

INITIATION À MATLAB® TNS COURS COMMUN

Matlab® : généralités	2
A.1. Présentation	2
A.2. Démarrage et répertoire de travail	2
A.3. Interpréter une ligne de commande	3
A.4. Aide en ligne (help)	5
A.5. Variables et espace de travail	5
A.5.1. <i>Format d'affichage dans la fenêtre de commande, double précision</i>	6
A.5.2. <i>Rappeler des commandes précédemment tapées</i>	7
A.5.3. <i>Variables en mémoire dans l'espace de travail</i>	8
A.5.4. <i>Sauvegarder des variables</i>	8
A.5.5. <i>Chargement de variables dans l'espace de travail</i>	8
A.6. Premier fichier script	8
A.7. Manipuler des vecteurs	9
A.7.1. <i>Opérations de base</i>	9
A.7.2. <i>Opérations élément par élément</i>	10
A.7.3. <i>L'opérateur « : » d'énumération</i>	10
A.8. Fonctions mathématiques particulières, manipulation de données	11
A.9. Affichage graphique	12
A.9.1. <i>Affichage graphique de courbes</i>	12
A.9.2. <i>Affichage de plusieurs graphiques sur une même fenêtre, gestion des fenêtres</i>	13
A.10. Scripts et fonctions	14
A.10.1. <i>Scripts, commentaires</i>	14
A.10.2. <i>Fonctions</i>	14
A.11. Images et surfaces	16
A.11.1. <i>Affichage d'images</i>	16
A.11.2. <i>Affichage graphique en 3D (surfaces)</i>	17

Matlab® : généralités

A.1. Présentation

Matlab[®] est un logiciel de calcul scientifique, dont la base est le calcul matriciel (MATrix LABoratory = MATLAB). C'est un logiciel interactif, où les instructions peuvent être directement tapées en ligne de commande, ou bien sauveées dans un fichier (un script). On peut également utiliser des fonctions (comme en C). Le langage est interprété (pas de compilation des programmes avant exécution), et utilise des bibliothèques de fonctions très nombreuses, ce qui fait de Matlab un outil puissant permettant d'écrire des programmes scientifiques beaucoup plus rapidement qu'avec n'importe quel autre langage traditionnel comme le C.

Ce document est une initiation destinée aux étudiants du cours de traitement numérique du signal en cours commun (ingénieurs 1 et licence 3). L'apprentissage consiste à faire sous Matlab tout ce qui est fait dans ce fascicule d'initiation. Les exercices sont orientés vers le traitement du signal.

Si vous désirez vous perfectionner, un bon nombre des documentations disponibles sur Matlab utilisent les versions 5.x ou antérieures, ce qui suffit amplement pour commencer : citons pour les débutants l'initiation à Matlab d'Hervé Carfantan, visible sur son site <http://www.ast.obs-mip.fr/article226.html> (et l'une des sources d'inspiration pour ce document), ainsi que le *Matlab Primer* disponible sur <http://math.ucsd.edu/~driver/21d-s99/matlab-primer.html> (une recherche sur *google* avec le mot-clé « *Matlab Primer* » vous donnera des adresses supplémentaires pour d'autres documents similaires). De nombreuses informations sont disponibles sur le site de *The Mathworks* (qui développe Matlab) www.mathworks.fr (ou [.com](http://www.mathworks.com) pour le site en Anglais) ainsi que des programmes développés par des utilisateurs. On trouve aussi des livres (des vrais livres avec du vrai papier) pour la programmation, les outils graphiques, les applications en communications numériques, ... dans les bibliothèques scientifiques. Et bien sûr, ne pas oublier de fureter dans les nombreuses pages d'aide (PDF) qui sont parfois plus prolixes que l'aide en ligne.

A noter : il existe d'autres logiciels de calcul scientifique dont le fonctionnement est proche de Matlab (ligne de commande, langage interprété, syntaxe simple, bibliothèques de fonctions), qui eux sont distribués gratuitement : Octave, sous licence GNU <http://www.octave.org>, qui d'un point de vue syntaxe est le plus proche de Matlab, et Scilab, distribué par un consortium lié à l'INRIA <http://www.scilab.org>.

A.2. Démarrage et répertoire de travail

Matlab est un langage interprété, c'est-à-dire que les instructions n'ont pas besoin d'être compilées préalablement, elles sont lues et interprétées directement par le logiciel.

