

↳ Terminer une commande par un point-virgule (**;**) empêche l'affichage du résultat de cette commande

```
x = 1:6;
```

Pour afficher le contenu d'une variable entrer simplement son nom.

```
x
```

2.2) Création d'une matrice par utilisation des fonctions offertes par Matlab :

↳ Utilisez par exemple (liste non exhaustive) et étudiez les résultats obtenus lors de l'utilisation des instructions suivantes :

```
U  = ones(3,4)
U  = ones(1,4)
Z  = zeros(3,4)
N  = rand(4,5)
N  = randn(5,4)
Id = eye(4)
Id = eye(3,4)
A  = magic(6)
```

On aura compris que **ones(l,c)** permet de créer une matrice remplie de 1, que **zeros(l,c)** permet de créer une matrice remplie de 0, que **rand(l,c)** et **randn(l,c)** construise des matrices avec des valeurs aléatoires suivant la loi uniforme ou gaussienne, et que **eyes** produit une matrice identité. Pour **magic**, utiliser la fonction **help**. Les dimensions de la matrice (en ligne et en colonne) sont déterminées par les paramètres **l** et **c**.

↳ Voir aussi **triu** et **tril** pour former une matrice supérieure ou inférieure.

2.3) Extraction ou extension d'une matrice existante :

Les termes d'une matrice peuvent être référencées par indexation entre parenthèse en ligne et en colonne; ainsi, **A(5, 3)** correspond à l'élément se trouvant à la 5ème ligne et 3ème colonne de la matrice A.

Pour extraire une matrice, essayer par exemple :

```
B = A(3:end, 2:4)          % ici, end dénote le dernier élément
```

puis, pour étendre une matrice :
B(2,4) = 9

On peut donc extraire un vecteur d'une matrice :
v = A(:, 3)
v = A(4, :)
Dans cette situation, la particule ' **:** ' définit toutes les lignes ou toutes les colonnes.

↳ Voir aussi **diag** pour extraire une diagonale d'une matrice.

↳ On peut également '**concaténer**' des matrices en ligne (si elles ont le même nombre de colonnes) ou en colonne (si elles ont le même nombre de lignes) . Exemple :
M1 = ones(5,3) , M2 = ones(3,5) *2,
M3 = [M1 ; M2'] **%** ; -> concaténation en ligne
M4 = [M1' , M2] **%** , -> concaténation en colonne

2.4) Autres commandes utiles:

↳ La commande **size** permet de connaitre la taille d'une matrice (ou d'un vecteur). Elle retourne 2 valeurs dont la 1ère est le nombre de ligne, et la 2ème le nombre de colonne : **size(B)**
On peut ne récupérer qu'une de ces 2 valeurs avec la syntaxe : **size(B,1)** ou **size(B,2)**

↳ Voir aussi la commande **length** (**help length**) qui renvoie la dimension maximale.

↳ La commande **reshape** permet de réorganiser une matrice à condition d'en garder le même nombre d'éléments. Essayer par exemple :
B = reshape (A,12,3)

↳ Il y a aussi la commandes **repmat** qui permet de dupliquer une matrice, les commandes **sort** ou **sortrows** qui permettent de la trier et les commandes **flipud** et **fliplr** qui servent à la retourner.

↳ la commande **who** permet de connaître les variables en cours dans l'espace mémoire de travail (Workspace). la commande **whos** informe de plus leur type et leur taille. Dans les versions plus récentes, ces informations sont visibles dans une fenêtre dédiée au Workspace.

↳ Dans le même ordre d'idée, la commande **exist** permet de savoir si une variable existe.

↳ **clear nom** supprime la variable dénommée **nom**, **clear** ou **clear all** les supprime toutes.

3) Opérations sur les matrices

3.1) Les opérateurs arithmétiques sont :

+ (addition), **-** (soustraction), ***** (multiplication matricielle),
/ (division matricielle), **^** (puissance matricielle)

Vous pouvez par exemple essayer : **A+2** ou **A/2**, ... mais nous vous laissons le soin d'expérimenter plus avant et par vous-même ces opérateurs.

