République Algérienne Démocratique et Populaire Ministère de l'Enseignement Supérieur

Et de la Recherche Scientifique

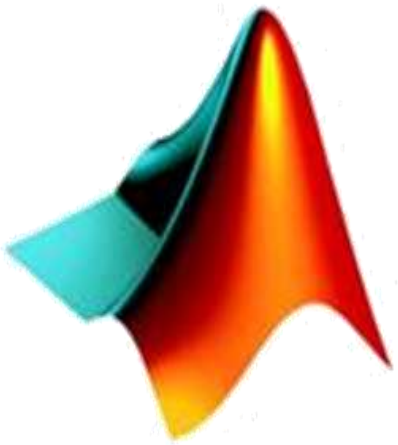
Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF

Faculté de Génie Mécanique

Département de Génie maritime

Polycopié de travaux pratique

Informatique 3



MATLAB®

Préparé par :

Dr. DEGHOUL.N

2024-2025

Avant-propos

Ce polycopié de travaux pratiques a été conçu dans le cadre de la formation des étudiants de deuxième année LMD de la filière Génie Maritime. Il a pour objectif de leur offrir une base solide et progressive pour découvrir et maîtriser les fonctionnalités essentielles du logiciel MATLAB.

Orienté vers la pratique, ce document guide l'étudiant étape par étape dans la prise en main de l'environnement MATLAB, la manipulation des variables, la gestion des données, la création de scripts et de fonctions, l'utilisation des structures de contrôle, ainsi que la visualisation graphique des résultats. Chaque chapitre aborde une thématique spécifique traitée en séance de travaux pratiques, illustrée par des exemples simples et concrets.

Ce support vise à développer chez l'étudiant des compétences techniques fondamentales, mais aussi une réelle autonomie dans la résolution de problèmes numériques, en préparation à des applications plus avancées au cours de sa formation.

Table des matières

Liste des figures

Introduction générale

Ce polycopié regroupe une série de travaux pratiques destinés à initier les étudiants à la programmation à l'aide de MATLAB, un environnement puissant largement utilisé dans les domaines de l'ingénierie, des sciences exactes et de l'analyse numérique.

À travers une série de TP construits de manière cohérente et adaptés au niveau de deuxième année Licence. Il vise à développer les compétences de base en programmation, en calcul numérique et en visualisation scientifique.

Dans le premier TP, les étudiants seront initiés à **l'**environnement MATLAB**,** à son interface graphique, à ses fenêtres principales et aux premières commandes de base à utiliser dans la console.

Le deuxième TP est consacré à la création de fichiers script**,** à la déclaration des variables et à l'exploration des types de données les plus utilisés dans MATLAB.

Le troisième TP traite de la lecture, de l'affichage et de la sauvegarde des données, indispensables pour interagir avec l'utilisateur et gérer les résultats de calculs.

Dans le quatrième TP, les étudiants apprendront à manipuler les vecteurs et les matrices, qui sont les structures centrales du logiciel MATLAB.

Le cinquième TP introduit les instructions de contrôle, notamment les boucles (for, while) et les conditions (if, switch), qui permettent de structurer des algorithmes.

Le sixième TP porte sur la création et l'utilisation de fonctions personnalisées, une étape importante pour rendre les programmes plus clairs, réutilisables et organisés.

Enfin, le septième TP est consacré au graphisme, avec la réalisation de tracés en 2D et en 3D.

Nous avons terminé notre polycopié par une bibliographie qui contient les différentes ressources utilisées dans l'ouvrage.

TP N°01 : Présentation de l'environnement MATLAB

1.1.Introduction

MATLAB (**MAT**rix **LAB**oratory) est un environnement de calcul numérique et un langage de programmation de haut niveau largement utilisé dans les domaines de l'ingénierie, des sciences et de l'économie. Développé par MathWorks, il est conçu principalement pour faciliter les opérations sur les matrices, mais il propose également des fonctionnalités étendues pour la visualisation de données, la simulation, le traitement du signal, l'optimisation et bien plus encore. Grâce à ses bibliothèques puissantes et à son interface conviviale, MATLAB permet aux utilisateurs de résoudre des problèmes complexes, de développer des algorithmes et de visualiser rapidement les résultats. Son efficacité en fait un outil incontournable tant dans les laboratoires académiques que dans les industries, où il est employé pour modéliser, analyser et tester des systèmes dans des domaines aussi divers que l'automatique, les télécommunications, l'aérospatiale, la finance ou encore la bio-informatique. En somme, MATLAB offre une plateforme flexible et performante pour aborder un large éventail de problèmes scientifiques et techniques.

1.2.Interface-utilisateur Matlab

Après avoir installé MATLAB sur votre ordinateur, recherchez le raccourci ou l'icône MATLAB sur votre bureau ou dans votre menu "Démarrer". Cliquez dessus pour ouvrir le logiciel.

MATLAB s'ouvre alors avec plusieurs fenêtres, notamment :

- Fenêtre de commandes (**Command Window**)
- Espace de travail (**Workspace**)
- Historique des commandes (**Command History**)
- Répertoire de travail (**Current Folder**)

L'interface-utilisateur de MATLAB varie légèrement en fonction de la version de MATLAB et du type de machine utilisée.

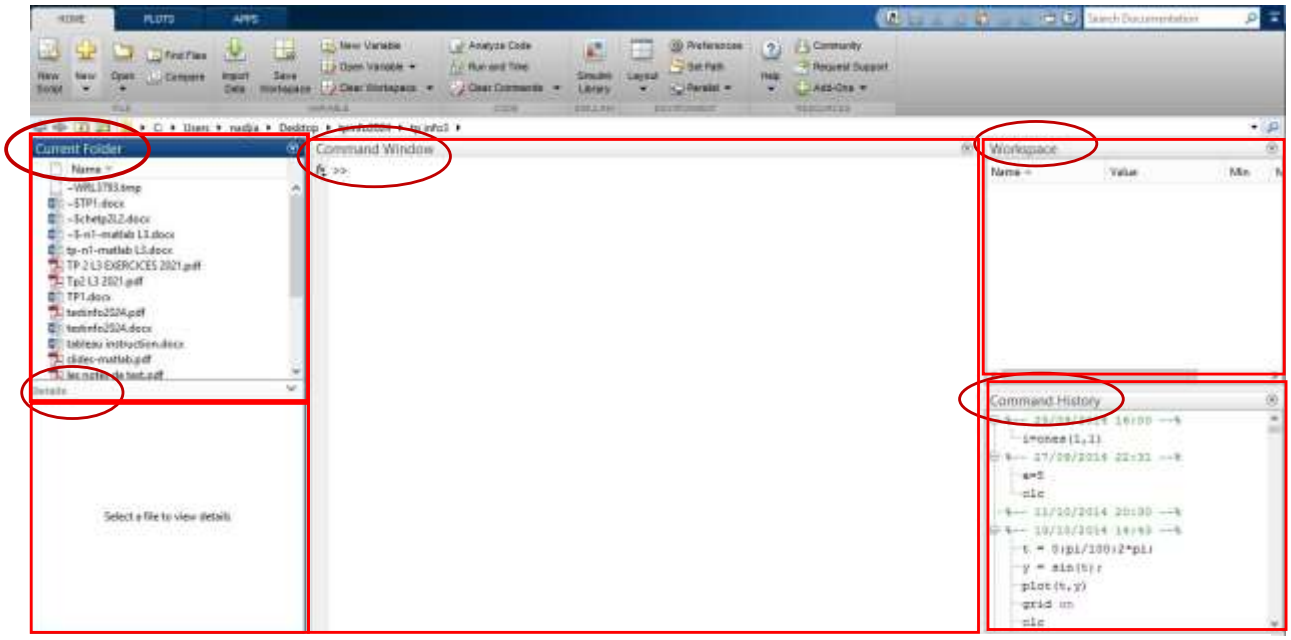


Figure 1. Interface-utilisateur Matlab

1.2.1. Fenêtre de commandes (Command Window)

Une fois dans MATLAB, la fenêtre principale qui s'affiche est la **fenêtre de commande** (***Command Window***). C'est ici que vous allez saisir les commandes et exécuter les scripts ou des fonctions.

L'invite de commande (>>) vous indique que MATLAB est prêt à recevoir des instructions qui sera exécutée après avoir tapée sur la touche « **Enter** ».

Figure 2. Fenêtre de Command Window

1.2.2. Historique de commandes (Command History)

Est une fenêtre qui affiche l'historique complet des commandes que vous avez exécutées lors de vos sessions de travail. Elle vous permet de consulter, réutiliser ou modifier des commandes sans avoir à les retaper manuellement dans la fenêtre de commande. C'est un outil précieux pour naviguer rapidement dans les commandes précédentes, surtout lors de travaux complexes ou répétitifs.