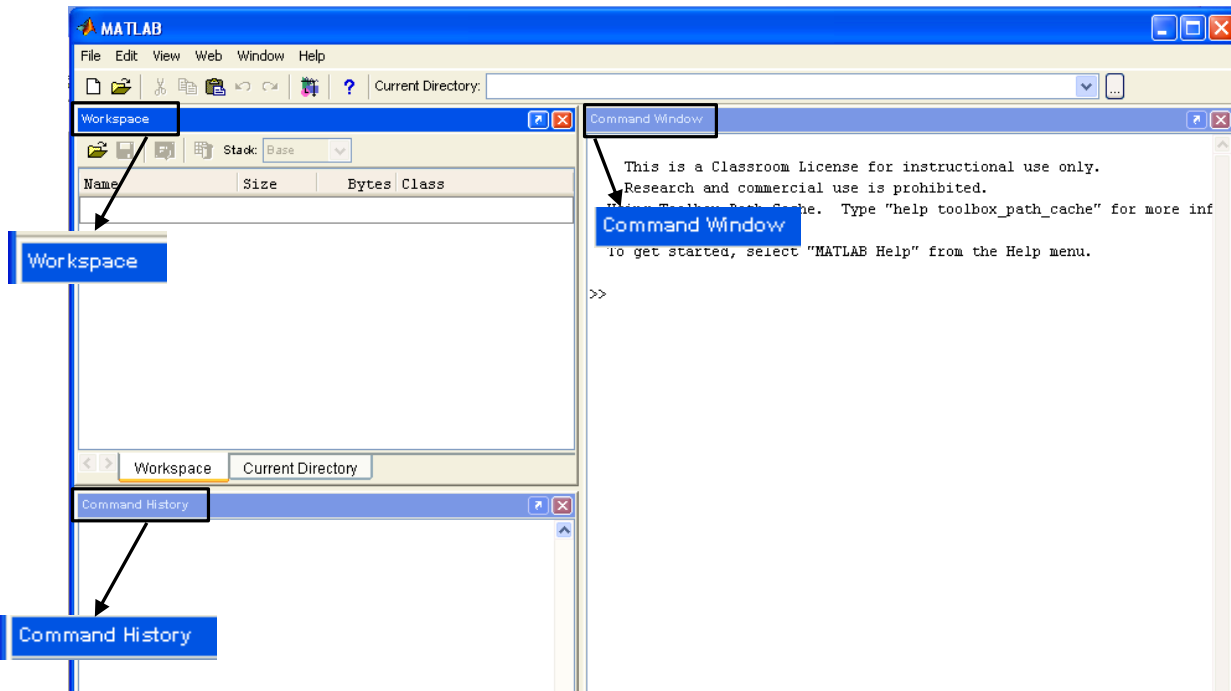
Après avoir lancé Matlab (double-cliquer sur l'icône Matlab), une fenêtre graphique apparaît. Elle est composée d'une barre de menu et de plusieurs fenêtres :

- une fenêtre de commande (*command window*)
- une fenêtre d'historique des commandes (*command history*)
- une fenêtre listant les variables présentes dans l'espace de travail (*workspace*)

Command window Les instructions sont tapées dans la fenêtre de commande, dès que le prompt » apparaît. Toute commande tapée est alors interprétée dès que la touche « Entrée » (ou « Enter ») a été frappée.


Command history Les instructions tapées dans la fenêtre de commande sont stockées en mémoire et affichées dans la fenêtre d'historique des commandes. On peut ainsi parcourir les commandes précédemment évaluées. Lorsque l'on quitte la session Matlab, les commandes stockées en mémoire sont sauveées dans un fichier d'historique.

Workspace L'espace de travail est la mémoire vive de la session. Toutes les variables définies dans la fenêtre de commande (ou définies lors de l'évaluation d'instructions dans des fichiers de scripts, voir les paragraphes 0 ou A.10) sont en mémoire dans l'espace de travail. On peut visualiser leur nom, taille, nombre d'octets (*Bytes*) et classe (réel, entier, caractère, variable logique...) dans la fenêtre *Workspace*. Quand on quitte la session, les variables stockées en mémoire dans l'espace de travail sont perdues.



Répertoire de travail Le répertoire par défaut est en général `c:\matlab\work`. Il faut donc changer de répertoire de travail, pour mettre ses fichiers dans un répertoire personnel. On suppose qu'il existe un disque D. Créons un répertoire de travail **TP_Matlab** sur le disque D :

```
>> cd d:
>> mkdir TP_Matlab
>> cd TP_Matlab
```

Un autre moyen de faire est d'utiliser la barre de menu, et de changer le *Current directory* en cliquant sur les trois petits points .

Créer ainsi un répertoire **Donnees** dans votre répertoire **TP_Matlab**.

Le répertoire dans lequel on se trouve est indiqué à côté de *Current directory*. On peut également utiliser la commande `pwd` (*print working directory*), et pour lister les fichiers dans le répertoire, la commande `dir` :

```
>> pwd
ans =

D:\TP_Matlab

>> dir

.          ..          Donnees
```

Si l'on utilise des fichiers qui sont dans le répertoire **TP_Matlab** à partir de la fenêtre de commande, pas de problème : Matlab trouve les fichiers qui sont dans le répertoire de travail courant. En revanche, si l'on veut utiliser d'autres fichiers situés dans d'autres répertoires, par exemple dans **Donnees** qui est un sous-répertoire du répertoire courant, il faut rajouter le chemin d'accès pour que Matlab puisse les trouver. Dans la barre de menu, l'onglet *File / Set Path* permet d'ajouter des répertoires (éventuellement avec leurs sous-répertoires) dans la liste des chemins existants.

Ajouter le répertoire **Donnees** dans la liste des chemins.

