1 Introduction

1.1 Langage de programmation

Un langage de programmation est un code de communication entre un humain et une machine (en générale un ordinateur) à l'aide de ces langages on peut écrire des programmes.

1.1.1 C'est quoi un programme

Liste d'instruction

↓

écrire (par vous)

↓

dans un fichier ou plusieur

↓

déstinées à être exécutés par l'ordinateur

1.1.2 Étapes de construction d'un programme

Problème → Algorithme → Transcription → Programme

↓ ↓

Manuel ↓

↓ ↓

Solution ← ← Exécution ← ← Ordinateur

Exemple :

PHP, C++, Java, Python, MATLAB, PASCAL, C ...

1.2 Pourquoi MATLAB?

MATLAB est une abréviation de MATrix LABoratory, elle est éveloppé par la société "The MathWorks". MATLAB est un système interactif de programmation scientifique, pour le calcul numérique et la visualisation graphique, basé sur la représentation matricielle des données. Elle peut s'interfacer avec d'autres langages comme le C, C++, Java et Fortran. Elle est une langage interprété (pas de compilation).

1.2.1 Avantages

1. Facilité d'utilisation, prise en main rapide.
2. Existence de toolboxes utiles pour l'ingénieur.
3. Possibilité de l'interfacer avec d'autres langages comme le C, C++, Java et Fortran.
4. Permet de faire du calcul parallèle.

1.2.2 Inconvinients

1. Limitation en mémoire.
2. Payant.

L'objectif de ce support de cours est double : la connaissance de ce logiciel est en soi indispensable parce qu'il est de plus en plus utilisé dans l'industrie et les banques pour développer des prototypes de logiciels et la mise en pratique des algorithmes d'analyse numérique étudiés.

2 Notions de base

2.1 Démarrage d'une session MatLab

Le lancement de MatLab sur un PC se fait tout simplement en l'appelant avec la commande MatLab dans une fenêtre. Cette fenêtre deviendra alors la fenêtre de commande MatLab. Sous Windows, il suffit de cliquer sur l'icône correspondante pour qu'une fenêtre similaire s'ouvre dans votre environnement. Il s'agit en effet d'une interface en mode texte avec l'interpréteur de langage MatLab dans laquelle on peut exécuter des instructions spécifiques ou lancer des programmes. L'affichage des résultats numériques se fera en mode texte dans cette même fenêtre ou dans une autre fenêtre en mode graphique (en faisant appel aux fonctions de visualisation de MatLab).

2.2 Environnement de MATLAB

MATLAB affiche au démarrage plusieurs fenêtres. Selon la version on peut trouver les fenêtres suivantes :

1. **Current folder** : indique le répertoire courant ainsi que les fichiers existants.

2. **Workspace** : indique l'espace courant de travail. Il liste les variables existantes avec leurs types et valeurs.

3. **Command History** : indique les commandes tapées. Historique des commandes lancées depuis l'invite de commande.

4. **Command Window** : indique l'invite de commande, utilisée pour formuler nos expressions et interagir avec MATLAB. C'est la fenêtre que nous utilisons tout au long de ce chapitre.

2.3 Variable

MATLAB utilise un langage interprété. Chaque expression écrite est interprétée et évaluée. La syntaxe est généralement de la forme:

2.3.1 Expression

Les expressions seront á exécuter dans la fenêtre de commande après les >>. Toutes les instructions en ligne s'éxécuteront après avoir tapé sur la touche Entrée ou Return.

>> expression;

Exemple

```
>> 5 + 5
ans =
      10
>> pi
ans =
      3.1416
```

tout les éléments de l'expression doivent être connu au moment de son évaluation par l'interpreteur.

2.3.2 Instructions

Est un ensemble structuré d'expressions ou effectation.

>> variable = expression;

Exemple

```
>> a = 10
a =
      10
>> a = a + ans
a =
      13.1416
```

Afin de cacher le résultat d'une commande, mettez un point-virgule ";" à la fin de la commande :

```
>> a = 10;
```

Si vous oubliez le point-virgule, ou si l'exécution d'un programme est trop longue, vous pouvez arrêter l'exécution en faisant Control C dans la fenêtre de commande.

NB :

- Les noms de variable doivent commencer par une lettre, et sont sensibles à la différenciation des caractères (majuscule/minuscule) et ne peuvent contenir aucun caractère spécial exceptée le tiret bas.

- 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-clefs réservé suivant est utilisé : for, end, if, while, function, return.

2.3.3 Opérateurs standards et caractères spéciaux

2.3.3.1 Les opérateurs arithmétiques

Les opèrations arithmètiques de base sont résumées comme suite

a + b	addition
a – b	soustraction
a / b	division
a * b	multiplication
^	puissance

2.3.3.2 Les opérateurs logiques

a & b ou and(a,b)	et logique
a | b ou or(a,b)	ou logique
~a ou not(a)	négation logique
false	valeur logique de faux
true	valeur logique de vrai

2.3.3.3 Les opérateurs de comparaison

a == b	égalité
a ~= b	inégalité
a > b	supérieur
a >= b	supérieur ou égal
a < b	inférieur
a <= b	inférieur ou égal

2.3.3.4 Les caractères spéciaux

MATLAB fournie une séries des caractères spéciaux :

- ( ) parenthèses
- = affectation
- , virgule
- ; point virgule
- % commentaire
- : jusqu'à