Si, pour les opérateurs *****, **/**, **^**, on veut qu'ils s'appliquent élément par élément (plutôt que matriciellement} , il faut qu'ils soient précédés d'un point : **.***, **./**, **.^**

Afin de vous familiariser avec ces types d'opérateurs, traiter les exemples suivant :

Soient deux vecteurs :
x = [10: -1: 1]' et **y = ones(10,1)**

↳ Effectuer le produit scalaire (ligne*colonne) de ces deux vecteurs avec de l'opérateur ***** :
x'*y

↳ puis le produit extérieur (colonne*ligne) :
x*y'

↳ enfin le produit terme à terme à l'aide de ".*" :
x.*y

Vous pouvez reprendre ces exemples avec les opérateurs **/** et **^**.

Matlab permet donc d'effectuer 3 types de produit, selon la nature des opérateurs et des opérandes.

↳ Les fonctions **sum** et **prod** permettent respectivement de réaliser la sommation ou le produit, des éléments d'une matrice en ligne (par défaut) ou en colonne.
N=1:9; N=reshape(N,3,3)
sum(N), sum(N,1), sum(N,2), prod(N), prod(N,1), prod(N,2),

↳ On peut également créer et manipuler des données complexes : **x + j*y**. Il est à noter ici que Les lettres i et j doivent être réservées à la notation de $\sqrt{-1}$, on évitera donc de les utiliser comme indice.

3.2) Calcul matriciel

↳ Inverser une matrice avec la fonction **inv**, exemple :
A = rand(4,4)+magic(4)
X = inv(A)
Et on peut vérifier que $AX = I$.
A * X

↳ Une matrice carrée est-elle inversible ? Le déterminant peut permettre de répondre à cette question. On le calcule avec la fonction **det** : **d = det(A)**.

↳ Calculer les valeurs propres λ et les vecteurs propres x pour une matrice carrée A tel que : $Ax = \lambda x$. grace à la fonction **eig** ; exemple
[x,l] = eig(A)
On peut vérifier que $Ax = \lambda x$.
A * x , x * l

↳ La décomposition en valeurs singulière d'une matrice A (telle que $A=U*V*W'$) peut être obtenue avec la fonction **svd**; exemple :
[U,V,W] = svd(A)
On vérifie que **U*V*W'** et égal à **A.**

3.3) Opérations statistiques de base

Pour les calculs statistiques habituels, on dispose des fonctions suivantes :

min (le minimum)	**max** (le maximum)	**mean** (moyenne)
median (la médianne)	**std** (écart type)	**var** (variance)

↳ Ces fonctions appliquent leurs calculs à chaque vecteur colonne de la matrice passée en paramètre. Notez que pour certaines fonctions (comme **mean**, **median**, **std**) on peut obtenir un résultat en ligne en passant la valeur 2 dans un paramètre supplémentaire. Exemple **mean(A,2)**. Pour avoir un résultat sur la matrice toute entière, on pourra soit utiliser **reshape**, soit appeler 2 fois la fonction. Exemple avec **max**.
N = rand(6,4) ;
max(reshape(N, 24, 1))
max(max(N))

↳ De plus les fonctions **corrcoef** et **cov** (qui suivent la même syntaxe) peuvent être utilisées pour calculer, la 1ère, la matrice des coefficients de corrélation, la 2ème, la matrice de covariance.

3.4) Autres fonctions mathématiques

↳ Nous indiquons ci-dessous une sélection des fonctions mathématiques élémentaires dont vous pouvez obtenir une liste exhaustive en tapant **help elfun**.

- Fonction trigonométriques : **cos sin tan acos asin atan**
- Fonction exponentielles : **abs sqrt exp log log10**
- Arrondis et restes : **floor fix ceil round mod rem**

↳ Il est à noter que les fonctions **log**, **sqrt** et **exp** ont leur version matricielle qui est **logm**, **sqrtm**, **expm**.