A.3. Interpréter une ligne de commande

Comme on vient de l'expérimenter, les instructions sont interprétées dès qu'on a tapé sur « Entrée ». On peut ainsi faire afficher l'aide en ligne générale, *via help*, qui liste les fonctions disponibles, par thème.

```
>> help
```

HELP topics:

```
matlab\general      - General purpose commands.
matlab\ops          - Operators and special characters.
matlab\lang         - Programming language constructs.
matlab\elmat        - Elementary matrices and matrix manipulation.
matlab\elfun        - Elementary math functions.
matlab\specfun      - Specialized math functions.
matlab\matfun       - Matrix functions - numerical linear algebra.
matlab\datafun      - Data analysis and Fourier transforms.
matlab\polyfun      - Interpolation and polynomials.
matlab\funfun       - Function functions and ODE solvers.
...                ...
```

Essayons quelques opérations très élémentaires...

```
>> 4+9
ans =

    13

>> 8/3
ans =

    2.6667

>> 9^4
ans =

    6561
```

`ans` est une variable par défaut interne à Matlab dans laquelle est stockée la dernière réponse (*answer*) obtenue. Pour voir le contenu de `ans` :

```
>> ans
ans =

    6561
```

La majorité des fonctions et constantes mathématiques usuelles sont disponibles.

```
>> help pi

PI      3.1415926535897....
PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897....

>> 2*sin(pi/3)
ans =

    1.7321

>> pi
ans =

    3.1416

>> exp(-sqrt(2))
ans =

    0.2431
```

A.4. Aide en ligne (help)

La fonction d'aide mentionnée tout au long de ce document est le `help`, qui renseigne à la fois sur les différentes rubriques et sur les fonctions elles-mêmes. On peut également passer par le menu (*Help / MATLAB Help*).

Si l'on veut des infos sur une rubrique particulière, par exemple `elfun`, on tape `help elfun`.

```
» help elfun
```

```
Elementary math functions.
```

```
Trigonometric.
```

```
sin          - Sine.
sinh         - Hyperbolic sine.
asin        - Inverse sine.
asinh       - Inverse hyperbolic sine.
cos         - Cosine.
cosh        - Hyperbolic cosine.
acos        - Inverse cosine.
acosh       - Inverse hyperbolic cosine.
tan         - Tangent.
...         ...
```

ou sur une fonction particulière : `help tan`.

```
» help tan
```

```
TAN      Tangent.
TAN(X) is the tangent of the elements of X.

See also ATAN, ATAN2.
```

Lorsque l'on cherche des informations sur une fonction, il faut toujours

1. Commencer par utiliser l'**aide en ligne de commande**
2. **Lire à partir du début** (remonter si besoin avec l'ascenseur) pour l'utilisation standard
3. Utiliser l'aide HTML (*onglet Help / Matlab Help* de la barre de menu) si l'on désire plus de détails

A.5. Variables et espace de travail

Dans Matlab, toute variable est une matrice de dimensions $m \times n$, avec m lignes, n colonnes, $m, n \geq 1$

- scalaire 1×1
- vecteur ligne $1 \times n$
- vecteur colonne $m \times 1$
- matrice $m \times n$

Un nom de variable doit respecter les règles suivantes :

- commencer par une lettre (a-z ou A-Z)
- avoir au plus 63 caractères
- ne contenir ni blancs, ni caractères de ponctuation, ni caractères accentués, ni opérateurs arithmétiques ($-$, $+$, \dots), ...

Matlab fait la différence entre les majuscules et les minuscules.

```
» r=2
r =
    2
```

```
» diametre=2*r
diametre =
    4
```

Matlab traite également les variables complexes ; le nombre complexe i (ou j) existe par défaut dans l'espace de travail.

```
>> sqrt(-1)
ans =

    0 + 1.0000i

>> a=2+3*i
a =

    2.0000 + 3.0000i

>> b=-3+5*j
b =

   -3.0000 + 5.0000i

>> a*b
ans =

  -21.0000 + 1.0000i

>> a'
ans =

    2.0000 - 3.0000i

>> a'*a
ans =

    13

>> abs(a)^2
ans =

    13.0000

>> i=1;j=1;
```

La dernière instruction a pour effet de changer la valeur par défaut de i et j qui deviennent des variables égales à 1. Si on recalculé a et b on trouve :

```
>> a=2+3*i
a =

    5

>> b=-3+5*j
b =

    2
```

A.5.1. Format d'affichage dans la fenêtre de commande, double précision

Lorsque l'on met un point-virgule à la fin d'une ligne, le résultat de l'opération n'est pas affiché, mais la commande a bien été exécutée.

```
>> circonference=2*pi*r;
>> surface=pi*r^2; volume_sphere=4/3*pi*r^3;
>> surface
surface =
```

```
12.5664
```

```
» volume_sphere  
volume_sphere =
```

```
33.5103
```

```
» direction='nord'  
direction =
```

```
nord
```

```
» distance=1001.36  
distance =
```

```
1.0014e+003
```

L'affichage par défaut est de 4 chiffres après la virgule, bien que les valeurs numériques soient en double précision. En double précision, les valeurs sont stockées sur 8 octets et ont au moins 15 chiffres significatifs, soit une précision de 10^{-16} . On peut choisir le format d'affichage à l'écran :

```
» help format
```

```
FORMAT Set output format.
```

```
All computations in MATLAB are done in double precision.  
FORMAT may be used to switch between different output  
display formats as follows:
```

```
FORMAT          Default. Same as SHORT.  
FORMAT SHORT    Scaled fixed point format with 5 digits.  
FORMAT LONG     Scaled fixed point format with 15 digits.  
FORMAT SHORT E  Floating point format with 5 digits.  
FORMAT LONG E   Floating point format with 15 digits.  
FORMAT SHORT G  Best of fixed or floating point format with 5 digits.  
FORMAT LONG G   Best of fixed or floating point format with 15 digits.  
FORMAT HEX      Hexadecimal format.  
FORMAT +        The symbols +, - and blank are printed  
                for positive, negative and zero elements.  
                Imaginary parts are ignored.  
FORMAT BANK     Fixed format for dollars and cents.  
FORMAT RAT      Approximation by ratio of small integers.
```

```
Spacing:
```

```
FORMAT COMPACT Suppress extra line-feeds.  
FORMAT LOOSE   Puts the extra line-feeds back in.
```

```
» format long  
» cos(pi/6)-sqrt(3)/2  
ans =  
1.110223024625157e-016
```

```
» format short
```