2.3.4 variables prédéfinies

MATLAB fournie un ensemble des constants prédéfinis :

ans	résultat de la dernière évaluation
i, j	le nombre imaginaire (racine carré de -1)
pi	$\pi = 3.1415...$
exp(1)	$e = 2.7183...$
eps	$\varepsilon = 2.2204e-016 \approx 2 \times 10^{-16}$
Inf	nombre infini
realmax	la valeur maximale absolue des réels
realmin	la valeur minimale absolue des réels
NaN	Not a Number 0/0

```
>> i
ans =
      0 + 1.0000i
>> pi
ans =
      3.1416
>> eps
ans =
      2.2204e-016
>> inf
ans =
      Inf
>> realmax
ans =
      1.7977e+308
```

```
>> realmin
ans =
       2.2251e-308
```

2.4 Programmer avec MATLAB

2.4.1 Fichiers scripts

Il est généralement commode d'écrire l'ensemble des instructions que l'on veut exécuter dans un fichier que l'on appelle un script que l'on va sauvegarder. Le nom du fichier pour la sauvegarde devra avoir le suffixe **.m**. Une fois sauvegardé, on pourra exécuter le script en tapant le nom du fichier (sans le suffixe). Remarquez que les variables à l'intérieur d'un script sont des variables globales auxquelles vous pouvez accéder dans la fenêtre de commande une fois le script exécuté.

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 = 7;
c = 3, d = 4;
e = a*b/(c+d)
```

⇒

```
>> test_script
e =
       1
```

Il s'agit de sauvegarder puis exécuter le script (menu Debug − > Save&Run ou Fleche verte ou F5). La sortie est affichée sur la ligne de commande.

2.4.2 M-files functions

Lorsque l'on fait appel à un même algorithme plusieurs fois en ne changeant que les variables d'entrée/sortie, il s'avère utile de pouvoir créer des fonctions. Pour créer une fonction, on ouvre un nouveau fichier dans lequel une fonction aura la syntaxe suivante :

```
function [out1,out2,...]=nomfonction(input1,input2,...)
    statements
end
```

où **input1,input2** sont les variables d'entrées et **out1,out2** de sortie. Le fichier où se trouve la fonction doit porter le nom **nomfonction.m**. Regardons l'exemple proposé par l'aide MATLAB :

```
function[s,p]=sommeproduit(a,b)
   s=a+b;
   p=a*b;
end
```

⇒

```
>> [s,p]=sommeproduit(4,5)
s=
     9
p=
     20
```

qui doit donc être sauvegardé dans un fichier appelé **sommeproduit.m**. Les variables utilisées à l'intérieur d'une fonction sont des variables locales, c'est-à-dire qu'elles n'existent que lors de l'appel de fonction et sont inaccessibles à partir de l'espace principale. Si une variable de même nom existe déjà en mémoire, en tant que variable globale, il n'y aura pas d'interférence entre ces deux données. Notons que pour les scripts par contre, les variables restent dans l'espace principal.

2.5 Commandes

Toutes les commandes sont en minuscules et en anglais. Lorsque l'on entre une commande, MATLAB affiche systématiquement le résultat de cette commande dans cette même fenêtre car Matlab est un langage interprété, c'est à dire qu'il exécute directement les commandes qu'on entre dans la fenêtre de commandes.

2.5.1 Commandes d'environnement

MATLAB garde en mémoire les variables qui ont été créées. On les voit en haut, à gauche, lorsque MATLAB dispose d'une interface graphique. On outre, on peut les afficher et les effacer par la ligne de commande :

who	donne la liste des variables présentes dans l'espace de travail.
whos	donne la liste des variables présentes dans l'espace de travail ainsi que leurs propriétés.
what	donne la liste des fichiers (.m) et (.mat) présents dans le répertoire courant.
clear var1... varn	efface les variables var1... varn de l'espace de travail.
clear	efface toutes les variables crées dans l'espace de travail.
exist var	vérifie si une fonction ou une variable existe dans le workspace

```
>> a=5 + 5
a =
     10
>> pi
ans =
     3.1416
```

```
>> who
Your variables are:
    a ans
>> whos
     Name    Size    Bytes Class Attributes
     a       1x1     8     double
     ans     1x1     8     double
>> what
    MATLAB Code files in the current folder
    C:\Users\omari\Documents\Research\Matlab\GUI

     GLCSimulation   file1   salesman   test1
     LEACH           ps      substr
>> clear a ans
>> who

>> exist a
ans =
    0
>> a = 10 ;
>> exist a
ans =
    1
```

- La commande clc permet d'effacer le contenu de la fenêtre de commande.

- Le symbole % dans une ligne a pour effet que le reste de la ligne ne sera pas exécuté ; ceci permet d'insérer des commentaires.

```
>> la = 10 % largeur
la =
    10
>> lo = 20 % longueur
lo =
    20
>> su = la * lo % surface = largeur * longueur
su =
    200
```

- Si une commande ne peut être écrite sur une seule ligne, il suffira d'ajouter à la fin de la ligne au moins trois points '...' et MATLAB concaténera cette ligne et la suivante (jusqu'à un maximum de 1024 caractères).

```
>> s = 1/2 + 2/3 + 3/4 + 4/5 + 5/6 ...
+ 7/8 + 8/9 + 9/10 + 10/11 + 11/12 ...
+ 12/13 + 13/14 + 14/15 + 15/16 + ...
16/17 + 17/18 + 18/19 + 19/20
s =
      15.5451
```

- Les commandes save et load permettent d'écrire (charger) toutes les variables du workspace dans le fichier matlab.mat. Si un nom de fichier est spécifié (save myfile ou load myfile) l'espace de travail est sauvegardé (chargé) conformément.

