Initiation Matlab 1

1 Introduction

Le nom MATLAB provient de MATrix LABoratory. L'objectif initial était de fournir un accès simplifié aux bibliothèques de fonctions des projets LINPACK et EISPACK (dédiées au calcul matriciel et à l'algèbre linéaire).

MATLAB est devenu un langage de référence pour l'analyse et la résolution de problème scientifique. Il intègre à la fois des solutions de calcul, de visualisation et un environnement de développement.

MATLAB a de nombreux avantages par rapport aux langages de programmation traditionnels (tel que le C/C++). Il permet le développement intéractif de part l'utilisation d'un langage interprété. La structure de données de base est le tableau ne nécessitant pas de dimensionnement. Il fournit de nombreuses fonctions préprogrammées regroupées en boîtes à outils (toolbox) pour de nombreux domaines (par ex : signal processing, statistics, control theory, optimization, ...).

De plus, MATLAB dispose d'un excellente documentation.

2 Démarrer Matlab

Lorsque vous lancez Matlab pour la première fois, l'écran ressemble à celui de la Figure 1. Le 'bureau' MATLAB est une fenêtre contenant d'autres sous-fenêtres. Les principaux outils disponibles depuis ce bureau sont :

- COMMAND WINDOW : invite de commande permettant de taper des instructions, d'appeler des scripts, d'exécuter des fonctions matlab.
- COMMAND HISTORY : historique des commandes lancées depuis l'invite de commande.
- WORKSPACE : il liste les variables en mémoire, il permet également de parcourir graphiquement le contenu des variables
- CURRENT DIRECTORY : un navigateur de fichier intégré à MATLAB pour visualiser le répertoire de travail courant et y effetuer les opérations classiques tel que renommer ou supprimer un fichier.
- le HELP BROWSER : un navigateur permettant de parcourir l'aide de MATLAB. L'aide est un outil précieux pour trouver les fonctions et apprendre leur fonctionnement (notamment le format des données à fournir en entrée ainsi que les valeurs renvoyées par la fonction).

Par la suite, il est conseillé de tester toutes les instructions précédées de `>>` dans la *command window*. Lorsque vous optenez une erreur, essayez d'en comprendre la signification. Avec un peu de pratique, vous verrez que les messages d'erreur sont en général explicites.

3 Matlab comme calculette

Comme tout langage de programmation, MATLAB dispose de fonctions de calculs mathématiques. Nous en voyons ici quelques exemples d'utilisation.