↳ Des fonctions de manipulation des nombres complexes sont également disponibles.

4) Les instructions conditionnelles

Elles sont construites à l'aide des opérateurs suivants :

Les opérateurs relationnels qui sont :
== (égal) **~=** (différent) **>** (supérieur) **>=** (supérieur ou égal)
< (inférieur) **<=** (inférieur ou égal)

Les opérateurs logiques qui sont :
& (et) **|** (ou) **~** (non)

Une expression booléenne s'exprime par la combinaison de ces opérateurs ; elle prend la valeur FAUX représentée par 0 ou VRAI représentée par toute autre valeur (différente de 0).

Après avoir affecté : **a=1 ; b=2 ; c=3;**, essayer :
a==a, b==c, a==c-b, b>a, b>c, (b==c-a & c>a) | a+b==c, …
L'utilisation du parenthésage permet d'ordonner et/ou regrouper des expressions booléennes.

Les instructions conditionnelles s'écrivent avec **if else elseif end** (il n'y a pas de **then**) combiné à une expression booléenne.

A essayer :
```
if (x) disp('VRAI'); else disp('FAUX'); end
```
avec x successivement égal à -1, 0, 1, 2

```
if (x==a)
   disp ('x est égal à a');
elseif (x==b)
   disp ('x est égal à b');
else
   disp(['x est égal à ' int2str(x)]);   % int2str convertit un entier en chaîne de caractères
end
```
avec x successivement égal à a, b, 9.

Voir aussi : **switch**

5) Boucle **for** et vectorisation d'un calcul

Saisir par l'éditeur de texte et sauvegarder les deux programmes suivants qui construisent le même vecteur :

a) **toto.m**
```
% sinus (commentaire)
k = 0;
for  t = 0 : .01 : 500
     k = k + 1;
     y(k,1) = sin(t);
end
```

b) **titi.m**
```
% sinus (commentaire)
     t = [0: .01 : 500]';
     y = sin(t);
```

Puis comparer leur temps d'exécution en lançant :

tic; toto; toc

et :

tic; titi; toc

la commande **tic** démarre un 'timer', et **toc** indique le temps écoulé en seconde depuis le dernier **tic.**
Vous avez du constater que l'exécution du programme **titi** était beaucoup plus rapide que celle de **toto**. Dans le cas de **titi**, les calculs sont présentés vectoriellement. Lorsque cela est possible cette méthode est donc préférable à l'usage d'une boucle **for** ou **while**.

↳ Voir aussi : - **while** (**help while**) qui suit une syntaxe 'assez' standard :

while (expression booléenne) ... **; end**

- **find** (§10).

6) Chaînes de caractères et format de conversion

6.1) Chaînes de caractères

Dans Matlab, les chaînes de caractères s'écrivent entre quote :

```
str1 = 'salut l''artiste' ; disp(str1)
```

↳ A l'intérieur d'une chaîne, pour qu'une quote soit prise comme une quote, elle doit être doublée.

↳ Les éléments d'une chaîne sont accessibles par indexation : **str1(9)**

Il existe de nombreuses fonctions qui s'appliquent sur les chaînes de caractères, nous n'en évoquons ici que quelques unes à titre d'exemple :

↳ Les fonctions **abs** ou **double** en donne le code ascii : **s=abs(str1)** ; **char** ou **setstr** font l'inverse : **char(s)**

↳ Une valeur numérique (i.e. un scalaire) peut être convertie en chaîne avec la fonction **num2str** :

```
str_pi = num2str(pi)          % nb : pi est prédéfini dans Matlab
```

↳ La chaîne obtenue peut être limitée à la partie entière en utilisant **int2str** :

```
str_pi = int2str(pi)
```

↳ Inversement, la conversion d'une chaîne en valeur numérique peut s'opérer avec **str2num** :

```
pi2 = str2num(str_pi)*2
```

↳ **strcmp** sert à comparer des chaînes**, upper** ou **lower** permettent de les convertir en majuscule ou minuscule

↳ **strtok** sépare une chaîne de son premier élément :

```
[str_first, str_left] = strtok(str1)
```

↳ La concaténation de chaînes peut s'effectuer avec **strcat** :

```
strcat(str_first,' a toi', str_left)
```

↳ etc…, Les fonctions associées aux chaînes de caractères peuvent être listées en tapant **help strfun**.

