République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira de Bejaia

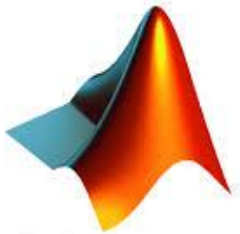
جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté de Technologie

Département d'**Hydraulique**

POLYCOPIE DE COURS

Module :



MATLAB

Master 1 Hydraulique

Option : Hydraulique Urbaine (HU)

Option : Ouvrages et Aménagements Hydrauliques (OAH)

Responsable du Module :

Dr. MERAH Ferhat

Option : Hydraulique Urbaine / Ouvrages et Aménagements Hydrauliques

Semestre : 1

UE : Méthodologique 1.1

Matière : Matlab

Objectifs de l'enseignement :

L'objectif est d'acquérir des connaissances de programmation par le logiciel MATLAB, pour son utilisation future dans la formulation et la résolution des problèmes complexes en hydraulique.

Connaissances préalables recommandées :

Afin de pouvoir suivre cet enseignement, des connaissances de base en algorithmique et en programmation, sont requises.

Contenu de la matière : *(Le détail du programme : voir Annexe)*

Chapitre 1 : Introduction et aspects élémentaires
Chapitre 2 : Les vecteurs
Chapitre 3 : Les matrices
Chapitre 4 : Eléments de la programmation Matlab
Chapitre 5 : Graphisme sous Matlab

Mode d'évaluation :

1- Durée de l'examen de fin de semestre : 1h 30
2- Durée de l'examen du rattrapage : 1h 30
3- Contrôles continus

Références bibliographiques :

[1] Apprendre et maîtriser MATLAB, M.Mokhtari, A. Mesbah, Edition Springer.
[2] Le site officiel de Matlab www.mathworks.com.
[3] Matlab pour les ingénieurs, A. Biran, M. Breiner, Pearson Education
[4] Introduction à MATLAB , Jean-Thierry Lapresté, Ellipses
[5] Calcul scientifique avec Matlab, Jonas Koko, Ellipses
[6] Applications mathématiques avec MATLAB, Analyse et analyse numérique: rappel de cours et exercices corrigés, Jolivet, Luc, Labbes, Rabah, Paris: Hermès Science Publications, 2005.
[7] Guide de l'analyse informatique, Sornet, Jacques, Paris, 1985.

Table des matières

Liste des figures

Liste des tableaux

Préambule

MATLAB (MATrix LABoratory), en français le laboratoire des matrices, est un langage de programmation de haut niveau de quatrième génération avec un environnement interactif pour le calcul numérique, la visualisation et la programmation. MATLAB est un langage de programmation développé par MathWorks[1].

Il permet des manipulations matricielles ; tracé des fonctions et des données ; implémentation d'algorithmes ; création d'interfaces utilisateurs ; interface avec des programmes écrits dans d'autres langages, notamment C, C++, Java et FORTRAN ; analyser les données ; développer des algorithmes ; et créer des modèles et des applications.

Il possède de nombreuses commandes et fonctions mathématiques intégrées qui nous aident dans les calculs mathématiques, la génération de tracés et l'exécution de méthodes numériques.

Matlab a débuté comme un langage de programmation matricielle où la programmation d'algèbre linéaire était simple. Il peut être exécuté à la fois sous des sessions interactives et en tant que travail par lots.
Ce cours donne aux étudiants une introduction douce et accrocheuse au langage de programmation MATLAB. Il est conçu pour leur donner la maîtrise du langage de programmation MATLAB. Des exemples MATLAB basés sur des problèmes, à la fin de chaque chapitre, ont été donnés de manière simple et facile pour rendre l'apprentissage rapide et efficace.

Ce cours a été préparé pour les étudiants de la 1ère année Master en Hydraulique (débutants en Matlab) afin de les aider à comprendre les fonctionnalités de base et avancées de MATLAB. Après avoir terminé ce présent cours, l'étudiant se retrouvera à un niveau modéré dans l'utilisation de MATLAB à partir duquel il pourra passer aux niveaux suivants.

L'objectif principal de ce cours est d'initier et de familiariser avec l'utilisation du logiciel Matlab. Ce cours est inspiré de plusieurs manuels et documents de cours de plusieurs universités et écoles **[3,4, 5, 6].** Il est fortement recommandé aux étudiants de découvrir la grande richesse de Matlab, une documentation plus complète avec différentes informations sont disponibles sur le site officiel de Matlab et celui de scilab**[1,2].**

On suppose que les apprenants ont quelques connaissances en programmation informatique et qu'ils comprennent des concepts tels que les variables, les constantes, les expressions, les instructions, etc. Si les étudiants ont déjà programmé dans un autre

[1] Mathworks : est un éditeur de logiciels américain, spécialisé dans les logiciels de calcul mathématiques. The MathWorks est connu pour être l'auteur de MATLAB et Simulink

langage de haut niveau comme le C, le Fortran ou le Java, alors ce sera très bénéfique et apprendre MATLAB sera comme un plaisir pour ces derniers.

MATLAB est utilisé dans toutes les facettes des mathématiques computationnelles. Voici quelques calculs mathématiques couramment utilisés où il est le plus couramment utilisé :

- Gestion des matrices et des tableaux ;
- Tracé et graphiques 2D et 3D ;
- Algèbre linéaire ;
- Équations algébriques ;
- Fonctions non linéaires ;
- Statistiques ;
- Analyse des données ;
- Calcul et équations différentielles ;
- Calculs numériques ;
- Ajustement des courbes ;
- Autres diverses fonctions spéciales.

MATLAB est largement utilisé comme outil de calcul en science et en ingénierie englobant les domaines de la physique, de la chimie, des mathématiques et de toutes les filières d'ingénierie. Il est utilisé dans une gamme d'applications, notamment [7,8,9] :

- Le traitement du signal et communications ;
- Le traitement d'images et de vidéos ;
- Le calcul numérique dans le système des réels ou des complexes ;
- Le calcul de probabilités ou les statistiques ;
- Le calcul intégral ou la dérivation ;
- L'optimisation ;
- Les tests et mesures ;
- L’automatisme ;
- Et dans plein d’autres utilisations.

Et si en standard, MATLAB propose des fonctions couvrant l'ensemble de ces domaines, si vos développements nécessitent de mettre en œuvre des programmes très poussés, il existe des fonctions plus spécifiques regroupées dans des TOOLBOX (que l'on peut traduire par "boîte à outils"). Ces toolbox sont des extensions évidemment payantes utiles, voire nécessaires, comportant des fonctions dédiées à ces domaines, pour des développements de niveau professionnel. On peut citer les extensions [1,2] :

- OPTIMIZATION pour l'optimisation ;
- IMAGE PROCESSING pour le traitement d'image.

Pour l'utilisateur débutant, ces toolbox ne sont pas nécessaires dans un premier temps, mais il est important de savoir que le moment venu, si le besoin se fait sentir, vous aurez la possibilité d'exploiter ces toolbox, et leur richesse fonctionnelle. [3]

Introduction générale

Cet ouvrage s'adresse aux étudiants en Master I Hydraulique avec ses deux spécialités à savoir « Hydraulique Urbaine » et « Ouvrages et Aménagements Hydrauliques » de l'université de Bejaia. Son objectif est de donner à l'étudiant un outil lui permettant de travailler de manière autonome à l'aide de questions détaillées et progressives, et d'une construction pas à pas des programmes à réaliser.

La meilleure façon d'apprendre est de l'essayer par l'étudiant lui-même. Travailler à travers des exemples, donnera à l'étudiant une idée pour la façon dont MATLAB fonctionne. Dans cette introduction, nous décrirons comment MATLAB gère des expressions numériques simples et des formules mathématiques.

Par ce document, nous souhaitons faciliter la prise en main de MATLAB, ainsi que présenter les outils disponibles pour résoudre des problèmes numériques qu'un étudiant, un ingénieur ou un scientifique, peut rencontrer dans le cadre de sa formation universitaire ou de son activité professionnelle. [3]

Le nom MATLAB signifie MATrix LABoratory. Une traduction littérale nous amène à voir MATLAB comme un laboratoire pour manipuler et exploiter des matrices. Nous reviendrons sur ce point, qui est un élément fondamental du langage MATLAB : la plupart des fonctions définies dans MATLAB le sont pour des grandeurs matricielles, et par extension, pour des données tabulées. MATLAB a été écrit à l'origine pour faciliter l'accès au logiciel matriciel développé par le LINPACK (package système linéaire) et les projets EISPACK (Eigen system package).

MATLAB est un langage de haute performance pour le calcul technique. Il intègre un environnement de calcul, de visualisation et de programmation. De plus, MATLAB est un environnement de langage de programmation moderne : il a des structures de données sophistiquées, contient des outils d'édition et de débogage intégrés et prend en charge la programmation orientée objet. Ces facteurs font de MATLAB un excellent outil pour l'enseignement et la recherche.

MATLAB présente de nombreux avantages par rapport aux langages informatiques conventionnels (par exemple, C, FORTRAN) pour résoudre des problèmes techniques. MATLAB est un système interactif dont l'élément de données de base est un tableau qui ne nécessite pas de dimensionnement. Le pack logiciel est disponible dans le commerce depuis 1984 et est maintenant considéré comme un outil standard dans la plupart des universités et industries du monde entier [10,11].

Il possède de puissantes routines intégrées qui permettent une très grande variété de calculs. Il dispose également de commandes graphiques faciles à utiliser qui rendent la visualisation des résultats immédiatement disponibles.

En plus de la documentation MATLAB qui est principalement disponible en ligne, nous recommandons les livres suivants : [10], [11], [12], [13]. Ils sont excellents dans leurs applications spécifiques.

Ce cours est structuré comme suit :

Le chapitre un portera sur les généralités et les aspects élémentaires du logiciel Matlab. Ensuite, les notions des vecteurs, leurs créations ainsi que leurs utilisations seront étudiées dans le chapitre deux. Le troisième sera consacré aux matrices et les fonctions appliquées à ces dernières. L'utilisation de la programmation par script Matlab sera traitée dans le chapitre quatre. Où on étalera sur les différentes commandes et instructions de programmation par fichiers. Le dernier chapitre étudié dans ce document est celui de la représentation graphique sous Matlab où l'étudiant verra comment utiliser les commandes pour tracer des graphes en 2D et 3D. ce travail sera terminé par une conclusion générale.

Chapitre I : Généralités et aspects élémentaires

I.1. Introduction

Nous supposons que les apprenants ont quelques connaissances en programmation informatique et comprennent des concepts et des notions de base comme les variables, les constantes, les expressions mathématiques, les déclarations, …etc. S'ils ont fait de la programmation dans n'importe quel autre langage de haut niveau comme C, C++, Fortran ou Java, alors ce sera très bénéfique et apprendre MATLAB sera comme un nouveau challenge pour eux.

Dans ce chapitre, l'environnement de Matlab est présenté. Un premier exemple introductif montre rapidement le principe de fonctionnement du logiciel. Nous présentons ensuite un ensemble de fonctions de base nécessaire pour débuter en Matlab.

I.2. Comprendre l'environnement MATLAB

En double-cliquant sur l'icône de MATLAB de raccourci (**MATLAB R2012a**) sur votre bureau Windows, MATLAB démarre, l'IDE (Integrated Development Environment, littéralement environnement de développement intégré) apparaît à l'écran. C'est la face visible de MATLAB, son interface graphique. Une fenêtre d'accueil appelée le bureau MATLAB apparaît. Le bureau est une fenêtre qui contient d'autres fenêtres. Les principaux outils, accessibles depuis le bureau, sont [3,4] :

- La fenêtre des commandes (***Command Window***) : à l'invite de commande « >> », l'utilisateur peut entrer les instructions à exécuter. Il s'agit de la fenêtre principale de l'interface ;
- L'historique des commandes (***Command history***) : historique des commandes que l'utilisateur a exécutées. Il est possible de faire glisser ces commandes vers la fenêtre de commande.
- L'espace de travail (***Workspace***) : permet de visualiser les variables définies, leur type, la taille occupée en mémoire…
- Le répertoire actuel ou courant (***Current folder***) : permet de naviguer et de visualiser le contenu du répertoire courant de l'utilisateur. Les programmes (fichiers *.m) de l'utilisateur doivent être situés dans ce répertoire pour être visible et donc exécutable ;
- Le navigateur d'aide (***Help***);
- Le bouton Démarrer de Matlab (***Start***) est un outil pour ouvrir rapidement certaines fonctionnalités de Matlab (outils, démonstrations et documentation). Il était présent sur les versions R2012a et antérieures. Ce bouton a été supprimé à partir de la version R2012b pour une utilisation sous forme d'onglets.

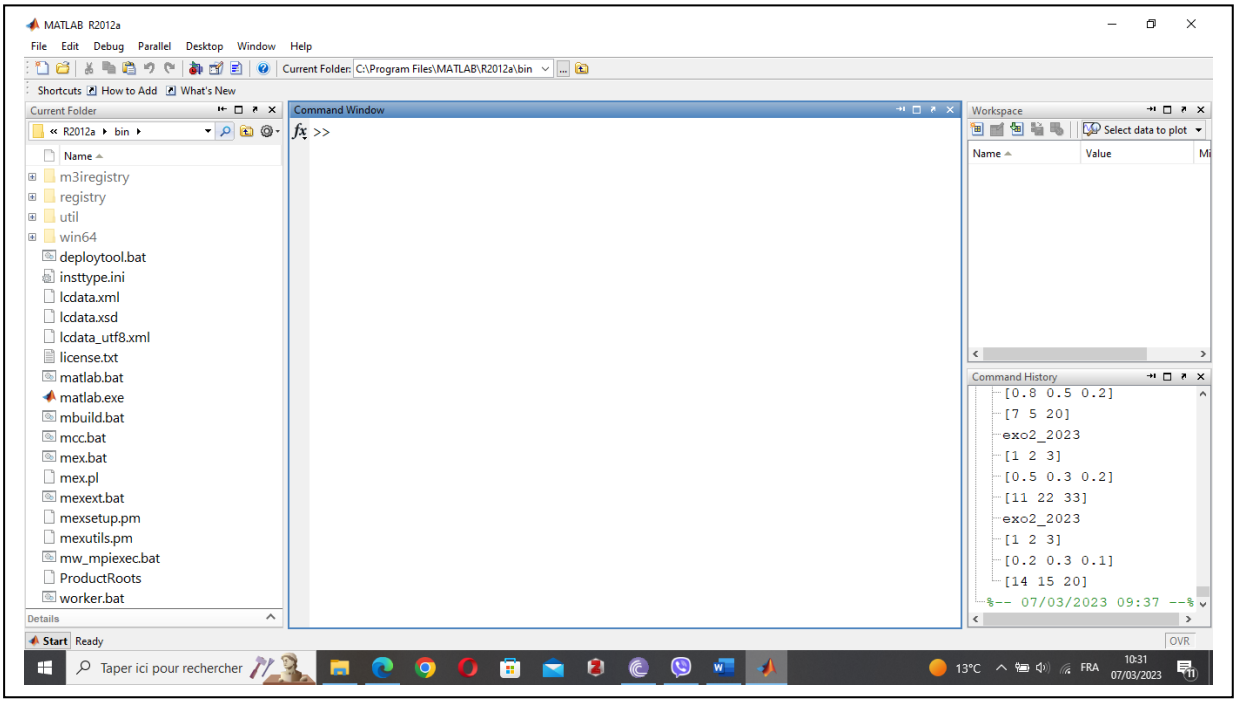


Figure I.1 : *Fenêtre principale de l'accueil Matlab*

Lorsque Matlab est démarré pour la première fois, l'écran ressemble à celui qui est montré dans la figure I.1. Cette illustration montre également la configuration par défaut de l'interface du bureau de Matlab. Vous pouvez personnaliser la disposition des outils et des documents en fonction de vos besoins.

I.2.1. L'aide du Matlab

La bonne utilisation de l'aide en ligne est fondamentale pour travailler correctement avec Matlab. Etant donné le très grand nombre d'instructions utilisables, il est hors de question de pouvoir mémoriser chacune d'elles avec sa syntaxe correspondante. A noter également que cette aide est uniquement disponible en langue anglaise. Elle est accessible de différentes manières :

- par la commande **help**, si l'on souhaite obtenir de l'aide sur certaines fonctions dont on connaît le nom, on utilise la fonction help. Par exemple :

Exemple:

```
>> help cos
 COS    Cosine of argument in radians.
   COS(X) is the cosine of the elements of X.
    See also acos, cosd.
   Overloaded methods:
     codistributed/cos
   Reference page in Help browser
     doc cos
```

- par la commande **doc**, en tapant directement doc nom de la commande dans la fenêtre de commandes

Exemple :

```
>>doc plot
plot
2-D line plot
Syntax
plot(Y)
plot(X1,Y1,...,Xn,Yn)
plot(X1,Y1,LineSpec,...,Xn,Yn,LineSpec)
plot(...,'PropertyName',PropertyValue,...)
plot(axes_handle,...)
h = plot(...)
```

Et l'affichage de sa **description**, les **indications** de la commande, des **exemples** et les **alternatives**, **voir aussi** les commandes de la même catégorie de **plot.**

- en consultant la documentation en ligne sur le site de la société **MathWorks**, à l'adresse https://fr.mathworks.com/help/

La commande **doc** offre une documentation complète de la commande ***recherchée***. La commande **help** est à privilégier pour une aide rapide. Elle donne accès à la syntaxe de toute commande.

I.2.2. Fenêtre des commandes

La fenêtre des commandes, appelée ***Command Window***, apparaît dans la partie centrale de l'interface. C'est dans cette fenêtre que vont être saisies les commandes juste-après le prompt Matlab « **>>** ». Le prompt « **>>** » rappelle l'interpréteur de commandes des shells UNIX.

Un moyen d'utiliser le logiciel est de saisir directement les commandes qu'on désire effectuer dans la ***Command Window***. C'est donc la fenêtre qui exécute les commandes MATLAB. On l'appelle aussi **le mode interactif, o**ù Matlab interprète et exécute les

commandes (instructions) directement après la saisie dans la sous fenêtre des 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.

La fenêtre de commandes donne accès à l'historique de toutes les commandes qui ont été déjà tapées sous MATLAB. Vous pouvez les retrouver et éditer ces commandes grâce aux touches de direction.

Utilisez les touches ⇑ et ⇓ pour parcourir les commandes exécutées précédemment, et utilisez ⇒, ⇐, et la touche «Entrée» pour éditer une commande. Pour relancer une commande, il suffit d'appuyer sur la touche «Entrée». Vous pouvez retrouver toutes les commandes commençant par un groupe de lettres. Pour cela, tapez les premières lettres de la commande que vous recherchez, puis appuyez plusieurs fois sur ⇑ pour parcourir les commandes correspondant à cette recherche. [albi]

Exemple :

```
>> r=5
r =
    5
>> s=pi*r^2
s =
  78.5398
>> p=2*pi*r
p =
  31.4159
```

I.2.3. Historique des commandes

La fenêtre Historique des commandes ***« Command History »*** contient un historique des commandes que vous avez déjà tapé dans la Fenêtre des commandes. Il est très utile de deux manières. Tout d'abord, il vous permet de voir d'un coup d'œil un enregistrement des commandes que vous avez saisies précédemment. Deuxièmement, il peut vous économiser un peu de temps de frappe. Si vous cliquez sur une entrée dans l'historique des commandes avec le bouton droit de la souris, il est mis en surbrillance et un menu d'options apparaît.

Vous pouvez, par exemple, sélectionner Copier, puis cliquer avec le bouton droit de la souris dans la fenêtre de commande et sélectionnez Coller, après quoi la commande que vous avez sélectionnée apparaîtra à l'invite de commande et sera prête à être exécuter ou à éditer. Il existe de nombreuses autres options que vous pouvez apprendre en expérimentant ; par exemple, si vous double-cliquez sur une entrée dans l'historique des commandes, elle sera exécutée immédiatement dans la fenêtre de commande.

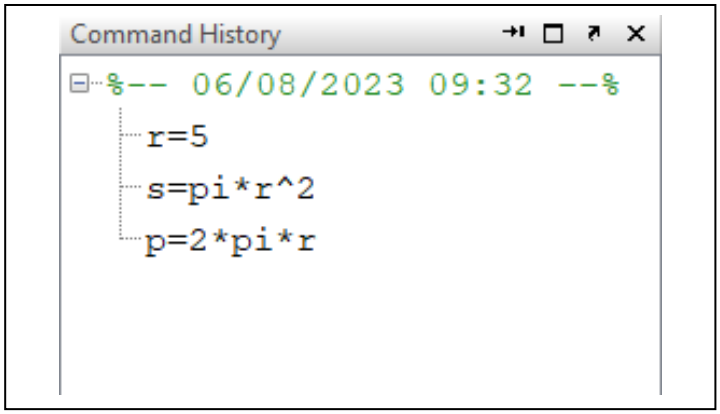


Figure I.2: *Copie écran de la fenêtre historique des commandes*

I.2.4. Workspace

Le « ***workspace »*** affiche le navigateur de l'espace de travail, une interface utilisateur graphique qui vous permet d'afficher et de gérer le contenu de l'espace de travail dans MATLAB. Il fournit une représentation graphique de l'affichage **whos** et vous permet d'effectuer l'équivalent des fonctions d'effacement, de chargement, d'ouverture et de sauvegarde. L'ordre de l'affichage des identificateurs de variables utilisées se fait en ordre alphabétique.

Le navigateur **Workspace** affiche également et met automatiquement à jour les calculs statistiques pour chaque variable, que vous pouvez choisir d'afficher ou de masquer.

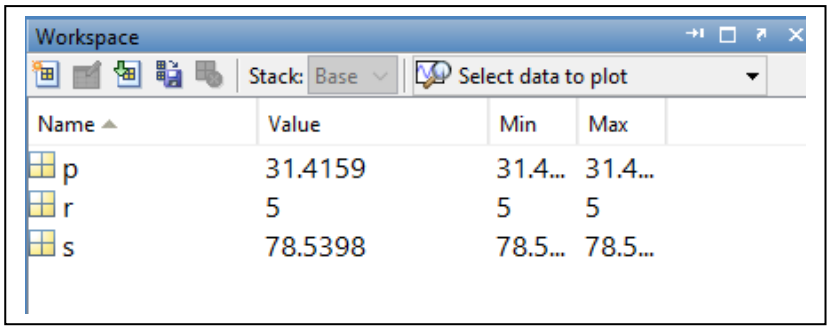


Figure I.3: *Copie écran de la fenêtre Workspace actuelle*

- *Quelques commandes de gestion du Workspace et les fichiers*

Dans le tableau suivant, quelques commandes de gestion du ***workspace*** à partir du ***command window*** :

Tableau I.1 : Commandes de gestion du workspace et des fichiers

Commande	Description
cd	Changer le répertoire courant
clc	Effacer la fenêtre de commande
clear (all)	Supprime toutes les variables de l'espace de travail
clear x	Effacer x Supprimer x de l'espace de travail
copyfile	Copier un fichier ou un répertoire
delete	Supprimer des fichiers
dir	Affiche la liste des répertoires
exist	Vérifie si des variables ou des fonctions sont définies
help	Afficher l'aide pour les fonctions MATLAB
lookfor	Recherche le mot spécifié dans toutes les entrées d'aide
mkdir	Créer un nouveau répertoire
movefile	Déplacer un fichier ou un répertoire
pwd	Identifier le répertoire courant
rmdir	Supprimer le répertoire
type	Afficher le contenu du fichier
what	Liste les fichiers MATLAB dans le répertoire courant Localiser les fonctions et les fichiers
which	
who	Afficher les variables actuellement dans l'espace de travail
whos	Afficher des informations sur les variables dans l'espace de travail

I.2.5. Le répertoire courant

Le répertoire courant ou le dossier actuel est un emplacement de référence utilisé par MATLAB pour rechercher et repérer des fichiers Matlab « *.m » et des sous répertoires. Ce dossier est parfois appelé répertoire courant, dossier de travail courant ou répertoire de travail actuel. Ce panneau vous permet d'accéder aux dossiers et fichiers du projet de travail.