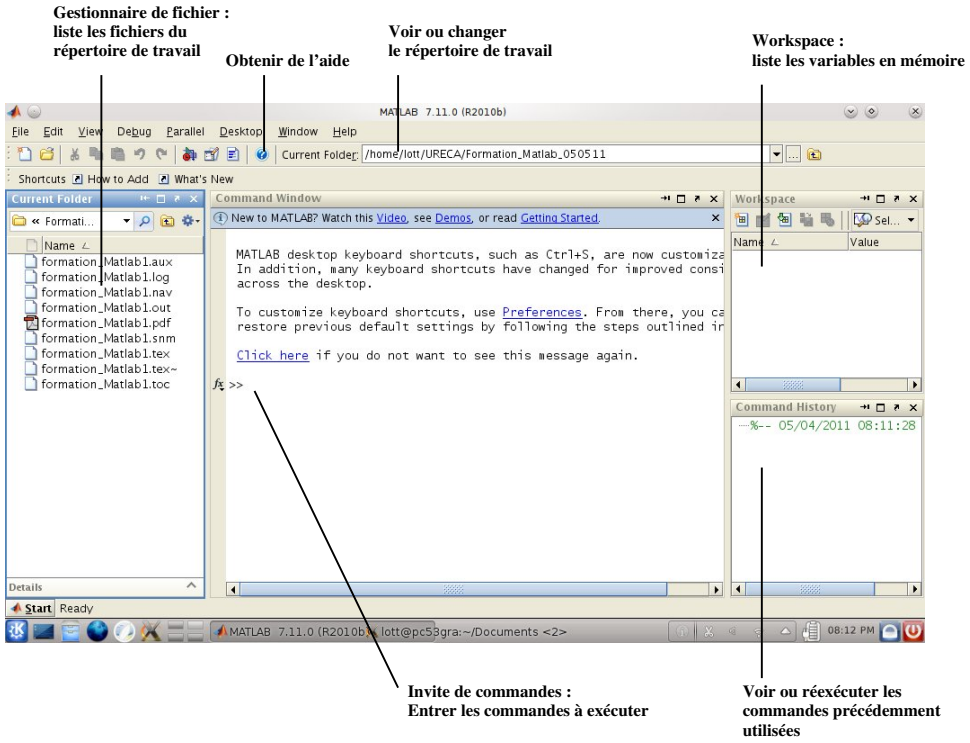

FIGURE 1 – L'interface graphique de l'environnement MATALB

+ (addition)	⇒	>> 3+2
- (soustraction)	⇒	>> 3-2
* (multiplication)	⇒	>> 3*2
/ (division)	⇒	>> 3/2
^ (puissance)	⇒	>> 3^2

Si l'on regarde ce que donne l'exemple d'addition :

```
>> 3+2
ans =
    5
```

On peut remarquer que lorsqu'aucune variable d'affectation n'est spécifiée pour stocker le résultat de l'opération, MATLAB stocke le résultat dans une variable appelée `ans` (diminutif pour `answer`). Cette variable sera écrasée à chaque nouvelle opération. Pour éviter cela, on peut spécifier le nom d'une variable pour stocker le résultat. On pourra alors réutiliser le résultat de l'opération plus tard. Par ex :

```
>>x = 3+2
x    =
    5
```

`x` prend alors la valeur `5`. Cette variable peut alors être réutilisée dans un calcul suivant :

```
>>x/2
ans =
    2.5000
```

3.1 Priorité des opérateurs

Les opérations sont évaluées en donnant la priorité aux opérateurs selon l'ordre suivant :

1. `()`
2. `^`
3. `* /`
4. `+ -`

Exemple 1 :

```
>> 3 + 2 * 4^2 → 35
           ---
            16
       -------
         32
```

Exemple 2 : `>>`$\underbrace{(\underbrace{(3+2)}_{5}*4)}_{20}$`^2` $\rightarrow 400$

A titre d'exercice, évaluer l'expression suivante

$$\frac{1}{12-6^2}+\frac{2}{3}$$

La réponse est `0.625`

4 Manipuler des variables

Une variable est un emplacement en mémoire permettant de stocker provisoirement une donnée. On réfère à l'emplacement en mémoire par le nom que l'on donne à la variable. Une variable en programmation n'a pas la même signification qu'en mathématiques.

On distingue plusieurs types de variable selon les données qu'elles servent à stocker (nombre, caractère alphanumérique, tableau, matrice, structure). Contrairement à d'autres langages de programmation, sous MATLAB le type des variable n'a pas besoin d'être spécifié, MATLAB infère le type d'une variable en fonction de la donnée que l'on y stocke.

Sous MATALB, les noms de variable doivent commencer par une lettre, sont sensibles à la casse (différenciation des caractères majuscule/minuscule) et ne peuvent contenir aucun caractère spécial excepté le tiret bas (_ , *underscore*). De même, il faut éviter d'utiliser comme nom de variable des noms déjà employés comme nom de fonctions (par ex : min, max, exp ...). MATLAB générera également une erreur si un des mots-clés réservés suivant est utilisé : `for`, `end`, `if`, `while`, `function`, `return`, `elseif`, `case`, `otherwise`, `switch`, `continue`, `else`, `try`, `catch`, `global`, `persistent`, `break`. Ces mots-clés font partie de la syntaxe du langage MATLAB, nous verrons dans la suite des exemples d'utilisation de certains de ces mots-clefs.

4.1 Utilisation d'une variable

Les variables sont créées lors de la première affectation.

A $\leftarrow$ 0 *on affecte la valeur 0 à la variable A, si A n'existait pas auparavant, elle est créée.*
B $\leftarrow$ 11 *on affecte la valeur 11 à la variable B*

On peut alors utiliser une variable dans un calcul ou pour affecter son contenu à une autre variable.