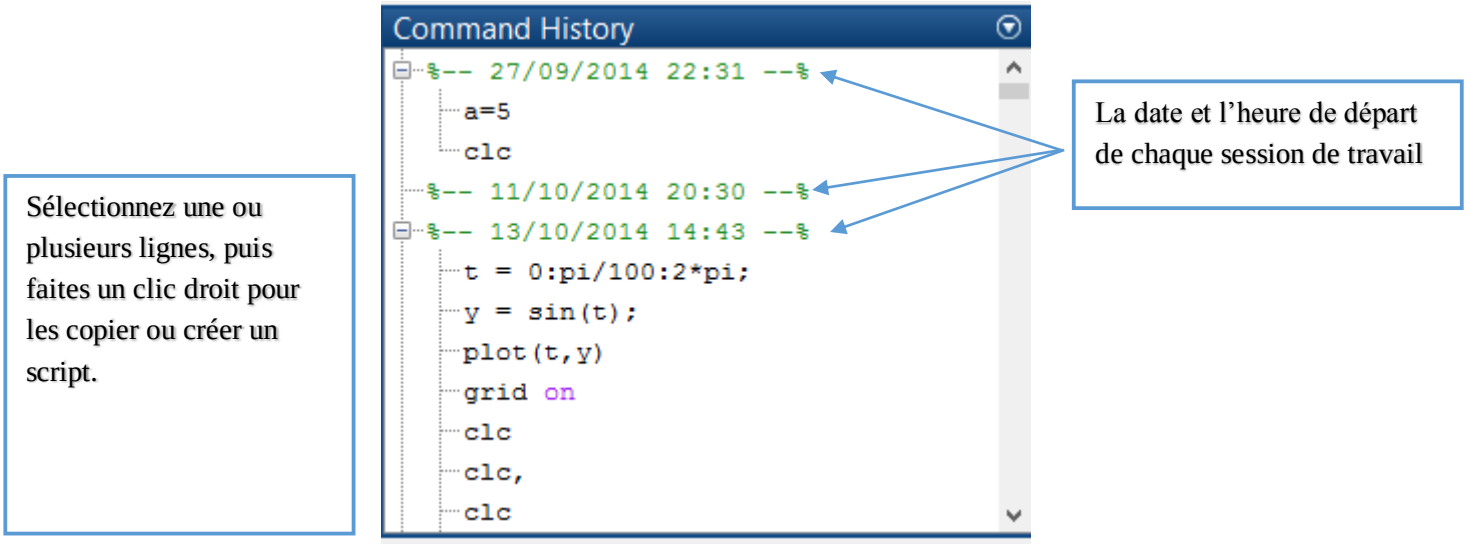


Figure 3. Fenêtre de Command History

1.2.3. Current folder

Le **Current Folder** dans MATLAB est une fenêtre qui affiche le contenu du répertoire de travail actuel, appelé **Current Directory**. Il permet de naviguer facilement dans les fichiers et dossiers de ce répertoire, facilitant ainsi l'accès aux scripts, fonctions, fichiers de données et autres ressources.

Le **Current Folder** offre des fonctionnalités pratiques comme l'ouverture, l'exécution, la suppression, le renommée des fichiers. En changeant de dossier dans le **Current Folder**, le répertoire de travail actif de MATLAB est automatiquement mis à jour, permettant une gestion simplifiée des fichiers pour les projets. Grâce à des options comme la recherche rapide et les actions contextuelles (clic droit), il permet de gagner du temps et d'organiser efficacement les fichiers nécessaires au développement en MATLAB.

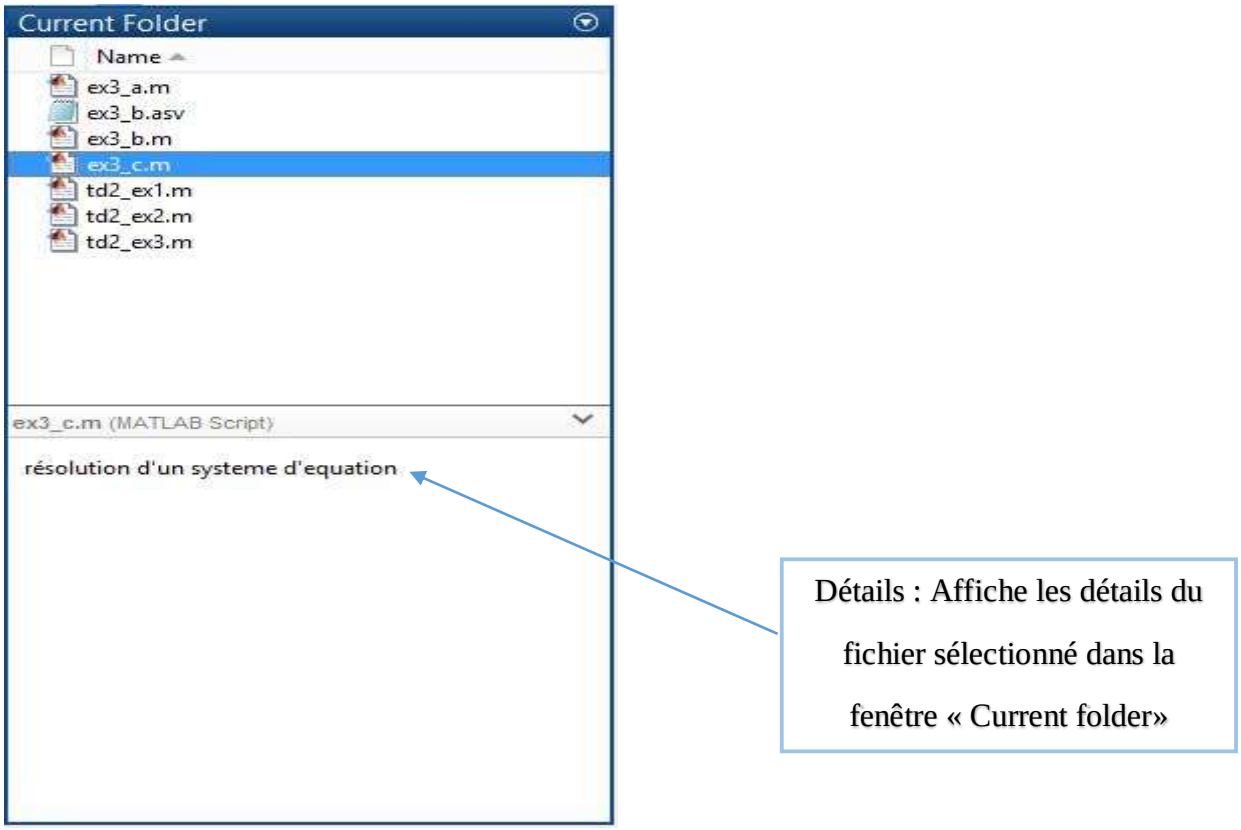


Figure 4. Fenêtre de Current folder

1.2.4. Espace de travail (**Workspace**)

Le **Workspace** est une fenêtre qui affiche toutes les variables actives pendant une session de travail. Il permet de visualiser les noms, types, tailles et valeurs des variables en mémoire, facilitant ainsi leur gestion et leur modification.

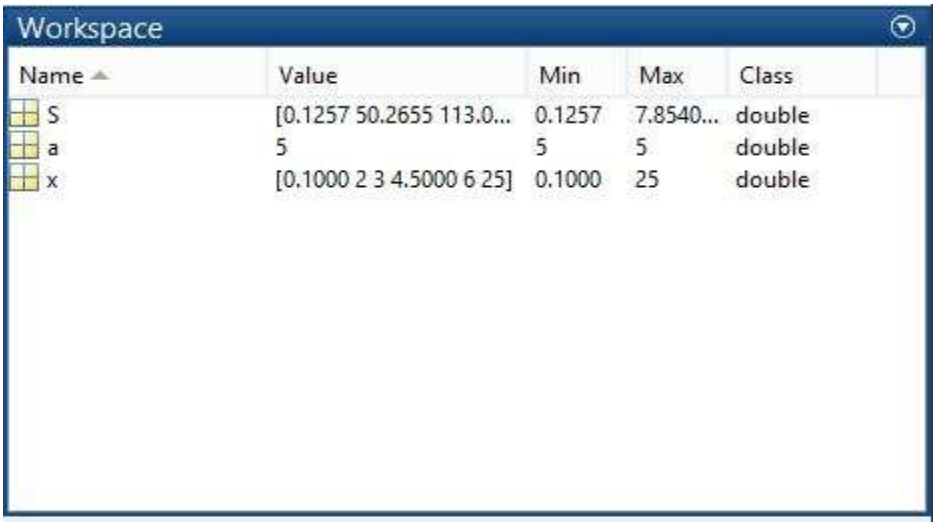


Figure 5. Fenêtre de Workspace

Pour afficher les variables existants dans le **workspace** utilisez les commandes **who** et **whos**

Who : affiche les variables existantes dans le workspace sans autres détails. **Exemple** :

```
Command Window
>> a=5;
>> b=[1 2 3];
>> who

Your variables are:

a  b
```

Whos : affiche les variables existantes dans le **workspace** avec détails tels que la taille, le type (classe) des variables, et la quantité de mémoire qu'elles occupent. **Exemple :**

```
Command Window
>> a=5;
>> b=[1 2 3];
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
  b           1x3                  24  double
```

Pour effacer une ou plusieurs variables dans MATLAB, vous utiliser la commande **clear**. Cela supprime les variables du **Workspace**, libérant ainsi la mémoire qu'elle occupait.

Voici quelques exemples :

Pour supprimer une variable spécifique, par exemple **a**, utilisez la commande **clear a** :

```
Command Window
>> a=5;
>> b=[1 2 3];
>> clear a
>> who

Your variables are:

b
```

Si vous voulez supprimer plusieurs variables, spécifiez leurs noms :

```
Command Window
>> a=5;
>> b=[1 2 3];
>> c=2;
>> d=4;
>> clear a b c
>> who

Your variables are:

d
```

Pour supprimer toutes les variables du **Workspace**, sans exception, utilisez la commande **clear** sans nom de variable :

```
Command Window
>> clear
>> who
>>
```

Pour supprimer toutes les variables du **Workspace**, à l'exception de celles que vous spécifiez, par exemple **b** utilisez la commande **clearvars -except b**

```
Command Window
>> a=5;
>> b=[1 2 3];
>> c=3.12;
>> d=4.5e5;
>> clearvars -except b   % supprime toute les variables sauf b
>> who

Your variables are:

b
```

1.3.Application :

1. Tapez la commande ***a=5*** puis Tapez ***a=5;*** Quel est le rôle du symbole (**;**) ?
2. Tapez les commandes ***who*** et ***whos***.
3. Utilisez ↑ pour modifier a : ***a=2;***
4. Tapez la commande ***b=a+2;*** . Réexécutez les commandes ***who*** et ***whos*** en utilisant ↑. Tapez ***clear b***
5. Tapez la commande ***whos***. en utilisant ↑ Tapez ***clear*** puis Tapez ***whos.***
6. Tapez la commande : ***clc .*** Quel est le rôle de cette commande ?