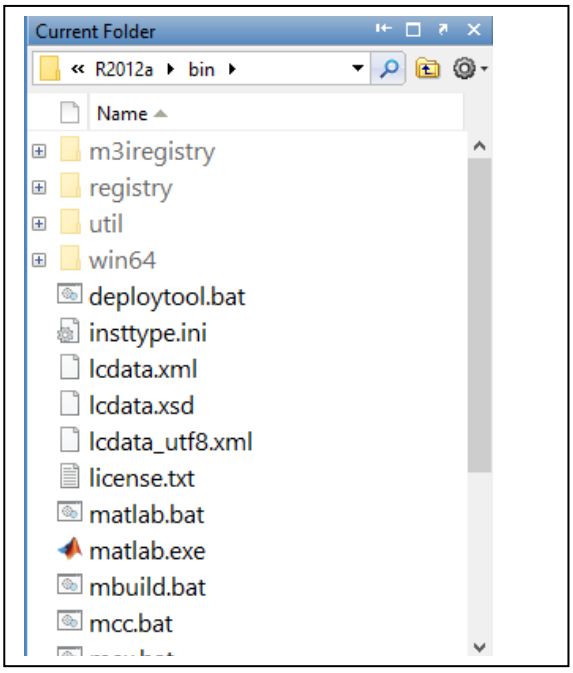


Figure I.4 : *Copie écran de fenêtre Dossier courant*

Le dossier courant par défaut est situé à l'endroit où l'exécutable du Matlab est installé (C:\Program Files\MATLAB\R2012a\bin) pour notre cas. Pour la bonne organisation de vos projets et travaux, vous devez tout d'abord créer votre répertoire sur un des disques. Après vous devez réorienter le dossier actuel du Matlab vers votre répertoire créé.

Exemple

- création du répertoire « programmes_matlab» sur le disque D:
- réorientation du dossier courant vers « programmes_matlab » soit par manipulation sur la fenêtre dossier courant ou bien par la fenêtre *Command Window* par la commande suivante :
  >> cd D:\programmes_matlab
- tous les programmes et projets créés seront enregistrés dans le répertoire « D:\programmes_matlab »

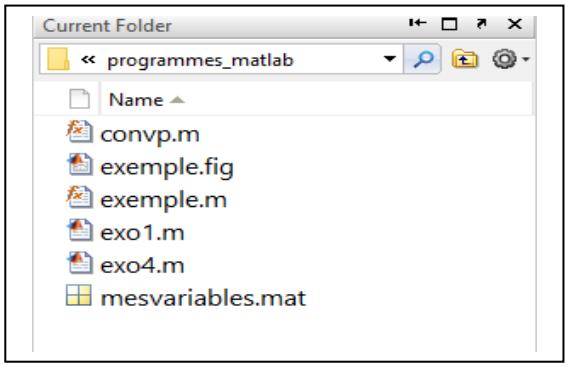


Figure I.5 : *Copie écran de fenêtre Dossier courant modifié*

I.3. Aspects élémentaires

I.3.1. Utiliser MATLAB comme calculatrice

Comme exemple de calcul interactif simple, tapez simplement l'expression que vous voulez évaluer. Commençons par le tout début. Par exemple, supposons qu'on veut calculer l'expression, 3*2-1. On va la saisir après le prompt Matlab (>>) comme suit :

```
>> 3*2-1

ans =

   5
```

Vous aurez remarqué que si on ne spécifie pas de variable de sortie, MATLAB utilise une variable par défaut **ans**, abréviation de ***answer***, pour stocker les résultats du calcul en cours. Notons bien que la variable **ans** est créée (ou écrasée, si elle existait déjà). Pour éviter cela, vous pouvez affecter une valeur à un nom de variable ou d'argument de sortie. Par exemple

```
>> R=3*2-1

R =

   5
>> 8*R
ans =
  40
```

Pour ce nouveau calcul, le résultat a été affecté à la variable R. Par contre le résultat de calcul de l'expression 8*R a été affecté à la variable **ans** déjà créée en écrasant son contenu par la valeur 40 au lieu de valeur 5 du calcul précédent.

I.3.2. Les variables sous Matlab

Dans la plupart des langages, et MATLAB n'échappe pas à la règle, on appelle variable un objet correspondant à une zone mémoire, et identifié par un nom. Cette notion est fondamentale, parce qu'elle permet de manipuler des valeurs (numériques ou non) et est à la base de la plupart des calculs et traitements que l'on peut réaliser avec un langage de programmation. [5,7]

Matlab ne fait pas de distinction entre variable entière, réelle, complexe, chaîne de caractères [3,9]. Ainsi les types de variables et dimension de variables n'ont pas à être déclarés préalablement avant leur utilisation et manipulation comme dans quelques langages de programmation, même pour les tableaux et les matrices. Il suffit simplement d'attribuer (=) une valeur à l'identificateur de la variable depuis ***Command Window***. Par

la suite, Matlab crée automatiquement la variable et lui attribue un espace mémoire dans le Workspace. Dans le cas où la variable existe déjà, Matlab modifie son contenu et, si nécessaire, effectue une allocation d'espace mémoire selon le besoin (exemple : redimensionnement d'un tableau de variables à de nouvelles dimensions). La création et la manipulation de variables est basée sur des expressions.

```
>> a=5
a =
    5
>> a=[1 2 3 4 5]
a =
    1    2    3    4    5
>> a='Bonjour'
a =
Bonjour
>> a=-1+3i
a =
 -1.0000 + 3.0000i
>> a=[1 2 3;4 5 6]
a =
    1    2    3
    4    5    6
>> b=a+5
b =
    6    7    8
    9   10   11
```

On remarque que la variable a au début c'était un scalaire entier, ensuite on lui affectant un vecteur, son type et ses dimensions ont changé. On lui attribuant la valeur 'Bonjour' la variable **a** est devenue une chaine de caractères… etc. le dernier contenu de la variable a manipulé dans l'expression « b=a+5 » est une matrice où a=$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

- *Rappel des types de variables*

Chaque variable a un nom (identificateur) et un type. Ce dernier correspond au genre d'information que l'on souhaite utiliser. Le nom de la variable obéit à des impératifs changeant selon les langages. Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces. Un nom de variable correct commence également impérativement par une lettre alphabétique. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

Les différents types pouvant définir une variable sont :

- entier pour manipuler des entiers ;
- réel pour manipuler des nombres réels avec toutes leurs formes;

- Complexe afin de manipuler les nombres complexes : dans Matlab, un nombre complexe est de la forme : z = a + bi . Stocké de façon interne sur 2x 8 octets, respectivement pour la partie réelle et la partie imaginaire. Les fonctions prédéfinies dans Matlab pour la manipulation des nombres complexes sont (real(z) , imag(z), abs(z), arg(z) et conj(z)).
- booléen pour manipuler des valeurs booléennes vrai ou faux ;
- chaîne de caractères pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases.

```
>> a=5      % a est un nombre entier
a =
    5
>> r=4.23      % a est un nombre réel
r =
    4.23
>> r2=1.028e-5      % r2 est un nombre réel avec exposant
r2 =
  1.0280e-05
>> ch='Bonjour'      % ch est une chaine de caractères
ch =
Bonjour
>> z=-1+3i       % z est un nombre complexe
z =
  -1.0000 + 3.0000i
>> b= true       % b est un booléen
b =
    1
```

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

MATLAB traite un seul type d'objet de variable : les matrices ! (d’où son nom est obtenu : Matrix Laboratory, le laboratoire des matrices)

1. Les scalaires sont des matrices 1 x 1 ;
2. les vecteurs lignes (rangées) sont des matrices 1 x n ;
3. les vecteurs colonnes sont des matrices n x 1 ;
4. les matrices de n x m.

I.3.4. Les opérateurs et fonctions mathématiques

a. Les opérateurs arithmétiques

Les opérateurs s'appliquent aux variables (scalaire, vecteurs et matrices) moyennant précautions et respect des règles d'usage (en général sur les matrices). Les éléments (les matrices) sont de type entier, réel ou complexe.

Par ordre de priorité des opérations, nous avons :

Tableau I.2 : *Les opérateurs arithmétiques*

^ : Puissance	. ^ : Puissance de chaque élément
* : Multiplication	. * : Multiplication de chaque élément
/ : Division à droite (ordinaire)	./ : Division à droite de chaque élément
\ : Division à gauche (*left division*)	. \ : Division à gauche de chaque élément
mod(a,b) : reste de la division de a sur b	+ : Addition - : Soustraction

L'utilisation des parenthèses dans des expression arithmétiques changent l'ordre de priorité.

b. Les fonctions mathématiques

De nombreuses fonctions de calcul sont prédéfinies dans MATLAB. Appliquées à un argument scalaire ou complexe, ces fonctions mathématiques s'appliquent également aux vecteurs et aux matrices en agissant sur chaque élément indépendamment. Ce qu'il faut retenir que pour appliquer une fonction à une valeur ou une variable, **il faut mettre cette dernière entre parenthèses**. Et aussi à ne pas oublier sous Matlab le nom de la fonction est toujours écrit en minuscule.

Tableau I 3 : *Quelques fonctions les plus utilisées sous Matlab*

Fonction	**Signification**	**Fonction**	**Signification**
abs(x) sign(x) sqrt(x) sin(x) cos(x) ceil(a)	|x| 1,-1 ou 0 $\sqrt{x}$ Le sinus de x (x en radian) Le cosinus de x la valeur entière supérieure	exp(x) log(x) log10(x) asin(x) round(a) fix(a)	e^x base e base 10 arc sinus en radian Valeur entière la plus proche de a la partie entière de a
real(z) conj(z) abs(z)	Partie réelle d'un complexe z Conjugué de z Le module du complexe z	imag(z) angle(z)	Partie imaginaire d'un complexe z argument de z en radian
a & b ou and(a,b) a | b ou or(a,b)	et logique ou logique	xor(a,b) ~a ou not(a)	Le ou exclusif La négation de a

c. Les opérateurs relationnels

On distingue les opérateurs de relation suivants :

Tableau I.4 : Opérateurs de relation sous Matlab

Symbole	Opération
<	Inférieur strictement
<=	Inférieur ou égal
>	Supérieur strictement
>=	Supérieur ou égal
==	Egal
~=	Différent

Appliqués aux opérandes, ils fournissent un résultat booléen.

Ces opérateurs relationnels sont utilisés dans des instructions de comparaisons et itératives (boucles). Ils sont associés quelques fois aux opérateurs logiques (or |, and &, not ~).

```
>> 8==5      % Est-ce que 8=5
ans =
   0         % Ici Matlab fournit le résultat faux (ans =0)
>> rep=10>=10       % rep reçoit le résultat du test logique
rep =
   1              % Ici Matlab fournit le résultat vrai dans rep (rep=1)
```

d. *Variables prédéfinies et constantes mathématiques*

Matlab possède quelques variables prédéfinies et des constantes mathématiques utilisées dans de différentes expressions mathématiques, on peut distinguer :

Tableau I.5 : Variables et constantes prédéfinies

Variable	Description
ans	Valeur de la dernière variable (ans)
eps	Précision relative en virgule flottante
i	Unité imaginaire d'un nombre complexe
inf	Infini (∞)
j	Unité imaginaire d'un nombre complexe
NaN	Pas un nombre
Pi	Le nombre π (3,14159 . . .)
realmin	Le plus petit nombre réel positif utilisable
realmax	Le plus grand nombre réel positif utilisable

I.4. Applications du chapitre I

I.4.1. Application 1

Cette application concernera l'utilisation du ***Command Window*** afin de gérer le **Workspace** et le **Dossier Courant** par :

- Création d'un répertoire sur le D:\ ;
- Modification du Dossier courant vers le répertoire créé ;
- Création de variables et réalisation de quelques calculs ;
- Manipulation du contenu du ***Workspace.***

```
>> cd d:\      % rediriger le dossier courant vers D:
>> mkdir mes_projets      % Créer un répertoire « mes_projets » sur D :
>> cd mes_projets      % rediriger le dossier courant vers D:\mes _projets
>> cd                  % visualiser l'adresse du dossier courant
d:\mes_projets
>> a=8                 % création de a par l'affectation de la valeur 8
a =
    8
>> b=2*a      % création de b par l'affectation du résultat de l'expression
b =
   16
>> a*
 a*
Error: Expression or statement is incomplete or
incorrect.      % Message d'erreur de la dernière commande saisie
>> a^3
ans =          % création de la variable ans car a^3 n'est pas affectée à un nom de variable
  512
>> who            % affichage des noms de variables existant dans le workspace
Your variables are :
a   ans  b

>> whos        % affichage des variables existant dans le workspace avec leurs détails
  Name      Size          Bytes  Class     Attributes
  a         1x1               8  double
  ans       1x1               8  double
  b         1x1               8  double
>> exist a       %Vérification de l'existence de la variable a dans le Workspace
ans =          % ans prend un résultat booléen 1 ou 0
    1
>> exist d       %Vérification de l'existence de la variable d dans le Workspace
ans =
   0
>> clear all       % supprimer le contenu du Workspace
>> clc             % effacer le contenu du Command window
```

I.4.2. Application 2

Utilisation du ***Command Window*** afin de :

- créer de différents types de variables ;
- insérer des fonctions arithmétiques et logiques dans des expressions ;

```
>> ro=1000;        % on utilise le symbole « ; » pour ne pas afficher le contenu de la variable
>> g=9.81,h=12     % on utilise le symbole « , » pour saisir plusieurs commandes sur une
                      seule ligne
g =
   9.8100
h =
   12
>> p=ro*g*h
p =
    117720
>> d=10 ;
>> s=pi*d^2/4      %utilisation de la constante pi
s =
  78.5398
>> z=3-2i              % fonctions appliquées à un nombre complexe
z =
  3.0000 - 2.0000i
>> real(z)             % partie réelle de z
ans =
    3
>> imag(z)             % partie imaginaire de z
ans =
   -2
>> abs(z)              % le module et l'angle de phase de z
ans =
   3.6056
>> angle(z)
ans =
  -0.5880
>> conj(z)             % le nombre conjugué de z
ans =
   3.0000 + 2.0000i
>> s,g,ans             % pour afficher le contenu de la variable, il suffit d'écrire son nom
s =
  78.5398
g =
   9.8100
ans =
  3.0000 + 2.0000i
```

I.4.3. Application 3

Utilisation du ***Command Window*** pour la modification des différents formats d'affichage d'une variable.

```
>> format short      % Format court, c'est le format par défaut
>> s
s =
   78.5398
>> format long          % Format long
>> s
s =
  78.539816339744831
>> format bank        % Format monétaire  (Banque)
>> s
s =
       78.54
>> format rat          % Format ratio de nombres entiers
>> s
s =
    8875/113
>> format hex         % Format hexadécimal
>> s
s =
   4053a28c59d5433b
>> format shorte        % Format court avec exponentiel
>> s
s =
   7.8540e+01
>> format longe            % Format long avec exponentiel
>> s
s =
    7.853981633974483e+01
```

I.5. Conclusion

Dans ce chapitre, nous avons présenté l'environnement du logiciel Matlab et ses constituants en détail. Un rappel sur les variables et leurs types a été donné ainsi que les fonctions et les différents opérateurs arithmétiques et logiques utilisés dans les expressions mathématiques. A la fin du chapitre trois (03) applications ont été proposées afin de manipuler les types de données et les informations liées aux ***workspace.*** Le chapitre suivant sera consacré à l'étude des vecteurs.

Chapitre II : Les vecteurs

II.1. Introduction

Comme son indique, Matlab est un langage de programmation pensé pour le calcul matriciel, où il excelle de terme de performances.

Tout le but d'un programme en Matlab est de faire passer un calcul classique à travers un calcul matriciel pour que Matlab le réalise très rapidement. Cela s'appelle la vectorisation. Nous allons voir en ce qui suit, comment Matlab crée et gère les vecteurs et les matrices.

II.2. Définition et rappels

Comme rappel, un tableau est une variable permettant de regrouper sous un même nom plusieurs valeurs de même type. Il existe 2 types de tableaux :

- les tableaux à une dimension : les vecteurs
- les tableaux à deux dimensions : les matrices

un vecteur est caractérisé par :

- son nom : identificateur unique pour tous ses éléments.
- sa taille : nombre d'éléments (cases du vecteur), ce nombre est constant et connu à l'avance pour certains logiciels, par contre dans Matlab, le nombre d'éléments n'est obligatoire d'être connu d'avance ;
- son indice : l'accès à un élément du vecteur se fait à l'aide du nom du vecteur suivi d'un indice ou du numéro de son emplacement (i , j , k …)

II.3. Création de vecteurs

II.3.1. Construction de vecteurs

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

On définit un vecteur **ligne** en donnant par énumération de ses éléments entre crochets « **[ ]** ». Les éléments sont séparés au choix par des **espaces** ou par des **virgules**.
Un vecteur ligne peut être aussi crée par description de la valeur initiale, l'incrément, et la valeur finale.

V1=[1,2,3,4,5] ou V1 = [1 2 3 4 5] V1 est un vecteur par énumération
V2=[Vi :inc :Vf] V2 est un vecteur par description
Avec :
Vi : valeur initiale ; inc : valeur d'incrémentation ou de désincrémentation (valeur positive ou négative et de type entier ou réel)
Vf : valeur finale.
On définit un vecteur **colonne** en donnant la liste de ses éléments séparés au choix par des **points virgules** (;) ou par des **retours chariots** (touche Entrée/Enter).

```
>>  v1=[1 2 3 4 5 6 7]          % Vecteur par énumération en utilisant « l'espace »
v1 =
   1    2    3    4    5    6    7
>> v1=[1:1:7]                  % Vecteur par description intervalle inc=1
v1 =
   1    2    3    4    5    6    7
>> V1=[1 :7]                   % Vecteur par intervalle si l'incrément =1 par défaut
v1 =
   1    2    3    4    5    6    7
>> V2=[1:0.5:3]                % Vecteur par description intervalle avec un incrément réel
V2 =
   1.0000    1.5000    2.0000    2.5000    3.0000
>> V3=[10:-2:0]                % Vecteur par intervalle avec un incrément négatif
V3 =
   10    8    6    4    2    0
>> V4=[2;5;8;4]                % Vecteur colonne en utilisant le « ; »
V4 =
   2
   5
   8
   4
```

II.3.2. Génération de vecteurs

Pour générer de grands vecteurs, on n'utilise pas l'énumération, on utilise plusieurs modes de construction de vecteurs.

La fonction linspace

y = **linspace**(x1,x2,n) génère n éléments. L'espacement entre les éléments est obtenu par l'expression : (x2-x1)/(n-1).

linspace est similaire à l'opérateur deux-points ":", mais donne un contrôle direct sur le nombre d'éléments à créer et inclut toujours les éléments de terminaison (x1 et x2).

"lin" dans le nom "**linspace**" fait référence à la génération de valeurs espacées linéairement par opposition à la fonction **logspace**, qui génère des valeurs espacées de manière logarithmique.

```
>> V5=linspace(1,2,8)      % génération linéaire de 8 éléments entre 1 et 2
V5 =
   1.0000   1.1429   1.2857   1.4286   1.5714   1.7143   1.8571   2.0000
>> V6=logspace(1,2,8)  % génération logarithmique de 8 éléments entre 1 et 2
V6 =
  10.0000  13.8950  19.3070  26.8270  37.2759  51.7947  71.9686 100.0000
```

La fonction rand

La fonction rand initialise des vecteurs lignes (rand(1, k)) ou colonnes (rand(1, k)) par des valeurs aléatoires entre 0 et 1 :

```
>> V7=rand(1,6)      %génération d'un vecteur ligne de 6 valeurs aléatoires entre 0 et 1
V7 =
   0.8147   0.9058   0.1270   0.9134   0.6324   0.0975
>> V8=rand(5,1)      %génération d'un vecteur  colonne de 5 valeurs aléatoires entre 0 et 1
V8 =
   0.9572
   0.4854
   0.8003
   0.1419
   0.4218
>> V9=10*rand(1,6) % à chaque exécution d'un rand, on génère de nouvelles valeurs
V9 =
   2.7850   5.4688   9.5751   9.6489   1.5761   9.7059
```

La fonction zeros

Cette fonction initialise un vecteur à des valeurs nulles (tous les éléments seront tous des zéros). Cette fonction sert généralement pour attribuer des données de départ pour de calculs de somme.