A ← B *on affecte la valeur contenue ds B à la variable A, A vaut à présent 11*
A ← A + 1 *on affecte la valeur contenue ds A incrémentée de 1 à la variable A, A vaut à présent 12*

Sous Matlab, l'affectation se fait à l'aide de l'opérateur `=`

```
>> A = 0
>> B = 11
>> A = B
>> A =  A + 1
```

4.2 Quelques types de variables : nombres, vecteurs (tableau 1D), matrices (tableau 2D)

- nombres :
`>>a = 0.66`
`>>deuxiemeNombre = 2/3`
`>>mon_nom_de_variable = -4`

- tableau 1D en ligne :
`>>rowvect1 = [1,2,3]` (on sépare les éléments par une virgule)
`>>rowvect2 = [a , mon_nom_de_variable , 12]`

- tableau 1D en colonne :
`>>colvect = [1;2;3]` (on sépare les éléments par un point virgule)

- matrice :
`>>maPremiereMatrice = [1,2,3;4,5,6;7,8,9] %matrice (3x3)`

Le caractère `%` permet de spécifier un commentaire dans le code : ce qui suit ne sera pas interprété par Matlab.

4.3 Accès aux éléments des tableaux

Accès à un élément unique :

Pour accéder à un élément unique d'une variable de type tableau 1D, on spécifie entre paranthèses après le nom de la variable l'indice de l'élément (l'indice peut être donner par le biais d'une variable) :

```
>>rowvect1(1) %1er élément
>>colvect(3) %3eme élément
>> ind = 2
>>rowvect1(ind)  (car ind = 2, ne fonctionne que si la variable utlisée contient un entier)
>>rowvect1(end) %dernier élément
```

Pour les tableaux à 2 dimensions, on spécifie entre paranthèses la position de l'élément dans le tableau en donnant en premier la ligne puis la colonne :

```
>>maPremiereMatrice(2,3) %élément de la 2eme ligne, 3eme colonne
```

Accès à un sous-ensemble :

Si l'on souhaite extraire un sous-ensemble d'un tableau, on spécifie pour chaque dimension un tableau des indices que l'on souhaite conserver. Pour un tableau 2D, on récupère alors l'intersection des lignes et colonnes spécifiées.

```
>>rowvect1([1,3]) %vecteur composé du 1er et 3eme élément
>>maPremiereMatrice([1,2],[2,3]) %matrice (2x2) extrait du coin supérieur droit de maPremiereMatrice
>>maPremiereMatrice(2,:) %2eme ligne de la matrice maPremiereMatrice
>>maPremiereMatrice(:,3) %3eme colonne de la matrice maPremiereMatrice
```

L'opérateur `:` (double point), lorsqu'utilisé pour accéder aux éléments d'un tableau, permet de conserver tous les indices de la dimension.

L'opérateur `:` permet également de créer un tableau 1D en ligne de la manière suivante :

`<debut>:<incrément>:<fin>`

si l'incrément n'est pas spécifié, il sera pris égal à 1.

Exemple de création de tableau avec `:` :
`>>D = 1:4`
`>>E = 1:0.1:2.5` (essayer avec 2.45 au lieu de 2.5)

Cette opération est utile pour :
— créer/initialiser des vecteurs
— accéder à des sous-ensembles de vecteurs ou matrices
`>>E(end-9:end) %10 derniers éléments de E`
— dans les boucles `for` (à venir)

5 Appeler/utiliser des fonctions de Matlab

Prototype d'une fonction type :

```
[out1,out2,...] = function_name(in1,in2,...)
```

- une fonction peut prendre des arguments d'entrée placés entre parenthèses après le nom de la fonction (ce sont les données à traiter).
- une fonction peut renvoyer une ou plusieurs valeurs de retour (ce sont le/les résultat(s) du traitement effectué(s) par la fonction)

Exemple 1 : la fonction `max` renvoie le maximum du vecteur donné en entrée, mais elle peut aussi renvoyer la position du maximum dans ce vecteur. Voir l'aide (`>>doc max`)

```
>>notes = [12 , 3 , 17 , 15 , 13  ]
>>max(notes) %affiche la note maximale
>>noteMax = max(notes) %stocke la note maximale ds la variable noteMax
>>[noteMax , ind] = max(notes) % ind contiendra la position de 'noteMax' dans 'notes'
```

Exemple 2 : la fonction `size` renvoie un vecteur contenant la taille selon chaque dimension de la variable passée en argument (voir l'aide `>>doc size`)

```
>>size(ind) %un scalaire est une matrice de dimension (1x1)
>>size(rowvect)
```

```
>>size(maPremiereMatrice)
>>dim = size(maPremiereMatrice) %sauvegarde du résultat ds la variable dim
>>[dim1 , dim2] = size(maPremiereMatrice)
```