A.5.2. Rappeler des commandes précédemment tapées

Pour se déplacer dans l'historique des commandes, on utilise les flèches du clavier ↑ et ↓. Pour rappeler toutes les commandes commençant par la chaîne de caractères « di », il suffit de taper di puis d'utiliser ↑ pour faire défiler les instructions. On peut par exemple changer la valeur de r, puis rappeler les commandes précédentes et refaire tous les calculs avec cette nouvelle valeur.

A.5.3. Variables en mémoire dans l'espace de travail

Les variables utilisées peuvent être de types (ou de classes) différent(e)s, ce qui est appelé **class** dans Matlab. Pour connaître la classe d'une variable, voir la fenêtre *Workspace* (ou utiliser la fonction **class**).

L'espace de travail contient toutes les variables créées dans la fenêtre de commande. Pour voir toutes les variables présentes dans l'espace de travail, voir la fenêtre *Workspace* ou bien utiliser **who** (liste des variables) ou **whos** (liste des variables avec la taille et la classe).

A.5.4. Sauvegarder des variables

On peut sauver tout ou une partie des variables de l'espace de travail dans des fichiers de données Matlab (avec l'extension **.mat**). Par exemple, sauvons la variable **surface** dans le fichier **sphere.mat** :

```
» save sphere surface
» dir

.      ..      sphere.mat  donnees
```

L'instruction **save** permet de sauver des données sous plusieurs formats (Matlab, ASCII, binaire, ... voir la quatrième série d'exercices). La syntaxe suivante permet de SAUVER TOUTES LES VARIABLES DE L'ESPACE DE TRAVAIL dans un fichier Matlab :

```
save nom_fichier
```

Si **nom_fichier** est omis, les variables sont sauvées par défaut dans le fichier **matlab.mat**. On peut aussi sauver une ou plusieurs variables dans un même fichier Matlab :

```
save nom_fichier variable1 variable2 ... variableN
```

```
» save
```

```
Saving to: matlab.mat
```

A.5.5. Chargement de variables dans l'espace de travail

Pour charger les fichiers de données Matlab dans l'espace de travail on utilise l'instruction

```
load nom_fichier
```

Si **nom_fichier** est omis, le fichier **matlab.mat** est chargé par défaut (un message d'erreur s'affiche s'il n'existe pas). Par exemple

```
» clear all
» load sphere
» who
...
» load
```


```
Loading from: matlab.mat
```

```
» who
...
```

A.6. Premier fichier script

Les instructions peuvent être écrites dans un fichier texte, avec l'extension « **.m** », qu'on appelle script. Lorsque le fichier est exécuté, les instructions sont interprétées comme si elles avaient été tapées en ligne de commande.

Pour commenter des lignes dans le fichier, on utilise le caractère « **%** » avant le commentaire. Toute commande située après « **%** » n'est pas prise en compte par Matlab, jusqu'à la ligne suivante.

Créer un nouveau fichier script en cliquant sur l'onglet *File/New/M-file* ou en cliquant sur l'icône . Y mettre les instructions suivantes :

```
% Calcul du volume d'un cone
% les unites sont en metres
% hauteur
h=2;
```



```
% rayon
r=0.4;
volume=pi*r^2*h/3
```

Sauver ce fichier sous le nom cone.m et l'exécuter.

```
>> cone
volume =
    0.3351
```

A.7. Manipuler des vecteurs

A.7.1. Opérations de base

```
>> v=[1 3 5 7 11 13 17 19 23 29]
v =
     1     3     5     7    11    13    17    19    23    29
```

Calculer les dimensions d'un vecteur :

```
>> s=size(v)
s =
     1    10
```

Calculer la longueur d'un vecteur :

```
>> l=length(v)
l =
    10
```

Accéder à un élément :

```
>> v(5)
ans =
    11
```

Produit scalaire : il faut respecter les règles mathématiques habituelles.

```
>> p=v*v
??? Error using ==> *
Inner matrix dimensions must agree.
```

```
>> p=v*v'
p =
    2394
```

On peut concaténer des vecteurs facilement :

```
>> c=[v v]
c =

Columns 1 through 11
     1     3     5     7    11    13    17    19    23    29     1

Columns 12 through 20
     3     5     7    11    13    17    19    23    29
```

ou modifier un élément :

```
>> v(8)=sqrt(v(4))  
v =
```

```
Columns 1 through 7
```

```
1.0000    3.0000    5.0000    7.0000   11.0000   13.0000   17.0000
```

```
Columns 8 through 10
```

```
2.6458    23.0000   29.0000
```

A.7.2. Opérations élément par élément

Il est possible et même recommandé de faire des calculs vectorisés, c'est-à-dire en traitant tous les éléments d'un vecteur en même temps. Pour faire des calculs en une seule instruction sur chaque élément d'un vecteur, on met un point avant l'opérateur :