```
>> who
Your variables are:
     a ans la lo s su
>> save
Saving to:
C:\Users\omari\Documents\Research\Matlab\GUI\matlab.mat
>> clear
>> who
>> load
Loading from: matlab.mat
>> who
Your variables are:
     a ans la lo s su
```

2.5.2 Commande help

Etant donné le trés grand nombre d'instructions utilisables, il est impossible de mémoriser chacune d'elle avec sa syntaxe correspondante. Il est donc essentiel d'utiliser l'aide. Pour trouver l'aide associée à une instruction, il vous suffit de taper **help commande**. Par exemple, taper :

```
>> help sin
SIN Sine.
SIN(X) is the sine of the elements of X.
See also asin, sind.
Reference page in Help browser
doc sin
```

helpwin ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations.

2.5.3 Fonctions mathématiques

Parmi les fonctions fréquemment utilisées, on peut noter les suivantes :

sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian)
sqrt(x)	la racine carrée de x $\sqrt{x}$
abs(x)	la valeur absolue de x $\|x\|$
exp(x)	e^x
log(x)	logarithme naturel de x $ln(x) = log_e(x)$
log10(x)	logarithme de la base 10 de x $ln(x) = log_{10}(x)$
ceil(x)	la valeur entière supérieure
floor(x)	la valeur entière inférieure
round(x)	la valeur entière la plus proche de x
mod(x, y)	le reste de la division entière de x sur y
real(c)	la partie réelle d'un nombre complexe
imag(c)	la partie imaginaire d'un nombre complexe
conj(c)	le conjugué d'un nombre complexe
angle(c)	l'argument d'un nombre complexe
abs(c)	le module d'un nombre complexe

3 Les nombre en Matlab

3.1 Gestion des données

MATLAB gère les nombres entiers, réels, complexes, les chaînes de caractères. Il est inutile de déclarer préalablement le type de la variable que l'on manipule il suffit simplement d'assigner une valeur au nom de la variable avec le caractère "=".

```
>> a = 1
a =
     1
>> b = 1.02
b =
     1.0200
>> x = 1.45e4
x =
     14500
>> c = 1 + 2.4i
c =
     1.0000 + 2.4000i
```

Remarques : Pas de différens types pour les nombres (Matlab fait lui même les conversions). Pas de déclaration de variables.

On peut avoir à lire les variables avec le format spécifié jusqu'à épuisement du fihier. (Le nombre de variables à lire et le nombre de données dans le fichier doivent correspondre.)

3.1.1 Input(Entrée) / Output(Sortie)

input(' ') : introduire une ou des variable par l'intermédiaire de clavier.

x=input('text');

chaine_caractère = input('text','s');

Exemple :

```
a=input('donner la valeur de a :');
b=input('donner la valeur de b :');
s=a+b;
s
```

⇒

```
>> somme
donner la valeur de a : 4
donner la valeur de b : 6
s =
      10
```

disp(' ') : permet d'afficher un tableau de valeurs numériques ou de caractére.

disp(variable)

disp(['un message :', num2str(variable)])

Exemple :

```
a=input('donner la valeur de a :');
b=input('donner la valeur de b :');
s=a+b;
disp(s)
disp(['la somme est :', num2str(s)])
```

⇒

```
>> somme
donner la valeur de a : 4
donner la valeur de b : 6
10
la somme est : 10
```

3.1.2 sprintf(' ')

Si on veut un format plus lisible, en particulier, afficher plusieurs variables sur la même ligne, on peut Utiliser la commande **fprintf**.

sprintf(format, variables)

3.1.2.1 Modèle d'édition de caractères

Lit les données avec le format spécifiée.

format = % L s

où : % est le symbole de début de format.

L est un entiers donnant la longeur total du champ (en nombre de caractères).

s est le symbole précisant que la donnée est de type chaine da caractère.

Exemple :

```
>> sprintf('% s', 'Bonjour')
ans =
      Bonjour
```

3.1.2.2 Modèle d'édition de réels

Lit les données avec le format spécifiée.

format = % L.D t

où : L est un entiers donnant la longeur total du champ.

D est le nombre décimales à afficher.

t spécifié le type de notation utilisée, c'est-à-dire :

t=d pour un entier

t=f pour un réel

Exemple :

```
>> a=1.5;
>> b=2;
>> sprintf('a = %f et b= %d',a,b);
a =1.5
b=2
```

3.1.3 Format d'affichage

Par défaut, MATLAB affiche les valeurs numériques réelles sous format de point fixe à 5 chiffres.

On peut changer la façon dont les valeurs numériques sont affichées à comme suit:

format short point fixe, 5 chiffres (aussi short g)

format long point fixe, 15 chiffres (aussi long g)

format short e point flottant, 5 chiffres

format long e point flottant, 15 chiffres

format rational format rationnel

```
>> format short g
>> 15/7
ans =
      2.1429
>> format long g
>> ans
ans =
      2.14285714285714
>> format short e
>> ans
ans =
      2.1429e+00
```

```
>> format long e
>> ans
ans =
      2.142857142857143e+00
>> format rational
>> ans
ans =
      15/7
```

3.2 Les Coordonnées Polaires

Un complexe est généralement représenté sous la forme algébrique ou cartésienne.

Un complexe peut également être représenté sous la forme polaire $\rho e^{i\theta}$, où ρ est son module et θ son argument.