Trouver une fonction et de l'aide sur son fonctionnement :

`>>lookfor fourier` (recherche de fonction par mot-clef, par exemple pour trouver la transformée de fourier discrète)
`>>help fft` (obtenir de l'aide sur la fonction fft en ligne de commande)
`>>doc fft` (ouvre un browser, aide plus détaillée. Le browser contient un moteur de recherche et des menus permettant de facilement trouver une fonction)

6 Ecrire des scripts M-files

Un fichier de script est un fichier externe contenant une suite d'instruction MATLAB. Les fichiers de script ont une extension de nom de fichier `.m`. Les M-files peuvent être des scripts qui exécutent simplement une suite d'instructions ou peuvent être des fonctions (nous verrons les fonctions plus loin).

Exemple : créer le script "test_script" (soit vous tapez `>>edit test_script.m`, soit vous faites 'File'->'New'->'Script' puis 'Save As' en spécifiant "test_script.m" comme nom) avec la suite d'instructions suivante :

```
clear all    %efface toutes les variables du workspace
a = 1
b = 2;
c = 3, d = 4;
e = a*b/(c+d),
scal = 11;
```

Sauvegardez puis exécutez le script (menu Debug->Save&Run ou Fleche verte ou F5). Observez la sortie sur la ligne de commande et conclure quant à l'utilisation des ; et ,

Il est également possible d'appeler le script depuis la ligne de commande, taper `>>test_script`

7 Programmer avec MATLAB

7.1 Opérateurs de comparaisons

Les opérateurs de comparaison sont :
`==` : égal à (`x == y`)
`>` : strictement plus grand que (`x > y`)
`<` : strictement plus petit que (`x < y`)
`>=` : plus grand ou égal à (`x >= y`)
`<=` : plus petit ou égal à (`x <= y`)
`~=` : différent de (`x ~= y`)

Le résultat d'une évaluation d'un test logique à l'aide d'opérateurs de comparaisons peut-être **Vrai** ou **Faux** qui sont respectivement représentés par les entiers 1 et 0 sous MATLAB (VRAI <==> 1 , FAUX <==> 0).

Exemple :

```
>>toto = 3 %on affecte à toto la valeur 3
>>titi = 4 %on affecte à titi la valeur 4
>>toto == 3 % égal à ?
```

```
>>toto == titi—
>>toto ~= 3 % différent de ?
>>toto ~= titi
>>toto > 3  % plus grand que ?
>>toto > titi
>>toto < 3  % plus petit que ?
>>toto < titi
>>toto >= 3 % plus grand ou égal à ?
>>toto >= titi
>>toto <= 3 % plus petit ou égal à ?
>>toto <= titi
```

7.2 Opérateurs logiques

Créer 4 variables booléennes pour tester les différents opérateurs

```
>>a = 1; b = 0; c = 1; d = 0;
```

Puis tester les 3 opérateurs logiques :

- l'opérateur `&` (ET logique),

```
>>a&c
>>a&b
>>b&d
>>b&a
```

Un ET logique entre 2 propositions renvoie VRAI si et seulement si les deux propositions sont vraies.

- l'opérateur | (OU logique),

```
>>a|c
>>a|b
>>b|d
>>b|a
```

Un OU logique entre 2 propositions renvoie FAUX si et seulement si les deux propositions sont fausses.

l'opérateur ~ (NON logique),

```
>>~a
>>~b
>>~b|a
>>~(b|a)
```

Exemple concret : on veut savoir si un individu a 25 ans révolus et mesure moins 180cm. Si le test `(age>25) & (taille<180)` retourne 1 les deux conditions sont vérifiées, si il retourne 0 au moins une des deux conditions est fausse.

```
>> age = 30; taille = 170; (age>25) & (taille<180)
>> age = 24; taille = 165; (age>25) & (taille<180)
>> age = 35; taille = 185; (age>25) & (taille<180)
>> age = 20; taille = 195; (age>25) & (taille<180)
```

A titre d'exercice, écrire l'équation logique permettant de savoir si un individu peut avoir accès au grand 8 du parc d'attraction. Un individu a accès au manège dans les cas suivants :
- il est majeur
- il a plus d'1m60

- il est accompagné d'un individu majeur.
Pour cela, créer 3 variables age , taille , accompagne et testé les différents cas de figure.