$v.^n$: élève chaque élément de v à la puissance n

$v.^{1/2}$ ou `sqrt(v)` : calcule la racine carrée de chaque élément de v

D'autres calculs peuvent être effectués de cette façon. Par exemple

```
>> v.^2  
ans =
```

```
Columns 1 through 7
```

```
1.0000    9.0000   25.0000   49.0000  121.0000  169.0000  289.0000
```

```
Columns 8 through 10
```

```
7.0000  529.0000  841.0000
```

```
>> 1./v  
ans =
```

```
Columns 1 through 7
```

```
1.0000    0.3333    0.2000    0.1429    0.0909    0.0769    0.0588
```

```
Columns 8 through 10
```

```
0.3780    0.0435    0.0345
```

A.7.3. L'opérateur « : » d'énumération

L'opérateur d'énumération s'utilise avec la syntaxe suivante

début : incrément : fin

où l'incrément n'a pas à être précisé quand il vaut 1.

```
>> ind1=3:10  
ind1 =
```

```
3    4    5    6    7    8    9   10
```

```
>> ind2=6:-1:1  
ind2 =
```

```
6    5    4    3    2    1
```

```

» temps=0:0.1:0.5
temps =

    0    0.1000    0.2000    0.3000    0.4000    0.5000

» temps(1:3)=-2
temps =

 -2.0000  -2.0000  -2.0000    0.3000    0.4000    0.5000

» temps(1:3)=[0.1 0.16 0.22]
temps =

    0.1000    0.1600    0.2200    0.3000    0.4000    0.5000

```

Première série d'exercices

A.8. Fonctions mathématiques particulières, manipulation de données

On peut obtenir différents types de vecteurs en utilisant directement des fonctions Matlab. Par exemple, la fonction **zeros** produit un vecteur de zéros, **ones** produit un vecteur de 1. Il faut préciser les dimensions lors de l'appel à la fonction : par exemple un vecteur ligne à 3 éléments a pour paramètres (1,3), pour indiquer 1 ligne et 3 colonnes.

```

» zeros(1,3)
ans =

    0    0    0

» ones(1,8)
ans =

    1    1    1    1    1    1    1    1

```

Le générateur de nombres pseudo-aléatoires de Matlab permet d'obtenir en une seule instruction un vecteur dont les éléments sont tirés au hasard. Pour générer des valeurs pseudo-aléatoires de loi uniforme sur [0, 1] (valeurs tirées au hasard uniformément dans l'intervalle [0, 1]) on utilise **rand**. On peut également générer des valeurs pseudo-aléatoires de loi gaussienne de moyenne nulle et de variance 1 avec **randn**.

```

» u=rand(1,5)
u =

    0.9501    0.2311    0.6068    0.4860    0.8913

» n=randn(4,1)
n =

 -0.4326
 -1.6656
  0.1253
  0.2877

```

On refait les deux instructions ci-dessus plusieurs fois d'affilée.

On constate que les valeurs changent à chaque fois, c'est-à-dire à chaque tirage.

Les fonctions mathématiques élémentaires (**help elfun**) s'appliquent directement aux vecteurs, en opérant sur chacun des éléments, comme par exemple les fonctions **exp**, **log**, **tan** ou **sqrt** utilisées précédemment. D'autres exemples suivent :

```

» v
v =

    Columns 1 through 7

    1.0000    3.0000    5.0000    7.0000   11.0000   13.0000   17.0000

    Columns 8 through 10

    2.6458   23.0000   29.0000

» log10(v)
ans =
                                % Logarithme base 10

    Columns 1 through 7

    0    0.4771    0.6990    0.8451    1.0414    1.1139    1.2304

    Columns 8 through 10

    0.4225    1.3617    1.4624

```

Parmi les fonctions de manipulation de données (`help datafun`), on remarque entre autres la fonction `sum` qui fait la somme de tous les éléments d'un vecteur.

```

» somme=sum(v)
somme =

    111.6458

» minv=min(-v)
minv =

    -29

» m=max(log(v))
m =

    3.3673

```

A.9. Affichage graphique

A.9.1. Affichage graphique de courbes

Pour afficher des courbes dans une fenêtre graphique, voir les fonctions de `help graph2d`. On présente ici que les fonctions les plus courantes. L'utilisation d'une instruction d'affichage graphique crée automatiquement une fenêtre graphique si aucune fenêtre n'a déjà été créée. On peut également initialiser une fenêtre graphique en faisant appel à l'instruction `figure(n)` où `n` est le numéro de la fenêtre. Les fenêtres sont numérotées à partir de 1.

```

» t=0:0.05:2;
» x=sin(pi*t);
» stem(t,x)
» stem(t,x,'r*')
» plot(t,x)

```

Pour afficher la même courbe en pointillés rouges :

```

» plot(t,x,'r:')

```

On peut tracer des courbes en échelle semi-logarithmique ou logarithmique (`semilogy`, `semilogx`, `loglog`).

```

» figure(2)
» z=exp(2*t);
» semilogy(t,z)
» hold on
» semilogy(t,1./z,'r')
» hold off