TP N°02 : Fichiers script et types de données et de variables

2.1.Modes de fonctionnement MATLAB

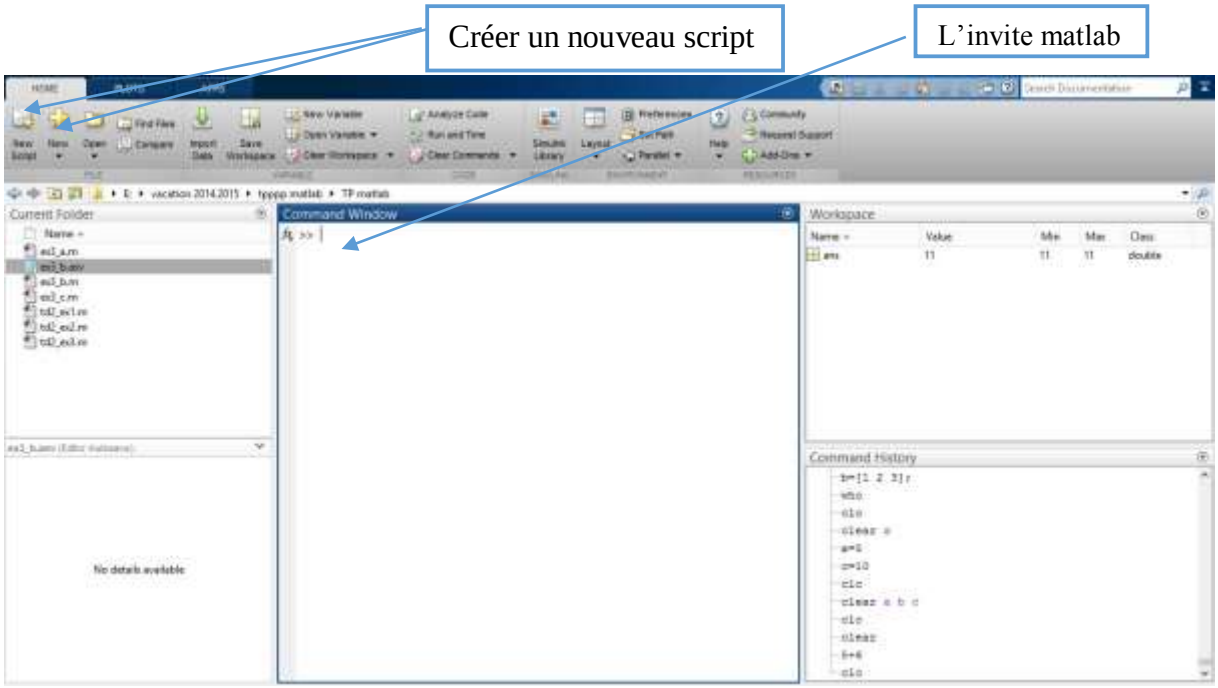
MATLAB offre trois principaux modes de fonctionnement adaptés à différents besoins de programmation et d'analyse.

- **le mode interactif**, où l'utilisateur peut exécuter des commandes directement dans la fenêtre de commande, obtenant immédiatement des résultats. Ce mode est idéal pour les tests rapides et les calculs simples.
- **mode script**, qui permet de regrouper plusieurs commandes dans un ***fichier.m*** et de les exécuter ensemble, facilitant ainsi l'automatisation de tâches répétitives.
- **mode fonction** permet de créer des fonctions avec des arguments d'entrée et de sortie, rendant le code plus modulaire et réutilisable. Ces fonctions sont également enregistrées dans des ***fichiers .m*** et permettent une organisation plus structurée des programmes. Ces modes permettent à MATLAB de s'adapter à une large gamme d'usages, depuis des opérations interactives simples jusqu'à des projets de programmation complexes.

2.2.Fichiers script

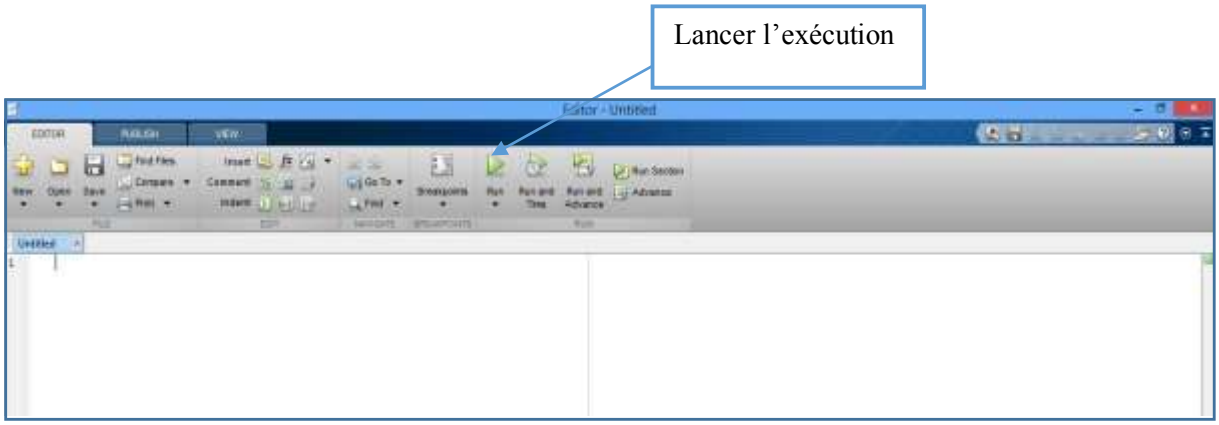
Un **fichier script** en MATLAB est un fichier texte contenant une séquence de commandes MATLAB qui est exécutée ensemble. Les fichiers script sont enregistrés avec l'extension **.m**

- Pour créer un script, ouvrez l'**éditeur MATLAB** (en utilisant la commande **edit** ou en cliquant sur **New Script**).



- Écrivez vos commandes MATLAB dans l'éditeur et enregistrez le fichier avec une extension (***.m***) par exemple **mon_script.m**

Pour exécuter le script Tapez simplement son nom sans l'extension ***.m*** comme exemple **mon_script** dans la fenêtre de commande, ou bien lancer directement (***Run***).



Commentaire : Vous pouvez ajouter des commentaires dans un script en utilisant le symbole **%**. Qui servent à expliquer le programme ou à ajouter des notes. (Les commentaires ne sont pas exécutés par MATLAB).

input : la fonction ***input*** est utilisée pour demander des entrées à l'utilisateur. Cela permet de récupérer des données depuis la console pour les utiliser dans des calculs ou des traitements dans un script ou une fonction.

x = input ('Entrez un nombre : ') ;

- Si l'utilisateur entre 5, alors la variable x sera assigné à la valeur 5.

2.3.Types de données et de variables

Les trois types de variables les plus courants dans Matlab sont : les réels, les complexes et les chaînes de caractères. Le type logique est utilisé pour représenter les résultats de certaines fonctions.

En MATLAB, toute variable est interprétée comme un tableau contenant des éléments d'un type spécifique. Il existe trois types principaux de tableaux. Les scalaires, qui sont des tableaux avec une seule ligne et une seule colonne. Les vecteurs, qui sont des tableaux formés soit d'une seule ligne, soit d'une seule colonne. Enfin, les matrices, qui sont des tableaux comportant plusieurs lignes et colonnes. Par conséquent, une variable MATLAB est toujours un tableau, qu'il soit désigné comme scalaire, vecteur ou matrice, en fonction de sa structure.

2.3.1. Les nombres Réels

Un **nombre réel** est un type de donnée numérique qui représente une valeur décimale ou entière sans composante imaginaire. Par défaut, les nombres réels sont stockés en **double précision** (type double), ce qui permet de représenter des nombres avec une grande précision (jusqu'à environ 15 chiffres significatifs).

Les nombres réels peuvent être déclarés directement comme variables. **Exemple :**

```
Command Window
>> a=5;
>> c=3.12;
>> d=4.5e5;
>> b=14e-5;
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
  b           1x1                   8  double
  c           1x1                   8  double
  d           1x1                   8  double
```

2.3.2. Les nombres complexes

Un **nombre complexe** peut être créé en ajoutant une partie imaginaire à une partie réelle à l'aide de la lettre **i** ou **j,** Les nombres complexes peuvent être représentés en **forme cartésienne** z=a+bi ou **forme polaire** $z=re^{i\theta}$. **Exemple** :

```
Command Window
>> z=3+5i;   % forme cartésiènne
>> Z=7*exp(3i); % forme polaire
```

MATLAB fournit plusieurs fonctions pour extraire et manipuler les parties réelles et imaginaires, ou pour calculer d'autres propriétés des nombres complexes. Par exemple :

real(z) : Extraire la partie réelle du nombre complexe z.

imag(z) : Extraire la partie imaginaire de z.

abs(z) : Calcule le module du nombre complexe z.

angle(z) : Calcule l'argument du nombre complexe z.

conj(z) : Calcule le conjugué d'un nombre complexe z.

2.3.3. Chaine de caractères

Une **chaîne de caractères** est un type de donnée utilisé pour représenter du texte, comme des mots, des phrases ou des symboles. MATLAB prend en charge plusieurs façons de travailler avec des chaînes de caractères, et il offre de nombreuses fonctions pour manipuler et traiter ces chaînes.

Il existe deux principales manières de gérer les chaînes : en utilisant les **tableaux de caractères** (ancien format) et les **objets de chaînes** (introduits dans MATLAB R2016b).

Vous pouvez créer un tableau de caractères en entourant le texte entre des apostrophes simples ('). **Exemple** :

```
Command Window
>> ch1='MAT' ;
>> ch2='LAB' ;
>> whos
  Name      Size            Bytes  Class    Attributes

  ch1       1x3                 6  char
  ch2       1x3                 6  char

>> ch=[ch1,ch2]  % concaténation pour obtenir MATLAB

ch =

MATLAB

>> ch(3)        % extrait le troisième caractère

ans =

T

>> ch(2:4)     % extrait les caractères 2 à 4

ans =

ATL
```

2.3.4. Le type logique

Le **type logique** est utilisé pour représenter des valeurs **vraies** ou **fausses**. Ces valeurs logiques sont soit **true** (vrai), soit **false** (faux). Elles sont souvent utilisées dans les conditions, les boucles et les opérations logiques. **Exemple** :

Ici, **a** à la valeur logique true et **b** à la valeur logique false. En interne, MATLAB représente true comme 1 et false comme 0

```
Command Window
>> a=true

a =

     1

>> b=false

b =

     0

>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x1                   1  logical
  b           1x1                   1  logical
```

Les valeurs logiques peuvent être utilisées dans des expressions conditionnelles et des opérations logiques. Voici quelques exemples :

```
Command Window
>> x=5;
>> y=3;
>> test1=x>y

test1 =

     1

>> test2=x<y

test2 =

     0

>> test3=(x>0)&&(y>0)  % Renvoie true car les 2 conditions sont vraies

test3 =

     1

>> test4=(x<0)||(y>0) % Renvoie true car une des conditions est vraie

test4 =

     1
```

2.3.5. Les vecteurs

En MATLAB, **un vecteur** est un tableau unidimensionnel qui peut contenir soit une rangée d'éléments (vecteur ligne) soit une colonne d'éléments (vecteur colonne).