Les opérateurs de comparaison et les opérateurs logiques sont utilisés essentiellement dans les instructions de contrôle `if` et `while`.

7.3 Branchement conditionnel (IF ... THEN ... ELSE ...)

On a parfois besoin d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée au préalable. Différentes formes d'instruction conditionnée existent sous MATLAB.

— *Forme 1*

```
if <expression booléenne>
    <suite d'instructions exécutée si l'expression est VRAI>
end
```

— *Forme 2*

```
if <expression booléenne>
    <suite d'instructions 1 exécutée si l'expression est VRAI>
else
    <suite d'instructions 2 exécutée si l'expression est FAUSSE>
end
```

L'indentation n'est pas obligatoire, mais elle permet de rendre le programme plus lisible.

— *Forme 3*

```
if <expression booléenne 1>
    <suite d'instructions 1 exécutée si l'expression 1 est VRAI>
elseif <expression booléenne 2
    <suite d'instructions 2 exécutée si l'expression 1 est FAUSSE
                                et que l'expression 2 est VRAI>
.
.
.
else
    <suite d'instructions n exécutée si aucune des expressions n'est VRAI >
end
```

Exemple : écrire un script `pile_face.m` pour simuler un tirage à pile ou face

```
x = rand(); %renvoie un nombre aléatoire compris
% entre 0 et 1 selon une loi uniforme
if x > 0.5
    disp('pile')
else
    disp('face')
end
```

Exercice : écrire un script `solve_eq_2.m` réalisant la résolution d'une équation du second ordre $ax^2 + bx + c = 0$.

Connaissant a,b,c (variables qu'on pourra initialiser au début du script), trouver x dans l'ensemble des nombres réels.

Rappel :

- si $\Delta = b^2 - 4ac$ est supérieur à 0, alors il existe 2 solutions réelles, $x1 = \frac{-b-\sqrt{\Delta}}{2a}$ et $x2 = \frac{-b+\sqrt{\Delta}}{2a}$
- si Δ est nul, alors il existe 1 solution unique, $x = \frac{-b}{2a}$
- si Δ est inféreur à 0, il n'y a pas de solution ds l'ensemble des réels. On affichera alors 'pas de solution'.

Indication : la racine carré s'obtient avec la fonction `sqrt` ou en élevant à la puissance $\frac{1}{2}$

7.4 Boucle for

La boucle `for` répéte une suite d'instruction un nombre prédéterminé de fois. Sa structure est la suivante :

```
for var = <list-of-values>
    <suite d'instruction>
end
```

La boucle `for` va executer la `<suite d'instruction>` pour chaque élément de la `<list-of-values>` en affectant l'élément à la variable `var`.

Exemple 1 :

```
i = 0; for k = 0:0.1:1
    i = i + 1;      disp(['Iteration ',num2str(i),', k vaut ',num2str(k)]);
end
```

En général, la liste de valeurs sert à indexer un vecteur (ou matrice), on doit alors se limiter à des valeurs entières.

Exemple 2 : (on rajoute plus 2 à tout le monde) :

```
note = [ 4 , 12 , 7 , 15 , 12 ]
for i = 1:length(note) %voir l'aide de length
    note(i) = note(i) + 2;
end
```

Exemple 3 : calcul de moyenne sur un vecteur. Ecrire le script suivant

```
note = [ 4 , 18 , 7 , 15 , 12 ]
moyenne = 0;
for i = 1:length(note)
    %on fait la somme des notes
    moyenne = moyenne + note(i);
end
%on divise la somme des notes par le nombre de notes
moyenne = moyenne / length(note)
```

Il existe bien entendu dans Matlab des fonctions permettant le calcul direct d'une moyenne, ce n'est qu'un exemple. Pour info, `>>mean(note)`

Exercice : Écrire un script qui calcule le maximum d'un vecteur

7.5 Boucle while

Il arrive que nous souhaitions répéter une suite d'instructions jusqu'à qu'une condition soit satisfaite. Si l'on ne connait pas le nombre d'itérations nécessaire à l'avance, une boucle `while` est préférable par rapport à une boucle `for`

```
while <expression booléenne>
    <suite d'instructions>
end
```

La `<suite d'instructions>` va être répétée tant que l'`<expression booléenne>` est vrai, ou dit autrement jusqu'à ce que l'`<expression booléenne>` soit fausse.