La fonction ones

Cette fonction initialise un vecteur à des valeurs égales à 1. Cette fonction sert généralement pour attribuer des données de départ pour de calculs de produit.

```
>> W1=zeros(1,6)      %génération d'un vecteur ligne de 6 valeurs nulles
W1 =
   0   0   0   0   0   0
>> W2=zeros(5,1)      %génération d'un vecteur  colonne de 5 valeurs aléatoires entre 0 et 1
W2 =
   0
   0
   0
   0
   0
>> W3=ones(1,6)       %génération d'un vecteur ligne de 6 valeurs égales à 1
W3 =
   1   1   1   1   1   1
>> W4=ones(5,1)       %génération d'un vecteur  colonne de 5 valeurs égales à 1
W4 =
   1
   1
   1
   1
   1
```

II.4. Opérations sur Les Vecteurs

II.4.1. Somme et soustraction de deux vecteurs

L'avantage du Matlab c'est d'utiliser les vecteurs et matrices comme des opérandes simples dans des opérations arithmétiques à l'opposé aux langages de programmations structurelles, où il faut parcourir les éléments du vecteur ou de la matrice par l'utilisation des indices de cases par des instructions répétitives (boucles). Vous pouvez additionner ou soustraire deux vecteurs. Les deux vecteurs d'opérandes doivent être du même type et avoir le même nombre d'éléments.

```
>> V1=[1 3 5 7 9]      %création de V1 de 5 éléments par énumération
V1 =
   1   3   5   7   9
>> V2=[2:2:10]         %création de V2 de 5 éléments par intervalle
V2 =
   2   4   6   8   10
>> Vs=V1+V2            % Vs : Vecteur somme de V1+V2
Vs =
   3   7   11   15   19
>> Vd=V2-V1            % Vd : Vecteur différence de V2+V1
Vd =
   1   1   1   1   1
```

II.4.2. Multiplication scalaire de vecteurs

Lorsque on multiplie un vecteur par un nombre, cela s'appelle la multiplication scalaire. La multiplication scalaire produit un nouveau vecteur de même type avec chaque élément du vecteur d'origine multiplié par le nombre.

```
>> V3=linspace(1,5,6)          % Création du vecteur V3 de 6 éléments entre 1 et 5
V3 =
   1.0000   1.8000   2.6000   3.4000   4.2000   5.0000
>> V4=4*V3                    % multiplication par un scalaire (valeur 4)
V4 =                          % Résultat de l'opération est un vecteur du même nombre d'éléments
   4.0000   7.2000  10.4000  13.6000  16.8000  20.0000
>> k=3;
>> V5=V3*k          % ici on multiplie le vecteur V3 par une variable scalaire k
V5 =
   3.0000   5.4000   7.8000  10.2000  12.6000  15.0000
```

II.4.3. Transposée d'un vecteur

L'opération de transposée, d'un vecteur ou d'une matrice, transforme un vecteur colonne en vecteur ligne et vice versa. L'opération de transposition est représentée par une apostrophe (') ou par la fonction standard **transpose.**

```
>> V6=V3'          % transposée du vecteur V3 par l'utilisation du ( ' )
V6 =
   1.0000
   1.8000
   2.6000
   3.4000
   4.2000
   5.0000
>> V7=transpose(V6)    % transposée du vecteur V6 par l'utilisation de la fonction transpose.
V7 =
   1.0000   1.8000   2.6000   3.4000   4.2000   5.0000
```

II.4.4. Opérations de multiplication et de division sur des vecteurs

Sous Matlab, il faut faire très attention aux tailles des vecteurs lors des opérations de multiplication et de division (sauf dans le cas d'élément par éléments).

```
>> V1,V2          %affichage du contenu des vecteurs V1 et V2
V1 =
    1    3    5    7    9
V2 =
    2    4    6    8   10
>> V1*V2           %essai de produit de deux vecteurs V1 et V2
Error using  *       %affichage un message d'erreur Problème de dimensions de  V1 et V2
Inner matrix dimensions must agree.
 % V1(1 ligne, 5 colonnes) et V2 (1 ligne, 5 colonnes), donc on ne peut pas exécuter le
produit de deux vecteurs.
 >> Ps=V1*V2'    % V2' devient (5 lignes, 1 colonne), comme ça on peut exécuter l'opération
du produit matriciel
Ps =
   190
>> W1=V1.*V2     %opération d'une multiplication terme par terme
W1 =
    2   12   30   56   90
>> W2=V2./V1           %opération d'une division terme par terme
W2 =
   2.0000   1.3333   1.2000   1.1429   1.1111
>> W3=V1.\V2          %opération d'une division gauche terme par terme
W3 =
   2.0000   1.3333   1.2000   1.1429   1.1111
%les opérations terme par terme (ou élément par élément) donnent un vecteur résultat du
même nombre d'éléments.
```

II.4.5. Application des fonctions sur des vecteurs

Assemblage de vecteurs

Matlab nous permet d'assembler des vecteurs pour créer de nouveaux vecteurs. Si on a deux vecteurs lignes W1 et W2 avec n et m nombre d'éléments, pour créer un vecteur ligne W de (n+m) éléments.

```
>> W1,W2      %affichage des vecteurs W1 et W2
W1 =
   1   3   5   7   9
W2 =
   2   4   6   8   10
>> W=[W1,W2]        %assemblage des vecteurs W1 et W2 dans W en utilisant la (,)
W =
   1   3   5   7   9   2   4   6   8   10
>> W=[W1 W2]       %assemblage des vecteurs W1 et W2 dans W en utilisant l'espace
W =
   1   3   5   7   9   2   4   6   8   10
>> Wm=[W1;W2]   %assemblage des vecteurs W1 et W2 sous forme d'une matrice dans
Wm en utilisant le ( ;)
Wm =
   1   3   5   7   9
   2   4   6   8   10
```

Taille d'un vecteur

Pour savoir le nombre d'éléments d'un vecteur V, on applique la commande **length**(V) soit affectée à une variable ou par génération du résultat **ans.** On utilise aussi la commande **size(V).** En utilisant cette commande, on aura 2 valeurs comme résultat. La première valeur c'est le nombre de lignes et la seconde c'est le nombre de colonnes.

Pour les vecteurs lignes, on aura (1, nbc). Pour les vecteurs colonnes (nbl, 1).

```
>> V=[1:3:12]          %création de V par intervalle
V =
   1   4   7   10
>> nb=length(V)      %nombre d'éléments du vecteur V affecté à la variable nb
nb =
   4
>> length(V)         %nombre d'éléments du vecteur V affecté à la variable ans
ans =
   4
>> size(V)
ans =                  % résultat une ligne et 4 colonnes
   1   4
```

Fonction trigonométrique sur un vecteur

```
>> Ws=sin(V)    %appliquer le sinus à un vecteur, on aura un vecteur comme résultat
Ws =
   0.8415   -0.7568    0.6570   -0.5440
>> Wc=cos(V)     % les valeurs du vecteur seront en radians
Wc =
   0.5403   -0.6536    0.7539   -0.8391
>> VV=acos(Ws)
VV =
   1.0000    2.2832    0.7168    2.5664
```

Opérations statistiques sur un vecteur

```
>> X=[1:2:9]                %Création d'un vecteur X par intervalle
X =
     1.00        3.00        5.00        7.00        9.00
>> max(X)                   %le maximum du vecteur X
ans =
     9.00
>> min(X)                   %le minimum du vecteur X
ans =
     1.00
>> mean(X)                  %la moyenne du vecteur X
ans =
     5.00
>> sum(X)                   %la somme des valeurs du vecteur X
ans =
    25.00
>> prod(X)                  %le produit des valeurs du vecteur X
ans =
   945.00
>> std(X)                   %l'écart type des valeurs du vecteur X
ans =
     3.16
>> var(X)                   %la variance des valeurs du vecteur X
ans =
    10.00
```

II.5. Applications du chapitre II

II.5.1. Application 1

Soient t les vecteurs $X1 = (5\ ,5\ ,5\ ,5\ ,5\ ,5)$ et $X2 = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$

Créer les deux vecteurs (X1 et X2) par deux méthodes :

```
>> X1=[5 5 5 5 5 5]            %1ère méthode pour créer X1 (énumération)
X1 =
   5    5    5    5    5    5
>> X1=5*ones(1,6)              %2ème méthode pour créer X1 (utilisation de ones)
X1 =
   5    5    5    5    5    5
>> X2=[2;4;6]                  %1ère méthode pour créer X2 (énumération colonne)
X2 =
   2
   4
   6
>> X2=[2:2:6]'                 %2ème méthode pour créer X2 (utilisation de la transposée)
X2 =
   2
   4
   6
```

II.5.2. Application 2

Soient X3 et X4 deux vecteurs de valeurs aléatoires entre 0 et 10.

Donner la taille de chaque vecteur ? modifier le 2ème élément de X3 par la valeur 3.5 ; annuler les 3 premiers éléments de X4 ; Assembler les deux vecteurs linéairement ;

```
>> X3=10*rand(1,5)          % création de X3 de valeurs aléatoires entre 0 et 10
X3 =
   8.1472   9.0579   1.2699   9.1338   6.3236
>> X4=10*rand(1,4)
X4 =
   0.9754   2.7850   5.4688   9.5751
>> length(X3)
ans =
    5
>> length(X4)                % taille du vecteur X4
ans =
    4
>> X3(2)=3.5                 % remplacement de le 2ème élément de X3 par la valeur 3.5
X3 =
   8.1472   3.5000   1.2699   9.1338   6.3236
>> X4(1:3)=0                 % annulation des 3 premiers éléments de X4
X4 =
     0     0     0   9.5751
>> X5=[X3 X4]                %Assemblage linéaire de X3 et X4
X5 =
   8.1472   3.5000   1.2699   9.1338   6.3236        0        0        0   9.5751
```

II.5.3. Application 3

Soit P un vecteur contenant les hauteurs de pluie (valeurs aléatoires entre 0 et 20) du mois de janvier.

- Calculer la pluie mensuelle totale et la moyenne de janvier ;
- Trouver les valeurs maximale et minimale de janvier ;
- Extraire les 10 premières valeurs dans P1 et les dernières valeurs dans P2

```
>> P=20*rand(1,30)
P =
  Columns 1 through 10
      19.30       3.15      19.41      19.14       9.71      16.01       2.84       8.44      18.31      15.84
  Columns 11 through 20
      19.19      13.11       0.71      16.98      18.68      13.57      15.15      14.86       7.84      13.11
  Columns 21 through 30
       3.42      14.12       0.64       5.54       0.92       1.94      16.47      13.90       6.34      19.00
>> pt=sum(P)
pt =
     347.68
>> Pm=mean(P)
Pm =
      11.59
>> mx=max(P)
mx =
      19.41
>> mn=min(P)
mn =
       0.64
>> P1=P(1:10)
P1 =
      19.30       3.15      19.41      19.14       9.71      16.01       2.84       8.44      18.31      15.84
>> P2=P(21:end)
P2 =
    3.42    14.12     0.64      5.54      0.92      1.94     16.47     13.90      6.34     19.00
```

II.6. Conclusion

Comme conclusion à ce chapitre, on peut dire qu'un rappel-définition sur des tableaux à une dimension (les vecteurs) a été exposé. Par la suite, une illustration de la construction et la génération des vecteurs ont été faites. Les différentes fonctions, utilisées lors des manipulations des vecteurs, sont montrées avec des exemples bien détaillés. Trois (03) applications avec corrigé sur la manipulation (opérations et caractéristiques) sur des vecteurs ont été formulées, à la fin de ce chapitre, avec des détails. L'étude des matrices (tableaux à deux dimensions) fera l'objet du chapitre suivant.

Chapitre III : les matrices

III.1. Définition et Introduction

Une matrice contient des informations de même type, organisées en lignes et en colonnes, elle est caractérisée par :

- un nom : identificateur unique pour tous ses éléments ;
- un nombre de lignes (m) et nombre de colonnes (n);
- ses 2 indices : (i, j) pour accéder à un élément de la matrice.

Dans ce chapitre, nous allons aborder un point important : les matrices ! C'est important, parce qu'historiquement MATLAB a été développé pour manipuler des matrices, et la plupart des fonctions tiennent compte du fait que potentiellement elles puissent être appliquées à des matrices. La création d'une matrice est réalisée : en entrant des éléments dans chaque ligne sous forme de nombres délimités par des virgules ou des espaces et en utilisant des points-virgules pour marquer la fin de chaque ligne. Les cas particuliers sont les vecteurs colonnes (lorsque n = 1) et les vecteurs lignes (lorsque m = 1).

III.2. Création des matrices

III.2.1. Construction des matrices

Pour construire une matrice dans MATLAB, on doit :

- commencer par un crochet, [
- séparer les éléments dans une rangée avec des espaces ou des virgules (,)
- utilisez un point-virgule (;) pour séparer les lignes
- terminer la matrice par un autre crochet,].

$A=[a_{11}, a_{12}, a_{13}, \ldots, a_{1n}\ ;\ a_{21}, a_{22}, a_{23}, \ldots a_{2n}\ ;\ \ldots\ ;\ a_{m1}, a_{m2}, a_{m3}, \ldots a_{mn}]$

Il faut faire attention, on doit avoir toujours le même nombre d'éléments dans chaque ligne.

```
>> Mat1=[1 3 2;5 6 7;4 9 8]   % création de Mat1 par énumération de 3 lignes et 3 colonnes
Mat1 =
   1   3   2
   5   6   7
   4   9   8
>> Mat2=[1,3,2;5,6,7]   % création de Mat2 en utilisant l'espace pour séparer les éléments
Mat2 =
   1   3   2
   5   6   7
```

Ou de créer ce type de tableau par des intervalles égaux

A=[1:5; 1:2:9 ;5:-1:1;…] , l'important c'est d'avoir dans chaque ligne le même nombre d'éléments.

```
>> Mat3=[1:4;2:2:8;10:-3:1]  % création de Mat3 par intervalles 4 éléments par ligne
Mat3 =
    1    2    3    4
    2    4    6    8
   10    7    4    1
```

par assemblage (agrégation) de vecteurs créés :

```
>> X1=[1   5   8   3]     % création de X1 par énumération de 4 éléments entiers
X1 =
    1    5    8    3
>> X2=[7:-0.5:5.5]      % création de X2 par intervalle de 4 éléments réels
X2 =
   7.0000    6.5000    6.0000    5.5000
>> X3=linspace(2,5,4)      % création de X3 de 4 éléments par la commande linspace
X3 =
    2    3    4    5
>> Mat4=[X1;X2;X3]    % création de Mat4 par assemblage des 3 vecteurs créés
Mat4 =
   1.0000    5.0000    8.0000    3.0000
   7.0000    6.5000    6.0000    5.5000
   2.0000    3.0000    4.0000    5.0000
% on remarque que les éléments de Mat4 sont de type réel par le fait que les éléments d'un
des 3 vecteurs sont réels.
```

III.2.2. Génération des matrices

Pour générer de grandes matrices en une seule commande, on n'utilise pas l'énumération ou les intervalles, on utilise les commandes déjà présentées dans le chapitre des vecteurs. Il suffit de mettre la commande en deux dimensions (remplacer le 1, de la création des vecteurs, par un autre nombre>1).

Quelques commandes de génération de matrices sont données ci-après :

zeros(n) : Génère une matrice carrée (n × n) avec tous les éléments = 0

zeros(m,n) : Génère une matrice m × n avec tous les éléments = 0

ones(n) : Génère une matrice carrée (n × n) avec tous les éléments = 1

ones(m,n) : Génère une matrice m × n avec tous les éléments = 1

eye(n) : Génère une matrice identité de dimension n × n

magic(n) : Génère une matrice magique de dimension n × n

rand(m,n) : Génère une matrice de dimension m × n de valeurs aléatoires entre 0 et 1.

```
>> M1=zeros(3)     %génération d'une matrice carrée de 3×3 de valeurs nulles
M1 =
   0   0   0
   0   0   0
   0   0   0
>> M1=zeros(3,3)   %génération d'une matrice carrée de 3×3 de valeurs nulles
M1 =
   0   0   0
   0   0   0
   0   0   0
>> M1=zeros(3,2) %génération d'une matrice rectangulaire  de 3×2 de valeurs nulles
M1 =
   0   0
   0   0
   0   0
>> M2=ones(3,2)   %génération d'une matrice rectangulaire  de 3×2 de valeurs =1
M2 =
   1   1
   1   1
   1   1
>> M3=rand(5,3)  %génération d'une matrice  de 5×3 de valeurs aléatoires entre 0 et 1
M3 =
   0.7577   0.7060   0.8235
   0.7431   0.0318   0.6948
   0.3922   0.2769   0.3171
   0.6555   0.0462   0.9502
   0.1712   0.0971   0.0344
>> I1=eye(3)       %génération d'une matrice carrée d'identité 3×3 où les valeurs de la diagonales =1
I1 =
   1   0   0
   0   1   0
   0   0   1
>> M4=magic(4)  %génération d'une matrice carré magique d'ordre 4 où la somme, des éléments sur
les lignes, les colonnes et les diagonales, est la même. Ici la somme = 34
M4 =
  16    2    3   13
   5   11   10    8
   9    7    6   12
   4   14   15    1
```

III.3. Accès aux éléments d'une matrice

Une fois la matrice créée, elle est automatiquement stockée et mémorisée dans le ***Workspace***. On peut y référer simplement en tant que matrice A. On peut alors visualiser un élément de cette matrice en spécifiant son emplacement.

Pour référencer un élément dans la $i^{ème}$ ligne et la $j^{ème}$ colonne de la matrice A, On écrit : A(i,j) Où :

A : l'identificateur de la matrice ;

i : numéro de la ligne (1≤i≤m) ;

j :numéro de la colonne ((1≤j≤n)

Il est utile de noter les possibilités suivantes :

- L'accès à un élément de la ligne i et la colonne j se fait par : A(i,j) déjà présenté ci-dessus ;
- L'accès à toute la ligne numéro i se fait par : A(i,:)
- L'accès à une partie d'une ligne i :A(i,2 :4)
- L'accès à toute la colonne numéro j se fait par : A(:,j)
- L'accès à une partie de la matrice se fait par : A(1:3,2:4)
- L'accès à une sous matrice A($[i_1,i_2]$,$[j_1,j_2]$) la sous matrice : exemple A([1,3],[2,4]) lignes(1,3) et colonnes (2,4).

N.B : On peut aussi accéder aux éléments d'une matrice par un indice unique qui est leur ordre de la matrice. Le premier élément d'une matrice est celui de la $1^{ère}$ ligne et la $1^{ère}$ colonne (1, 1). Le second élément est le (2,1) ainsi de suite.

En général, on ne sert pas de cette méthode. Mais certaines fonctions (exemple ***find***) renvoient, comme résultat, les numéros d'ordre de certains éléments d'une matrice.

Exemple :

```
>> A=[7 4 3 ; 5 8 6]          % création de A de dimension (2×3)
A =
      7.00(1)    4.00(3)    3.00(5)
      5.00(2)    8.00(4)    6.00(6)
>> A(2)                % afficher du 2ème élément de A avec indice unique
ans =
      5.00
>> A(4)                % afficher du 4ème élément de A avec indice unique
ans =
      8.00
```

```
>> A=[2:5;7 8 9 6;linspace(4,8,4)]    %Création de la matrice A
A =
   2.0000   3.0000   4.0000   5.0000
   7.0000   8.0000   9.0000   6.0000
   4.0000   5.3333   6.6667   8.0000
>> A(3,2)              %Accès à l'élément de la 3ème ligne et 2ème colonne de la matrice A
ans =
   5.3333
>> A(2,:)       %Accès à toute la 2ème ligne de la matrice A
ans =
    7    8    9    6
>> A(:,3)           %Accès à toute la 3ème colonne de la matrice A
ans =
   4.0000
   9.0000
   6.6667
>> A(1:2,2:3)     %Accès à la sous matrice délimitée pour les lignes de la 1ère jusqu'à la 2ème
                    et de la 2ème  à la 3ème pour les colonnes de la matrice A
ans =
    3    4
    8    9
>> A([1,3],[2,4])    %Accès à la sous matrice pour les lignes n°1 et n°3  et pour les colonnes
la n°2 et n°4 de  la matrice A
ans =
   3.0000   5.0000
   5.3333   8.0000
```

III.4. Opérations sur les matrices

Dans cette partie, on présentera les opérations matricielles de base et couramment utilisées. On peut citer les opérations suivantes :

- Addition et soustraction de matrices ;
- Opérations scalaires des matrices ;
- Transposée d'une matrice ;
- Matrices de concaténation ;
- Multiplication matricielle ;
- Déterminant d'une matrice ;
- Inverse d'une matrice.

III.4.1. Addition et soustraction des Matrices

Les opérations de base appliquées aux matrices sont l'addition et la soustraction. Les deux matrices d'opérandes doivent avoir le même nombre de lignes et de colonnes.

```
>> A=[1 4 7 8 ;1:2:7]   %Création d'une matrice A de 2 lignes et 4 colonnes
A =
   1   4   7   8
   1   3   5   7
>> B=2*ones(2,4)     %Création de la matrice B de (2×4)  à l'aide de la commande ones
B =
   2   2   2   2
   2   2   2   2
>> S=A+B            % S est la somme des deux matrices A et B de dimensions (2×4)
S =
   3   6   9  10
   3   5   7   9
>> D=A-B          % D est la matrice différence entre les éléments de A et B
D =
  -1   2   5   6
  -1   1   3   5
```

III.4.2. Opérations scalaires par des matrices

Lorsque on additionne, soustrait, multiplie ou divise une matrice par un nombre, cela s'appelle l'opération scalaire sur les matrices.

Les opérations scalaires produisent une nouvelle matrice avec le même nombre de lignes et de colonnes avec chaque élément de la matrice d'origine ajouté, soustrait, multiplié ou divisé par le nombre.

```
>> format bank    %utilisation du format d'affichage bank (2 chiffres après la virgule)
>> A=10*rand(3)       %Création d'une matrice carrée A d'ordre 3 d'éléments aléatoires
A =
       8.15       9.13       2.78
       9.06       6.32       5.47
       1.27       0.98       9.58
>> A*3                %multiplication des éléments de A par le scalaire 3
ans =
      24.44      27.40       8.35
      27.17      18.97      16.41
       3.81       2.93      28.73
>> A/2          %division des éléments de A sur le scalaire 2
ans =
       4.07       4.57       1.39
       4.53       3.16       2.73
       0.63       0.49       4.79
>> A-2          %soustraction des éléments de A par le scalaire 2
ans =
       6.15       7.13       0.78
       7.06       4.32       3.47
      -0.73      -1.02       7.58
>> sin(A)      %application de la fonction sin sur les éléments de A  (éléments en Radians)
ans =
       0.96       0.29       0.35
       0.36       0.04      -0.73
       0.96       0.83      -0.15
%dans tous les cas, on va avoir un résultat matrice de même dimension que A
```

III.4.3. Transposée d'une matrice

L'opération de transposition d'une matrice change les lignes et les colonnes d'une matrice. Il est représenté par un guillemet simple (') ou en utilisant la commande ***transpose***(A)

```
>> A        %affichage de la matrice A créée précédemment
A =
       8.15      9.13      2.78
       9.06      6.32      5.47
       1.27      0.98      9.58
>> A'              % transposée de matrice A par l'utilisation du ( ' )
ans =
       8.15      9.06      1.27
       9.13      6.32      0.98
       2.78      5.47      9.58
>> transpose(A')    % la transposée  en utilisant de la fonction transpose.
ans =
       8.15      9.13      2.78
       9.06      6.32      5.47
       1.27      0.98      9.58
```

III.4.4. Agrégation des matrices (concaténation)

On peut assembler (concaténer) deux matrices pour créer une matrice plus grande. La paire de crochets '[]' est l'opérateur de concaténation (d'assemblage).
MATLAB autorise deux types de concaténations :

- Concaténation horizontale ;
- Concaténation verticale.