Vecteur ligne : Un vecteur ligne est créé en mettant les éléments entre crochets [ ] et en les séparant par des espaces ou des virgules. **Exemple** :

```
Command Window
>> v=[1 2 3 4 5]   %vecteur ligne de 5 éléments

v =

     1     2     3     4     5

>> w=[1,2,3,4,5] %vecteur ligne de 5 éléments

w =

     1     2     3     4     5
```

Vecteur colonne : Un vecteur colonne est créé en plaçant les éléments entre crochets et en les séparant par des points-virgules (**;**).**Exemple** :

```
Command Window
>> v=[1;2;3;4;5]   %vecteur colonne avec 5 éléments

v =

     1
     2
     3
     4
     5
```

2.3.6. Les matrices

Pour créer une matrice, vous utilisez des crochets [] et séparez les éléments avec des espaces (pour les colonnes) et des points-virgules ; (pour les lignes). **Exemple** :

```
Command Window
>> A=[1 2 3;4 5 6;7 8 9] % crée une matrice A (3*3)

A =

     1     2     3
     4     5     6
     7     8     9
```

MATLAB, inclut des constantes prédéfinies :

- **pi :** la valeur π
- **eps :** Le plus petit incrément possible entre deux nombres distincts
- **ans :** Variable par défaut où MATLAB stocke le résultat d'une expression si aucune autre variable n'est spécifiée.
- **NAN :** (Not a Number) Représente une valeur non définie.
- **Inf :** (Infini) Représente une valeur infinie

```
Command Window
>> pi

ans =

    3.1416

>> eps

ans =

   2.2204e-16

>> NaN

ans =

   NaN

>> inf

ans =

   Inf
```

2.4.Les opérations arithmétiques

Les opérations arithmétiques en MATLAB permettent d'effectuer des calculs numériques sur des scalaires, des vecteurs, et des matrices. Elles incluent :

- L'addition (+),
- La soustraction (-),
- La multiplication (*),
- La division (/),
- L'exponentiation (^).

MATLAB offre également des opérations élément par élément, comme la multiplication (**.***), la division (. /), Et l'exponentiation (.^), où les opérations sont appliquées individuellement à

chaque élément des matrices. Ces opérations sont essentielles pour manipuler et traiter efficacement les données numériques dans divers contextes scientifiques et techniques.

2.5.Les opérations logiques

- = = : Égalité
- ~ = : Inégalité
- > : Supérieur
- < : Inférieur
- >= : Supérieur ou égal
- <= : Inférieur ou égal
- & : et
- || : ou

TP N°03 : Lecture, Affichage et Sauvegarde des données

3.1.Sauvegarde des données

La sauvegarde des données en MATLAB est un aspect essentiel pour préserver les variables et les résultats des calculs pour une utilisation future. La méthode la plus courante pour enregistrer des données consiste à les stocker dans des fichiers **.mat** (fichiers binaires spécifiques à MATLAB), mais il est également possible de les sauvegarder dans d'autres formats, comme des fichiers ***texte*** ou ***Excel***.

3.1.1. Sauvegarde des données dans un fichier .mat

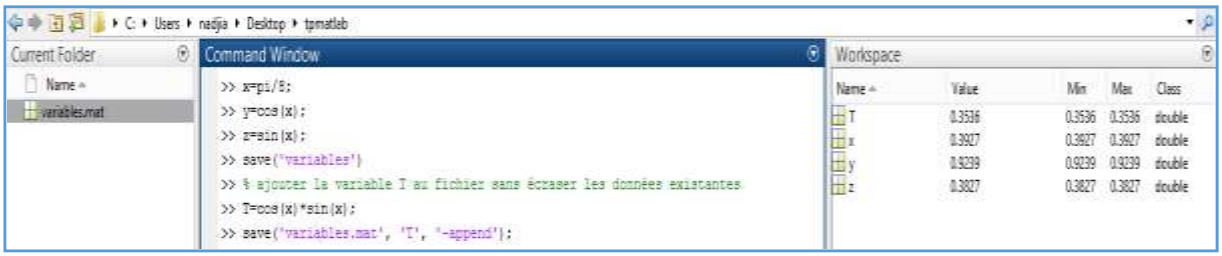
Vous utilisez la commande **save**, qui permet de stocker les variables du **Workspace** dans un fichier au format **.mat**. Voici plusieurs façons de sauvegarder les variables :

3.1.1.1.Sauvegarder toutes les variables

3.1.1.2.Sauvegarder des variables spécifiques

3.1.1.3.Sauvegarde partielle ou incrémentielle

Il est possible de sauvegarder les données de manière incrémentielle, en ajoutant de nouvelles variables à un fichier existant.



3.1.1.4.Sauvegarde en format texte ASCII

3.2.Lecture des données

Pour **lire des variables** à partir de différents types de fichiers ou du **workspace**, plusieurs fonctions sont disponibles.

3.2.1. Lecture de variables depuis un fichier.mat

La commande ***load*** permet de lire des variables enregistrées dans un **fichier.mat**

```
Command Window
>> % Charge uniquement la variable 'x'
>> load('variables', 'x');
>> % Charge toutes les variables du fichier
>> load('variables');
```

3.2.2. Lecture de variables depuis un fichier texte

Utilisez la fonction ***load***, pour lire des variables à partir de fichier text.

```
data = load('data.txt');    % Charge les données numériques du fichier texte
```

3.2.3. Lecture de variables depuis un fichier Excel

Utilisez les fonctions ***readtable*** ou ***xlsread***, pour lire des variables à partir de fichier Excel.

```
data = readtable('data.xlsx');   % Charge le fichier Excel en tant que table
```

3.3.Affichage des données

3.3.1. Affichage simple avec la fonction *disp*

La fonction **disp** est la méthode la plus simple et la plus directe pour afficher le contenu d'une variable à l'écran. Elle permet d'afficher des chaînes de caractères, des nombres, des matrices, des vecteurs, etc. **Exemple** :

```
Command Window
>> a = 5;
>> disp(a); % Affiche : 5
     5
```

Pour afficher un message

```
Command Window
>> a = 5;
>> disp('Le résultat est :'); disp(a);
Le résultat est :
     5
```

3.3.2. Affichage formaté avec *fprintf*

La commande **fprintf** permet un affichage plus flexible et formaté, particulièrement utile pour contrôler la mise en page et le format des données numériques ou textuelles.

%d : Indique que x est un entier (entier simple).
%.2f : Affiche un nombre décimal (flottant) avec 2 chiffres après la virgule.
\n : Indique un saut de ligne à la fin de chaque ligne affichée.

```
Command Window
>> x = 10;
>> y = 3.1416;
>> fprintf('La valeur de x est : %d\n et La valeur de y est : %.2f\n', x, y);
La valeur de x est : 10
 et La valeur de y est : 3.14
```

Dans cet exemple, **%d** est utilisé pour afficher un entier, et **%.2f** pour afficher un nombre flottant avec deux décimales.

Exemple :

Supposons que vous souhaitiez afficher la valeur d'une variable x ainsi que son carré de manière formatée. Voici un script simple utilisant **fprintf** pour cela :

script.m

```
1 -    clear all;
2 -    close all;
3 -    clc;
4      % Script qui utilise fprintf pour afficher des valeurs formatées
5      % Définir une variable x
6 -    x=5;
7      % Calculer le carré de x
8 -    carre_x=5^2;
9      % Afficher la valeur de x et de son carré avec fprintf
10 -   fprintf ('La valeur de x est : %d\n', x);
11 -   fprintf('Le carré de x est : %.2f\n', carre_x);
```

Résultat attendu

Command Window

```
La valeur de x est : 5
Le carré de x est : 25.00
```

3.3.3. Affichage formaté avec *num2str*

La fonction **num2str** (abréviation de number to string) permet de convertir une valeur numérique (scalaire, vecteur ou matrice) en une chaîne de caractères**.** Elle est très utile lorsqu'on veut combiner du texte et des nombres dans des messages ou pour l'affichage formaté sans utiliser fprintf.

```
clear all; close all ; clc;
v = [1 2 3];
s = num2str(v);
disp(['Vecteur : ' s])
```