```

Il est possible d'ajouter du texte sur les figures : titre et label pour les axes avec `title`, `xlabel`, `ylabel`. On peut aussi créer une légende ou placer du texte sur les figures en utilisant la barre de menu des fenêtres graphiques (*Insert/ Legend* ou *Insert/ Text*).

```

» figure(1)
» title('Courbe x = sin(\pi t)')
» xlabel('t (secondes)')
» ylabel('x')

```

On peut rajouter un quadrillage sur la figure, changer les axes, etc.

```

» grid
» axis([0 1 0.1 10])

```

A.9.2. Affichage de plusieurs graphiques sur une même fenêtre, gestion des fenêtres

On peut afficher plusieurs courbes sur un même graphique en utilisant `hold on` et `hold off`, ou plusieurs graphiques sur une même fenêtre en utilisant la fonction `subplot`, ou encore passer d'une fenêtre à une autre avec `figure`.

```

» figure(2)
» t=-1:0.1:1;
» w1=1/2;
» plot(t, sin(2*pi*w1*t))
» hold on
» plot(t, cos(2*pi*w1*t),'m')
» plot(t, sin(2*pi*w1*t),'og')
» hold off

```

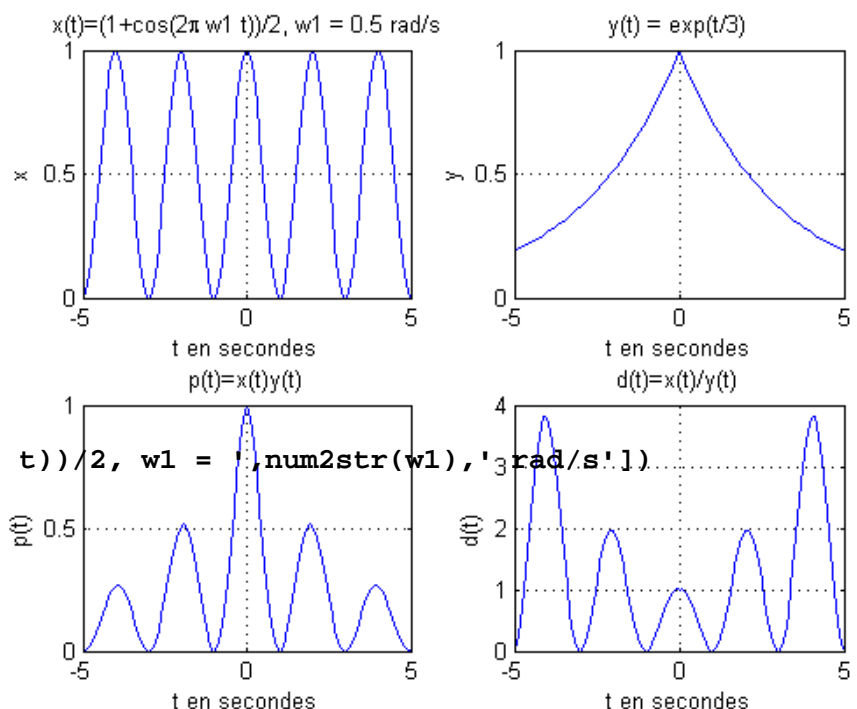
L'instruction `hold on` maintient le stylo graphique sur la fenêtre, alors que `hold off` le relève. L'instruction `hold` seule fait passer d'un état à l'autre.

Pour afficher plusieurs graphiques dans une même fenêtre, on utilise `subplot` de la façon suivante :

```

» t=-5:0.1:5;
» w1=0.5;
» x=(1+cos(2*pi*w1*t))/2;
» y=exp(-abs(t/3)) ;
» figure(2)
» subplot(2,2,1)
» plot(t,x)
» grid
» xlabel('t en secondes')
» ylabel('x')
» title(['x(t)=(1+cos(2\pi w1 t))/2, w1 = ',num2str(w1),' rad/s'])
» subplot(2,2,2)
» plot(t,y)
» grid
» xlabel('t en secondes')
» ylabel('y')
» title('y(t) = exp(t/3)')

```



```

» subplot(2,2,3)
» plot(t,x.*y)
» grid
» title('p(t)=x(t)y(t)')
» xlabel('t en secondes')
» ylabel('p(t)')
» subplot(2,2,4)
» plot(t,x./y)
» grid
» title('d(t)=x(t)/y(t)')
» xlabel('t en secondes')
» ylabel('d(t)')

```

Les 4 figures de la fenêtre peuvent être vues comme les 4 éléments d'une matrice 2x2, numérotés de 1 à 4 en suivant les lignes de gauche à droite :

1	2
3	4

Pour revenir à la 1^{ère} fenêtre, effacer la fenêtre, puis fermer toutes les fenêtres :

```

» figure(1)
» clf
» close all

```

A.10. Scripts et fonctions

A.10.1. Scripts, commentaires

Nous avons déjà abordé au paragraphe 0 la notion de script. On rappelle donc que l'on peut stocker dans un fichier toute suite d'instructions. Le fichier doit avoir l'extension `.m`. Ce fichier peut être appelé depuis la fenêtre de commande en tapant simplement

```
» nom_fichier
```

et toute la séquence de commandes qu'il contient sera exécutée. On ouvre un nouveau fichier et on y tape les lignes suivantes :

```
% Nom du programmeur : Mézigue
```

```
r=input('Entrez la valeur du rayon en m : ');
```

```
disp(['La surface vaut ', num2str(pi*r^2),'m2']) % Affichage surface
```

On sauve le fichier sous le nom
disque.m

puis dans la fenêtre de commande on tape le nom du fichier (sans le `.m`) ce qui produit la séquence interactive suivante

```
» disque
```

```
Entrez la valeur du rayon : 5
```

```
La surface vaut 78.5398
```

Toutes les variables présentes dans l'espace de travail sont accessibles lors de l'exécution du script, et toute variable créée ou modifiée par le script l'est également dans l'espace de travail.