Lorsque on concatène deux matrices en séparant celles-ci à l'aide de virgules, elles sont simplement ajoutées horizontalement. C'est ce qu'on appelle la concaténation horizontale. On doit avoir le même nombre de lignes dans les deux matrices.
Sinon, si on assemble deux matrices en les séparant par des points-virgules, elles sont ajoutées verticalement. C'est ce qu'on appelle la concaténation verticale. L'exigence principale c'est d'avoir le même nombre de colonnes dans les deux matrices.

```
>> A=10*rand(3,4)     %création de la matrice A de (3×4)
A =
      9.65      9.57      1.42      7.92
      1.58      4.85      4.22      9.59
      9.71      8.00      9.16      6.56

>> B=magic(3)        %création de la matrice carrée B par la commande magic
B =
      8.00      1.00      6.00
      3.00      5.00      7.00
      4.00      9.00      2.00
>> C=3*ones(4)       %création de la matrice carrée c par la commande ones
C =
      3.00      3.00      3.00      3.00
      3.00      3.00      3.00      3.00
      3.00      3.00      3.00      3.00
      3.00      3.00      3.00      3.00
>> Ah=[A,B]     %assemblage horizontal des deux matrices A et B
Ah =                 La matrice A                        La matrice B
      9.65      9.57      1.42      7.92      8.00      1.00      6.00
      1.58      4.85      4.22      9.59      3.00      5.00      7.00
      9.71      8.00      9.16      6.56      4.00      9.00      2.00
>> Av=[A;C]    %assemblage vertical des deux matrices A et C
Av =
      9.65      9.57      1.42      7.92
      1.58      4.85      4.22      9.59   } La matrice A
      9.71      8.00      9.16      6.56
      3.00      3.00      3.00      3.00
      3.00      3.00      3.00      3.00
      3.00      3.00      3.00      3.00   } La matrice C
      3.00      3.00      3.00      3.00
```

III.4.5. Produit matriciel

Considérons deux matrices A et B, si A est une matrice de dimension (m x n) et B est une matrice de (n x p), elles peuvent être multipliées ensemble pour produire une matrice C de dimension (m x p). La multiplication matricielle n'est possible que si le nombre de colonnes de A est égal au nombre de lignes de B.

Dans le produit matriciel, les éléments des lignes de la première matrice sont multipliés par les colonnes correspondantes de la seconde matrice.

Chaque élément de la (i, j)ème position, dans la matrice résultante C, est la somme des produits des éléments de la ième ligne de la première matrice avec l'élément correspondant de la jième colonne de la deuxième matrice.

Le produit matriciel dans MATLAB est effectué à l'aide de l'opérateur « * ».

```
>> A=[2:2:8;linspace(3,6,4);9 7 5 3 ]   % matrice A de dimensions(3×4)
A =
   2   4   6   8
   3   4   5   6
   9   7   5   3
>> B=fix(10*rand(3))      % matrice B de dimensions(3×3) (partie entière des valeurs
aléatoires entre 0 et 10)
B =
   0   6   3
   8   7   6
   9   7   1
>> P=A*B     % demande un produit matriciel de A*B
Error using  *          % génération d'un message d'erreur car le nombre de colonnes de A
Inner matrix dimensions must agree.       % n'est pas égal au nombre de lignes de B
>> P=B*A            % produit matriciel B*A génère un la matrice P (ça passe)
P =
  45   45   45   45
  91  102  113  124
  48   71   94  117
```

III.4.6. Fonctions usuelles appliquées aux matrices

MATLAB fournit plusieurs fonctions pour manipuler les matrices, trier, inverser, calculer le déterminant, déterminer les dimensions, permuter, remodeler ou décaler contenu de la matrice. Le tableau suivant résume les principales fonctions affectant ou effectuant des opérations sur des matrices. Noter que les fonctions scalaires courantes, (sin, exp, etc...) peuvent aussi s'appliquer à des matrices.

Tableau III.1: *Fonctions appliquées aux matrices*

Fonction	Rôle
size(A)	Dimension de la matrice (n et m)
length(A)	La plus grande dimension de la matrice (n ou m)
sin(A)	Déjà expliquée
numel(A)	Nombre d'éléments de la matrice ou (n*m)
find(A)	Localise tous les éléments non nuls de A et renvoie leurs positions.
find(A>5)	Localise (par leurs indices) les éléments >5 dans A
diag(A)	Retourne le vecteur de la diagonale d'une matrice carrée
sort(A)	Trie les éléments d'une matrice en ordre croissant ou décroissant
reshape(V,n1,n2)	Remodeler le vecteur V sous forme de matrice en n1 lignes et n2 colonnes
sum(A)	Retourne un vecteur somme des colonnes de A
sum(A,1)	Retourne un vecteur somme des colonnes de A
sum(A,2)	Retourne un vecteur somme des lignes de A
det(A)	Retourne le déterminant de la matrice A
inv(A)	Calcule la matrice inverse de A
triu(A)	Retourne la partie supérieure de A

tril(A)	Retourne la partie inférieure de A
rank(A)	Retourne le nombre de dimensions de A (vecteur =1, matrice=2)
min(A)	Retourne un vecteur minimum de chaque colonne (même chose max…)
eig(A)	Renvoie un vecteur des valeurs propres de la matrice A.
trace(A)	Retourne la somme de la diagonale
A(:)	Affiche la matrice A sous forme d'un vecteur colonne
A(:)'	Affiche la matrice A sous forme d'un vecteur rangée

```
>> A=linspace(1,10,16)      %création d'un vecteur linéaire de 16 valeurs entre 1 et 10
A =
1.00  1.60  2.20  2.80  3.40  4.00  4.60  5.20  5.80  6.40  7.00 7.60  8.20  8.80  9.10 10.00
 >> B=reshape(A,4,4)       %remodeler le vecteur A sous forme d'une matrice carrée B
B =
   1.00   3.40   5.80   8.20
   1.60   4.00   6.40   8.80
   2.20   4.60   7.00   9.40
   2.80   5.20   7.60  10.00
>> diag(B)          %affiche le vecteur de la diagonale de B
ans =
    1
    4
    7
   10
>> trace(B)        %la somme de la diagonale de B
ans =
   22
>> eig(B)           %les valeurs propres de B
ans =
  23.2393
  -1.2393
  -0.0000
   0.0000
>> sum(B)         %le vecteur  somme des colonnes de B
ans =
   7.6000  17.2000  26.8000  36.4000
>> det(B)          % le déterminant de B
ans =
  3.4079e-30
>> triu(B)          %affiche la partie supérieure de B (sous la diagonale =0)
ans =
   1.0000   3.4000   5.8000   8.2000
        0   4.0000   6.4000   8.8000
        0        0   7.0000   9.4000
        0        0        0  10.0000

>> size(B)        %affiche le nombre de lignes et de colonnes de B
ans =
   4    4
```

III.5. Application du chapitre III

III.5.1. Application 1

a. Créer la matrice suivante avec l'utilisation des intervalles (une seule commande)

$$A=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 0.3 & 0.7 & 1.1 & 1.5 & 1.9 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 \\ 15 & 12 & 9 & 6 & 3 \\ -8 & -5 & -2 & 1 & 4 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

b. Créer les sous-matrices suivantes à partir de la matrice A (une seule commande pour chaque matrice)

$$B=\begin{pmatrix} -8 & 1 & 4 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad C=\begin{pmatrix} 0.7 & 1.1 & 1.5 \\ 0.8 & 0.7 & 0.6 \end{pmatrix} \qquad D=\begin{pmatrix} 0.3 & 1.1 & 1.9 \\ 0.9 & 0.7 & 0.5 \\ 15 & 9 & 3 \end{pmatrix}$$

c. Donner les caractéristiques suivantes (size(A), find(A==1), sum(A), la diagonale de D et le nombre d'éléments de B)

III.5.2. Application 2

a. Créer Mat une matrice carrée (4×4) d'éléments entiers aléatoires entre 0 et 20 ;
b. Extraire toute la 1ère ligne dans le vecteur V1, la dernière colonne dans V2 ;
c. Assembler (Concaténer) une ligne composée de valeurs= 1 à la fin de Mat ;
d. Annuler les valeurs de la 3ème ligne de Mat.

III.5.3. Application 3

Soit un système de réseau d'adduction de 7 conduites, Vnum, Vdeb et Vlong constituent les vecteurs numéros des conduites, leurs débits et longueurs respectifs. Réaliser les tâches suivantes dans le ***command windows***:

1. Calculer leurs diamètres (Vdiam) respectifs en utilisant D=$\sqrt{Q}$;
2. Créer le vecteur Vvit pour les 7 conduites ; V=(Q/S)
3. Rassembler tous les vecteurs colonnes sous forme d'une matrice A de 5 colonnes ; (Vnum Vdeb Vlong Vdiam Vvit)
4. Trouver la Vitesse Moyenne des 7 conduites ;
5. Trouver le petit diamètre du réseau d'adduction.

```
>> A=[1:5;0.3:0.4:1.9;0.9:-0.1:0.5;15:-3:3;-8:3:4;ones(1,5);zeros(1,5)]
A =
      1.00      2.00      3.00      4.00      5.00
      0.30      0.70      1.10      1.50      1.90
      0.90      0.80      0.70      0.60      0.50
     15.00     12.00      9.00      6.00      3.00
     -8.00     -5.00     -2.00      1.00      4.00
      1.00      1.00      1.00      1.00      1.00
         0         0         0         0         0
>> B=A(5:7,[1 4 5])    %extraction de B à partir de A, les lignes (5 6 7) et les colonnes (1 4 5)
B =
     -8.00      1.00      4.00
      1.00      1.00      1.00
         0         0         0
>> C=A(2:3,2:4)       %extraction de C à partir de A, les lignes (2 et 3) et les colonnes (2 3 4)
C =
      0.70      1.10      1.50
      0.80      0.70      0.60
>> D=A(2:4,1:2:5)     %extraction de D à partir de A, les lignes (2 3 4) et les colonnes (1 3 5)
D =
      0.30      1.10      1.90
      0.90      0.70      0.50
     15.00      9.00      3.00
>> size(A)      %dimensions de A dans le vecteur ans de deux valeurs   7 lignes et 5 colonnes
ans =
      7      5
>> [nblg   nbcol]=size(A)     %dimensions de A dans deux scalaires nblg= 7 lignes et nbcol= 5 colonnes
nblg =
      7
nbcol =
      5
>> ind1=find(A==1)  %ind1 : vecteur des indices de A où les valeurs =1
ind1 =                    A =
     1                         1.00   (1)      2.00  (8)      3.00 (15)      4.00 (22)      5.00 (29)
     6                         0.30   (2)      0.70  (9)      1.10 (16)      1.50  (23)     1.90 (30)
    13                         0.90   (3)      0.80  (10)      0.70  (17)      0.60  (24)     0.50 (31)
    20                        15.00 (4)      12.00 (11)      9.00 (18)      6.00 (25)      3.00 (32)
    26                        -8.00  (5)     -5.00  (12)     -2.00  (19)      1.00  (26)     4.00 (33)
    27                         1.00 (6)      1.00  (13)      1.00 (20)      1.00  (27)     1.00 (34)
    34                         0     (7)      0   (14)         0   (21)         0    (28)      0 (35)
>> vectsom=sum(A)       % vectsom : vecteur somme des colonnes de la matrice A
vectsom =
     10.20     11.50     12.80     14.10     15.40
>> diagD=diag(D)      % diagD : vecteur colonne de la diagonale de la matrice D
diagD =
      0.30
      0.70
      3.00
>> nbelB=numel(B)     % nbelB : scalaire renferme le nombre d'éléments de la matrice B
nbelB =
      9
```

```
>> Mat=fix(20*rand(4))    % création de la matrice Mat
Mat =
      16.00      12.00      19.00      19.00
      18.00       1.00      19.00       9.00
       2.00       5.00       3.00      16.00
      18.00      10.00      19.00       2.00
>> V1=Mat(1,:)    % extraction du vecteur de la 1ère ligne
V1 =
      16.00      12.00      19.00      19.00
>> V2=Mat(:,end)       % extraction du vecteur colonne  de la dernière colonne
V2 =
      19.00
       9.00
      16.00
       2.00
>> Mat=[Mat;ones(1,4)]    % ajout d'une ligne de valeurs = 1 à la fin de la matrice Mat
Mat =
      16.00      12.00      19.00      19.00
      18.00       1.00      19.00       9.00
       2.00       5.00       3.00      16.00
      18.00      10.00      19.00       2.00
       1.00       1.00       1.00       1.00
>> Mat(3,:)=0            % annuler la 3ème ligne de Mat
Mat =
      16.00      12.00      19.00      19.00
      18.00       1.00      19.00       9.00
         0          0          0          0
      18.00      10.00      19.00       2.00
       1.00       1.00       1.00       1.00
```

```
>> Vnum=[1:7]       %création du vecteur Vnum : numéros de conduites
Vnum =
     1     2     3     4     5     6     7
>> Vdeb=rand(1,7)   %création du vecteur Vdeb : débits passant par les conduites (aléatoires)
Vdeb =
    0.71     0.03     0.28     0.05     0.10     0.82     0.69
>> Vlong=100*rand(1,7)   %création de Vlong : longueurs de conduites (aléatoires *100)
Vlong =
    93.40    67.87    75.77    74.31    39.22    65.55    17.12
>> Vdiam=sqrt(Vdeb)   %calcul du vecteur Vdiam : les diamètres calculés
Vdiam =
    0.84     0.18     0.53     0.21     0.31     0.91     0.83
>> Vvit=4*Vdeb./(pi*Vdiam.^2)      %calcul du vecteur Vvit : les vitesses
Vvit =
    1.27     1.27     1.27     1.27     1.27     1.27     1.27
>> A=[Vnum'   Vdeb'    Vlong'   Vdiam'   Vvit']    %assemblage des  5 vecteurs en une matrice
A =
     1     0.71    93.40     0.84     1.27
     2     0.03    67.87     0.18     1.27
     3     0.28    75.77     0.53     1.27
     4     0.05    74.31     0.21     1.27
     5     0.10    39.22     0.31     1.27
     6     0.82    65.55     0.91     1.27
     7     0.69    17.12     0.83     1.27
>>Vmoy = mean(Vvit)
Vmoy=
    1.27
>>DiamMin=min(Vdiam)
DiamMin =
    0.18
```

III.6. Conclusion

Dans ce chapitre, on a présenté un rappel général du principe des matrices. En suite la construction et la génération des matrices ont été illustrées. Les différentes fonctions, utilisées lors des manipulations des matrices, sont montrées avec des exemples bien détaillés. A la fin du chapitre, trois (03) applications corrigées sur la manipulation (opérations et caractéristiques) sur des matrices ont été exposées avec des explications. Après l'utilisation du ***command window*** dans la présentation des vecteurs et des matrices, on passera à la programmation en utilisant les fichiers Matlab.

Chapitre IV : Eléments de la programmation sous Matlab

IV.1. Introduction

Jusqu'ici, nous avons toujours considéré que l'on travaillait dans la fenêtre de commandes ***(Command Window)***, en créant des variables dans l'espace mémoire de MATLAB (le ***workspace***), et en utilisant différentes fonctions prédéfinies.

Cette approche est très pratique pour démarrer, ou pour réaliser de petits calculs, qui ne nécessitent pas trop de lignes de commandes. Mais elle n'est pas appropriée dès que l'on s'attaque à des problèmes complexes, qui demandent des manipulations de commandes plus structurées ou plus nombreuses.

On va voir dans ce qui suit, comment utiliser MATLAB comme un véritable langage de programmation, en passant des fichiers de commandes (que l'on peut sauvegarder et donc réutiliser), en écrivant nos propres fonctions et en utilisant des structures de contrôle [3,12].

Pour rappeler, Matlab propose deux modes de fonctionnement :

– **Mode interactif :** où Matlab interprète et exécute les commandes (instructions) directement après la saisie dans la sous fenêtre des commandes déjà utilisé ;

– **Mode exécutif :** il s'agit de l'exécution d'un programme (script) en langage Matlab (ligne par ligne) saisie dans un fichier avec l'extension "*.m"

Un script Matlab est un simple fichier texte, qui comporte une suite d'instructions ou de commandes MATLAB, toutes séparées par une virgule (,), un point-virgule (;) ou un retour à la ligne (touche entrée). Ces commandes sont identiques à celles que l'on peut employer directement dans la fenêtre ***command Window*** de MATLAB.

L'avantage de ce mode est que ces commandes sont stockées dans un fichier « *.m », il est donc aisé de les rappeler pour les réexécuter, même dans une nouvelle session de Matlab. Comme tout langage, Matlab possède aussi un certain nombre d'instructions syntaxiques (boucles simples, conditionnelles, etc...) et de commandes élémentaires (lecture, écriture, etc...).

IV.2. Démarrer un script Matlab

Pour la création des scripts Matlab, on utilisera toujours l'éditeur intégré au logiciel « ***Editor*** » qui se lance à partir de la fenêtre de commande en cliquant sur l'icône ***new M-file*** ou ***open file*** dans la barre des menus. Une fois le fichier créé et enregistré sous un nom valide, on peut l'exécuter en tapant son nom - sans le suffixe .m - dans la fenêtre des commandes.

La première étape est de créer un répertoire (ou en choisir un existant) où vous allez ranger les fichiers  créés. En utilisant l'onglet ***Current Directory***, on se place

dans le répertoire personnel (probablement sur le disque D:\). On crée ensuite un répertoire pour les projets Matlab à réaliser, par exemple « Mes_projets » (dans l'onglet ***Current Directory***, bouton droit de la souris et ***New → Folder***). Dans ce répertoire, on peut, par exemple, créer un sous-répertoire pour le 1er projet pratique : Projet1. Ensuite, on crée un nouveau m-file Matlab (menu principal, ***File → New → M-file***) ou en utilisant l'outil-bouton (New Script)

La commande ***edit*** tapée dans le ***Command Window*** déclenche aussi la création d'un nouveau script Matlab. Exemple : ***edit*** Fich1.m.

Apparaît alors une nouvelle fenêtre ressemblant à un éditeur de texte, c'est le ***M-file Editor***. Dans cette fenêtre, on rentre toutes les commandes qu'on désire que Matlab exécute.

Exemple d'un script Matlab :

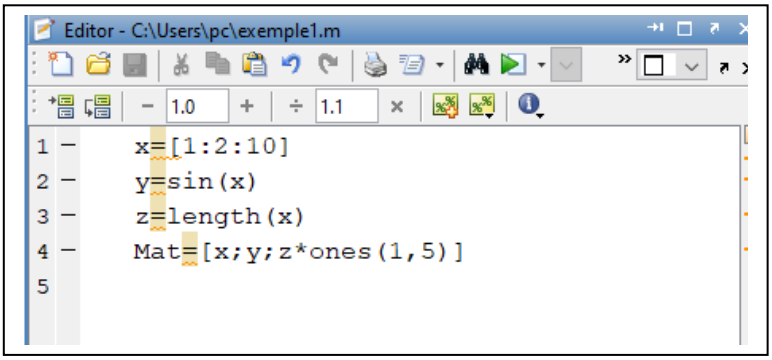


Figure IV.1: *Exemple d'un script Matlab dans la fenêtre Editor*

L'exécution du script Matlab se fait soit :

- par le ***Command Window*** en spécifiant juste le nom du script (du fichier) sans l'extension, c'est pourquoi, il est conseillé de choisir des noms explicites, permettant une identification rapide de l'utilité des scripts créés.

- ou bien en appuyant sur l'outil de l'exécution de l'éditeur de script.

NB : L'exécution d'un script Matlab est visible (à suivre) sur le ***Command Window*** pour les résultats numériques et textuelles, par contre les courbes et les graphiques sont générés dans la fenêtre ***figure*** (chapitre V).

IV.3. Les instructions de base Matlab

IV.3.1. Entrée des données – Lecture

On peut facilement demander à l'utilisateur de nous donner une valeur. Cette valeur peut alors être affectée à une variable. L'utilisateur peut saisir des informations au clavier grâce à la commande ***input***.

Sa syntaxe est la suivante : Vr = ***input*** ('message')

Où : Vr : la variable qui reçoit la donnée introduite

La donnée introduite peut être :

De n'importe quel type (nombre ou chaine de caractères) ;

Scalaire, vecteur ou matrice

input *:* instruction de lecture sous Matlab ;

('message') : texte associé à la lecture. Exemple : 'faites entrer la valeur' ou 'Vr= '

Exemple sur la fenêtre *editor* de Matlab :

```
1. clear, clc   %supprime toutes les variables du workspace et efface le Command Window
2. rayon=input('donner la valeur du rayon ');   %lecture d'un scalaire réel
3. surf=pi*rayon^2   % utilisation de la variable introduite dans une expression
4. peri=2*pi*rayon
```

L'exécution sur la fenêtre *Command Window*

```
donner la valeur du rayon 5    % affichage du message associé et  attente de l'introduction de la valeur, une fois la donnée est introduite, l'exécution continuera jusqu'à la fin du script.
surf =
      78.50
peri =
      31.40
```

IV.3.2. Sortie des variables - Ecriture

L'affichage des variables se fait par plusieurs manières (les plus utilisées) :

- par la spécification de leurs noms sans point-virgule ;
- la commande ***disp***(X), (pour *disp*lay), affiche une variable X (chaine de caractères, réel, …) sur la fenêtre de *Command Window*.

Si on veut afficher plusieurs variables du même type avec *disp*, on doit ajouter les crochets aux parenthèses de ***disp : disp*** ([x,y])

L'opérateur [ ] ne peut qu'afficher des variables de même type (numériques ou chaine de caractères).

Si l'on veut afficher des valeurs numériques avec des messages textuels associés, il faut les convertir en chaînes avec la commande ***num2str : Num***ber To ***Str***ing (convertir un numérique vers une chaine des caractères).

Exemple sur la fenêtre ***editor*** de Matlab :

```
1-  clear,clc
2-  forme=input('la forme est : '); % valeur attendue est une chaine de caractères
3-  rayon=input('donner la valeur du rayon '); % valeur attendue est un réel
4-  surf=pi*rayon^2;
5-  peri=2*pi*rayon;
6-  disp(surf)   %afficher le contenu de la variable surf
7-  peri         %afficher le contenu de la variable peri
8-  disp([surf,peri]) %afficher les deux variables réelles
9-  disp(['la forme est : ',forme])  %afficher le message et le contenu de la variable forme
10- disp(['Surface = ',num2str(surf),' Perimetre = ',num2str(peri)])  %afficher les messages et
    les contenus des variables avec conversion vers la chaine de caractères
```

L'exécution sur la fenêtre Command Window

```
la forme est : 'cercle'   % la forme est un cercle  (chaine de caractères il faut utiliser ls (‘’)
donner la valeur du rayon 4    % entrée d'une valeur numérique
      50.27  % affichage de surf
      25.13    % affichage de peri
      50.27       25.13     % affichage de surf et peri en même temps
la forme est : cercle    % affichage de deux textes : le message et la valeur de la variable forme

Surface = 50.27   Perimetre = 25.13  % affichage des textes et les variables résultat converties
```

Introduction et affichage d'un vecteur et d'une matrice :

```
1-  clear, clc
2-  vecteur=input('les valeurs du vecteurs sont : '); % ( ; ) donc le vecteur ne sera pas affiché
3-  matrice=input('les éléments de la matrice sont : ') % pas de ( ; ) donc la matrice sera
    affichée
4-  moy=mean(vecteur);
5-  som=sum(vecteur);
6-  s=size(matrice);
7-  Vc2=matrice(:,2)%affichage de la 2eme colonne
8-  disp([moy,som]) %les deux variables sont réelles
9-  disp(['taille de la matrice = ',int2str(s)])
```

```
valeurs du vecteurs sont:[2:2:10]   %introduction avec [] d'un vecteur (intervalle ou énumération)
les éléments de la matrice sont: [1:4;4 5 7 8] %introduction de la matrice
matrice =
     1.00     2.00     3.00     4.00
     4.00     5.00     7.00     8.00
Vc2 =          %affichage  de la 2ème colonne de matrice
     2.00
     5.00
     6.00    30.00   % affichage de moyenne et la somme du vecteur
taille de la matrice = 2  4    % affichage des dimensions de matrice avec conversion
```

IV.3.3. Instruction conditionnelle if

En général, les instructions conditionnelles sont des structures qui permettent le contrôle de l'exécution des commandes. Ces structures permettent une exécution conditionnelle de lignes de commandes. Ces lignes de commandes forment ce que l'on appelle communément un bloc. Une condition est générée par l'application d'un opérateur relationnel à deux opérandes. Les opérateurs relationnels et logiques ont été présentés au chapitre I.