Exemple 1 : on cherche le premier élément négatif d'un vecteur

```
vec = [1,1,1,1,1,1,-1,1,-1]
i = 1;
while vec(i) >= 0
    i = i+1;
end
i %indice du premier élément négatif
vec(i) %premier élément négatif
```

Réessayer avec `vec = [1,1,1,1,1,1,1,1,1]`, que ce passe-t-il ?

Modifions le script pour prendre en compte ce cas limite :

```
vec = [1,1,1,1,1,1,-1,1,-1]
i = 1;
while  (i <= length(vec)) && (vec(i) >= 0)
    i = i+1;
end
if i <= length(vec)
   i %indice du premier élément négatif
   vec(i) %premier élément négatif
else
   disp('Tous les éléments sont positifs')
end
```

Réessayer avec vec = [1,1,1,1,1,1,1,1,1]

7.6 Instructions break

L'instruction `break` : provoque l'arrêt de la première boucle `for`, `while` englobante.

L'instruction `for` ci-dessous est stoppée au premier i tel que t(i) est nul :

```
t = -2:10;
for i = 1:length(t)
   if t(i) == 0
      break;
   end
end
```

7.7 Instructions continue

L'instruction `continue` : dans une instruction `for`, `while`, l'instruction `continue` provoque l'arrêt de l'itération courante, et le passage au début de l'itération suivante.

Supposons que l'on parcoure un tableau t pour réaliser un certain traitement sur tous les éléments, sauf ceux qui sont négatifs :

```
for i = 1:length(t)
  if (t(i) < 0 )
      continue;   %   on passe au i suivant dans le for
  end
  ...             %   traitement de l'élément courant
end
```

Numéros	Element du M-file	Description
(1)	Ligne de définition de fonction	Définie le nom de la fonction ainsi que le nombre et l'ordre des arguments d'entrée et de sortie
(2-5)	Texte d'aide	Description du programme qui sera affichée lorsqu'on demandera de l'aide sur la fonction
(6+)	Corps de la fonction	Code de la fonction réalisant les calculs nécessaires.

TABLE 1 – Détail des éléments constituant une fonction MATLAB

8 Création d'une fonction

Nous avons évoqué précédemment l'utilisation de fonctions MATLAB préprogrammées telle la fonction `max()` et `size()`. Tout comme pour les scripts, il est possible de créer ses propres fonctions au sein d'un M-file. Contrairement aux scripts, les fonctions peuvent accepter des données en entrée et renvoyer des données en sortie. Chaque fonction a son propre espace en mémoire et ne voie pas les données du workspace. Seules les données qu'on aura spécifié comme arguments d'entrée seront utilisables par la fonction.

A titre d'exemple, nous allons créer une fonction de calcul de moyenne. Ds le répértoire courant, créer le fichier `moyenne.m` (`>>edit moyenne.m`) contenant le code suivant :

```
function [mu,minv,maxv] = moyenne( tab )                              (1)
%MOYENNE(TAB) calcul de moyenne sur un tableau 1D                     (2)
%Pour un tableau 'tab' à 1 dimension [mu,minv,maxv] = moyenne( tab )  (3)
%retourne la moyenne 'mu', le minimum 'minv' le maximum 'maxv' des    (4)
%éléments de 'tab'.                                                   (5)
mu = 0;                                                               (6)
for i = 1:length(tab)                                                 (7)
  mu = mu + tab(i);                                                   (8)
end                                                                   (9)
mu = mu / length(tab);                                               (10)
minv = min(tab);                                                     (11)
maxv = max(tab);                                                     (12)
```

La première ligne d'une fonction commence avec le mot-clé `function`. Cette ligne donne le nom de la fonction ainsi que la liste des arguments d'entrée et de sortie. Dans le cas de cette fonction, elle ne prend qu'un seul argument en entrée mais peut renvoyer de 1 à 3 valeurs. Il est possible mais pas obligatoire de rajouter en commentaires sous la première ligne de définition de la fonction un descriptif de ce que fait la fonction. Ces lignes en commentaires seront affichés lorsqu'on demandera de l'aide sur la fonction. Il est important de noter que le nom du fichier `.m` contenant notre fonction doit être formé du nom de la fonction suivi de l'extension `.m` . Pour notre exemple de calcul de moyenne, le fichier devra s'appeler `moyenne.m`.