Vous devez vous-même écrire les informations d'aide pour un script ou une fonction que vous avez créés. Pour les scripts, les commentaires situés en début de fichier (à partir de la 1^{ère} ligne) seront pris en compte par le `help`.

A.10.2. Fonctions

Nous avons déjà utilisé des fonctions dans les paragraphes précédents, comme par exemple `log10` qui a un paramètre d'entrée et un paramètre de sortie, ou `size` qui a plusieurs paramètres d'entrée et de sortie.

Les fonctions permettent de passer des variables en argument et de renvoyer des variables en sortie. Les fonctions sont des fichiers de type texte avec l'extension « .m », mais elles doivent respecter une syntaxe particulière : la PREMIÈRE LIGNE d'une FONCTION doit respecter impérativement la SYNTAXE SUIVANTE :

```
function [sortie1, sortie2, ..., sortieN] = nom_fonction(entree1, entree2, ..., entreeM)
```

où « entree1, ..., entreeM » sont les arguments d'entrée, et « sortie1, ..., sortieN » les arguments de sortie. La forme générale est la suivante :

```
function [sortie1, sortie2, ..., sortieN] = nom_fonction(entree1, entree2, ..., entreeM)
% Commentaires
  INSTRUCTIONS
sortie1 = ... ;
...
sortieN = ... ;
```

Toute fonction doit se trouver dans un FICHIER DU MÊME NOM QUE LE NOM DE LA FONCTION. Une fonction de nom `nom_fonction` doit donc être enregistrée dans le fichier `nom_fonction.m` (on peut dans la version 7 définir plusieurs fonctions dans le même fichier `nom_fonction.m`, mais elles ne peuvent être appelées que depuis le fichier `nom_fonction.m` et pas depuis l'espace de travail ou un autre fichier).

Les commentaires doivent commencer à la 2^{ème} ligne juste après la définition de fonction (située sur la 1^{ère} ligne). Les commentaires doivent contenir une description de la fonction avec sa syntaxe, la date de réalisation, le nom de la personne qui a écrit le programme, *etc.*

On rappelle qu'avant de créer sa propre fonction, il est impératif de vérifier si elle n'existe pas déjà sous Matlab, en cherchant dans *Help / MATLAB Help / Index*. Vous pouvez aussi utiliser `exist` ou `lookfor` en ligne de commande pour rechercher des fonctions existant sous Matlab.

Créez la fonction `minmax` qui renvoie en sortie les éléments minimum et maximum d'un vecteur, et pour une matrice les éléments minimum et maximum de chaque colonne.

```
function [mini, maxi] = minmax(x)
% fonction [mini, maxi] = minmax(x)
% Cette fonction renvoie le minimum et le maximum d'un
% vecteur x.
% Si x est une matrice, mini et maxi sont les
% éléments minimum et maximum de chaque colonne de x.
%
% Auteur : Mézigue
% Date : 29/02/2006

mini=min(x);
maxi=max(x);
```

Les variables `x`, `mini` et `maxi` sont locales à la fonction. Elles n'existent que dans la fonction et n'ont aucun rapport avec les variables de l'espace de travail, qui donc ne sont pas modifiées. Les variables définies dans la fonction ne sont pas accessibles depuis l'espace de travail. Si l'on tape `maxi` (variable interne de la fonction `minmax`) dans la fenêtre de commande, on constate que cette variable n'est pas définie.

```
>> maxi
??? Undefined function or variable 'maxi'.
```

Exemple d'utilisation :

```
>> [m1,m2]=minmax(v);
```

Si l'on veut que le résultat s'affiche à l'écran :

```
>> [m1,m2]=minmax(v)
```

```
m1 =
    -1

m2 =
     1
```

On a souvent besoin de créer ses propres fonctions lorsque l'on programme avec Matlab. Il est **impératif de vérifier tout d'abord que le traitement que l'on désire effectuer n'existe pas déjà**. On peut chercher dans le *Help / MATLAB Help* via l'onglet *Index* ou *Search*.

Par exemple, si l'on veut faire une fonction qui calcule le plus grand élément d'un tableau, *largest element* en anglais, on peut rechercher les programmes qui ont à voir avec le terme *largest*. On constate qu'une fonction `max` existe, et il est donc impératif de l'utiliser plutôt que d'en programmer une autre !

Deuxième série d'exercices

A.11. Images et surfaces

A.11.1. Affichage d'images

En Matlab, toute matrice peut être considérée comme une image que l'on peut afficher avec `image` ou `imagesc`.