Le mot réservé ***if*** indique le début d'une structure conditionnelle. Il est nécessairement suivi d'une condition. Les instructions1 seront exécutées si et seulement si la condition est évaluée vraie (vaut 1 ou ***true***). Si cette condition donne un faux (vaut 0 ou **false**), ce sont les instructions2 qui seront exécutées.

La syntaxe d'un ***if double*** est la suivante :

if (condition)
<instructions1>
else
<instructions2>
end

On peut également imbriquer des ***if*** . . . ***else*** les uns dans les autres à l'aide de l'instruction **elseif**. La syntaxe est présentée ci-après :

if (cond1)
<instructions>
elseif (cond2)
<instructions>
elseif (cond3)
...
Else
<instructions>
end

NB : Les segments de ***elseif*** est ***else*** sont optionnels, par contre la parution de ***if*** et ***end*** est obligatoire. Le mot THEN , qu'on trouve presque dans tous les langages, n'existe pas sous Matlab.

Voici un exemple de manipulation d'un IF double dans le script : exemple_if1.m

```
1-    clear, clc
2-    ph=input('faites entrer le Ph de l''eau : ');
3-    if ph>6.5 & ph<8.5   %après la condition ou else on n'écrit pas les instruction à exécuter
4-    disp('l''eau est consommable')
5-    else
6-    disp('l''eau n''est pas consommable')
7-    end
```

L'exécution sur ***Command Window*** donnera :

```
faites entrer le Ph de l'eau : 9   %valeur introduite au ph
l'eau n'est pas consommable      %résultat selon la condition appliquée
>>exemple_if1                    % réexécution du script
faites entrer le Ph de l'eau : 6.8
l'eau est consommable
```

Dans ce qui suit, on trouvera un exemple d'un IF multiple dans le script : exemple_if2.m

```
1-    clear, clc
2-    Q=input('faites entrer le débit de la conduite : ');
3-    D=input('donner le diamètre de la conduite : ');
4-    V=4*Q/(pi*D^2)
5-    if V<0.5
6-    disp('la vitesse d''écoulement est faible risque de dépôt')
7-    elseif V>=0.5 & V<2
8-    disp('la vitesse d''écoulement est acceptable')
9-    else
10-   disp('la vitesse est excessive risque d''érosion')
11-   end
```

L'exécution sur ***Command Window*** donnera :

```
faites entrer le débit de la conduite : 0.25
donner le diamètre de la conduite : 0.3
V =
    3.5368
la vitesse est excessive risque d'érosion
```

```
>> exemple_if2
faites entrer le débit de la conduite : 0.28
donner le diamètre de la conduite : 0.5
V =
   1.4260
la vitesse d'écoulement est acceptable
```

IV.3.4. Instruction conditionnelle switch

Comme une structure if, la structure switch est une structure conditionnelle, c'est-à-dire qu'elle comporte différents blocs d'instructions qui seront exécutés de manière conditionnelle. C'est une structure qui n'est pas très utilisée comme ***if.***

Cependant, ici il n'y a pas de conditions, le critère de choix est la valeur ordinale (scalaire, chaine de caractères ou de type d'énumération créé) d'une expression (ou d'une variable). Cette valeur, que l'on appelle cas (***case***), permet la sélection du bloc à exécuter.

La syntaxe est la suivante :

switch *expression du choix*

case *expression du 1er cas*

instructions

case *expression du 2ème cas*

instructions

...

Otherwise

Insturctions

end

Exemple avec la commande switch :

```
1-      clc,clear
2-      note = input('Introduisez la notation  : ');
3-      switch note
4-         case 'A'
5-            disp(' Excellent travail');
6-         case 'B'
7-            disp(' C''est du bon travail');
8-         case {'C','D'}
9-            disp(' C''est du travail moyen ');
10-        case 'F'
11-           disp(' Très faible note, module à refaire ');
12-        otherwise
13-           disp(' Mauvaise notation introduite');
14-     end
```

Un exemple d'exécution sur le ***Command Window*** donnera :

```
Introduisez la notation  : 'C'
 C'est du travail moyen
>>exemple_switch
Introduisez la notation  : 'X'
 Mauvaise notation introduite
```

IV.3.5. Boucle conditionnelle (while …)

Ce mécanisme permet de répéter une série d'instructions tant qu'une condition est vérifiée. Sa syntaxe est :

while expression

instructions …

end

Le terme expression est une expression logique. Si cette dernière est évaluée à vrai, le bloc instructions est exécuté. Puis, expression est de nouveau testé. L'exécution du bloc est répétée tant que l'expression logique donne un **vrai**. Si cette condition est testée fausse, on sort de cette boucle.

NB : Cette instruction itérative est utilisée lorsque le nombre de répétitions n'est pas connu d'avance et aussi lorsque dans « expression » il y a plusieurs conditions reliées avec des opérateurs logiques (et, ou).

Exemple :

Dans l'exemple suivant, on demandera à un utilisateur d'introduire un coefficient de ruissellement, tant que la valeur introduite n'est pas valide, on lui redemandera de rentrer une nouvelle valeur jusqu'à avoir introduit un coefficient valide.

```
1-      clc,clear
2-      Cr=input('Donner le coefficient de ruissellement du bassin:');
3-      while Cr<0|Cr>1
4-      Cr=input('Donner le coefficient de ruissellement du bassin:');
5-      end
6-      disp('Vous avez introduit un coefficient valide')
```

Lorsqu'on exécute ce fichier, le résultat suivant s'affiche :

```
Donner le coefficient de ruissellement du bassin:-1
Donner le coefficient de ruissellement du bassin:2
Donner le coefficient de ruissellement du bassin:3
Donner le coefficient de ruissellement du bassin:0.8
Vous avez introduit un coefficient valide
```

IV.3.6. Boucle itérative (for …)

Cette boucle exécute le bloc d'instructions interne autant de fois que spécifié par une variable (indice) jouant un rôle de compteur. Sa syntaxe est :

for *indice = début:pas:fin*

instructions …

end

La variable « indice » est initialisée à la valeur « début » et évolue jusqu'à la valeur « fin » par pas d'incrémentation « pas ». A chaque itération, l'ensemble d'instructions est exécuté. Généralement, variable est un scalaire, et souvent un entier. Comme le Fortran, Matlab peut utiliser des variables (indice et pas) de type réel (positif ou négatif).

Si la valeur du pas est égale à la valeur 1, l'écriture de la boucle peut être de la forme suivante :

for indice = début:fin

instructions …

end

Exemple d'utilisation de l'instruction **for**

```
1-      clc,clear,
2-      s=0;
3-      for i=1:2:9
4-          s=s+i^2;
5-          disp([' Valeur de i = ',int2str(i)])
6-          disp([' Valeur de la somme = ',int2str(s)])
7-      end
```

L'exécution sur le ***Command Window*** donnera :

```
Valeur de i = 1        % valeur de début de i=1
Valeur de la somme = 1
Valeur de i = 3        % incrémentation à i de la valeur pas=2
Valeur de la somme = 10
Valeur de i = 5
Valeur de la somme = 35
Valeur de i = 7
Valeur de la somme = 84
Valeur de i = 9          % valeur de fin de i=9
Valeur de la somme = 165
```

L'exemple suivant utilise les valeurs d'une suite d'un vecteur de données :

```
1-      clc,clear
2-      s=0;
3-      for i=[13,9,21,32]
4-      s=s+i^2;
5-      disp([' Valeur de i = ',int2str(i)])
6-      disp([' Valeur de la somme = ',int2str(s)])
7-      end
```

```
Valeur de i = 13     % la variable indice i prend la 1ère valeur du vecteur =13
 Valeur de la somme = 169
 Valeur de i = 9    % la variable indice i prend la 2ème valeur du vecteur =9
 Valeur de la somme = 250
 Valeur de i = 21
 Valeur de la somme = 691
 Valeur de i = 32      % la variable indice i prend la dernière valeur du vecteur =32
 Valeur de la somme = 1715
```

NB : MATLAB permet d'utiliser une boucle imbriquée dans une autre boucle. La section suivante montre quelques exemples pour illustrer le concept.

Syntaxe

La syntaxe d'une instruction de la boucle **for** et ***while*** imbriquée dans MATLAB est la suivante :

Exemple 1

```
for i=1 :2 :9
   for j=1 :4
     instructions…
   end
end
```

Exemple 2 :

```
while a>2
   for j=1 :4
     instructions…
   end
end
```

Exemple 3 :

```
while a>2
   while and(b==5,c<=1)
     instructions…
   end
end
```

IV.4. Utilisation des fonctions sous Matlab

Jusqu'ici, on a vu un certain nombre de fonctions prédéfinies, que l'on peut appliquer à des scalaires ou à des matrices de données. Ces fonctions peuvent être utilisées directement en ligne de commandes, soit dans la fenêtre de commandes ou sur les scripts Matlab. L'autre type de fonction, c'est la fonction créée par l'utilisateur pour un besoin spécifique[5,11].

Cette fonction est un groupe d'instructions qui réalisent ensemble une certaine tâche. Dans MATLAB, les fonctions sont définies dans des fichiers séparés. Le nom du fichier et de la fonction doit être le même. De manière analogue, il est possible de définir ses propres fonctions.

Les fonctions opèrent sur des variables dans leur propre espace de travail, également appelé espace de travail local, distinct de l'espace de travail (***Workspace***) auquel vous accédez à l'invité de commande MATLAB, appelé espace de travail de base.

IV.4.1. Création d'une fonction

Sur l'éditeur des scripts Matlab, faire ***File*** → ***New*** →***Function***. Cette fonction doit être enregistrée dans le même répertoire du script Matlab appelant.

Les fonctions peuvent accepter plusieurs arguments d'entrée et renvoyer plusieurs arguments de sortie.

La syntaxe de l'entête d'une fonction est la suivante :

function [res1,res2, ..., resN] = ma_fonct(don1,don2,…,donM)

avec :

function : mot clé Matlab pour commencer une fonction

[res1,res2,…resN] : les N noms de variables des résultats de sortie ;

ma_fonct : le nom attribué à la fonction créée ;

(don1,don2,…,donM) : les M arguments (données) nécessaires pour la fonction, on les appellent aussi les arguments formels.

NB : Plusieurs cas de figures de déclarations d'une fonction peuvent se présenter :

function ma_fonct (arg1, arg2, …)

Lorsqu'il n'y a pas de variable résultat de sortie ;

function V_res = ma_fonct(arg1, arg2, …)

une fonction avec une seule variable résultat de sortie (V_res) ;

function [V_res1, V_res2, …] = ma_fonct(arg1, arg 2, …)

une fonction avec plusieurs variables résultat de sortie (V_res1, V_res2,…). Ici cette fonction joue le rôle des subroutines (Fortran) ou des procédures (Pascal).

Exemple d'un *script function* calcul.m :

```
1-    function [som, moy ] = calcul(n1,n2,n3,n4)
2-    som=n1+n2+n3+n4
3-    moy=som/4
4-    end
```

IV.4.2. L'appel à la fonction réalisée sous Matlab

Cet appel se fait à partir d'un script appelant (comme un programme principal) ou à partir d'une autre fonction. Tous ces programmes (script principal et le script fonction) doivent être enregistrés dans le même répertoire.

Dans ce qui suit, on fera appel à la fonction ***calcul.m*** créée dans la séquence précédente par le ***script_principal.m*** créé dans le même répértoire.

```
1-    clc,clear
2-    note1 =input('la note 1 = ');   % introduction des valeurs effectives
3-    note2 =input('la note 2 = ');
4-    note3 =input('la note 3 = ');
5-    note4 =input('la note 4 = ');
6-    [s   my]= calcul(note1,note2,note3,note4);  % appel à la fonction calcul.m
7-    disp(['la somme est ',num2str(s)])   % affichage des résultats
8-    disp(['la moyenne est ',num2str(my)])
```

L'exécution sur le ***Command Window*** donnera :

```
la note 1 = 14
la note 2 = 15
la note 3 = 18
la note 4 = 12
la somme est 59
la moyenne est 14.75
```

IV.5. Les Polynômes sous MATLAB

IV.5.1. Définition

Par définition, un polynôme P est un vecteur « rangée » qui contient les coefficients du polynôme classés par ordre décroissant. Un polynôme de degré n est représenté par un vecteur de taille (n+1).

Un polynôme de la forme $P(x)= a_nx^n+ a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+...+a_1x+a_0$ est représenté sous Matlab par le vecteur ligne $P= [a_n, a_{n-1}, a_{n-2}, ..., a_1, a_0]$.

Exemple :

Le polynôme $P= 3X^3- x^2 +2X+1$ s'écrit sous Matlab : P = [3 -1 2 1] , pour séparer les coefficients du polynôme, on peut utiliser soit les virgules ou des espaces.

IV.5.2. Fonctions appliquées aux polynômes

Matlab dispose d'une grande variété de fonctions applicables aux polynômes. Dans ce qui suit, on présentera les commandes les plus courantes.

- *La commande polyval*

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

```
1-      clc,clear
2-      P=[3 -1 2 1]   % création du polynôme P(x)= 3X^3- x^2 +2X+1
3-      P0=polyval(P,0)    % Evaluation de P(0)= 3(0)^3- (0)^2 +2(0)+1
4-      P1=polyval(P,1)
5-      P2=polyval(P,2)   % Evaluation de P(2)= 3(2)^3- (2)^2 +2(2)+1
```

Le résultat est :

```
P =
   3   -1    2    1
P0 =
   1
P1 =
   5
P2 =
  25
```

- *La commande roots*

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

```
1-      clc,clear
2-      P=[3 -1 2 1]   % création du polynôme P(x)= 3X^3- x^2 +2X+1
3-      format short
4-      r=roots(P)
```

Résultat :

```
P =
  3.00       -1.00        2.00        1.00
r =                    % r est un vecteur colonne de 3 racines
   0.3480 + 0.8933i
   0.3480 - 0.8933i
  -0.3627
```

- *La commande poly*

Cette fonction retourne un polynôme depuis ses racines stockées dans un vecteur.

P = ***poly***(r) où r est un vecteur renvoie un vecteur ligne dont les éléments sont les coefficients du polynôme dont les racines sont les éléments de r.

- *La commande polyint*

La commande ***polyint***(P) : intègre analytiquement un polynôme P.

polyint(p,k) renvoie un polynôme représentant l'intégrale du polynôme p, en utilisant une constante scalaire d'intégration k. Si k=0, cette commande s'écrira : ***polyint***(P)

- *La commande polyder*

La fonction ***polyder*** calcule la dérivée des polynômes.

Syntaxe

k = ***polyder***(p) renvoie la dérivée du polynôme p. l'opérande p est un vecteur dont les éléments sont les coefficients d'un polynôme en puissances décroissantes.

```
1.  clc,clear
2.  P=[3 -2 -1 2 ]     % création du polynôme P(x)= 3X^3+2x^2 - X+2
3.  r1=roots(P)      % r1 contiendra les racines du polynôme P
4.  format short
5.  r=poly([r1])*P(1)   %  dans r il y aura la reconstruction du polynôme P
6.  P2=polyint(P,2)     % intégration du polynôme P avec une constance c=2
7.  P3=polyder(P)       % dérivation du polynôme P dans le vecteur P3
```

Le résultat de ce script dans le ***Command Window***

```
P =                 %P est le polynôme de départ
    3    2    -1    2
r1 =                %r1 les racines du polynôme  P
  -1.3098
   0.3216 + 0.6369i
   0.3216 - 0.6369i
r =                                 %r  est le polynôme P depuis les racines r1
    3.0000    2.0000   -1.0000    2.0000
P2 =                                %P2 est l'intégration du polynôme P
    0.7500    0.6667   -0.5000    2.0000    2.0000
P3 =                                %P3 est la dérivée du polynôme P
    9    4    -1
```

IV.6. Résolutions des Systèmes Linéaires

IV.6.1. Définition

Un système linéaire est un ensemble de n équations à p variables de la forme :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1p}x_p = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2p}x_p = b_2 \\ \qquad\qquad \cdots\ \cdots \\ \qquad\qquad \cdots\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{np}x_p = b_n \end{cases}$$

Où : $a_{ij} \in R$, $i \in \{1,2, \ldots,n\}$, $j \in \{1,2, \ldots,p\}$.

Il faut toujours réécrire le système d'équations sous la forme :

(A) (X)=(b) ou A : matrice des coefficients de (n×p), X : vecteur colonne des p inconnus, b : vecteur colonne de second membre de p éléments

Exemple :

$$\begin{cases} x + 3y - z = 3 \\ 2x - y + z = 1 \\ -x + 2y + 2z = 2 \end{cases}$$

Les variables x, y et z seront représentés par x_1, x_2, x_3

Les coefficients de la matrice A : a_{11}=1 ; a_{12}=3 ; a_{13}=-1 ; a_{21}=2 ; a_{22}=-1 ; a_{23}=1 ; a_{31}=-1 ; a_{32}=2 ; a_{33}=2

Le vecteur b : b_1=3 ; b_2=1 ; b_3 =2

A = $\begin{bmatrix} 1 & 3 & -1 \\ 2 & -1 & 1 \\ -1 & 2 & 2 \end{bmatrix}$ x= $\begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix}$ ou x= $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ le second membre b= $\begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$

IV.6.2. Résolution du système d'équations

Matlab fournie plusieurs techniques afin de résoudre un système d'équations. Néanmoins, il reste à l'utilisateur de vérifier la compatibilité du système linéaire (s'il existe solution).

Par exemple, si le déterminant de A est non-nul alors le système accepte une solution. Par contre, si le déterminant est nul le système peut accepter un nombre infini de solution ou un ensemble vide des solutions.

- La commande X = A\B fait résoudre le système linéaire AX = B ;
- La commande X = B/A fait résoudre le système linéaire XA = B (noter le slash).

La commande X = B*inv(A) fait résoudre le système linéaire XA = B.

```
1.  clc,clear
2.  A=[1 3 -1; 2 -1 1; -1 2 2]
3.  b=[3;1; 2]
4.  d=det(A)
5.  x=A\b
6.  x2=inv(A)*b
```

La solution sera :

```
A =                 % matrice des coefficients
    1    3   -1
    2   -1    1
   -1    2    2
b =                 %vecteur du second membre
    3
    1
    2
d =          %valeur du déterminant
   -22
x =          % le vecteur solution 1ère  méthode
   0.7273
   0.9091
   0.4545
x2 =          % le vecteur solution 2ème   méthode
    0.7273
    0.9091
    0.4545
```

IV.7. Applications du chapitre IV

IV.7.1. Application 1

X étant une variable réelle, écrire un script qui calcule la valeur de Y donnée par :

$$\begin{cases} Y = X^2 & si \quad X < 2.7 \\ Y = X^3 & si \quad 2.7 \leq X \leq 6.9 \\ Y = X^4 & si \quad X > 6.9 \end{cases}$$

```
1-      x=input('faites entrer x =');    %introduction de la valeur de x
2-      if x<2.7
3-         y=x*x;                % y=x^2
4-      elseif and(x>=2.7, x<=6.9)
5-         y=x^3;                % y=x^3
6-      else
7-         y=power(x,4);      % y=x^4
8-      end
9-      disp(['la valeur de y =',num2str(y)])   % affichage de la valeur de y
```

L'exécution donnera :

```
faites entrer x =7        %exécution 1
la valeur de y =2401
>>application4_1        %nom du script de cette application afin de l'exécuter
faites entrer x =3        %exécution 2
la valeur de y =27
```

IV.7.2. Application 2

1- Ecrire une fonction1 qui convertit en coordonnées polaires (r, θ) associées aux coordonnées cartésiennes (x,y) sachant que :

$r = \sqrt{x^2 + y^2}$

- θ = arctang $|y/x|$ si x > 0 (la fonction arc tangente sous Matlab : **atan** (alpha))
- θ =- arctang $|y/x|$ si x < 0
- θ =π/2 si x = 0 et y > 0
- θ =- π/2 si x = 0 et y< 0

2- Ecrire une fonction2 qui convertit l'angle θ trouvé en « radian » vers le « degré »

Dans un script principal faites lire deux coordonnées cartésiennes (x,y) ;

Faites appel à la fonction 1 pour convertir (x,y) en polaire (r, θ). Un deuxième appel à la fonction 2 pour convertir l'angle trouvé vers le degré.

Fonction 1 : **conversion.m** (conversion de coordonnées)

```
1-  function [ r1,t1 ] = conversion(x1,y1)
2-  r1=sqrt(x1^2+y1^2);
3-  if x1>0
4-      t1=atan(abs(y1/x1));
5-  elseif x1<0
6-      t1=-atan(abs(y1/x1));
7-  else
8-       if y1>0
9-        t1=pi/2;
10-      else
11-      t1=-pi/2;
12-      end
13- end
14- end
```

Fonction 2 : **radeg.m** (conversion d'angle)

```
1-    function degre= radeg(t1)
2-      degre=t1*180/pi;
3-    end
```

Le script principal : **application4_2.m**

```
1-    clc,clear
2-    x=input('entrer la valeur x= ');
3-    y=input('entrer la valeur y= ');
4-    [r,t]=conversion(x,y);
5-    angle_degre=radeg(t);
6-    disp(['r = ',num2str(r)])
7-    disp(['angle = ',num2str(angle_degre),'  degrés'])
```

NB : les trois fichiers seront enregistrés dans le même répertoire.