6.2) Conversion de chaînes avec format

Le format permet d'indiquer sous quelle forme une variable doit convertir en chaîne de caractères
Ce formatage s'exprime, à l'intérieur d'une chaîne de caractère par un caractère spécial précédé du caractère '**%**'.
Exemple avec la fonction **sprintf** qui permet justement de formater des données en chaîne de caratères :

```
a=3 ; b=5 ;
str = sprintf('%d + %d = %d \n', a, b, a+b)
```

La chaîne de caractères est délimitée par les quotes (‘). Les variables ou expressions qui suivent (après la virgule) seront associées dans l'ordre au format indiqué par le caractère **%**. Par exemple le 1er **%d** est associé à la variable **a**, il indique que cette variable doit être convertie sous la forme d'un entier. A la fin de la chaîne, on trouve un caractère spécial : **\n** qui correspond au saut de ligne.

↳ Les principaux formats de conversion sont : **%d** et **%i** pour les entiers, **%f** pour les floats et les doubles et **%e** pour une notation scientifique puis **%c** pour un caractère de type quelconque et **%s** pour une chaîne de caractères.

On peut insérer dans un format la taille du champ de conversion :
Comparer **sprintf(‘%s’,’toto’)**, et **sprintf(‘%10s’,’toto’)**

Pour une expression numérique, on peut en préciser la partie entière et décimale. Essayez par exemple :
sprintf(‘%d’,9), sprintf(‘%8d’,9), sprintf(‘%8.6d’,9),

↳ Des caractères dits 'drapeaux' peuvent être associés au format qui permettent de plus d'agrémenter et cadrer la conversion. Par exemple l'insertion du caractère **–** juste après le **%** permet de cadrer à gauche. Il y a par exemple également le **+** qui permet d'introduire le signe (+ ou -), le **0** qui remplit le début d'un champ numérique par des 0, ou l'espace qui rajoute un espace en l'absence du signe -.

↳ Les principaux caractères spéciaux sont **/t** pour la tabulation et **/n** pour le saut de ligne.

En fait, l'ensemble de ces spécifications suivent celles du langage C auquel vous pouvez vous reporter.

La fonction **sscanf** est l'inverse de **sprintf**, elle permet d'extraire, d'une chaîne de caractères, des variables selon le format. Exemple :

```
str = '1 + 2.5 = 3 + 0.5'
A = sscanf(str, '%d + %f = %d + %f')
```

7) Entrées - Sorties

↳ Entrée clavier : L'instruction **input** permet de demander à l'utilisateur de votre programme (lorsqu'il est lancé en mode immédiat) de saisir une valeur au clavier, ce qui peut le rendre plus interactif : **val = input(‘entrez une valeur : ’) ;** La saisie se terminant par un retour chariot.

↳ Sortie écran : On a déjà constaté que la réponse d'une commande Matlab s'affichait à l'écran si elle ne se terminait pas par un point-virgule. On a également rencontré la commande **disp** qui affiche une (ou des) chaîne(s) de caractères : **disp([‘val vaut ', int2str(val), ‘ au moins’]);**. Il y a aussi **fprintf** évoquée ci-après.

↳ Fichier : L'instruction **save** permet d'enregistrer simplement des variables dans un fichier.mat elle se présentent ainsi : **save nom_du_fichier liste_des_variables**; exemple :

```
A = [ 1 : 2 : 9] , B = rand(3,3),
save mesdonnees val A B
```

Les données ainsi sauvegardées peuvent être directement récupérées par l'instruction **load** :

```
clear val A B;
load mesdonnees
```

Si aucune option n'est précisée **save** enregistre les données dans un fichier **.mat** en format binaire. Avec l'option **–ascii** on peut les sauvegarder en ascii dans un fichier d'extension **.txt** (**save –ascii mesdonnees val A B**) mais dans ce cas, elles ne pourront être lu avec **load** que si le nombre de lignes et de colonnes sont les mêmes pour tous les enregistrements (comme une matrice) ce qui n'est pas le cas dans l'exemple.