```
>> c = 2 + 2i        % forme algébrique ou cartésienne
c =
      2.0000 + 2.0000i
>> theta = angle(c)        % angle de c
theta =
      0.7854
>> ro = abs(c)        % module de c
ro =
      2.8284
>> ro*exp(theta*i)        % forme polaire
ans =
      2.0000 + 2.0000i
>> [theta, ro] = cart2pol(2,2)        % conversion au polaire
theta =
      0.7854
ro =
      2.8284
>> [x y] = pol2cart(theta, ro)        % conversion au cartésien
x =
      2.0000
y =
      2
```

3.3 Instructions de contrôle

Comme dans la plupart des langages il existe des instructions de contrôle de la forme for, while ou if. On notera ici qu'en raison du caractère interprété du langage il faut, dans la mesure du possible, éviter de les utiliser et les remplacer par la notion de boucle implicite que l'on verra plus loin.

3.3.1 For

La boucle for parcourt un vecteur d'indices et exécute à chaque pas toutes les instructions délimitées par l'instruction end. La syntaxe est de la forme:

for compteur = début : pas : fin
expression,
expression;
end

Exemple

```
>>x=1; for k=1:4,x=x*k, end
x =
     1
x =
     2
x =
     6
x =
     24
```

3.3.2 While

Il arrive que nous souhaitions répéter une suite d'instructions jusqu'á 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. La boucle while exécute une suite de commandes jusqu'à ce qu'une condition soit satisfait. La syntaxe est de la forme:

while relation
expression;
end

Exemple

```
>>x=1
while x<14
   x=x+5
end
x =
      6
x =
      11
x =
      16
```

3.3.3 If

La syntaxe est de la forme:

if relation	**if relation**	**if relation**
expression;	**expression;**	**expression;**
end	**else**	**elseif relation**
	expression;	**expression;**
	end	**else**
		expression;
		end

Exemple

```
if moy >= 16
      mention = "très bien";
elseif moy >= 14
      mention = "bien";
elseif moy >= 11
      mention = "Assez bien";
elseif moy >= 10
      mention = "Passable";
else
      mention = "Ajourné(e)";
end
```

4 Vecteurs et Matrices

Les vecteurs et matrices peuvent être construits directement selon la syntaxe suivante :

- Ils sont délimités par des crochets,
- Les éléments sont entrées ligne par ligne,
- Les éléments appartenant à la même ligne sont séparés par des virgules ou des espaces,
- Les différentes lignes doivent comporter le même nombre d'éléments et sont séparées par des points-virgules.

4.1 Les Vecteurs

Par défaut, le vecteur est une ligne à plusieurs colonnes.

Un vecteur ligne peut être créé par énumération de ces valeurs (coordonnées) séparées par des espaces ou des virgules:

```
>> v = [1 2.3 4 5]
v =
    1.0000   2.3000   4.0000   5.0000
>> v = [1, 2.3, 4, 5]
v =
    1.0000   2.3000   4.0000   5.0000
```

Un vecteur ligne peut être aussi créé par description de la valeur initiale, l'incrément, et la valeur finale :

```
>> v = [1 : 1 : 10]
v =
    1   2   3   4   5   6   7   8   9   10
```

La fonction linspace(i, j, k) génère un vecteur de k éléments allant de i à j :

```
>> v = linspace(1, 10, 10)
v =
     1   2   3   4   5   6   7   8   9   10
```

Un vecteur colonne peut être créé par énumération de ces valeurs (coordonnées) séparées par des points-virgules:

```
>> v = [1 ; 2.3 ; 4 ; 5]
v =
    1.0000
    2.3000
    4.0000
    5.0000
```

Un vecteur colonne peut être obtenu à partir d'un vecteur ligne en utilisant les opérations « trans-conjugué » (') et « transposé » (.'):

```
>> v = [1 2.3 4 5]
v =
    1.0000   2.3000   4.0000   5.0000
>> v' % transposé et trans-conjugé
ans =
    1.0000
    2.3000
    4.0000
    5.0000
>> w = [1 2.3 4+i 5]
w =
    1.0000   2.3000   4.0000 + 1.0000i   5.0000
>> w' % trans-conjugué
ans =
    1.0000
    2.3000
    4.0000 - 1.0000i
    5.0000
```

```
>> w.' %transposé
ans =
      1.0000
      2.3000
      4.0000 + 1.0000i
      5.0000
```

Les fonctions zeros et ones initialisent des vecteurs lignes (zeros(1, k) et ones (1, k)) ou colonnes (zeros(k, 1) et ones(k, 1)) par des zéros ou des uns :

```
>> v = zeros(1,5)
v =
     0   0   0   0   0
>> v = zeros(5,1)
v =
     0
     0
     0
     0
     0
>> v = ones(1,5)
v =
     1   1   1   1   1
>> v = ones(5,1)
v =
     1
     1
     1
     1
     1
```

4.1.1 Opérations sur Les Vecteurs

```
>> v = [1 2 3 4]; w = [2 4 6 8];
>> v + w % addition
ans =
     3   6   9   12
```

```
>> v – w % soustraction
ans =
     -1    -2    -3    -4
>> v * w' % produit scalaire
ans =
     60
>> v .* w % produit terme-à-terme
ans =
     2    8    18    32
>> v ./ w % division terme-à-terme
ans =
     0.5000    0.5000    0.5000    0.5000
>> v + 3 % addition d'un scalaire
ans =
     4    5    6    7
>> v - 3 % soustraction d'un scalaire
ans =
     -2    -1    0    1
>> v * 3 % produit avec un scalaire
ans =
     3    6    9    12
>> v / 3 % division sur un scalaire
ans =
     0.3333    0.6667    1.0000    1.3333
>> v .^2 % mise à une puissance
ans =
     1    4    9    16
>> x = [1 2 3]; y = [4 5 6 7]; z = [8 9];
>> v = [x y z] % concaténation
v =
     1    2    3    4    5    6    7    8    9
>> length(v) % taille
ans =
     9
>> a = [1 2]; b = [1; 2];
```