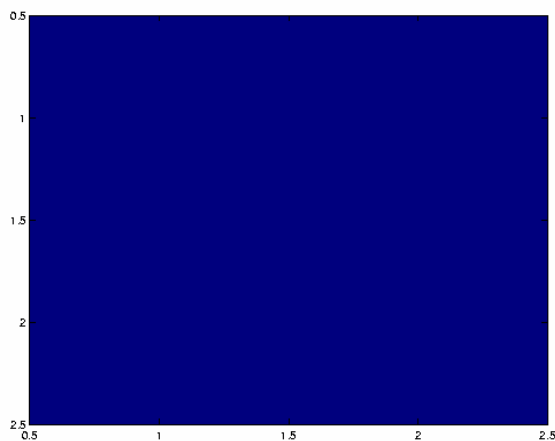
```
» Z=rand(2)
```

```
Z =
```

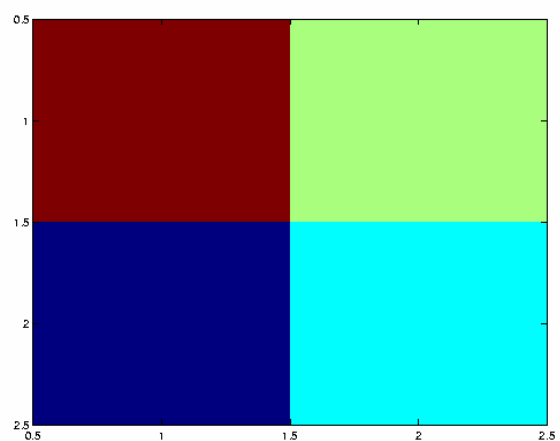
```
    0.9501    0.6068
    0.2311    0.4860
```

```
» figure(1)
» image(Z)
» imagesc(Z)
```

```
% Affiche une image
% Met à pleine échelle et affiche
```



`image(Z)`



`imagesc(Z)`

La fonction `image` utilisée avec des matrices affiche la matrice comme une image, en utilisant une échelle d'intensité correspondant à tous les entiers entre 0 et 255. On peut donc associer à cette échelle 256 niveaux de gris différents (`colormap('gray')`). On peut également associer à ces 256 valeurs des couleurs différentes, avec des tons chauds (`colormap('hot')`), ou froids (`colormap('cool')`), ou métalliques (`colormap('bone')`), ..., voir le `help graph3D` pour une liste complète. La fonction `imagesc` met les valeurs à afficher à pleine échelle (utilise toute l'échelle entre 0 et 255). Ainsi l'image de gauche correspond à des valeurs allant de 0 à 1 (valeurs de `Z`) représentées avec une échelle totale de 256 valeurs : on ne voit pas, à l'œil, de différence entre les quatre cadrans. Dans l'image de droite, la plus

petite valeur de z est associée à 0 et la plus grande à 255, et cette mise à l'échelle permet de visualiser les différentes valeurs.

A.11.2. Affichage graphique en 3D (surfaces)

Pour afficher des surfaces dans une fenêtre graphique, voir les fonctions de `help graph3d`. On ne présente ici que les fonctions les plus courantes.

On veut représenter sous forme d'une surface les valeurs prises par une fonction de deux variables x et y . Par exemple, on veut évaluer la fonction $z(x,y)$, en fonction des deux variables x et y , cette fonction étant définie par $z(x,y) = xy$. On veut calculer des valeurs de z pour $0 \leq x \leq 2$ et $-1 \leq y \leq 2$. En choisissant une grille de points espacés de 1, on devra évaluer z pour tous les couples de coordonnées possibles $(x,y) \in \{0, 1, 2\} \times \{-1, 0, 1, 2\}$. La fonction `meshgrid` permet d'obtenir ces points :

```

>> x=0:2; y=-1:2;
>> [X,Y]=meshgrid(x,y)
X =
    0    1    2
    0    1    2
    0    1    2
    0    1    2
Y =
   -1   -1   -1
    0    0    0
    1    1    1
    2    2    2

```

L'ensemble des couples de points $(x(i, j), y(i, j))$ pour $i=1, \dots, \text{length}(y)$ et $j=1, \dots, \text{length}(x)$ est généré grâce à `meshgrid`, et l'évaluation de la fonction peut se faire en une seule instruction matricielle :

```

>> Z=X.*Y;
>> mesh(X,Y,Z)
>> title('Fonction z=xy')

```

Soit la fonction $z(x,y) = \cos(\pi x) \sin(\pi y)$ que l'on veut évaluer sur une grille de points espacés de 0,1 avec $1 \leq x \leq 2$ et $-1 \leq y \leq 1$. On doit donc évaluer la fonction z pour tous les couples de coordonnées possibles $(x,y) \in \{1; 1,1; 1,2; \dots; 2\} \times \{-1; -0,9; \dots; 1\}$. Les instructions Matlab sont simples :

```

>> x=1:0.1:2;
>> y=-1:0.1:1;
>> [X,Y] = meshgrid(x,y);
>> Z=cos(pi*X).*sin(pi*Y);
>> mesh(X,Y,Z)

```

Pour avoir une surface pleine :

```

>> surf(X,Y,Z)
>> title('Fonction z=cos(\pi x) sin(\pi y)')
>> xlabel('x'), ylabel('y')

```

Comme pour l'affichage d'une matrice avec `image(sc)`, on peut changer la carte des couleurs (`colormap`) pour afficher en niveaux de gris, avec des tons chauds ou froids, ... voir dans le `help graph3d`.

```

>> colormap('bone')
>> colormap('gray')
>> colormap('pink')

```

La fonction `contour` permet d'afficher les lignes de niveau :

```

>> contour(X,Y,Z)

```