Le tableau 1 donne un récapitulatif des éléments constituants une fonction MATLAB.

- Créer un vecteur (par ex, `>>note = [2,18,5,15]`) et appeler la fonction `moyenne()` des manières suivantes :

```
>>moyenne(note)
>>a = moyenne(note)
>>[a,b] = moyenne(note)
>>[a,b,c] = moyenne(note)
```

- Essayer d'accéder à l'aide (`>>help moyenne`)

Exercices :

— Ecrivez une fonction de conversion de degrés Fahrenheit vers des degrés Celsius. (Rappel : $C = \frac{5}{9}(F - 32)$)

— Ecrivez une fonction prenant 2 arguments d'entrée et 2 arguments de sortie qui permettra de déterminer la taille en cm et le poids en kg d'une personne à partir de sa taille en pouces (1 in = 2.54 cm) et son poids en livres (1 lb = 0.453 kg).

9 Graphique 2D

9.1 Fonction élémentaire

On souhaite afficher la fonction $y = sin(3\pi x)$ sur $0 < x < 1$. Pour cela, on évalue un certain nombre de points sur l'intervalle.

```
>>N = 10; h = 1/N; x=0:h:1
>>y = sin(2*pi*x)
```

Pour afficher les données sur un graphique, on utlise la fonction `plot()`. Elle prend un ou 2 tableaux 1D en entrée. Si utilisée avec 2 tableaux, ceux-ci doivent être de même taille.

```
>>figure(1), plot(y)
>>figure(2), plot(x,y)
```

Comparez les figures 1 et 2, et conclure quant à l'utilisation d'1 ou 2 tableaux.

La fonction `figure` permet de créer un nouvelle fenêtre pouvant contenir des graphes

Il peut être utile de fermer toutes les fenêtres de graphique en début de script. Pour cela, on appelle la fonction `close all`

Effectuer un nouvel affichage pour `N = 100`

9.2 Titres, labels et grille

Pour rajouter un titre et des labels d'axes à notre figures, on utilise

```
>>title('Graph de y = sin(3 pi x)')
>>xlabel('Axe x')
>>ylabel('Axe y')
```

Une grille en pointillés peut égalment être rajoutée

```
>>grid on %affichage de la grille
>>grid off %masquage de la grille
```

9.3 Styles de ligne et couleurs

Par défaut, les lignes sont bleues en trait plein. Il est possible de modifier le style et la couleur de la ligne en rajoutant un 3ème argument à la fonction `plot()`. La figure 9.3 présente les couleurs et styles de ligne disponibles ainsi que les codes associés. Par exemple, pour afficher notre courbe avec un point vert par donnée, nous utiliserons l'option `'g.'`.

```
>>plot(x,y,'g.')
```

Colours		Line Styles	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

9.4 Plusieurs courbes sur le même graphique

Un nouvel appel de la fonction `plot` efface le graphique précédent. Ce n'est pas souhaitable si l'on souhaite afficher plusieurs courbes sur le même graphe. Pour empécher matlab d'effacer le graphe, on utilise `hold on`

```
>>plot(x,sin(3*pi*x),'r-'), hold on
>>plot(x,cos(3*pi*x),'kx')
>>title('Plusieurs courbes'), xlabel('Axe x'),ylabel('Axe y')
>>legend('Sinus','Cosinus')
```

Une légende peut être affichée donnant la liste des styles de lignes suivie de la description spécifiée par l'utilisateur

9.5 Affichage de plusieurs graphes côte à côte (subplot)

Une fenêtre d'un graphique peut être divisée en sous-fenêtres suivant un tableau de dimension (mxn). On peut alors afficher dans chaque sous-fenêtre une ou plusieurs courbes. Les fenêtres sont indexées de 1 à $m \cdot n$ en ligne en partant du coin supérieur gauche. (`subplot(2,2,k)`, pour la k ième fenêtre d'un tableau (2x2)). Les instructions `hold` et `grid` fonctionnent sur le subplot courant