↳ On trouve bien d'autres fonctions d'entrée-sortie dont la syntaxe est empruntée au langage C comme **fopen**, **fclose**, **fwrite** ou **fread** pour ne citer que les plus courantes. Faire un **help iofun** pour en découvrir d'autres.

↳ Parmi elles, il y a la fonction **fprintf** qui permet l'écriture de données formatées comme pour **sprintf** : **fprintf(fid, format, expressions associées au format)** ou **fid** est un identifiant de fichier (file Id) rendu par une fonction d'ouverture de fichier comme **fopen**. Exemple :

```
myfid = fopen('monfichier','w') ;          % 'w' => ouverture en écriture
a=3 ; b=5 ;
fprintf(myfid, '%d + %d = %d \n', a, b, a+b);   % Ecriture formatée
fclose (myfid) ;                           % Fermeture du fichier
```

On peut utiliser **fprintf** sans le paramètre **fid**, dans ce cas, la sortie se fait sur l'écran.

Comme **sscanf** faisait l'inverse de **sprintf** pour les chaînes de caractère, On dispose de la fonction **fscanf** qui fait l'inverse de **fprintf** pour lire les données d'un fichier de façon formatée.

```
myfid = fopen('monfichier','r');           % 'r' => ouverture en lecture
D = fscanf(myfid,'%d + %d = %d \n')        % Lecture formatée des données
fclose (myfid);
```

8) Représentations graphiques

Un des attraits et points forts de Matlab est la possibilité de tracer des courbes de manière très simple sans que l'utilisateur n'ait à se préoccuper des ressources et des mises à l'échelle nécessaire pour mener à bien cette tâche à bien

8.1) Partie 1 : Les sorties graphiques se font dans une fenêtre dédiée appelée figure. **figure** est également la commande qui permet de créer une fenêtre.

↳ L'instruction **plot** est la plus simple. Soit **x=[-20:20]' ;** et **y=x.^2 ;**
L'instruction **plot(y)** trace en ordonnée toutes les composantes du vecteur y avec leur indice (à partir de 1) en abscisse. Vous remarquerez que si aucune figure courant n'est active, **plot** en crée une (c'est vrai aussi pour les autres fonctions d'affichage graphique).
Quand y est un vecteur de dimension n très grande on peut n'en visualiser qu'une partie avec **plot(y(a:b))** ou qu'un point sur p avec **plot(y(1:p:n))**.

Attention : lorsque les vecteurs représentent des valeurs d'une fonction continue échantillonnée, les tracés obtenus peuvent surprendre. Il ne faut jamais perdre de vue que Matlab travaille toujours avec des grandeurs discrètes.

Quand y est une fonction du vecteur x, l'instruction **plot(x,y)** permettra de visualiser y(x). L'instruction **plot(x,y,x,z)** ou **plot(x,[y;z])** permet de visualiser y(x) et z(x) sur le même graphique. (***nb : il est impératif que les vecteurs x,y et z soient de même dimension.***).

Des marqueurs et des couleurs peuvent être précisés, entre quotes, à l'instruction **plot** : dans l'instruction : **plot(x,y,'*r', x,z,'+m')**, * et + sont des marqueurs, r et m sont des couleurs. Faire **help plot** pour connaitre les autres possibilités.

Une autre façon de réaliser plusieurs tracés graphiques sur une même figure est d'utiliser l'instruction **hold on** qui permet de ne pas effacer la fenêtre courante :

```
figure; hold on;
plot(x,y);
plot(x,z);
hold off;
plot(x,y);
```

hold off interrompt la suite de tracé graphique sur la même fenêtre.