Résultat attendu

Command Window

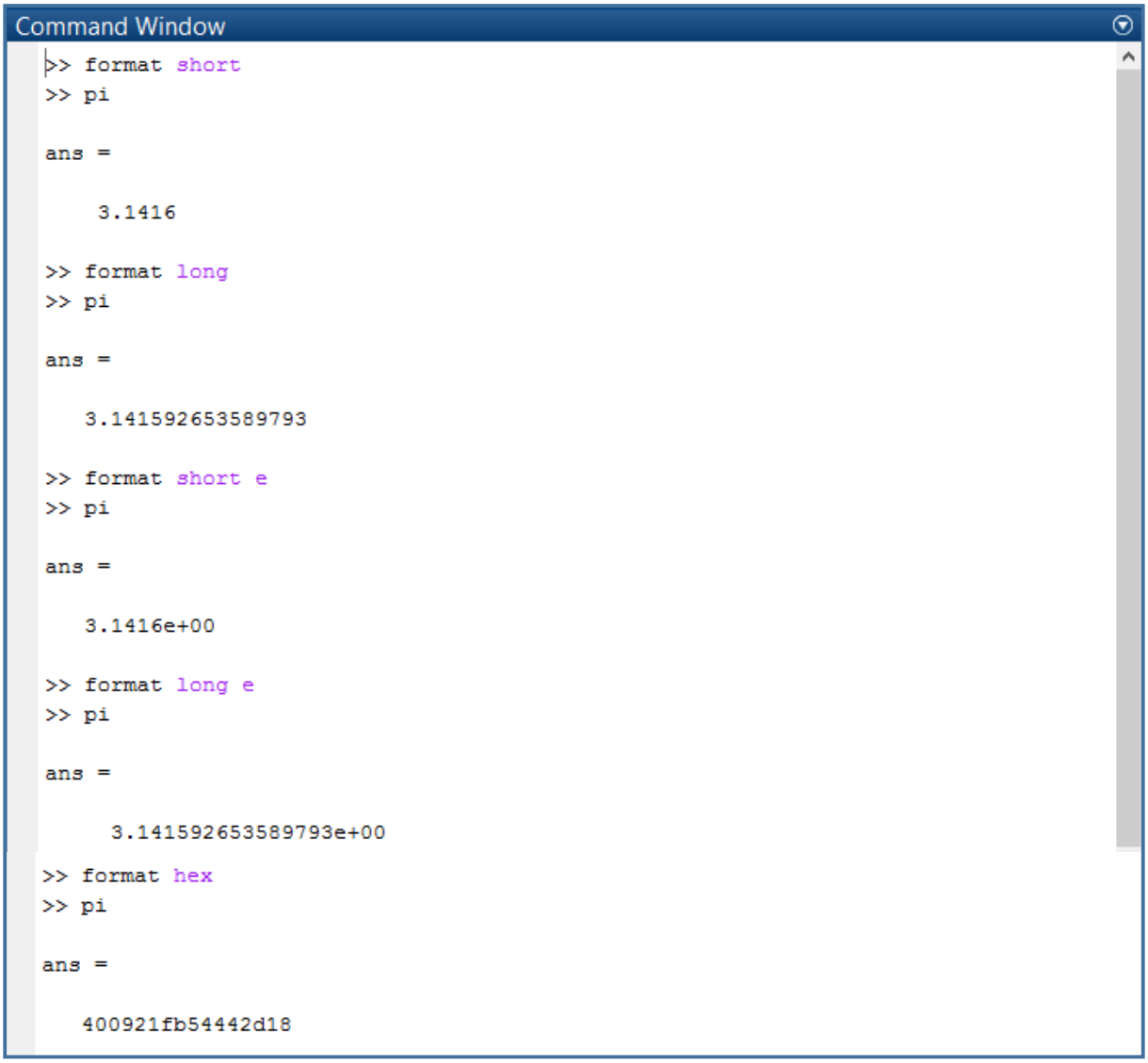
```
Vecteur : 1  2  3
fx >>
```

3.4.Format d'affichage

Le format d'affichage détermine comment les nombres et les résultats sont affichés dans la fenêtre de commande. Vous pouvez utiliser la commande format pour modifier le style d'affichage des nombres. Cela n'affecte pas la précision réelle des calculs, mais seulement la

façon dont les résultats sont affichés à l'écran. Par défaut le format est le format court à 5 chiffres.

Voici un aperçu des différents formats d'affichage disponibles en MATLAB.



Remarque : Pour voir le format actuelle on a la commande : **get(0,'format')**

TP N°04 : manipulation des Vecteurs et matrices

4.1.Les vecteurs

Un **vecteur** est un tableau unidimensionnel qui peut contenir soit une rangée d'éléments (vecteur ligne) soit une colonne d'éléments (vecteur colonne). Les vecteurs sont essentiels en programmation MATLAB, car ils permettent de stocker et de manipuler des séries de données numériquement.

Un vecteur ligne est créé en mettant les éléments entre crochets [ ] et en les séparant par des espaces ou des virgules. **Exemple** :

```
Command Window
>> v=[1 2 3 4 5]   %vecteur ligne de 5 éléments

v =

     1     2     3     4     5
```

Un vecteur colonne est créé en plaçant les éléments entre crochets et en les séparant par des points-virgules (**;**). **Exemple** :

```
Command Window
>> v=[1;2;3;4;5]   %vecteur colonne avec 5 éléments

v =

     1
     2
     3
     4
     5
```

4.1.1. Création de vecteurs Avec l'opérateur deux points (:)

L'opérateur **(:)** est utilisé pour générer des vecteurs en spécifiant le premier élément, et le dernier élément. (L'incrément est par défaut égal 1). **Exemple** :

```
Command Window
>> v=[1:5]   % Crée un vecteur [1 2 3 4 5]

v =

     1     2     3     4     5
```

Pour spécifier un pas (incrément), utilisez la syntaxe **début : incrément : fin.** Exemple :

```
Command Window
>> v = [0:2:8]  % Crée un vecteur [0 2 4 6 8 ]

v =

     0     2     4     6     8
```

Remarque : On peut éviter de mettre les crochets si les composants d'un vecteur varient d'un pas constant

4.1.2. Création de vecteur Avec la fonction *linspace*

La fonction **linspace** génère un vecteur avec un nombre spécifié d'éléments, également espacés entre une valeur de début et une valeur de fin. **Exemple :**

```
Command Window
>> v = linspace(0, 10, 5)  % Crée un vecteur de 0 à 10 avec 5 éléments

v =

         0    2.5000    5.0000    7.5000   10.0000
```

4.1.3. Fonctionnalités des vecteurs

MATLAB offre plusieurs fonctions utiles pour travailler avec des vecteurs :

size (v) : Retourne la taille du vecteur v, Ici le vecteur v est une matrice 1x5. **Exemple** :

```
Command Window
>> v = [1 2 3 4 5]

v =

     1     2     3     4     5

>> size(v)

ans =

     1     5
```

length(v) : Retourne la longueur du vecteur. **Exemple** :

```
Command Window
>> v = [1 2 3 4 5]

v =

     1     2     3     4     5

>> length (v)

ans =

     5
```

Transposition d'un vecteur

Pour changer un vecteur ligne en vecteur colonne (ou vice versa), vous utiliser l'opérateur de transposition **(')** ou à l'aide de la fonction **transpose**. **Exemple :**

```
Command Window
>> v = [3 4 5]

v =

     3     4     5

>> Vt= transpose(v)

Vt =

     3
     4
     5

>> v= Vt'

v =

     3     4     5
```

Accéder aux éléments d'un vecteur

Les éléments d'un vecteur peuvent être accédés à l'aide d'index et même modifier celui-ci directement. **Exemple** :

Remarque : (MATLAB commence à compter les index à partir de 1).

```
Command Window
>> v = [0 1 2 3 4 5 6 7 8 9 10]

v =

     0     1     2     3     4     5     6     7     8     9    10

>> v(6)      % Accéder au 6ème élément

ans =

     5

>> v(6)=9      % remplacer le 6ème élément par 9

v =

     0     1     2     3     4     9     6     7     8     9    10
```

La suppression d'un élément d'un vecteur. Exemple :

```
Command Window
>> v = [0:2:10]

v =

     0     2     4     6     8    10

>> v(3)=[]    % suppression de l'élément de l'indice 3

v =

     0     2     6     8    10
```

La suppression de plusieurs éléments d'un vecteur. Exemple :

```
Command Window
>> v = [0:10]

v =

     0     1     2     3     4     5     6     7     8     9    10

>> v(1:5)=[]   % suppresion des éléments de l'indice 1 à 5

v =

     5     6     7     8     9    10
```

On peut ajouter une valeur à un élément du vecteur. Exemple :

```
Command Window
>> v = [0:10]

v =

     0     1     2     3     4     5     6     7     8     9    10

>> v(1)=v(1)+10  % on ajoute la valeur 10 à l'élément d'indice 1

v =

    10     1     2     3     4     5     6     7     8     9    10
```

Concaténation des vecteurs : Il est aussi possible de concaténer des vecteurs. Exemple :

```
Command Window
>> v1=[1 2 3];
>> v2=[4 5 3];
>> v=[v1 v2]

v =

     1     2     3     4     5     3
```

4.1.4. Les opérations arithmétiques

Les vecteurs en MATLAB peuvent être manipulés par des opérations élémentaires, telles que l'addition, la multiplication, etc.

L'addition

```
Command Window
>> v1 = [1 2 3];
>> v2 = [4 5 6];
>> % l'addition de deux vecteurs v1 et v2
>> v3=v1+v2

v3 =

     5     7     9
```

La soustraction

```
Command Window
>> v1 = [1 2 3];
>> v2 = [4 5 6];
>> % soustraction
>> v4=v1-v2

v4 =

    -3    -3    -3
```

La multiplication

```
Command Window
>> v1=[1 2 3]; % vecteur 1*3
>> v2=[4;5;3]; % vecteur 3*1
>> v = v1*v2   % (1*3)*(3*1) = (1*1)

v =

    23

>> w = v2*v1  % (3*1)*(1*3) = (3*3)

w =

     4     8    12
     5    10    15
     3     6     9
```

Multiplication et division élément par élément

Pour multiplier ou diviser élément par élément deux vecteurs de même taille, on utilise l'opérateur (.*) pour la multiplication et (**.** /) Pour la division. Exemple :

```
Command Window
>> v = [0:5];
>> w = [6:11];
>> x = v.*w    % multiplication élément par élément

x =

     0     7    16    27    40    55

>> y = v./w    % division élément par élément

y =

         0    0.1429    0.2500    0.3333    0.4000    0.4545
```

Puissance élément par élément

Pour élever chaque élément d'un vecteur à une puissance donnée, on utilise l'opérateur (.^).

```
Command Window
>> v = [0:5];
>> x=4;
>> w = v.^x

w =

     0     1    16    81   256   625
```

Opération avec un scalaire

```
Command Window
v =

     0     1     2     3     4     5

>> v1= v+5

v1 =

     5     6     7     8     9    10

>> v2= v-2

v2 =

    -2    -1     0     1     2     3

>> v3 = v*4

v3 =

     0     4     8    12    16    20

>> v4 = v/2

v4 =

         0    0.5000    1.0000    1.5000    2.0000    2.5000
```

La racine carrée

Vous pouvez facilement calculer la **racine carrée** de chaque élément d'un vecteur en utilisant la fonction **sqrt**, qui applique l'opération élément par élément. Exemple :

```
Command Window
>> v = [0:5];
>> racine_carree = sqrt (v) %Calcul de la racine carrée élément par élément

racine_carree =

         0    1.0000    1.4142    1.7321    2.0000    2.2361
```

Remarque : La fonction **sqrt** ne fonctionne que sur des nombres réels positifs. Si le vecteur contient des nombres négatifs, MATLAB renverra des nombres complexes pour les racines carrées. Exemple :

```
Command Window
>> v = [0 -1 -4 5];
>> racine_carree = sqrt (v) %Calcul de la racine carrée élément par élément

racine_carree =

   0.0000 + 0.0000i   0.0000 + 1.0000i   0.0000 + 2.0000i   2.2361 + 0.0000i
```

La comparaison entre les vecteurs

MATLAB compare chaque élément de deux vecteurs de même taille et renvoie un vecteur logique contenant des valeurs de **1** (vrai) ou **0** (faux).