Exemple :

```
>>subplot(2,2,1), plot(x,sin(3*pi*x))
>>    ylabel('sin 3 pi x')
>>subplot(2,2,2), plot(x,cos(3*pi*x))
>>    ylabel('cos 3 pi x')
>>subplot(2,2,3), plot(x,sin(6*pi*x))
>>    ylabel('sin 6 pi x')
>>subplot(2,2,4), plot(x,cos(6*pi*x)), hold on, plot(x(1:10:end),cos(6*pi*x(1:10:end)),'r.')
>>    ylabel('sin 6 pi x')
```

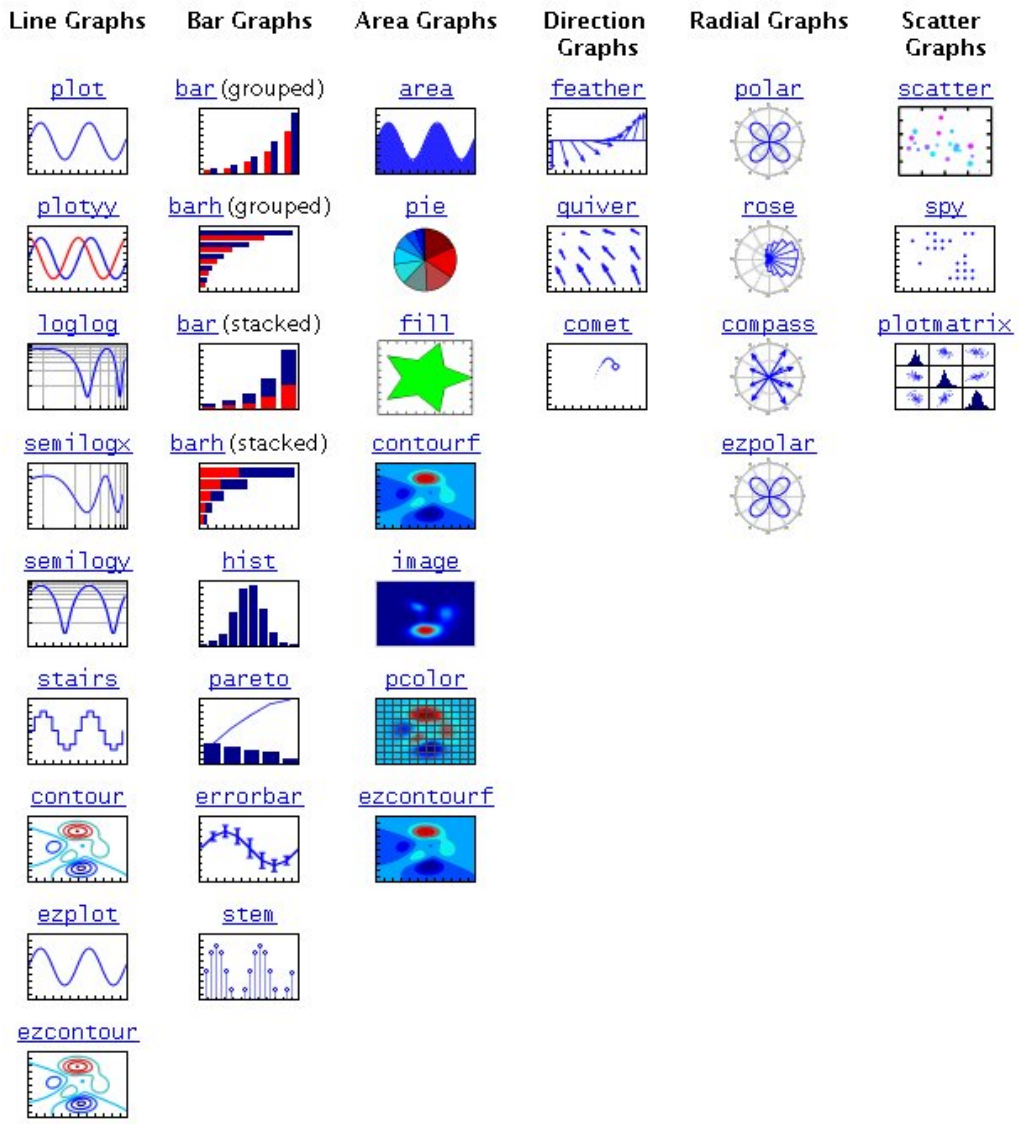

FIGURE 2 – Liste des fonctions graphiques 2D