Un exemple d'exécution :

```
entrer la valeur x= 10
entrer la valeur y= 3
r = 10.4403
angle = 16.6992 degrés
```

IV.7.3. Application 3

Trouver les solutions du système suivant :

$$\begin{cases} 3x + 2z = 1 \\ 3y + z + 3t = 2 \\ x + y + z + t = 0 \\ 2x - y + z - t = -1 \end{cases}$$

Les variables x, y,z et t seront représentés par x_1, x_2, x_3, x_4

$$A = \begin{bmatrix} 3 & 0 & 2 & 1 \\ 0 & 3 & 1 & 3 \\ 1 & 1 & 0 & 1 \\ 2 & -1 & 1 & -1 \end{bmatrix} , \; X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} \quad et \quad b = \begin{bmatrix} 1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

Les coefficients de la matrice A : $a_{11}=3$; $a_{12}=0$; $a_{13}=2$; $a_{14}=1$; $a_{21}=0$; $a_{22}=3$; $a_{23}=1$; $a_{24}=3$; $a_{31}=1$; $a_{32}=1$; $a_{33}=0$; $a_{34}=1$; $a_{41}=2$; $a_{42}=-1$; $a_{43}=1$; $a_{44}=-1$

Le vecteur b : b_1=1 ; b_2=2 ; b_3 =0 ; b_4 =-1

```
1. clc, clear % Effacer le window command et supprimer toutes les variables du workspace
2. A=[3 0 2  1; 0 3 1 3; 1 1 0 1;2 -1 1 -1] % Création de la Matrice A
3. b=[1 2 0 -1]' % b Vecteur colonne
4. X=A\b % Résolution du système
```

Le résultat obtenu dans le *window command* est :

```
A =

    3    0    2    1
    0    3    1    3
    1    1    0    1
    2   -1    1   -1
b =

    1
    2
    0
   -1
X =

   -0.5000
   -1.0000
    0.5000
    1.5000
```

VI.8. Conclusion

Le chapitre de la programmation sous Matlab est très important pour l'étudiant. Créer des scripts Matlab pour les problèmes à résoudre est la finalité de l'étudiant. Dans ce chapitre, on a présenté la structure des programmes Matlab, ensuite on a défini les différentes instructions de base (lecture, affichage, Test, boucles et fonction) du logiciel. La création et la résolution des polynômes sont illustrées avec des exemples à l'appui. Savoir créer et résoudre les systèmes d'équations linéaires est la dernière partie théorique de ce chapitre. Comme les précédents chapitres, trois (03) applications-exemples corrigées sur la création des scripts Matlab sont présentés avec des explications. La dernière partie à voir dans ce cours est les représentations graphiques sous Matlab. Ce point sera le sujet du chapitre suivant.

Chapitre V : Graphisme sous Matlab

V.1. Introduction

MATLAB, outre de permettre de faire des calculs numériques de très haut niveau, il peut aussi produire des graphiques impressionnants, de type 2D ou 3D.

Matlab possède un grand choix de commandes pour créer toutes sortes de représentations graphiques : graphiques standards en 2D ou 3D avec axes linéaires, semi-log ou logarithmiques, histogrammes, en escalier, en fromage, rosette, représentation de coordonnées polaires, de courbes de niveau (« contour surface »), de mailles régulières (« mesh », « grid »), parmi d'autres. Tous les objets d'un graphique peuvent être formatés pour obtenir l'apparence désirée. Texte, légendes et commentaires ainsi que des aides à la visualisation (commentaires, grilles) peuvent être facilement ajoutés. Un même graphique peut contenir plusieurs jeux de données, plusieurs graphiques peuvent être placés sur une même page, etc. Ce chapitre présente les commandes les plus importantes pour élaborer les représentations graphiques les plus communes. Il existe cependant un grand nombre de commandes et scripts pour l'élaboration de graphiques plus sophistiqués. [5,7]
Dans toutes les représentations graphiques, le logiciel se base sur des données discrètes rangées dans des matrices ou des vecteurs colonnes. Par exemple, 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. La capacité de tracer librement des fonctions et des données mathématiques est l'étape la plus importante, et ce chapitre est rédigé pour aider les étudiants à le faire.

V.2. Gestion des fenêtres graphiques

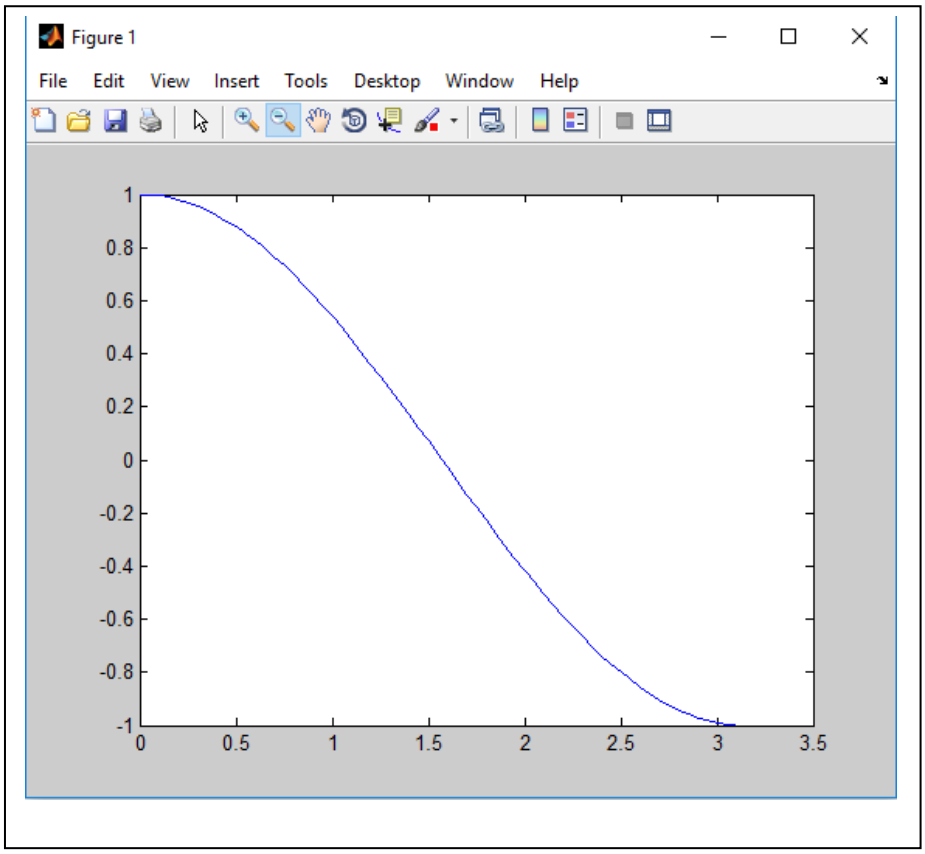
Les représentations graphiques sont réalisées dans un environnement **figure** qu'on peut contrôler. Par défaut, toute commande de représentation graphique déclenche la création d'une fenêtre **figure nommée** ***figure1*** *contenant la représentation.*

Par défaut, une nouvelle instruction graphique sera affichée dans la même fenêtre et écrasera la figure précédente. On peut ouvrir une nouvelle fenêtre graphique par la commande : **figure**. Chaque fenêtre se voit affecter un numéro suivant la précédente **(n)**. Ce numéro est visible dans la barre de titre de la fenêtre sous forme d'un titre par défaut. Le résultat d'une instruction graphique est par défaut affiché dans la dernière fenêtre graphique ouverte qui est la fenêtre graphique active. On rend active une fenêtre graphique précédemment ouverte en exécutant la commande **figure(n),** où **n** désigne le numéro de la figure.

La commande **close** permet de fermer la fenêtre graphique active. On ferme une fenêtre graphique précédemment ouverte en exécutant la commande **close(n)**,

où **n** désigne le numéro de la figure. Il est également possible de fermer toutes les fenêtres graphiques en tapant la commande : **close all**.

La figure V.1 représente la fenêtre graphique ***Figure 1*** où est affiché le graphe de la fonction **cosinus** entre **0 et π**, résultat de la commande **fplot('cos',[0 pi]).**



FigureV.1: *Environnement Figure de Matlab*

V.3. Graphiques à 2 dimensions (2D)

V.3.1. La commande plot

Pour tracer le graphique d'une fonction, vous devez suivre les étapes suivantes :

- Définir x, en spécifiant la plage de valeurs de la variable x, pour laquelle la fonction doit être tracée ;
- Définir la fonction, y = f(x) ;
- Utiliser la commande **plot**, comme **plot(x, y) ;**

L'exemple suivant démontrerait le concept. Traçons la fonction simple $y = x^2$ pour la plage de valeurs de x de -20 à 20, avec un pas d'incrémentation de 5.

Créez un script Matlab et tapez les instructions suivantes :

```
1. X = [-20:5:20]; % création du vecteur X
2. Y = X.^2;   % création de Y en fonction de X
3. plot(X, Y)   % Tracer les point y en fonction de X
```

L'exécution de ce script donnera :

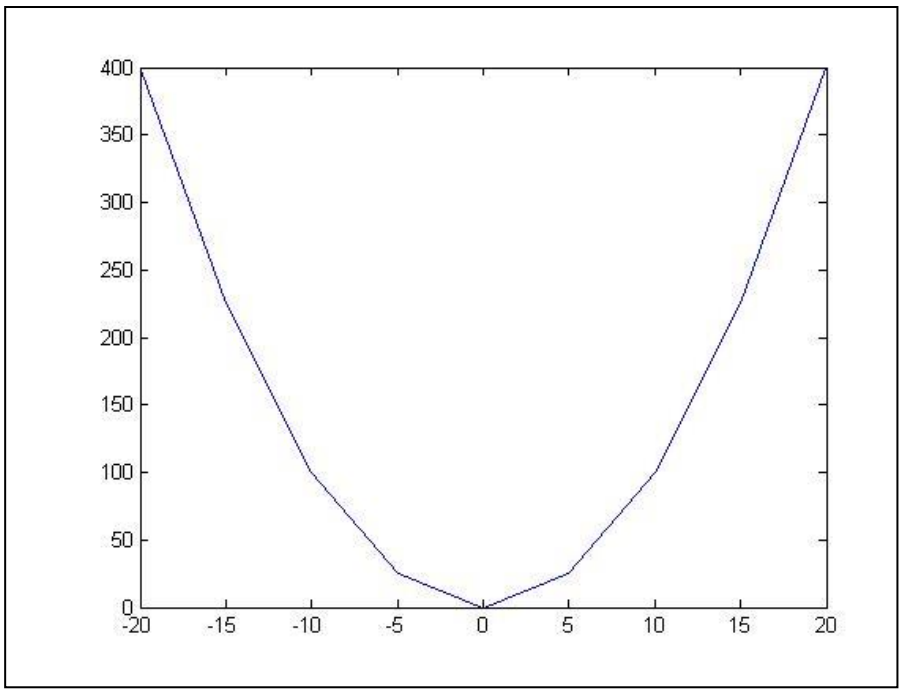


Figure V.2: *Représentation graphique de la fonction Y = X²*

V.3.2. Ajout des caractéristiques du graphe

MATLAB vous permet d'ajouter des caractéristiques descriptives au graphe dessiné à savoir : un titre, des étiquettes le long des axes x et y, des quadrillages de grille et également d'ajuster les axes pour embellir le graphique. Les commandes suivantes sont utilisées pour ajouter ces caractéristiques :

- Les commandes **xlabel** et **ylabel** génèrent des étiquettes le long des axes x et y ;
- La commande **title** permet de mettre un titre sur le graphique ;
- La commande **grid on** ou **grid** permet de mettre un arrière-plan sous forme d'un quadrillage sur le graphique, on peut le faire disparaitre par la commande **grid off** ;
- La commande **axis**([x_{min} x_{max} y_{min} y_{max}]) permet d'introduire les limites minimales et maximales des axes X et Y ;
- La commande **axis auto** permet de choisir automatiquement les valeurs limites de chaque axe ;
- La commande **axis equal** permet de générer le tracé avec les mêmes facteurs d'échelle et les mêmes espaces sur les deux axes ;
- La commande **legend**('show') permet de montrer la légende, par contre la commande **legend**('hide') permet de la cacher.

Exemple :

```
1. X = [0 : 0.01 : 2*pi]; % création du vecteur X (les angles sous Matlab sont en Radians)
2. Y = cos(X);   % création de Y en fonction de X
3. plot(X, Y)    % Tracer les point y en fonction de X
4. xlabel('x'), ylabel('Cos(x)'), title('Graphe de Cosinus')
5. grid on, axis([0 pi -1 1]) % quadrillage et modification des limites des axes
6. legend('cos(x)') % affichage de la légende
```

Le graphe obtenu est présenté sue la figure ci-dessous.

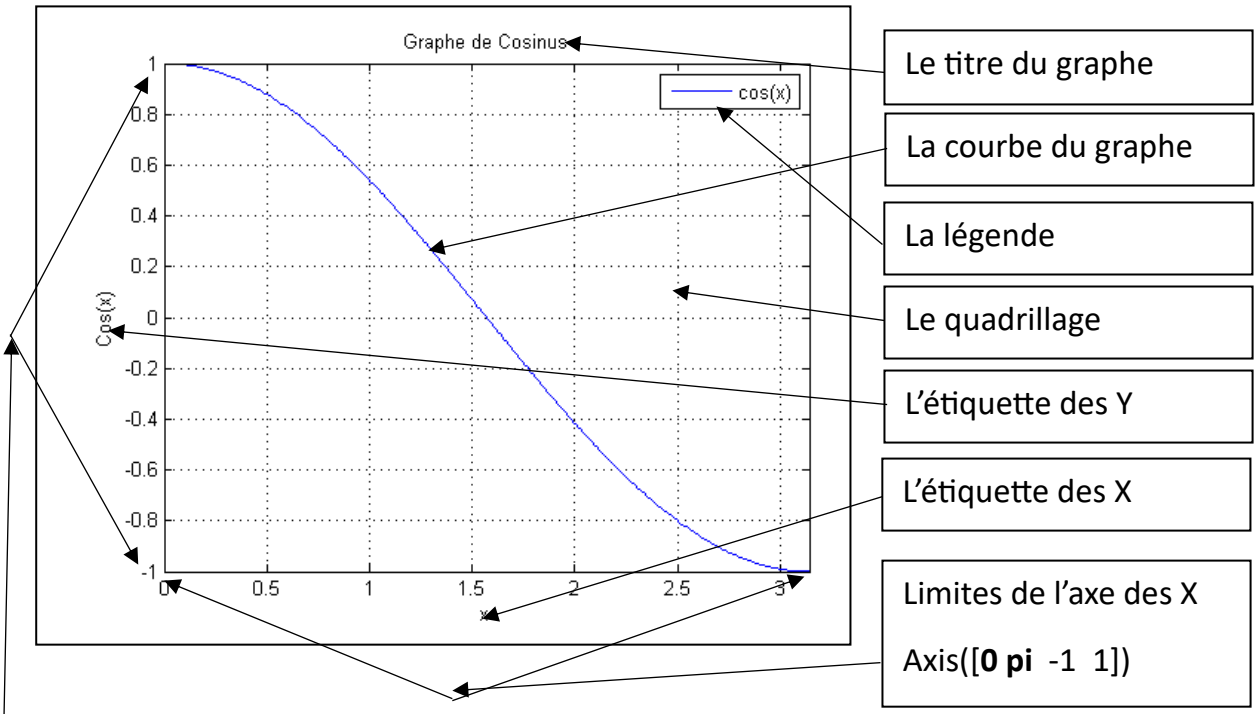


Figure V.3 : *Représentation de la fonction y=cos(x) avec les caractéristiques du graphe*

V.3.3. Dessiner plusieurs courbes sur le même graphique

Afin de dessiner plusieurs courbes sur le même graphique, on utilise toujours la commande plot (x, y1, x, y2,…) en incluant des couples (x, y1 , x, y2). L'exemple suivant illustre le concept :

Exemple

Tapez un le script Matlab suivant :

```
1. x = linspace(1,100, 100); % créer une liste de 100 éléments
2. y1 = x;  % 1ère courbe
3. y2 = 2*x+1;  % 2ème courbe
4. y3 = -2*x+3;  % 3ème courbe
5. plot(x, y1, x, y2, x, y3)
6. legend('x', '2x+1', '-2x+3')
7. grid
8. title (' Graphe avec trois courbes')
9. xlabel('x'), ylabel('y')
```

Le graphe obtenu est illustré dans la figure suivante :

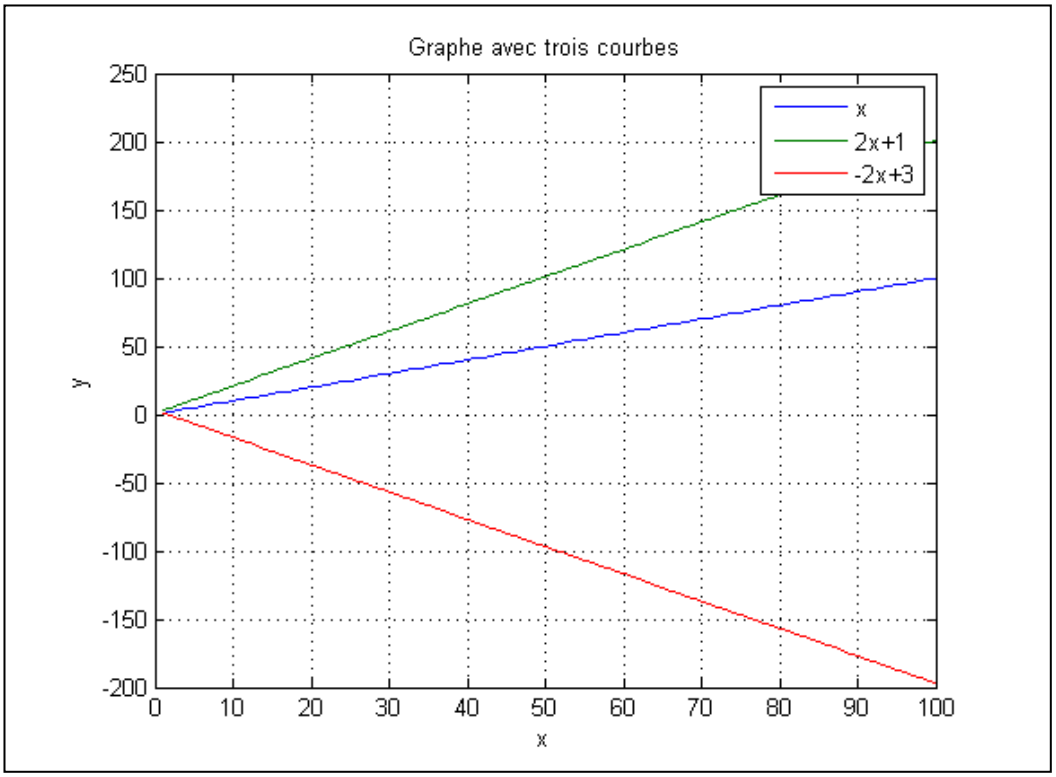


Figure V.4 : *Plusieurs courbes sur un même graphique*

V.3.4. Graphique avec spécification

La commande **plot** (x, y, LineSpec) affiche un graphe avec une spécification donnée par LineSpec.

LineSpec : La spécification de la ligne est formée au plus de trois valeurs : **style de ligne, type de marquer et la couleur de la courbe.**

Par exemple **plot** (x, y, '-+b') affiche une courbe en bleu avec un style de ligne continu (caractère « - ») et un type de marqueur (+).

• Noter bien que la spécification 'type de marqueur' est utilisée pour les points choisis (vecteurs x et y), par contre la spécification 'style de ligne' couvre toute la courbe.

Tableau V.1 : *Type de trait dans la spécification d'un graphique*

Symbole	Style de trait (le ligne)
-	Ligne continue ou solide (la ligne par défaut)
--	Ligne discontinue
:	Ligne pointillées
-.	Ligne tiretée (Tiret point)

TableauV.2 : *Type de marqueur dans une spécification*

Symbole	Type de marqueur
o	Cercle
+	Signe plus
*	Astérisque
s	Carrée (Square)
d	Diamant
v	Triangle vers le bas
^	Triangle vers le haut
<	Triangle pointant vers la gauche
>	Triangle pointant vers la droite
.	Point
p	Pentagramme
h	Hexagramme
x	Croix

Tableau V.3 : *Couleur de trait d'une courbe*

Lettre d'alphabet	Couleur
y	Jaune (**Y**ellow)
m	Magenta (**M**agenta)
r	Rouge (**R**ed)
b	Bleue (**B**lue)
g	Vert (**G**reen)
k	Noir (Blac**k**)
c	Cyan (**C**yan)
w	Blanc (**W**hite)

Dans le script suivant, les symboles de spécification de graphe sont ajoutés au programme précédent.

```
1. x = linspace(1,50, 10); % 10 éléments
2. y1 = x;
3. y2 = 2*x+1;
4. y3 = -2*x+3;
5. plot(x, y1, '-b', x, y2, '--or',x, y3, ':vk')
6. legend('x', '2x+1', '-2x+3')
   % y1 en bleu et ligne solide
   % y2 en rouge et ligne discontinue, marquer o
   % y3 en noir et ligne en pointillé, marqueur triangle vers le bas
```