La comparaison entre les vecteurs se fait élément par élément en utilisant des opérateurs de comparaison. Exemple :

```
Command Window
>> v1 = [1, 2, 3];
>> v2 = [1, 0, 3];
>> v = v1 == v2  % comparaison d'égalité
v =
     1     0     1
>> w = v1 ~= v2  % comparaison d'inégalité
w =
     0     1     0
>> y = v1 > v2   % comparaison de supériorité
y =
     0     1     0
>> z = v1 >= v2  % Comparaison de supériorité ou égalité
z =
     1     1     1
>> t= v1 < v2    % Comparaison de l'infériorité
t =
     0     0     0
>> I= v1 <= v2    % Comparaison de l'infériorité ou égalité
I =
     1     0     1
```

Comparaison avec scalaire

```
Command Window
>> v=[ 5 12 4 3 8 4.5 10];
>> x = v > 10
x =
     0     1     0     0     0     0     0
```

4.1.5. Fonctions intégrées pour les vecteurs

MATLAB fournit plusieurs fonctions intégrées pour manipuler les vecteurs :

- **sum(v)** : Somme des éléments du vecteur.
- **prod(v)** : Produit des éléments du vecteur.
- **max(v)** : Renvoie la valeur maximale dans le vecteur.
- **min(v)** : Renvoie la valeur minimale dans le vecteur.
- **mean(v)** : Renvoie la moyenne des éléments du vecteur.
- **median(v)** : Renvoie la médiane des éléments.
- **sort(v)** : Trie les éléments du vecteur.
- **cumsum(v) :** Somme cumulé des éléments du vecteur.
- **cumprod(v)** : Produit cumulé des éléments du vecteur.

```
Command Window
>> v=[ 1 2 3 4 ];
>> sum(v)       % Somme des éléments du vecteur v
ans =
    10
>> prod(v)      %Produit des éléments du vecteur v
ans =
    24
>> max(v)       % Renvoie la valeur maximale dans le vecteur v
ans =
     4
>> min(v)       %Renvoie la valeur minimale dans le vecteur v
ans =
     1
>> mean(v)      % Renvoie la moyenne des éléments du vecteur v
ans =
    2.5000
>> cumsum(v)    % Somme cumulé des éléments du vecteur v
ans =
     1     3     6    10
>> cumprod(v)   % Produit cumulé des éléments du vecteur v
ans =
     1     2     6    24
>> median(v)    % Renvoie la médiane des éléments
ans =
    2.5000
>> sort(v)      % Trie les éléments du vecteur v
ans =
     1     2     3     4
```

4.1.6. Exemple pratique :

Voici un exemple pratique qui illustre comment manipulé un vecteur dans MATLAB, y compris des opérations courantes comme la création, la sélection d'éléments, l'ajout, la suppression et quelques calculs de base.

Code MATLAB

```
clear all; close all ; clc;
%  Créer et Afficher le vecteur v
v= [5, 10, 15, 20, 25, 30, 35];
disp('Le vecteur est :'); disp(v);
%  Accéder à des éléments spécifiques du vecteur
element_3 = v(3); % Accéder au 3e élément
disp(['Le 3e élément est : ', num2str(element_3)]);
% Accéder aux éléments 2 à 5 du vecteur
sous_v = v(2:5);
disp('Les éléments 2 à 5 du vecteur v sont :'); disp(sous_v);
%  Ajouter un élément au vecteur
 v = [v, 40];      % Ajouter l'élément 40 à la fin du vecteur
disp('Le vecteur v après ajout de 40 est :'); disp(v);
%  Supprimer un élément du vecteur
v(3) = []; % Supprimer le 3e élément
disp('Le vecteur v après suppression du 3e élément est :'); disp(v);
v_double = v * 2; % Multiplier chaque élément par 2
disp('Le vecteur v après multiplication par 2 :'); disp(v_double);
v_plus_10 = v + 10; % Ajouter 10 à chaque élément
disp('Le vecteur v après ajout de 10 à chaque élément :'); disp(v_plus_10);
somme_v = sum(v); % Somme des éléments
moyenne_v = mean(v); % Moyenne des éléments
max_v = max(v);  % Maximum  des éléments
min_v = min(v); % Minimum des éléments
% Afficher les résultats des calculs
disp(['La somme des éléments du v est : ', num2str(somme_v)]);
disp(['La moyenne des éléments du v est : ', num2str(moyenne_v)]);
disp(['Le maximum du vecteur est : ', num2str(max_v)]);
disp(['Le minimum du vecteur est : ', num2str(min_v)]);
```

Résultats :

```
Command Window
  Le vecteur est :
       5    10    15    20    25    30    35

  Le 3e élément est : 15
  Les éléments 2 à 5 du vecteur v sont :
      10    15    20    25

  Le vecteur v après ajout de 40 est :
       5    10    15    20    25    30    35    40

  Le vecteur v après suppression du 3e élément est :
       5    10    20    25    30    35    40

  Le vecteur v après multiplication par 2 :
      10    20    40    50    60    70    80

  Le vecteur v après ajout de 10 à chaque élément :
      15    20    30    35    40    45    50

  La somme des éléments du v est : 165
  La moyenne des éléments du v est : 23.5714
  Le maximum du vecteur est : 40
  Le minimum du vecteur est : 5
fx >>
```

4.1.7. Travail demandé :

Exercice N°1 :

Créez un script MATLAB qui :

1. Définit un vecteur de 5 éléments (au choix).
2. Calcule et affiche :

- La somme des éléments du vecteur.
- La moyenne des éléments.
- Le produit des éléments.

3. Affiche les résultats à l'aide de la fonction **fprintf** pour un formatage clair.

Exercice N° 2 :

Créez un script MATLAB qui :

1. Définit un vecteur contenant 50 valeurs également espacées entre 0 et 100 en utilisant la fonction **linspace**.
2. Calcule et affiche :

- La moyenne des éléments du vecteur.
- La valeur maximale et minimale du vecteur.

Exercice N° 3 :

Créez un script MATLAB qui effectue les opérations suivantes sur deux vecteurs V et W :

1. Créez deux vecteurs A et B de taille 6.
2. Effectuez les opérations élément par élément suivantes :

- Addition
- Soustraction
- Multiplication
- Division

3. Affichez chaque résultat avec des messages appropriés.
4. Calculez la somme de tous les éléments de chaque vecteur.
5. Affichez la moyenne des deux vecteurs.

Exercice N° 4 :

Créez un vecteur v = [12, 34, 56, 78, 90, 13, 45]. Ensuite :

- Trouvez les éléments de v qui sont supérieurs à 50.
- Comptez combien d'éléments du vecteur sont inférieurs ou égaux à 30.

4.2.Les matrices

Pour créer une matrice, vous pouvez utiliser des crochets [] et séparer les éléments avec des espaces (pour les colonnes) et des points-virgules ; (pour les lignes). Exemple :

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7]

A =

     1     2     3
     4     4     2
     9     8     7
```

Vous pouvez accéder à un élément spécifique d'une matrice en indiquant son indice de ligne et de colonne. Exemple :

```
Command Window
>> A(3,2) % élément a la 3eme ligne et 2eme colonne

ans =

     8
```

4.2.1. Opération sur les matrices

MATLAB permet d'effectuer des opérations courantes sur les matrices comme l'addition, la soustraction, la multiplication, etc.

Addition et soustraction

Les matrices doivent avoir la même taille. Exemple

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> B=[1 0 3;4 4 -2;-1 0 7];
>> D=A+B  % l'addition de deux matrices A et B

D =

     2     2     6
     8     8     0
     8     8    14

>> S=A-B % la soustraction

S =

     0     2     0
     0     0     4
    10     8     0
```

Multiplication de matrices

La multiplication de matrices en MATLAB utilise l'opérateur * et respecte les règles de l'algèbre matricielle (le nombre de colonnes de la première matrice doit correspondre au nombre de lignes de la seconde).

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> B=[1 0 3;4 4 -2;-1 0 7];
>> P=A*B % multiplication de deux matrices A et B

P =

     6     8    20
    18    16    18
    34    32    60
```

Multiplication élément par élément

Utilisez (**.***) pour multiplier chaque élément d'une matrice par l'élément correspondant d'une autre matrice de même taille.

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> B=[1 0 3;4 4 -2;-1 0 7];
>> P=A.*B          % multiplication élément par élément

P =

     1     0     9
    16    16    -4
    -9     0    49
```

Division élément par élément

Utilisez **. /** Pour diviser chaque élément d'une matrice par l'élément correspondant d'une autre matrice de même taille.

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> B=[1 2 3;4 4 -2;-1 4 7];
>> D=A./B   % division élément par élément

D =

     1     1     1
     1     1    -1
    -9     2     1
```

4.2.2. Fonctions utiles pour les matrices

Transposée d'une matrice

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> transposee_A = A'

transposee_A =

     1     4     9
     2     4     8
     3     2     7
```

Déterminant d'une matrice carrée

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> det_A = det(A)  % Calcul du déterminant de A

det_A =

   -20
```

Inverse d'une matrice carrée

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> inv_A = inv(A)  % Calcul de l'inverse de A

inv_A =

   -0.6000   -0.5000    0.4000
    0.5000    1.0000   -0.5000
    0.2000   -0.5000    0.2000
```

Dimensions d'une matrice

Vous pouvez obtenir les dimensions d'une matrice avec la fonction **size**.

```
Command Window
>> A=[1 2 3;4 4 2;9 8 7];
>> size(A)

ans =

     3     3
```

Matrice identité

```
Command Window
>> I = eye(3)  % Matrice identité 3x3

I =

     1     0     0
     0     1     0
     0     0     1
```

Matrice de zéros

```
Command Window
>> zeros_matrix = zeros(2, 3)  % Matrice 2x3 de zéros

zeros_matrix =

     0     0     0
     0     0     0
```

Matrice de uns

```
Command Window
>> ones_matrix = ones(2, 4)  % Matrice 2x4 de uns

ones_matrix =

     1     1     1     1
     1     1     1     1
```

Matrice aléatoire

```
Command Window
>> A = rand(2,3)  % Matrice 2x3 avec des éléments aléatoires

A =

    0.8147    0.1270    0.6324
    0.9058    0.9134    0.0975
```

Concaténation des matrices

Vous pouvez concaténer des matrices horizontalement ou verticalement.