8.2) Partie 2 :

↳ Une matrice peut être représentée selon une échelle de couleur grâce à la fonction **imagesc** :

```
Z=x*y';
Imagesc(Z) ;
colorbar ;              % Permet d'afficher l'échelle de couleur sur la figure
colormap (winter) ;     % Permet le choix d'une 'map' de couleurs dont certaines sont prédéfinies
                        % comme winter  (jet est la map par défaut). Faire un help graph3D
                        % pour connaitre les autres maps.
```

↳ La visualisation des graphes des fonctions de 2 variables (graphes 3-D) se fait de façon tout aussi simple grâce aux instructions **mesh(Z)** ou **surf(Z)** par exemple, où Z est une matrice (à 2 dimensions) dont les composantes correspondent aux valeurs prises par la fonction. Essayez par exemple :

```
Z=x*y';
figure; mesh(Z)                          % surface définie par une grille
figure; surf(Z);                         % surface remplie
figure; surf(x,y,Z);  shading interp ;   % shading interp permet de lisser les couleurs
```

8.3) Partie 3 : Voici ci-dessous un autre exemple qui vous montre d'autres possibilités

```
x = [-2:0.1:2]' ;
y = sin(x);
X = x * ones(length(x), 1);
Y = ones(length(x),1) * y';
%
figure; % taper les instructions à la suite pour voir l'effet produit par chacune
surf (x, y, X.^2 + Y.^2);                % surface colorée
view(-30, 40)                            % angle de vue (azimut, élévation)
surfc(x, y, X.^2 + Y.^2);                % trace des contours
surfl(x, y, X.^2 + Y.^2);                % trace une surface avec des conditions de lumière
light('Position',[1 -1 1]);                  % ajoute une lumière (non disponible pour Octave-3.2.4)
light('Position',[-1 -1 1], 'Color',[.5 .5 .5]);   % lumière basse (non disponible pour Octave-3.2.4)
box on;                                  % encadre les axes (ou pas avec off)
grid on;                                 % marque un grillage par les axes (ou pas avec off)
title('Mettre un titre à la figure','FontSize',14);
xlabel('libellé pour l''axe des x');
ylabel('libellé pour l''axe des y');
text(-2.5, 0.5, 3, 'positionne un texte aux coordonnées spécifiées', 'FontSize',14);
```

8.4) Partie 4 :

↳ Avec Matlab, on dispose de fonctions graphiques déjà spécialisées comme par exemple **hist** pour les histogrammes, **bar** pour une représentation en barre, **pie** pour un camembert, etc...

↳ Il est possible grâce la fonction **subplot** de présenter plusieurs graphiques dans une même figure. Soit par exemple : **x=[1:100]'**

```
figure
subplot(2,2,1)        % On prévoit 2 lignes de 2 graphiques en colonne dont on va tracer le 1er ...
plot (x, sin(x))
subplot(2,2,2)        % ... puis le 2ème...
bar (x)
subplot(2,2,3)        % ... puis le  3ème...
pie (x)
subplot(2,2,4)        % ... et enfin  le 4ème
hist (x)
```

↳ Parfois, la représentation graphique n'est pas ajustée aux axes comme on le souhaiterait. La fonction **axis** permet d'y remédier de plusieurs façons :

- par une définition explicite des bornes des axes : **axis([XMIN XMAX YMIN YMAX]);** y ajouter **ZMIN ZMAX** pour une représentation 3D.
- par des définitions prédéfinies : **axis tight**, **axis equal** (faire un **help axis** pour en savoir plus).

↳ Lors d'un exemple précédent on a utilisé le paramètre '**FontSize**' pour définir la taille des caractères. En fait il existe aussi la fonction **set** qui permet de définir les propriétés d'un objet graphique. Exemple :

```
h = figure;                 % h est un pointeur (handle) sur l'objet figure.
get(h) ;                    % cette fonction permet de connaître les valeurs des propriétés associées
set(h,'color',[1 1 1]);     % à l'objet, que l'on peut modifier avec set
```

De la même façon on peut récupérer le pointeur de n'importe quel objets graphique (**h=title, h=plot**, etc...) pour en modifier les propriétés. Sachez aussi que pour personnaliser une représentation graphique, il est aussi possible d'intervenir directement par les boîtes de dialogues de la fenêtre accessible par son menu : **Edit>Figures Properties**, et en cliquant sur l'élément à modifier.