Le résultat obtenu est présenté dans la figure ci-dessous

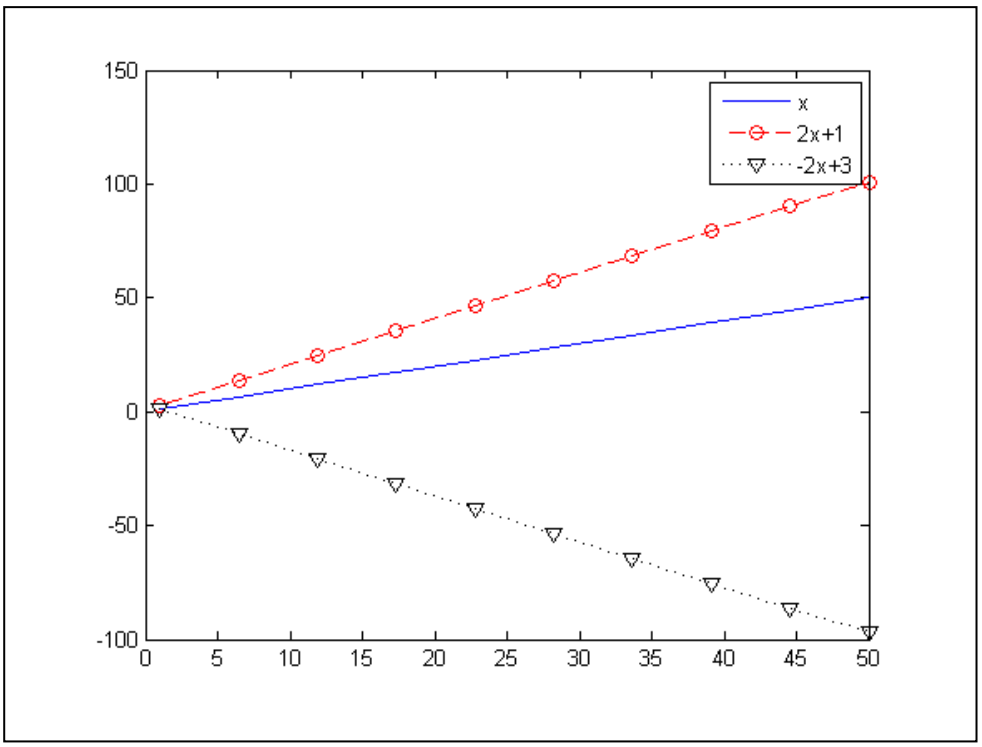


Figure V.5: *Représentation graphique de 3 courbes avec des spécifications*

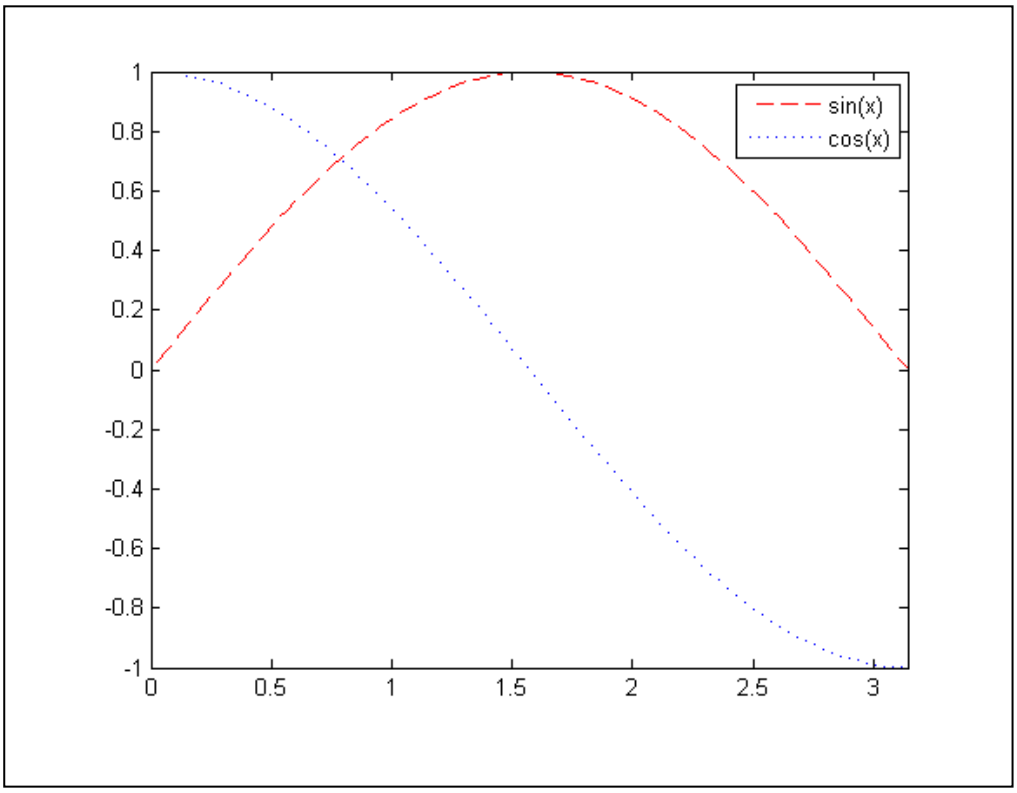
V.3.5. Ajout de courbes à un graphe

Chaque fois que la commande de traçage plot est exécutée, MATLAB efface l'ancien tracé et en dessine un nouveau. Si on souhaite superposer deux tracés ou plus (ajouter une courbe dans le graphique initial, on tape **hold on**. Cette commande demande à MATLAB de conserver les anciens graphiques et de dessiner les nouveaux graphiques par-dessus les anciens. Il reste en vigueur jusqu'à ce qu'on tape **hold off.**

Voici un exemple utilisant les commandes plot et hold on.

```
1. x=-pi:pi/100:pi;
2. y=sin(x);z=cos(x);
3. figure % sur un nouveau graphe
4. plot(x,y,'--r') %(l'option '--' spécifie un tracé discontinu en rouge)
5. hold on % mode superposé pour ajouter une autre courbe
6. plot(x,z,':'); %(l'option ':' spécifie un tracé pointillé couleur bleue par défaut)
7. axis([0 pi -1 1]); % redimensionnement des axes
8. legend('sin(x)','cos(x)')
9. hold off % fermeture de superposition dans la figure
```

Le graphe généré est illustré comme suit :



FigureV.6 : *Ajout d'une courbe à un graphique*

V.3.6. Génération de plusieurs graphiques sur une figure

Lorsqu'on veut créer un tableau de tracés sur une même figure, chacun de ces graphiques est appelé sous-graphique.

La commande **subplot** permet d'afficher plusieurs représentations graphiques sur une même figure.

La syntaxe de la commande est : **subplot** (m,n,i) où

- m est le nombre de sous-fenêtres verticalement ;
- n est le nombre de sous-fenêtres horizontalement ;
- i sert à spécifier dans quelle sous-fenêtre doit s'effectuer l'affichage. Les fenêtres sont numérotées de gauche à droite et de haut en bas. Voir l'exemple suivant illustrant l'utilisation de la commande **subplot**.

Voici un exemple utilisant les commandes **subplot**

```
1. figure,x=-pi:0.01:pi
2. y1=cos(x),y2=sin(x),y3=tan(x),y4=3*x.^2,y5=asin(x),
3. subplot(2,3,1), plot(x,y1), title('cosinus'), grid
4. subplot(2,3,2), plot(x,y2), title('sinus'), grid
5. subplot(2,3,3), plot(x,y3), title('tangente'), grid
6. subplot(2,3,4), plot(x,y4), title('Courbe 3*x^2'), grid
7. subplot(2,3,[5 6]), axis([-1 1 -pi pi]),plot(x,y5), title('arc-sinus'), grid %affichage de
   cette courbe sur les fenêtres 5 et 6
```

Lorsque le fichier est exécuté, Matlab génère le graphique suivant :

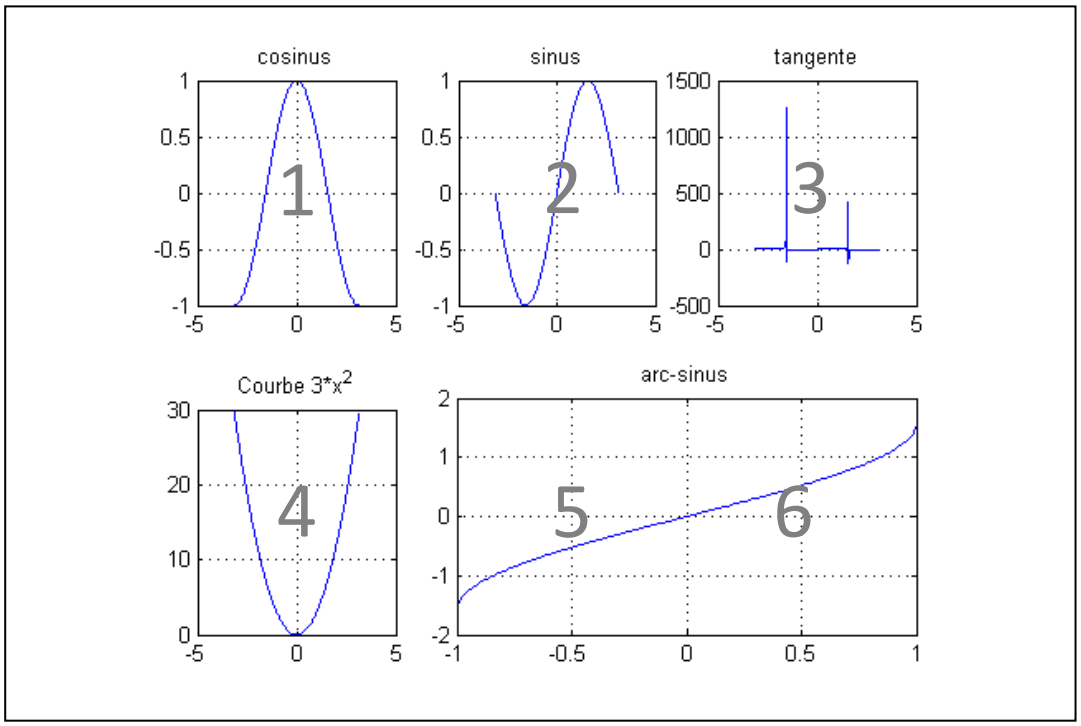


Figure V.7 : *Représentation de 5 sous graphes sur une seule figure*

V.3.7. Graphe d'une fonction (Utilisation de fplot)

La commande **fplot** permet de tracer le graphe d'une fonction sur un intervalle donné.

La syntaxe est : **fplot**('nomf', [xmin xmax]) où

- nomf est soit le nom d'une fonction Matlab incorporée, soit une expression définissant une fonction de la variable x, soit le nom d'une fonction utilisateur.
- [xmin , xmax] est l'intervalle pour lequel est tracé le graphe de la fonction.

On obtient le graphe de la fonction incorporée sinus entre $-\pi$ et π par :

fplot('sin',[-pi pi]). L'exemple suivant présente une combinaison de l'utilisation des commandes **fplot** et **subplot .**

```
1. figure
2. subplot(2,3,1), fplot('cos',[-pi  pi]), title('cosinus'), grid
3. subplot(2,3,[2 5]), fplot('sin',[-pi  pi]), title('sinus'), grid
4. subplot(2,3,3), fplot('tan',[0  pi]), title('tangente'), grid
5. subplot(2,3,4), fplot('3*x.^2',[-10  10]), title('Courbe 3*x^2'), grid
6. subplot(2,3, 6), fplot('asin',[-1  1]), title('arc-sinus'), grid
```

La séquence une fois exécutée, Matlab génère une figure avec les sous-graphiques suivants :

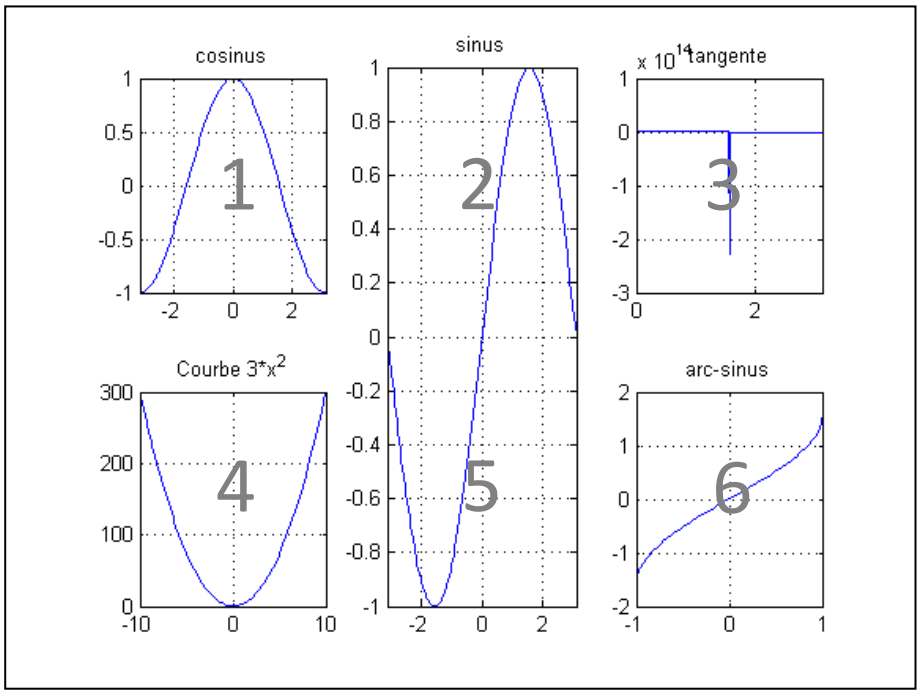


Figure V.8 : *Présentation des graphes de fonctions avec fplot*

NB : Il est possible de dessiner plusieurs fonctions sur la même figure avec un seul **fplot**. Il faut pour cela utiliser la commande **fplot** de la manière suivante :

fplot('[nomf_1 , nomf_2 , nomf_3]', [x_{min} , x_{max}])

où nomf_1, nomf_2, nomf_3 est soit le nom d'une fonction MATLAB incorporée, soit une expression définissant une fonction de la variable x, soit le nom d'une fonction utilisateur exemple cos(x), tan(x) ou 2*x.^2.

fplot('[sin(x),cos(x)]', [-2*pi 2*pi])

V.4. Graphiques à 3 dimensions (3D)

Comme un graphique en 2D est défini par une liste de couples (x,y) on peut définir une courbe en 3D par une liste de triplets (x,y,z). Puisque l'instruction **plot** attendait deux arguments, le vecteur des abscisses, et celui des ordonnées, l'instruction **plot3** en attend trois : le vecteur des abscisses x, le vecteur des ordonnées y et le vecteur des hauteurs z.

V.4.1. Graphique linéaire (Courbes) en 3D

Donc la commande utilisée pour schématiser des courbes en 3D est **plot3**(x, y, z).
L'utilisation de **plot3** est similaire à la commande **plot** (paramètres, linespec, légende, titre, les axes et les étiquettes des axes).

Exemple et résultat graphique

```
1. x=-2*pi:0.1:2*pi,y=cos(x),z=sin(x) %création des vecteurs x, y et z
2. plot3(x,y,z,'o:r'),grid % tracer une courbe 3D en rouge, ligne pointillée avec
   marquer 'o'
3. xlabel('x'),ylabel('y'),zlabel('z'),title('graphe 3D') % ajout de l'étiquette de l'axe de z
4. legend('courbe en 3D')
```

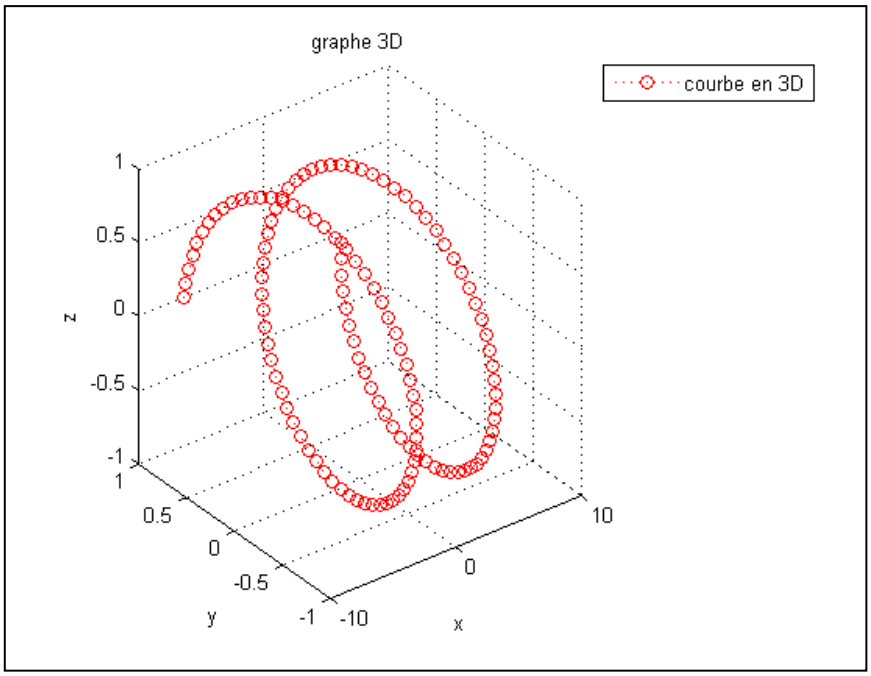


Figure V.9 : *Exemple de courbe en 3D*

Une fois le tracé apparait, on peut modifier la rotation ou l'orientation des axes. Ceci est possible soit en utilisant la flèche (dans l'environnement **figure**) soit en tapant la commande « **view** » suivie par les paramètres azimut et élévation » dans le script Matlab.

view(az, el) : définit l'angle de vue (ou d'orientation) d'un tracé tridimensionnel. L'azimut, **az,** est la rotation horizontale autour de l'axe z, mesurée en degrés à partir de l'axe y négatif. Les valeurs positives indiquent une rotation dans le sens antihoraire du point de vue. **el** est l'élévation verticale du point de vue en degrés. Les valeurs positives d'élévation correspondent au déplacement au-dessus de l'objet ; les valeurs négatives correspondent au déplacement en dessous de l'objet. (Aide du logiciel Matlab)

V.4.2. Graphique surfacique dans le 3D

Ce type de représentation graphique est utilisé pour des graphes en 3D comme tracer l'allure d'écoulement d'un fluide, l'allure des forces agissant sur un obstacle…etc.

Il existe deux commandes de base pour tracer des surfaces dans l'espace 3D : **mesh** et **surf**. Le premier produit une surface transparente en « maille » ; et le dernier produit un ombré opaque. Il existe deux manières différentes d'utiliser chaque commande, une pour tracer des surfaces dans lesquelles la coordonnée z est donnée en fonction de x et y, et une pour des surfaces paramétriques dans lesquelles x, y et z sont tous donnés en fonction de deux autres commandes paramètres. Illustrons le premier avec du **mesh** et le second avec du **surf**.

V.4.2.1. Graphique surfacique (mesh)

Pour tracer une fonction à deux variables ; c'est à dire une fonction de la forme : z=f(x, y) ; on suit les étapes suivantes :

- on définit un maillage régulier, c'est à dire on va construire deux matrices X et Y dont leurs colonnes sont identiques et prennent la valeur des deux vecteurs x et y. Ceci est possible en utilisant la commande « **meshgrid** ».

[X,Y] = meshgrid(x,y) renvoie les coordonnées de la grille 2D sur la base des coordonnées contenues dans les vecteurs x et y. X est une matrice dont chaque ligne est une copie de x, et Y est une matrice dont chaque colonne est une copie de y. La grille représentée par les coordonnées X et Y comporte length(y) lignes et length(x) colonnes.

Pour mieux comprendre cette notion du maillage essayons ensemble l'exemple suivant :

x = [-1 0 1];

y = [9 10 11 12];

[X,Y] = meshgrid(x,y)

• on donne la formule permettant de définir la matrice Z : Z= f(X,Y)

• on utilise la commande « mesh » pour tracer le graphe 3D : mesh(X,Y,Z)

ci-après un exemple classique : On veut tracer la fonction f définie :

(x,y) →$x * e^{-x^2-y^2}$ définie dans le domaine x[-2,2] et y [-2,2].

```
1. x=[-2:0.2:2]; % valeurs du vecteur x
2. y=[-2:0.2:2]; % valeurs du vecteur y
3. [X,Y]=meshgrid(x,y); %création de maillage
4. Z=X.* exp(-X.^2-Y.^2); % fonction Z=f(X,Y)
5. mesh(X,Y,Z) % graphique surfacique en 3D
6. xlabel('x'),ylabel('y'),zlabel('z'),title('graphe surfacique 3D')
```

Le tracé en 3D obtenu est illustré ci-après :

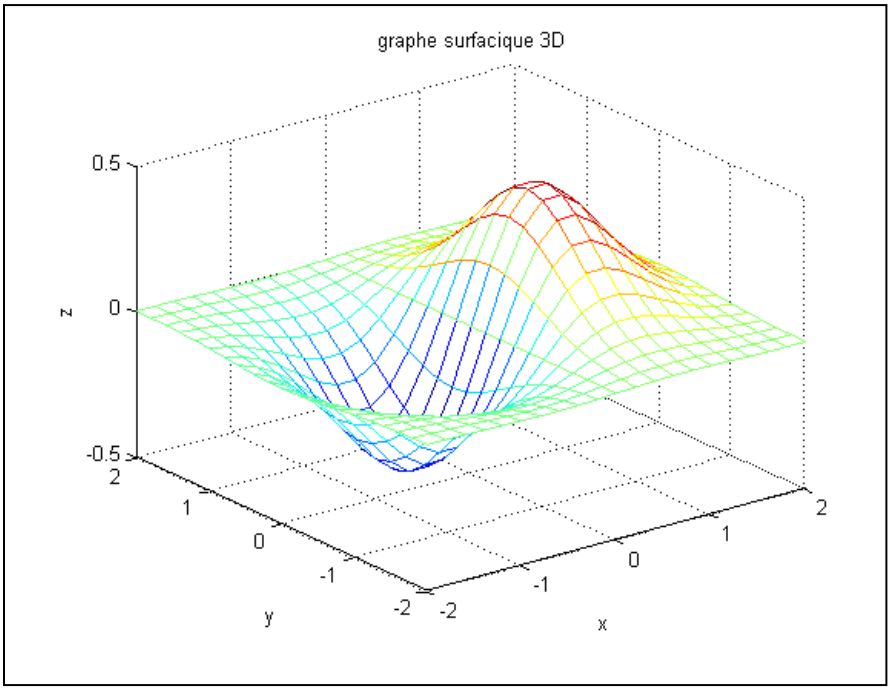


Figure V.10 : *représentation graphique surfacique d'une fonction à 2 variables en 3D*

V.4.3.2. Graphique d'une surface paramétrée (surf)

La commande **surf** permet de tracer une surface paramétrée d'équations.

$$(E) \quad \begin{cases} x = f1(u,v), \\ y = f2(u,v), \\ z = f3(u,v). \end{cases}$$

Pour tracer la fonction paramétrée d'équation (E) pour $u \in]\, u_{min}, u_{max}\, [$ et $v \in]\, v_{min}, v_{max}\, [$, *on procède comme la commande* ***mesh*** *:*

- Création d'un maillage carré de coté h, de l'aire $[\, u_{min}, u_{max}\,] \times [\, v_{min}, v_{max}\,]$ *à l'aide de la commande meshgrid,*

[U , V] =meshgrid ($u_{min} : h : u_{max}$, $v_{min} : h : v_{max}$)

- Obtention $\begin{cases} X = f1(U,V), \\ Y = f2(U,V), \\ Z = f3(U,V). \end{cases}$
- Représenter de la surface en 3 D à l'aide de la commande **surf.**

Exemple :

Soit les fonctions paramétrées suivantes :

$$x = v\cos(u),$$
$$y = v\sin(u)$$
$$z = 2u.$$

Sur le domaine [0 , 2π] pour u et [0, 2] pour v avec une longueur de maillage de h=0.2, on aura le script suivant :

```
1. [U,V] = meshgrid(0 : .2 : 2*pi, 0 : .2 : 2.1); %création du maillage
2. X = V.*cos(U);   %fonction paramétrée de X = f1(U,V)
3. Y = V.*sin(U);   %fonction paramétrée de Y = f2(U,V)
4. Z = 2*U;   %fonction paramétrée de Z = f3(U,V)
5. surf(X,Y,Z)   % tracer la surface dans le 3D
```

On aura le dessin suivant :

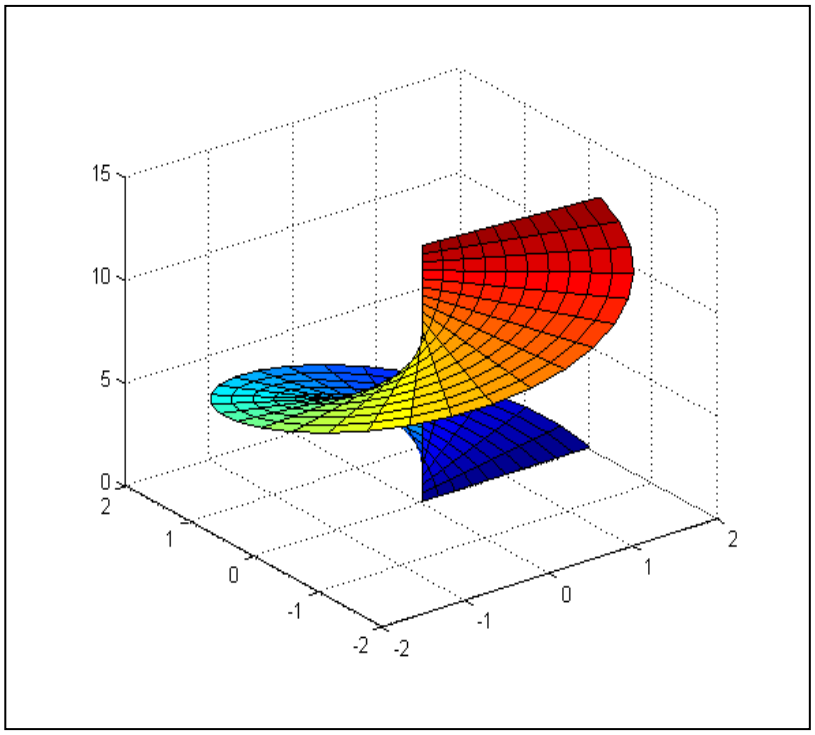


Figure V.11 : *représentation de surface paramétrée avec surf*

V.4.3.3. Tracé de courbes de contour

Pour tracer les courbes z=Cte d'une surface définie par $z=f(x,y)$, on peut utiliser la commande contour. Elle s'utilise comme les commandes précédentes (mesh et surf), mais fournit un graphe en 2D dans le plan (x,y). Plusieurs paramètres optionnels peuvent être spécifiés, notamment le nombre de courbes de contours à afficher.