Concaténation horizontale

```
Command Window
>> A=[1 2 3;4 5 6]; % matrice 2*3
>> B=[1 2 ;4 4 ];   % matrice 2*2
>> H = [A B]        %Concaténation horizontale

H =

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

Concaténation verticale

```
Command Window
>> A=[1 2 3;4 5 6]; % matrice 2*3
>> B=[1 2 3;4 4 4];   % matrice 2*3
>> V = [A ;B]        %Concaténation verticale

V =

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

Opérations sur les lignes et les colonnes

Somme des éléments d'une matrice par ligne ou par colonne

```
Command Window
>> A=[1 2 3;4 5 6];
>> sum_col = sum(A)  % Somme de chaque colonne

sum_col =

     5     7     9

>> sum_row = sum(A, 2)  % Somme de chaque ligne

sum_row =

     6
    15
```

Produit des éléments d'une matrice par ligne ou par colonne

```
Command Window
>> A=[1 2 3;4 5 6];
>> prod_col = prod(A)  % Produit de chaque colonne

prod_col =

     4    10    18

>> prod_row = prod(A, 2)  % Produit de chaque ligne

prod_row =

     6
   120
```

Extraction de sous-matrices

Pour extraire une sous matrice, vous utilisez la syntaxe suivante :
Sous_matrice = matrice (lignes, colonnes) ; Où lignes et colonnes représentent les indices des lignes et des colonnes que vous voulez extraire.

Extraction d'une seule ligne
Supposons que vous ayez une matrice A et que vous souhaitiez extraire la 2ème ligne :

```
Command Window
>> A=[1 2 3;4 5 6];
>> ligne_2 = A(2, :)  % Extraire toute la 2ème ligne

ligne_2 =

     4     5     6
```

Extraction d'une seule colonne
Supposons que vous ayez une matrice A et que vous souhaitiez extraire la 3ème colonne:

```
Command Window
>> A=[1 2 3;4 5 6];
>> colonne_3 = A(:, 3)  % Extraire toute la 3ème colonne

colonne_3 =

     3
     6
```

Extraction d'un bloc de sous-matrice

Vous pouvez extraire un bloc de sous-matrice. Par exemple, si vous souhaitez extraire les éléments de la 1ème et 2ème ligne, ainsi que les colonnes 2 et 3, vous pouvez faire ceci :

```
Command Window
>> A=[1 2 3;4 5 6];
>> sous_matrice = A(1:2, 2:3)  % Extraire les lignes 2 à 3 et colonnes 1 à 2

sous_matrice =

     2     3
     5     6
```

Extraction de plusieurs lignes/colonnes non consécutives

Si vous voulez extraire des lignes ou des colonnes qui ne sont pas adjacentes, vous pouvez spécifier les indices directement dans un vecteur. Par exemple, pour extraire la 1ère et 3ème ligne, ainsi que la 2ème et 3ème colonne :

```
Command Window
>> A=[1 2 3;4 5 6;7 8 9];
>> sous_matrice = A([1, 3], [2, 3])  % Extraire les lignes 1 et 3, colonnes 2 et 3

sous_matrice =

     2     3
     8     9
```

Extraction d'une sous-matrice entière avec *end*

L'opérateur end en MATLAB représente le dernier indice d'une dimension (ligne ou colonne). Il est très utile pour extraire des sous-matrices quand vous ne connaissez pas à l'avance les dimensions exactes de la matrice.

Exemple : Extraire de la 2ème à la dernière ligne et la 2ème à la dernière colonne :

```
Command Window
>> A=[1 2 3 5;4 5 6 9;7 8 9 2];
>> % Extraire à partir de la 2ème ligne/colonne jusqu'à la dernière
>> sous_matrice = A(2:end, 2:end)

sous_matrice =

     5     6     9
     8     9     2
```

Modifier une sous-matrice

Vous pouvez aussi modifier une sous-matrice en affectant de nouvelles valeurs à une section de la matrice. Par exemple, remplacer la sous-matrice des lignes 1 et 2 et colonnes 1 et 2 par une nouvelle matrice :

```
Command Window
>> A=[1 2 3 5;4 5 6 9;7 8 9 2];
>> A(1:2, 1:2) = [10 11; 12 13]

A =

    10    11     3     5
    12    13     6     9
     7     8     9     2
```

La fonction find

La fonction **find** est utilisée pour trouver les indices des éléments non nuls ou qui satisfont une condition logique spécifique dans un tableau, un vecteur ou une matrice. . Elle est très utile pour extraire les positions des éléments qui répondent à un critère donné. Exemples :

find(x) : Renvoie les indices des éléments non nuls.

```
Command Window
>> x = [0, 2, 0, 4, 0, 6];
indices_non_nuls = find(x)

indices_non_nuls =

     2     4     6
```

find(x > condition) : Renvoie les indices des éléments qui satisfont la condition donnée.

```
Command Window
>> x= [5, 10, 15, 20, 25, 30];
indices_sup_15 = find(x > 15)

indices_sup_15 =

     4     5     6
```

[ligne, colonne] = find(x) : Renvoie les indices des lignes et des colonnes des éléments non nuls dans une matrice.

```
Command Window
>> A = [3 0 3; 0 5 0; 5 0 9];
>> [ligne, colonne] = find(A)  % Trouver les indices des éléments non nuls

ligne =

     1
     3
     2
     1
     3


colonne =

     1
     1
     2
     3
     3
```

find(x > condition, n) : Renvoie les **n** premiers indices qui satisfont la condition.

Vous pouvez également récupérer les valeurs des éléments qui répondent à une condition en utilisant les indices retournés par **find.** Exemple :

```
Command Window
>> x= [5, 10, 15, 20, 25, 30];
>> indices = find(x > 10, 2)  % Trouver les 2 premiers éléments supérieurs à 10

indices =

     3     4

>> valeurs_sup_10 = x(indices)  % Récupérer les valeurs

valeurs_sup_10 =

    15    20
```

4.2.3. **Exemple pratique** :

Voici un exemple complet d'un script qui illustre différentes opérations sur une matrice.

Code MATLAB :

```
clear all; close all ; clc;
% Créer une matrice A
A = [1 2 3; 4 5 6; 7 8 9];
% Afficher la matrice
disp('Matrice A :'); disp(A);
% Calculer la transposée de A
A_transpose = A';
disp('Transposée de A :'); disp(A_transpose);
% Calculer le déterminant de A
det_A = det(A);
disp('Déterminant de A :'); disp(det_A);
% Créer une matrice identité de même taille que A
I = eye(size(A));
disp('Matrice identité :'); disp(I);
% Ajouter et soustraire deux matrices
B = [9 8 7; 6 5 4; 3 2 1];
addition = A + B;
soustraction = A - B;
disp('Addition de A et B :'); disp(addition);
disp('Soustraction de A et B :');
disp(soustraction);
% Multiplication élément par élément
multiplication_element = A .* B;
disp('Multiplication élément par élément :'); disp(multiplication_element);
 % Division élément par élément
division_element = A ./ B;
disp('Division élément par élément de A par B :'); disp(division_element);
```

Résultats :

Command Window

```
Matrice A :
     1     2     3
     4     5     6
     7     8     9

Transposée de A :
     1     4     7
     2     5     8
     3     6     9

Déterminant de A :
   6.6613e-16

Matrice identité :
     1     0     0
     0     1     0
     0     0     1

Addition de A et B :
    10    10    10
    10    10    10
    10    10    10

Soustraction de A et B :
    -8    -6    -4
    -2     0     2
     4     6     8

Multiplication élément par élément :
     9    16    21
    24    25    24
    21    16     9

Division élément par élément de A par B :
    0.1111    0.2500    0.4286
    0.6667    1.0000    1.5000
    2.3333    4.0000    9.0000
```

4.2.4. Travail demandé :

Exercice N°1 :

- Créez la matrice A

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- Affichez les dimensions de la matrice A
- Accédez au troisième élément de la première ligne
- Remplacez l'élément en position (2,2) par la valeur 0
- Ajouter la valeur 10 à chaque élément de la matrice A
- Transposez la matrice A

Exercice N°2 :

- Créez deux matrices **M** et **N**

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad N = \begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 3 & 1 & 2 \end{pmatrix}$$

- Multipliez chaque élément de la matrice M par les éléments correspondants de la matrice N.
- Divisez chaque élément de la matrice M par les éléments correspondants de la matrice N.

Exercice N°3 :

Utilisez la matrice **D** suivante

$$D = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 1 & 2 & 3 & 4 \\ 31 & 32 & 33 & 34 \end{pmatrix}$$

- Extraire la sous-matrice formée par les deux premières lignes et les deux premières colonnes.
- Extraire la dernière colonne de **D**.
- Extraire les éléments de la deuxième et troisième ligne, et des colonnes deux et trois de la matrice.
- Calculez la somme de tous les éléments de la matrice.

Exercice N°4 :

- Créez une matrice **B** de taille 5×5 avec des valeurs aléatoires entières comprises entre 1 et 20.
- Trouvez les indices des éléments de B qui sont supérieurs à 10.
- Remplacez tous les éléments de B supérieurs à 10 par 10.

4.3.Résolution d'un système d'équation linéaire

Pour résoudre un système d'équations linéaires en MATLAB, on peut utiliser plusieurs méthodes, dont la plus courante consiste à utiliser la fonction \ (backslash) ou bien l'inversion de matrice **inv**.