Dans Matlab, il, existe beaucoup de fonctions qui réalisent des représentations graphiques des données et même des vidéos, il n'est pas envisageable de toutes les mentionner ici, faites un **help** sur **graph2d**, **graphe3d** et **specgraph** pour vous en rendre compte et trouver la fonction de représentation graphique qui vous intéresse.

↳ La fonction **drawnow** permet de rendre effectives Les sorties graphiques.

↳ La fonction **saveas** permet de sauvegarder une figure dans un format à spécifier.

9) Ecrire une fonction

C'est le mot clé **function** qui permet de définir une fonction. Une fonction doit être écrite dans un fichier séparé dont le nom devra être celui de la fonction. Elle pourra être réutilisée par plusieurs scripts ou autres fonctions. Une fonction peut elle-même en comporter plusieurs autres à la fin, qui ne seront utilisables que par la fonction elle-même.

La syntaxe d'une fonction se présente ainsi :

function [s_1, s_2, ..., s_n] = nom_fonction(e_1, e_2, ..., e_n)

où les e_i sont les paramètres d'entrée, et les s_i ceux de sortie. Une fonction peut ne pas avoir de paramètre. Suit quelques exemples de fonctions simples, à saisir dans un seul fichier qu'on nommera **bonjour.m**.

```
function bonjour                % fonction sans paramètre
disp('bonjour');
sms = 'au revoir'
showsms (sms)
y = fact(5)
%-----------------------------------
function showsms(sms)           % fonction avec un paramètre en entrée
disp(sms);
%-----------------------------------

%-----------------------------------
function [y] = fact(x)          % fonction avec un paramètre en entrée et en sortie
if (x==0) y=1; else y = x * fact(x-1); end
%-----------------------------------
```

↳ On aura remarqué la **récursivité** de la dernière fonction (**fact**).

↳ Tous les paramètres de sortie doivent avoir été affectés par la fonction avant qu'elle ne se termine, par contre il est possible de ne pas renseigner les derniers paramètres d'entrée à l'appel d'une fonction.

Les fonctions **nargin** est **nargout** permettent de connaitre les nombres d'argument en entrée et en sortie. Avec **vararing** et **varargout**, il est aussi possible de passer les paramètres sous forme d'une liste.

↳ A voir aussi : **global**, **persistant**, ...

10) Compléments « en vrac »

10.1) Compléments divers

↳ La variable **pi** est déjà défini ; ainsi que **i** et **j** pour les nombres complexe.

↳ Les formats d'affichage des valeurs numériques peuvent être précisés par la commande **format** suivi d'une option. Exemple : **pi**, **format long**, **pi**. Par défaut c'est le **format short** qui est utilisé.

↳ **find** est une fonction souvent utilisée pour trouver les indices des éléments d'un tableau qui répondent à une condition. Cela permet par exemple d'éviter une boucle for. Soit par exemple : **x=1:10**; pour remplacer tous les chiffres pairs par 0 on peut coder :

```
k=find(~rem(x,2));  x(k)=0
```

Dans le même ordre d'idée, voir aussi : **all**, **any**, **intersect**, **setdiff**, ...

↳ La commande **date** donne la date, **clock** également mais sous forme décimale et en précisant l'heure la minute et la seconde.