```
>> length(a)
ans =
     2
>> length(b)
ans =
     2
>> size(a) % nombre de lignes et de colonnes
ans =
     1    2
>> size(b)
ans =
     2    1
```

4.1.2 Adressage et Indexage

```
>> v = [1:1:5]*2
v =
     2    4    6    8    10
>> v(3) % 3ème élément
ans =
     6
>> v(2:4) % sous-vecteur du 2ème au 4ème élément
ans =
     4    6    8
>> v(2:end) %sous-vecteur du 2ème au dernier élément
ans =
     4    6    8    10
>> v([5 3 1]) % les éléments v(5), v(3), et v(1)
ans =
     10    6     2
```

4.1.3 Opérations Statistiques

```
>> x = [2 4 5 6 7];
>> max(x) % maximum
ans =
     7
```

```
>> min(x) % minimum
ans =
     2
>> sum(x) % somme des valeurs
ans =
     24
>> prod(x) % produit des valeurs
ans =
     1680
>> mean(x) % moyenne des valeurs
ans =
     4.8000
>> median(x) % la valeur médiane (après le tri)
ans =
     5
>> var(x) % la variance des valeurs
ans =
     3.7000
>> std(x) % l'écart type des valeurs
ans =
     1.9235
```

$$\text{sum(x)}=\sum_{i=1}^{n} x_i$$

$$\text{prod(x)}=\prod_{i=1}^{n} x_i$$

$$\text{mean(x)}=\overline{x}=\frac{\text{sum(x)}}{n}$$

$$\text{mean(x)}=\begin{cases} x(p+1) & \text{si } n=2p+1 \\ \dfrac{x(p)+x(p+1)}{2} & \text{si } n=2p \end{cases}$$

$$\text{var(x)}=\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2$$

$$\text{std(x)}=\sqrt{\text{var(x)}}$$

4.1.4 Tri d'un Vecteur

Deux opérations particulièrement utiles à effectuer sur des vecteurs sont les opérations de tri et de recherche.

La commande **sort(v)** retourne un vecteur trié en ordre croissant depuis un vecteur d'entrée v.

En fait, la commande **sort** retourne deux vecteurs: vecteurs des valeurs triées et vecteurs de leurs indices avant le tri, par exemple

[svecteur ivecteur] = sort(v)

Les commandes **sort(v, 'ascend')** et **sort(v, 'descend')** retournent un vecteur trié en ordre croissant (décroissant respectivement).

Le tri d'un vecteur est basé sur les caractéristiques des éléments du vecteur d'entrée:

a. Les entiers et les réels: Selon leurs valeurs.

b. Les complexes: Selon le module puis l'argument (ou la phase).

c. Les caractères: Selon le code ascii.

4.1.5 Recherche d'un Elément dans un Vecteur

La commande **find(v)** retourne un vecteur des indices des valeurs non-nulles d'un vecteur v.

find(cond v) retourne les indices des valeurs satisfaisantes d'une condition sur le vecteur v.

La commande **find(v, k)** ou **find(v, k, 'first')** retourne au plus un vecteur des k premiers indices des valeurs non-nulles d'un vecteur v.

La commande **find(v, k, 'last')** retourne au plus un vecteur des k derniers indices des valeurs non-nulles d'un vecteur v.

Les commandes **find(cond v, k, 'first')** et **find(cond v, k, 'first')** filtrent le vecteur v selon la condition introduite et prennent les k premiers ou k derniers indices.

4.1.6 Insertion, Modification et Suppression d'un Elément dans un Vecteur

La modification d'un élément (ou plusieurs éléments) dans un vecteur peut être effectué par de son index (ou leurs index) comme suit:

v(i_1): l'élément i1 du vecteur v.

v(i_1 : i_2): un sous-vecteur de v à partir de l'index i_1 jusqu'à l'index i_2.

v(i_1 : pas : i_2): un sous-vecteur de v à partir de l'index i_1 jusqu'à l'index i_2 avec un pas.

v([$i_1 i_2 \ldots i_n$]): un sous-vecteur de v en prendre en considération les indices $i_1, i_2, \ldots, i_n$.

La suppression d'un ou plusieurs éléments d'un vecteur peut être effectuée en utilisant le vecteur vide [].

L'insertion d'un nouvel élément dans un vecteur nécessite l'index exact de l'élément afin de concaténer la partie gauche, le nouvel élément et la partie droite, en construisant le nouveau vecteur:

```
>> v = [2 4 8 10 12]
v =
    2   4   8   10   12
>> x = 6; i = 3; % élément à insérer et son index
>> v = [v(1:i-1) x v(i:end)] % concaténer partie gauche + élément + partie droite
v =
    2   4   6   8   10   12
```

4.2 Les matrices

Les matrices suivent la même syntaxe que les vecteurs.

Les composantes des lignes sont séparées par des virgules ou des espaces et chaque ligne est séparée de l'autre par un point virgule.

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

La commande **repmat(val, lin, col)** crée une matrice pleine des valeurs dupliqués .

Les commandes **zeros(lin, col)** et **ones(lin, col)** créent des matrices des zéros et des uns, respectivement.

La matrice identité peut être obtenue par la commande **eye(lin, col)**

Une matrice peut être produise à partir d'une autre matrice ou un autre vecteur en utilisant la commande **reshape(Vec, [lin, col])** en changeant sa dimension (le nombre d'éléments doit être identique)

```
>> v = [1 2 3 4 5 6 7 8 9 10 11 12] % vecteur ligne de 12 éléments
v =
    1   2   3   4   5   6   7   8   9   10   11   12
>> A = reshape(v, 3, 4) % matrice identité de 3 × 4
A =
    1   4   7   10
    2   5   8   11
    3   6   9   12
```