Pour tracer les lignes de niveau de la fonction *f(x,y)* pour $x_{min} < x < x_{max}$ et $y_{min} < y < y_{max}$ on procède de la même manière utilisée précédemment

Création d'un maillage, de maille de longueur *h*, du domaine construit par :

$[x_{min}, x_{max}]$ x $[y_{min}, y_{max}]$ grâce à la commande **meshgrid**.

[X,Y] = meshgrid (x_{min}: h :x_{max}, y_{min}: h : y_{max}).

Définition de la fonction aux nœuds de ce maillage : Z = f(X,Y) ;

- Affichage des lignes de niveau grâce à la commande contour : **contour(X,Y,Z)**

Exemple

Pour schématiser la fonction $f(x,y) = x^* e^{-(x2+y2)}$ sur le domaine [-2 , 2] x [-2 , 2] en prenant un maillage de maille de longueur h=0.1, on écrit le script suivant :

```
1. x=[-2:0.1:2]; % Intervalle des x et longueur de maille= 0.1
2. y=[-2:0.1:2]; % Intervalle des y et longueur de maille= 0.1
3. [X,Y]=meshgrid(x,y); %création du maillage
4. Z=X.*exp(-X.^2-Y.^2); % fonction de deux variables
5. contour(X,Y,Z,20) % courbes de niveaux avec un nombre total de 20 courbes
6. xlabel('x'), ylabel('y')
```

L'application du mode contour engendrera le schéma suivant :

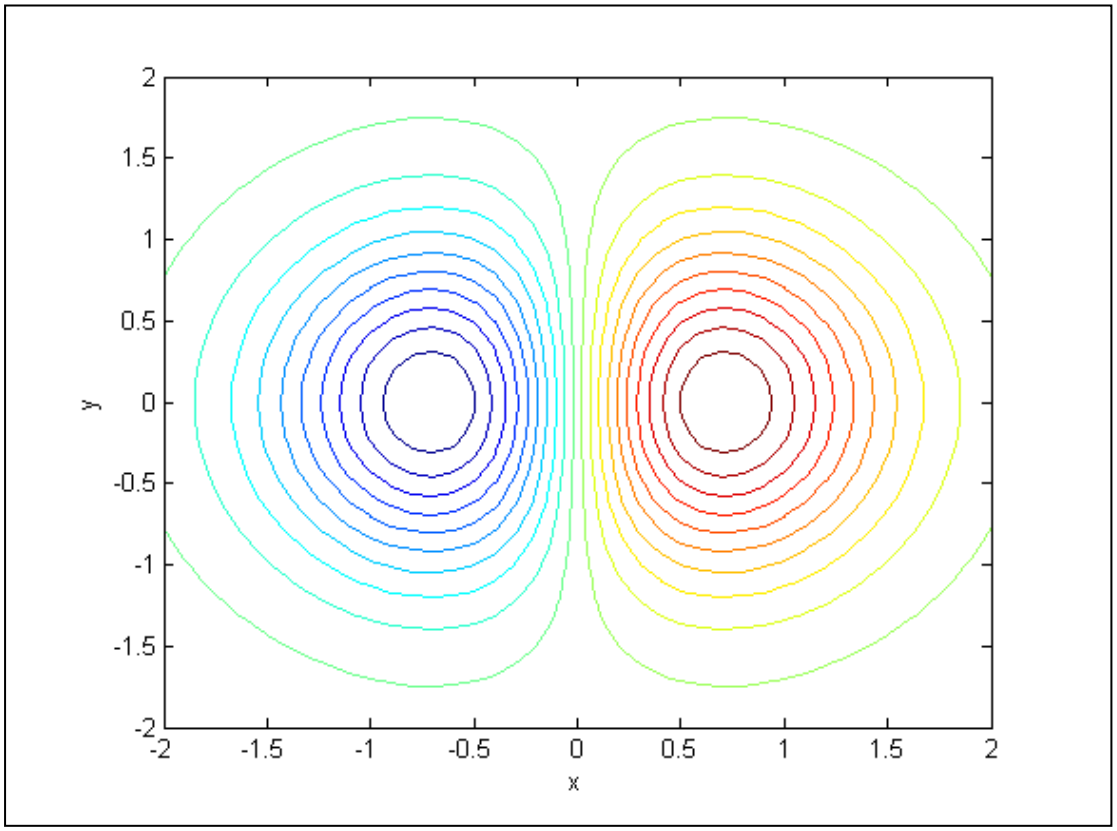


Figure V.12 : *Représentation de courbe à 2 variables avec contour*

V.5. Applications du chapitre V

V.5.1. Application 1

Ecrire un script Matlab permettant de

- Tracer la courbe représentant la fonction sin(2x) sur l'intervalle [0 , 2*pi] ;
- Ajouter dans le même graphique la fonction y = sin(t) dans l'intervalle [0 , 2*pi] avec un pas de 0.2 ;
- Ajouter les caractéristiques du graphe des 2 courbes.

```
1. figure % ouverture d'une nouvelle figure
2. fplot('sin(3*x)', [0 2*pi],'-r') % tracé de la fonction sin(3x) par la commande fplot
3. t=0:0.2:2*pi; % création du vecteur t dans l'intervalle [0 2π] avec un pas de 0.2
4. y=sin(t); % création du vecteur y
5. hold on % superposition d'une courbe supplémentaire
6. plot(t,y,':b') % tracé de la fonction y=f(t)
7. title('courbes trigonométriques')
8. xlabel('t'),ylabel('y')
9. legend('sin(3x)','cos (t)')
10. hold off % suspension de la superposition
```

Le graphe obtenu est dans la figure suivante :

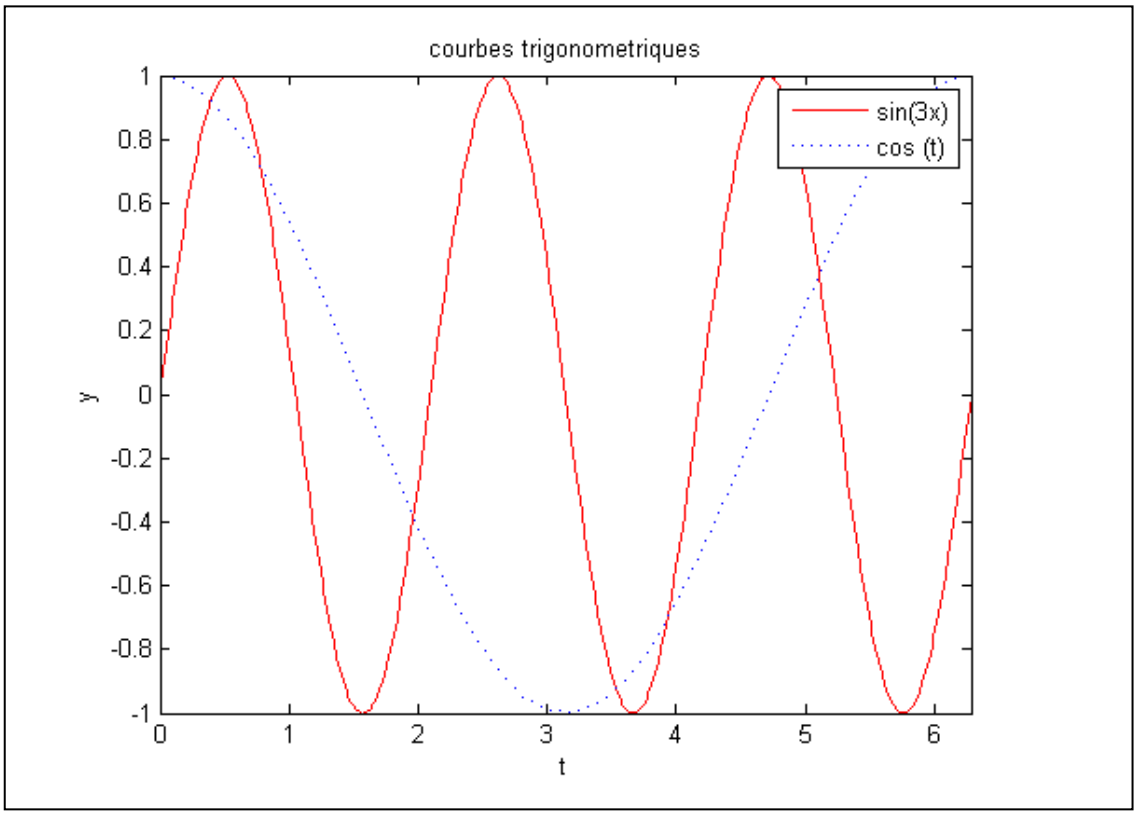


Figure V.13 : *Graphe de l'application V.1*

V.5.2. Application 2

En utilisant des sous fenêtres d'une figure, tracer le graphe des fonctions suivantes :

- $x \rightarrow x^2 \sin(x)$ $x \in [0, 2\pi]$;
- $x \rightarrow \sqrt{3x^2}$ $x \in [-3, 3]$;
- $x \rightarrow \cos(x)$ $x \in [0, \pi]$;
- $x \rightarrow x^3 \cos(x)$, $z \rightarrow \sin(x)$ $x \in [0, 2\pi]$;

```
1. figure
2. x=0:0.01:2*pi;
3. y1=(x.^2).*sin(x);
4. y2=(x.^3).*cos(x);
5. z=sin(x);
6. subplot(2,2,1),plot(x,y1,'-r'),title('courbe y1=x^2*sin(x)'),grid
7. subplot(2,2,2),fplot('sqrt(3*x^2)',[-3  3],':m'),title('courbe  sqrt(3*x^2)'),grid off
8. subplot(2,2,3),fplot('cos',[0  pi],'-.b'),title('courbe cos(x)'),grid
9. subplot(2,2,4),plot3(x,y2,z,'-k'),title('courbe 3Dimensions'),grid
```

Le graphe obtenu sera comme ci-après :

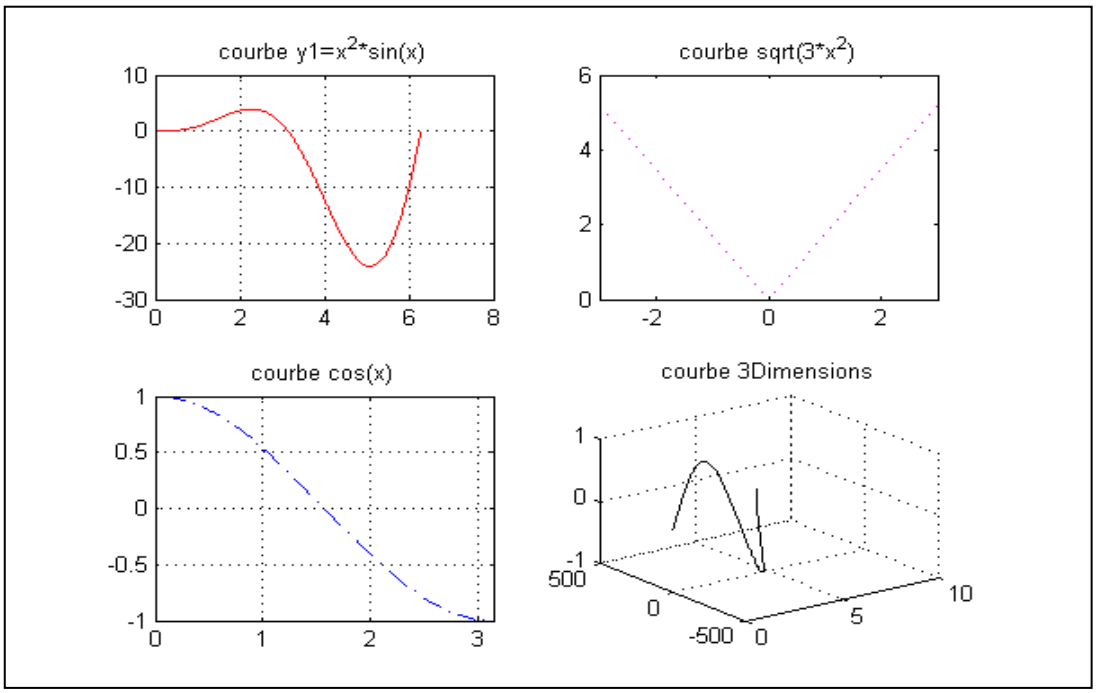


Figure V.14 : *Graphe résultat de l'application V.2*

V.5.3. Application 3

Suivez les étapes suivantes pour faire l'application 3 : [3]

- Créer le maillage carré d'un côté c=0.2 pour x ϵ[-1,1] et y ϵ[-3,3];
- Détermination de la fonction z=f(X,Y)= $Y^2 - X^3$;

- En utilisant des sous fenêtres d'une figure, tracer le graphe de la même fonction de surface z= f(X,Y) en utilisant les différentes commandes **mesh**, **meshc**, **surf**, **surfl**.

```
1. x = -1:0.2:1;
2. y = -3:0.2:3;
3. [X,Y] = meshgrid(x,y);
4. Z = Y.^2 - X.^3;
5. subplot(2,2,1),mesh(X, Y, Z);
6. xlabel('x'); ylabel('y'); zlabel('z'); title('Utilisation de mesh'),axis([-1 1 -3 3 0 10]);
7. subplot(2,2,2),meshc(X, Y, Z);
8. xlabel('x'); ylabel('y');zlabel('z'); title('Utilisation de meshc');
9. subplot(2,2,3),surf(X, Y, Z);
10. xlabel('x'); ylabel('y');zlabel('z'); title('Utilisation de surf'); ,axis([-1 1 -5 3 0 10]);
11. subplot(2,2,4),surfl(X, Y, Z);
12. xlabel('x'); ylabel('y');zlabel('z'); title('Utilisation de surfl');
```

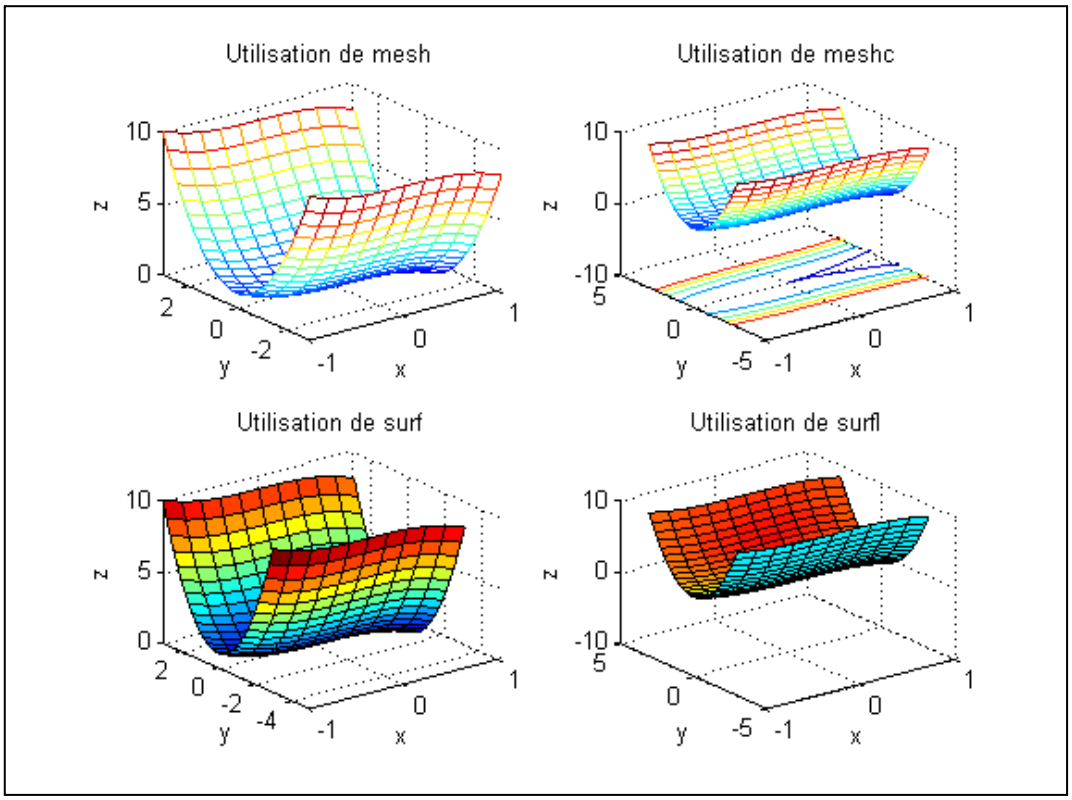


Figure V.15 : *Graphe résultant de l'application V.3*

Avec :

- mesh, qui trace une série de lignes entre les points de la surface en mode «lignes cachées» ;

- meshc, qui fonctionne comme mesh mais en ajoutant les courbes de niveau dans le plan (x,y) ;
- surf, qui «peint» la surface avec une couleur variant selon la cote ;
- surfl, qui «peint» la surface comme si elle était éclairée ;

VI.6. Conclusion

Les fonctions graphiques, incluant les fonctions de tracé 2D et 3D, permettent de visualiser les données et de communiquer les résultats. Comme déjà mentionné dans ce chapitre, il est possible de personnaliser les tracés de manière interactive ou programmatique.

Dans ce chapitre, nous avons exposé les différentes présentations graphiques utilisées sous Matlab soit sur plan ou dans l'espace. Pour tracer les fonctions linaires, nous avons utilisé les commandes plot et plot3. Pour améliorer la visibilité des graphiques, nous avons ajouté les caractéristiques des graphes. Nous avons vu comment présenter plusieurs graphiques sur une seule figure. Nous avons passé au tracé des graphes de 2 variables en utilisant la commande **meshgrid** pour la génération d'un maillage, et **mesh**, **surf** pour dessiner les graphes en 3D. A la fin du chapitre 3 applications, de graphisme sous Matlab, sont présentées et argumentées.

Conclusion générale

Ce présent polycopié permet à l'étudiant, de Master 1 Hydraulique de l'université de Bejaia, de découvrir un langage complet et efficace avec son environnement de travail ainsi que les méthodes de base de programmation mathématiques. Dans ce document, on trouvera les différents chapitres constituants le cours. Dans le 1er chapitre, nous avons présenté l'environnement du logiciel Matlab et ses constituants en détail. Un rappel sur les variables et leurs types a été donné ainsi que les fonctions et les différents opérateurs arithmétiques et logiques utilisés dans les expressions mathématiques. Dans le 2ème et 3ème chapitre, nous nous sommes intéressés aux vecteurs et matrices. Où nous avons étudié ces structures qui sont la base de fonctionnement de Matlab. Cela s'est passé par une illustration de la construction et la génération des vecteurs et des matrices ainsi que la présentation des différentes fonctions, utilisées lors des manipulations des vecteurs et des matrices. Le chapitre de la programmation sous Matlab est très important pour l'étudiant. Créer des scripts Matlab afin de résoudre des problèmes dans la spécialité est la finalité de l'étudiant dans l'utilisation de ce Logiciel. Dans ce chapitre, on a présenté la structure des programmes Matlab, ensuite on a défini les différentes instructions de base (lecture, affichage, Test, boucles et fonction) du logiciel. La création et la résolution des polynômes sont illustrées avec des exemples à l'appui. Savoir créer et résoudre les systèmes d'équations linéaires est la dernière partie théorique de ce chapitre.

Les fonctions graphiques, incluant les fonctions de tracé 2D et 3D, permettent de visualiser les données et de communiquer les résultats. Comme déjà mentionné dans le dernier chapitre de ce polycopié, il est possible de personnaliser les tracés de manière interactive ou programmatique. Dans ce chapitre, nous avons exposé les différentes présentations graphiques utilisées sous Matlab soit dans le plan ou dans l'espace.

REFERENCES BIBLIOGRAPHIQUES

[1] Documentation en ligne de MathWorks : MATLAB Documentation - MathWorks Suisse

[2] Documents en ligne de scilab : https://www.scilab.org/tutorials

Espaces d'entraide :

- Developpez.com : Club des développeurs MATLAB & Simulink : actualités, cours, tutoriels, programmation, FAQ, forum, livres, sources (developpez.com)
- Matlab Central : Home - MATLAB Central (mathworks.com)

Cours mis en ligne par des Universités :

[3] Représentations graphiques [MATLAB, pour la résolution de problèmes numériques] (mines-albi.fr)
[4] http://www-gmm.insa-toulouse.fr/~roussier/poly_info_icbe.pdf
[5] https://www.ljll.math.upmc.fr/~postel/matlab/
[6] *http://w3.gel.ulaval.ca/~fortier/MAT19961/index2.html*
[7] https://homepages.laas.fr/yariba/enseignement/manuel-matlab.pdf
[8] Formation Matlab pour débutant - Cours MATLAB (cours-gratuit.com)
[9] https://homepages.laas.fr/yariba/enseignement/slides-matlab.pdf

Ouvrages - Livres

[10] JEAN-PIERRE GRENIER, « DEBUTER EN ALGORITHME AVEC MATLAB ET SCILAB ». EDITION ELLIPSE, 2007, 160 PAGES
[11] JEAN-THIERRY LAPRESTE. « INTRODUCTION A MATLAB ». ELLIPSES. 4IEME EDITION, 2015, 240 PAGES
[12] MOHAMED FADHEL SAAD, « PROGRAMMER EN MATLAB ». EDITIONS-ENI, 2020, 407 PAGES

[13] MOHAND MOKHTARI. « APPRENDRE ET MAITRISER MATLAB ». SPRINGER. BERLIN : SPRINGER, 1998, 728 PAGES.