↳ Lorsque qu'on réalise un tirage aléatoire avec **rand** ou **randn**, on peut maitriser l'initialisation du random en utilisant le paramètre **'seed'** ; exemple :

```
rand('seed',0) ; x=rand(1,10) ,      % l'exécution de ces 2 instructions à la suite produit le même
                                     % tirage
rand('seed', sum(100*clock)) ; x=rand(1,10) ,  % l'exécution de ces 2 instructions à la suite
                                               % produit des tirages différents.
```

Il y a aussi **randperm** qui retourne une permutation aléatoire ce qui permet de mélanger des données. Qui saura dire ce que fait cette ligne de code :

```
x=1:10, p=find(~rem(x,2)), ip=randperm(length(p)), x(p)=x(p(ip)) % elle ne mélange que les
                                                                 % chiffres pairs.
```

↳ **Tableau cellulaire** : c'est un tableau dont les éléments sont de type quelconque ce qui peut s'avérer pratique pour stocker des éléments de tailles différentes (comme des chaînes de caractères par exemple). Il se définit à l'aide des accolades que ce soit par l'indexation ou par les valeurs affectées. Exemples équivalents :

```
Tcell{1}='Toto';  Tcell{2}='Rantanplan'; Tcell{3}=pi;      % définition par indexation
Tcell = {'Toto', 'Rantanplan', pi}                         % définition par affectation
```

Exemple de conversions à comparer :

```
rand('seed',0);
x = [[5:10]'; 10*rand(5,1)];
T2dcarx = num2str(x)          % conversion de valeurs numériques en tableau 2D de caractères
T1dstrx  = cellstr(T2dcarx)   % conversion d'un tableau de caractères (2D) en un tableau
                              % cellulaire (1D) de chaînes de caractères
Tcellx = num2cell(x)          % conversion d'un tableau numérique en tableau cellulaire.
```

Pour manipuler les contenus numériques des cellules, il faut utiliser les accolades et non pas les parenthèses : **Tcellx{5}+Tcellx{7}** convient alors que **Tcellx(5)+Tcellx(7)** provoque une erreur.

10.2) Commandes de contrôle

↳ **pause(n)** : Suspension du programme là où cette commande est inserée soit pour n secondes si ce paramètre est passé, soit jusqu'à une frappe au clavier.

↳ **break** : Interrompt une boucle en cours

↳ **return** : Interrompt la fonction ou le script en cours (retour à l'appelant)

↳ Pour arrêter une exécution en cours faites <**Control-C**>.

↳ **keyboard** : Permet de prendre la main là où il est placé dans un code. Cela peut s'avérer utile pour débugger un programme. Pour reprendre l'exécution, il faut taper **return**.

error : Provoque l'arrêt de l'exécution en cours (i.e. rend la main à Matlab) en affichant la chaîne de caractères qui doit être passée en parametre. Si cette dernière est vide, alors, l'instruction **error** n'est pas prise en compte.

↳ **exit** ou **quit** : Interrompt l'éxecution de Matlab (termine la session Matlab).

↳ Certaines **commandes systeme**s (UNIX en général) peuvent être passées directement au prompt de Matlab (comme **ls**, **dir**, **cd**, **pwd**, …) mais il est aussi possible d'utiliser le caractère **!** à la suite duquel on peut passer une commande système quelconque.

Références

Le document présenté n'a pour prétention que d'aider à mettre un pied à l'étier aux néophytes de Matlab. Il est en effet assez limité par rapport à l'ensemble des fonctions et boites à outils disponibles sous ce logiciel. Pour une connaissance plus approfondie, nous vous livrons quelques références utiles, mis à part les manuels propres au logiciel Matlab :

Il y a une très vaste bibliothèque de livres en anglais. En français voici trois références :

- « Apprendre et maîtriser MATLAB »
  Mokhtari, SPRINGER VERLAG. ISBN : 3-540-62773-1 (10/1997) ~730p. ~54€ *
- « Traitement numérique du signal :simulation sous MATLAB »
  Blanchet, HERMES. ISBN : 2-86601-667-X (02/1998) ~320p. ~29€ *
- « Introduction à MATLAB »
  Jean-Thierry Lapresté, ELLIPSES. ISBN 2-7298-2401-6 ~230 p. ~21€ *

* Indications de pages approximatives, prix relevés sur internet en décembre 2011.

Adresses web utiles :

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

http://octave.sourceforge.net/

http://www.mathworks.com/