```
>> A = reshape(v, 4, 3) % matrice identité de 4 × 3
A =
     1    5    9
     2    6   10
     3    7   11
     4    8   12
>> A = reshape(v, 2, 6) % matrice identité de 2 × 6
A =
     1    3    5    7    9   11
     2    4    6    8   10   12
>> A = reshape(v, 6, 2) % matrice identité de 6 × 2
A =
     1    7
     2    8
     3    9
     4   10
     5   11
     6   12
```

4.2.1 Propriétés des matrices

Matlab fournie quelque opérations pour extraire les propriétés de base des matrices :

```
>> A = [1 2 1; 1 1 2; 2 2 1]
A =
     1    2    1
     1    1    2
     2    2    1
>> det(A) % déterminant
ans =
     3
>> size(A) % taille (lignes , colonnes)
ans =
     3    3
```

```
>> diag(A) % le vecteur du diagonal
ans =
     1
     1
     1
>> trace(A) % la trace (somme su diagonal)
ans =
     3
>> triu(A) % matrice triangulaire supérieure
ans =
     1   2   1
     0   1   2
     0   0   1
>> tril(A) % matrice triangulaire inférieure
ans =
     1   0   0
     1   1   0
     2   2   1
>> A' % le transposé
ans =
     1   1   2
     2   1   2
     1   2   1
>> A1 = inv(A) % l'inverse
A1 =
    -1.0000    0    1.0000
     1.0000   -0.3333   -0.3333
     0   0.6667   -0.3333
>> A*A1
ans =
     1.0000   0   -0.0000
     0   1.0000   0
     0   0   1.0000
```

La commande **rank** retourne le rang d'une matrice.

4.2.2 Opérations sur les matrices

En algèbre, Il y a plusieurs opérations de base qui s'effectuent sue les matrices. Néanmoins, ces opérations souvent exigent des pré-conditions avant s'exécuter :

a. A + B, A - B : (addition et soustraction) A et B ont la même dimension, sauf pour les scalaires.

b. A.*B, A./B : (multiplication et division terme à terme) A et B ont la même dimension, sauf pour les scalaires.

c. A*B : (multiplication) le nombre des colonnes de A égale au nombre de lignes de B, sauf pour les scalaires.

d. A/B : (division) A et B sont carrées et de même dimension, et B est singulière (inversible, ou déterminant non nulle) , sauf pour les scalaires.

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1   2   3
     4   5   6
     7   8   9
>> A + 2
ans =
     3   4   5
     6   7   8
     9   10   11
>> A - 2
ans =
    -1   0   1
     2   3   4
     5   6   7
>> A * 2
ans =
     2   4   6
     8   10   12
     14   16   18
```

```
>> A^2 % A*A
ans =
     30    36    42
     66    81    96
     102    126    150
>> A.^2 % mettre au carré les éléments de A
ans =
     1    4    9
     16    25    36
     49    64    81
>> 2.^A % mettre chaque élément comme une puissance
ans =
     2    4    8
     16    32    64
     128    256    512
>> A/2
ans =
     0.5000    1.0000    1.5000
     2.0000    2.5000    3.0000
     3.5000    4.0000    4.5000
```

Comme pour les vecteurs, la commande **find(A)** trouve les valeurs non nulles dans une matrice. **find(cond A)** trouve les valeurs satisfaisantes de la condition en question.

4.2.3 L'indexage dans une matrice

La modification d'un élément (ou plusieurs éléments) dans une matrice peut être effectué par de son index comme suit:

A(i), A(i, j): l'élément de l'index i ou (i, j) de la matrice A.

A(i: j), A(i_1: i_2, j_1: j_2): une sous-matrice de A à partir de l'index linéaire i jusqu'à l'index j (i_1 et i_2 index des lignes, j_1 et j_2 index des colonnes).

A(i: pas: j), A(i_1: pas: i_2, j_1: pas: j_2): une sous-matrice de A à partir de l'index i jusqu'à l'index j avec un pas (i_1 i_2 index des lignes, j_1 j_2 index des colonnes).

A([i_1 i_2 ... i_n]), A([i_1 i_2 ... i_n] ,[j_1 j_2 ... j_n]): une sous-matrice de A en prendre en considération les indices i_1 i_2 ... i_n (pour l'indexage ligne-colonne, on considère i_1 i_2 ... i_n des indices des lignes, j_1 j_2 ... j_n indices des colonnes).

4.2.4 L'Affectation et la Modification et Suppression dans une matrice

L'affectation d'un scalaire dans une matrice indicée généralise le scalaire sur toute la matrice.

L'affectation d'une matrice à une autre exige l'égalité des tailles des deux matrices.

A(1:2, 1:2) = 2 : affectation d'un scalaire dans une sous-matrice indicés

A = 2 : affectation d'un scalaire dans une matrice sans indice rend la matrice comme scalaire

La suppression d'un ou plusieurs éléments d'un vecteur peut être effectuée en utilisant le vecteur vide []:

A(1,:) = [] : supprimer une ligne

A(:,2) = [] : supprimer une colonne

A(4) = [] : supprimer un élément

La concaténation des matrices se fasse de la même manière que les vecteurs :

```
>> A = ones(2,2)
A =
     1    1
     1    1
>> B = zeros(2,2)
B =
     0    0
     0    0
>> C = eye(2)
C =
     1    0
     0    1
>> D = [A B; B C]
D =
     1    1    0    0
     1    1    0    0
     0    0    1    0
     0    0    0    1
```

4.2.5 Les fonctions appliquées sur les matrices

```
>> C = [-9 -5; -8 -4]
C=
    -9    -5
    -8    -4
```

```
>> sin(A)
ans =
    -0.4121    0.9589
    -0.9894    0.7568
>> abs(A)
ans =
     9    5
     8    4
>> sort(A)
ans =
    -9    -5
    -8    -4
>> sum(A)
ans =
    -17    -9
>> sum(sum(A))
ans =
    -26
>> prod(A)
ans =
     72    20
>> prod(prod(A))
ans =
     1440
>> min(A)
ans =
    -9    -5
>> max(A)
ans =
    -8    -4
>> mean(A)
ans =
    -8.5000    -4.5000
```

5 Polynômes

Définition

Un polynôme de la forme $P(x) = a_n x^n + \cdots + a_1 x^1 + a_0$ est représenté en Matlab par un vecteur ligne $[a_n, \ldots, a_1, a_0]$ (ou vecteur colonne).

Afin d'évaluer un polynôme pour une valeur fixe a, on utilise la commande **polyval(P, a)**.

```
>> P = [1 2 1] % définir le polynôme x^2 + 2x + 1
P =
     1   2   1
>> polyval(P, 0), polyval(P, 1), polyval(P, 2)
ans =
     1 % P(0)
ans =
     4 % P(1)
ans =
     9 % P(2)
```

Opérations arithmétiques

Matlab fournie deux opérations arithmétiques de multiplication et de division à travers les commande **conv** et **deconv**. Il reste au programmeur de programmer les deux autres opérations d'addition et de soustraction.

C = conv(A, B) est la convolution des tableaux A et B, c'est à dire les coefficients du produit des deux polynômes.

[Q, R] = deconv(A, B) est la dé-convolution des tableaux A et B où Q est le quotient de la division et R est le reste (A = conv(B ,Q) + R).

```
>> P1 = [1 1 1], P2=[1 1] % P1 = x^2 + x + 1, P2 = x + 1
P1 =
     1   1   1
P2 =
     1   1
>> conv(P1, P2) % P1*P2
ans =
     1   2   2   1 % P1 * P2 = x^3 + 2x^2 + 2x + 1
>> [Q R] = deconv(P1, P2) % P1/P2
Q =
     1   0 % Quotient = x
R =
     0   0   1 % Reste = 1
```

Racines et Interpolations

La commande **roots(P)** fait extraire les racines d'un polynôme P.

La commande **poly(V)** retourne un polynôme depuis ses racines stockées dans le vecteur V.

```
>> P = [1 -5 6]
P =
     1   -5   6
>> roots(P)
ans =
     3.0000
     2.0000
>> poly([2 3])
ans =
     1   -5   6
```

La commande **Polyfit(X, Y, n)** permet d'approximer un polynôme de dégrée n qui passent approximativement par les points (X, Y). Cette approximation P(X(i))=Y(i) est au sens des moindres carrées : trouver P tel que $\sqrt{\sum_i (P(x_i) - y_i)^2}$ est minimal.

```
>> X = [0 1 2], Y=[-1 0 3]
X =
     0   1   2
Y =
     -1   0   3
```

```
>> polyfit(X, Y, 0)
ans =
      0.6667 % P(x) = 2/3 (avec erreur)
>> polyfit(X, Y, 1)
ans =
      2.0000   -1.3333 % P(x) = 2x -1/3 (avec erreur)
>> polyfit(X, Y, 2)
ans =
      1.0000   0   -1.0000 % P(x) = x^2 - 1 (meilleur)
```

Dérivation et Intégration d'un polynôme

La commande **polyder(P)** retourne la dérivée d'un polynôme P.

La commande **polyint(P)** retourne l'intégral d'un polynôme P. Si on veut calculer l'intégral entre deux point x1 et x2, on utilise la commande polyval pour évaluer l'intégral sur ces deux points puis on applique la soustraction.

```
>> P = [1 2 1] % P(x) = x^2 + 2x + 1
P =
      1   2   1
>> Pder = polyder(P) % P'(x) = 2x + 2
Pder =
      2   2
>> Pint = polyint(P) % integral(P(x)) = 1/3x^3 + x^2 + x
Pint =
      0.3333   1.0000   1.0000   0
>> polyval(Pint, 2) - polyval(Pint, 1) % integral(P(x)) entre 1 et 2
ans =
      6.3333
```

Résolutions des équations et des inéquations, linéaires et non-linéaires

Matlab utilise la commande **solve** afin de résoudre les équations et les inéquations, les systèmes linéaires et non linéaires.

Tout d'abord, il est nécessaire de déclarer des variables symboliques des équations (ou inéquations, . . .) en utilisant la commande **syms**.

Il faut noter que solve retourne des solutions sous forme symbolique ou formelle au lieu de numérique (racine de 2 à la forme numérique de 1.4142 et la forme formelle 2^(1/2)).

```
>> solve(x^2-1) % résoudre x^2 - 1 = 0
Undefined function or variable 'x'.
>> syms x % definer la variable symbolique x
>> solve(x^2-1) % résoudre x^2 - 1 = 0
ans =
    1
    -1
>> solve(x^2-1==1) % résoudre x^2 - 1 = 1
ans =
    2^(1/2)
    -2^(1/2)
```

Matlab peut aussi résoudre des équations de forme générale ou paramétrique.

```
>> syms x a b c
>> solve(a*x+b == 0)
ans =
    -b/a
>> solve(a*x^2 + b*x + c == 0)
ans =
    -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
    -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

Pour les équations à plusieurs variables, Matlab permet de choisir la variable pour laquelle on veut résoudre.

```
>> solve(a*x + b, x) % résoudre pour x
ans =
    -b/a
>> solve(a*x + b, a) % résoudre pour a
ans =
    -b/x
>> solve(a*x + b, b) % résoudre pour b
ans =
    -a*x
```

La commande solve nous aide aussi à résoudre un système linéaire ou non-linéaire en présentant ses lignes d'équations.

```
>> syms x y
>> S = solve(x+y == 5, x-y == 1) % système linéaire
S =
    x: [1×1 sym]
    y: [1×1 sym]
>> [S.x S.y]
ans =
    [ 3, 2]
>> solve(x^2 + 1) % équation non-linéaire
ans =
    i
    -i
>> S = solve(x^2 + y == 0, x + y^2 == 0) % système non-linéaire
S =
    x: [4×1 sym]
    y: [4×1 sym]
>> [S.x S.y]
ans =
    [ 0, 0] % sol. 1
    [ -1, -1] % sol. 2
    [ (3^(1/2)*i)/2 + 1/2, 1/2 - (3^(1/2)*i)/2] % sol. 3
    [ 1/2 - (3^(1/2)*i)/2, (3^(1/2)*i)/2 + 1/2] % sol. 4
```

La commande solve permet aussi de résoudre une inéquation ou un système linéaire ou non linéaire des inéquations.

```
>> solve(x^2 - 9 <= 0)
ans =
    Dom::Interval([-3], [3]) % [-3, 3]
>> solve(x^2 - 9 >= 0)
ans =
    Dom::Interval([3], Inf) % ] − ∞, −3] ∪ [3, +∞[
    Dom::Interval(-Inf, [-3])
>> solve(x^2 - 9 > 0)
ans =
    Dom::Interval(3, Inf) % ] − ∞, −3] ∪ [3, +∞[
    Dom::Interval(-Inf, -3)
```

6 Graphisme en Matlab

6.1 Représentation graphique sous MATLAB

Pour représenter des courbes du type y = f(x) ou des surfaces z = f(x; y), les données x,y,z doivent être des vecteurs colonnes (x et y) ou des matrices (z) aux dimensions compatibles. L'instruction de dessin correspondante (par exemple **plot(x,y)** pour tracer des courbes planes) est alors utilisée et éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes, etc...). La visualisation du résultat s'effectue dans une fenêtre graphique (avec possibilité de zoom, de rotation, d'impression).

6.1.1 Exemple de représentation graphique en deux dimensions (2D)

6.1.1.1 Fonction élémentaire

La commande plot dont la syntaxe est la suivante :

plot(x,y,s)

permet de tracer des graphiques (courbes ou nuages de points) de vecteurs de dimensions compatibles (y en fonction de x).

6.1.1.2 Styles de lignes 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 3 ème argument à la fonction plot( ). Par exemple, pour afficher notre courbe avec un point vert par donnée, nous utiliserons l'option 'g.'

$>>$ **plot(x,y,'g.')**

NB : Le type de tracé est par défaut le trait continu. De même, MATLAB fixe une couleur par défaut si elle n'est pas spécifiée.

Les styles de lignes disponibles ainsi que les codes associés sont résumés dans le tableau ci-dessous :

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

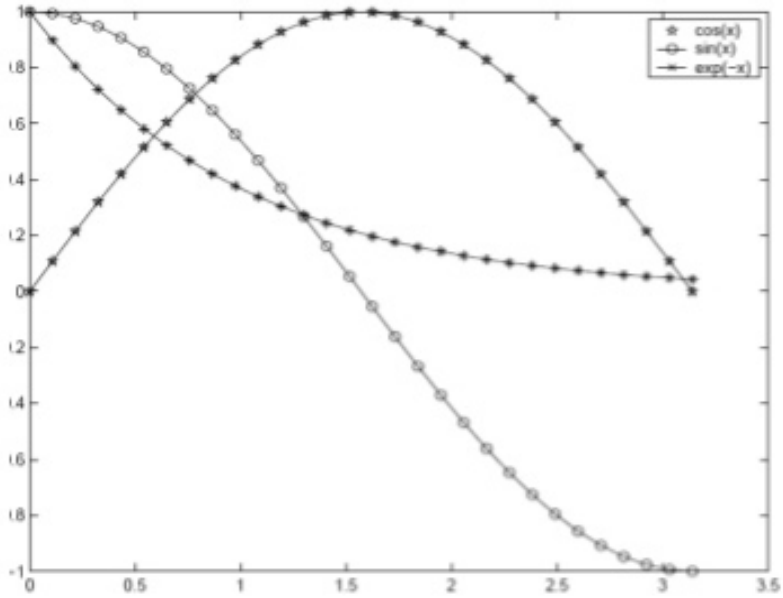
Exemple

```
plot(x,cos(x),x,sin(x),x,exp(-x)) % Matlab va automatiquement utiliser des couleurs
    différentes pour chaque courbe
plot(x,cos(x),'o-r',x,sin(x),'x-b',x,exp(-x),'*-g') % pour spécifier le type de symbole
    et la couleur à utiliser pour chaque courbe
legend('cos(x)','sin(x)','exp(-x)') % pour rajouter une légende
```

NB : Pour superposer des courbes par des appels successifs à cette fonction, il faut auparavant avoir utilisé la commande hold on, comme suit :

```
hold on
plot(x,cos(x),'o-r')
plot(x,sin(x),'x-b')
plot(x,exp(-x),'*-g')
legend('cos(x)','sin(x)','exp(-x)')
```

le résultat de ce script est illustré par la Figure suivante.



6.1.1.3 Affichage de plusieurs graphes côte à côte

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 :

```
>> 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')
```