Considérons un système linéaire de la forme :

$$\boldsymbol{AX=B}$$

Où **A** est la matrice des coefficients, **X** est le vecteur des inconnues, et **B** est le vecteur des constantes.

Supposons le système suivant :

$$\begin{cases} \mathbf{2x+3y-5z = 5} \\ \mathbf{-2x+3y-5z = -2} \\ \mathbf{2x+3y-4z = 3} \end{cases}$$

Écrivez le système sous forme matricielle

$$\mathbf{A=}\begin{pmatrix} \mathbf{2} & \mathbf{3} & \mathbf{-5} \\ \mathbf{-2} & \mathbf{3} & \mathbf{5} \\ \mathbf{2} & \mathbf{3} & \mathbf{-4} \end{pmatrix} \qquad \mathbf{B=}\begin{pmatrix} \mathbf{5} \\ \mathbf{-2} \\ \mathbf{3} \end{pmatrix}$$

Résolvez pour x (qui contient les valeurs de x, y, et z) en utilisant la méthode \.

Où bien en utilisant l'inverse de la matrice A (si elle est inversible).

Solution en MATLAB :

```
clear all; close all ; clc;
%  Définir la matrice A et le vecteur B
A = [2 3 -5; -2 3 -5; 2 3 -4];
B = [5; -2; 3];

% Résoudre pour X
X = A \ B;  % ou bien  X = inv(A) * B;

% Afficher la solution
x= X(1);
y= X(2);
z= X(3);

disp('Les valeurs de x, y, et z sont :');
disp(x);disp(y);disp(z);
```

```
Command Window
Les valeurs de x, y, et z sont :
    1.7500

   -2.8333

    -2
```

4.3.1. Travail demandé :

Exercice N°1 :

Résoudre le système suivant :

$$\begin{cases} x+2y-3z = 5 \\ -2x+y-5z = 6 \\ 2x+3y+4z = 7 \end{cases}$$

4.4.Les polynômes

Les polynômes en MATLAB sont représentés sous forme des vecteurs contenant leurs coefficients ordonnés par ordre décroissant

Le polynôme $P(x)=2x^3-4x^2+3x-5$ est représenté par le vecteur **P = [2 -4 3 -5]**

```
Command Window
>> P=[2 -4 3 -5]

P =

     2    -4     3    -5
```

4.4.1. Fonctions utiles pour les matrices

Evaluation d'un polynôme

Pour évaluer un polynôme en un certain point x, on utilise la fonction **polyval**

Exemple : évaluez $P(x)=2x^3-4x^2+3x-5$ en **x=3**

```
Command Window
>> P = [2 -4 3 -5];  % Coefficients du polynôme P
   x = 2;            % Point d'évaluation
   y = polyval(P, x)

y =

     1
```

Les racines d'un polynôme.

La fonction **roots** permet de trouver les racines d'un polynôme.

Exemple : Trouver les racines du polynôme $P(x)=2x^3-4x^2+3x-5$

```
Command Window
>> P = [2 -4 3 -5];
racines = roots(P)

racines =

   1.9023 + 0.0000i
   0.0488 + 1.1453i
   0.0488 - 1.1453i
```

Multiplication de deux polynômes

Pour multiplier deux polynômes, on utilise la fonction **conv**.

Exemple : Multiplier $P(x)=2x^3-4x^2+3x-5$ par $Q(x)=x-5$

```
Command Window
>> P = [2 -4 3 -5];
>> Q=[1 -5];
>> produit = conv(P, Q)

produit =

     2   -14    23   -20    25
```

Division de polynômes

Pour diviser deux polynômes, utilisez la fonction deconv qui renvoie le quotient et le reste.

Exemple : Diviser $P(x)=2x^3-4x^2+3x-5$ par $Q(x)=x-5$

```
Command Window
>> P = [2 -4 3 -5];
>> Q=[1 -5];
>> division = deconv(P, Q)

division =

     2     6    33
```

La dérivation d'un polynôme

La fonction **polyder** permet de calculer la dérivée d'un polynôme.

Exemple : Trouver la dérivée de $P(x)=2x^3-4x^2+3x-5$

```
Command Window
>> P = [2 -4 3 -5];
>> D = polyder (P) % la dérivée

D =

     6    -8     3
```

Intégration d'un polynôme

La fonction **polyint** permet de calculer la primitive (intégrale) d'un polynôme.

Exemple : Calculer la primitive de $P(x)=2x^3-4x^2+3x-5$

```
Command Window
>> P = [2 -4 3 -5];
>> I = polyint (P) % l'integrale

I =

    0.5000   -1.3333    1.5000   -5.0000         0
```

4.4.2. Exemple pratique :

1. Représentez le polynôme $P(x)=4x^4-3x^3+2x^2-x+7$
2. Trouvez les racines de $P(x)$
3. Évaluez $P(x)$ pour $x=3$.
4. Calculez la dérivée de $P(x)$.
5. Multipliez $P(x)$ par le polynôme $Q(x)=x^2+1$.

Solution en MATLAB

```
clear all; close all ; clc;
% Représenter le polynôme P(x) = 4x^4 - 3x^3 + 2x^2 - x + 7
P = [4 -3 2 -1 7];

% Trouver les racines de P(x)
racines = roots(P);
disp('Les racines de P(x) sont :');
disp(racines);

% Évaluer P(x) pour x = 3
x = 3;
valeur_Px = polyval(P, x);
disp(['P(3) = ', num2str(valeur_Px)]);

% Calculer la dérivée de P(x)
derivee_P = polyder(P);
disp('La dérivée de P(x) est :');
disp(derivee_P);

% Représenter le polynôme Q(x) = x^2 + 1
Q = [1 0 1];

% Multiplier P(x) par Q(x)
produit_PQ = conv(P, Q);
disp('Le produit des polynômes P(x) et Q(x) est :');
disp(produit_PQ);
```

Résultats

Command Window

```
Les racines de P(x) sont :
   0.9512 + 0.8307i
   0.9512 - 0.8307i
  -0.5762 + 0.8749i
  -0.5762 - 0.8749i

P(3) = 265
La dérivée de P(x) est :
    16    -9     4    -1

Le produit des polynômes P(x) et Q(x) est :
     4    -3     6    -4     9    -1     7
```

TP N°05 : Instructions de contrôle (boucles for et while, instructions if et switch)

5.1. Structures itératives (ou répétitives)

5.1.1. La boucle for

La boucle for permet de faire se répéter un certain nombre de fois une (ou plusieurs) instruction (s) ; le nombre de fois est explicitement connu par l'utilisateur

Syntaxe générale

for ***variable = début : pas : fin***

% Instructions à exécuter

end

Exemples :

```
for   i = 0 :5 ;
    disp(i) ;
end
```

```
for   i = 0 :10 ;
    i=2*i;
   disp(i) ;
end
```

```
for   j = 0 :2:10 ;
    disp(j) ;
end
```

Exemple 1 : Calculer et afficher les carrés des nombres de 1 à 5.

Solution :

```
for i = 1:5
    i = i^2;
    disp('Le carré de i est '); disp(i);
end
```

Exemple 2 : afficher tous les multiples de 3 entre 1 et 30.

Solution :

```
for i = 1:30
    if mod(i, 3) == 0
        disp(i);
    end
end
```

Dans cette boucle for, on utilise mod(i, 3) == 0 pour vérifier si i est un multiple de 3 ou non. Si oui, il est affiché.

5.1.2. La boucle while

La boucle while répète un bloc de code tant qu'une condition est vraie. Elle est utile lorsque le nombre d'itérations n'est pas fixé et dépend d'une condition.

Syntaxe générale

while condition

% Instructions à exécuter

end

Exemples :

```
i=0;
while  i <5
i = i +1;
disp(i);
end
```

```
i =5;
while  i >0
i = i -1;
disp(i);
end
```

Exemple 1 : Calculer le produit des entiers jusqu'à ce qu'il dépasse 100.

Solution :

```
produit = 1;
i = 1;
while produit <= 100
    produit = produit * i;
    i = i + 1;
end
disp('Le produit des entiers jusqu'à ce point est : '); disp(produit);
```

Ici, la boucle while continue de multiplier les entiers jusqu'à ce que le produit dépasse 100.

5.1.3. Travail demandé

Exercice N°1 :

Ecrire un programme matlab pour calculer la somme s=1+2/2 !+3/3 !.....on arrête le calcul quand s>2.

5.2.Structure conditionnelle : tests if

La structure **if** permet d'exécuter des instructions en fonction d'une ou plusieurs conditions.

Syntaxe générale

if ***condition***
% Code exécuté si la condition est vraie
end

if ***condition***
% Code exécuté si la condition est vraie
else
% Code exécuté si la condition est fausse
end

if ***condition***
% Code exécuté si la condition est vraie
elseif ***autre_condition***
% Code exécuté si l'autre condition est vraie
else
% Code exécuté si aucune des conditions n'est vraie
end

Exemple N°1 : Tester le script suivant, pour $i = 5$ et $i = 10$.

```
i= input('entrer i=');
if    ( i ==10)
j=2*pi;
disp (j);
end
```

Exemple N°2 : Tester le script suivant, pour $x=3$ *et* $x=18$.

```
x= input('entrer x=');
if    mod (x,2)==0
 disp ('x est pair')
else
 disp ('x est impair')
end
```

Exemple N°3 : Tester le script suivant, pour différentes valeurs de i

```
i= input('entrer i=');
if    i >0
 disp ('i est positif')
elseif i <0
 disp ('i est négatif')
else
 disp ('i est null')
end
```

5.2.1. Travail demandé :

Exercice N°1 :

Ecrire un programme matlab pour trouver le plus grand de trois nombre (a,b,c).

5.3.Structure conditionnelle : switch

La structure switch est utilisée lorsque l'on a plusieurs options possibles basées sur la valeur d'une variable.

Syntaxe générale :

switch variable

case valeur1

% Code si variable == valeur1

case valeur2

% Code si variable == valeur2

otherwise

% Code si aucune des valeurs précédentes ne correspond

end

Exemple : Tester le script suivant, pour différentes valeurs.

```
saison_num = 4;  % Changez ce nombre pour tester d'autres valeurs

switch saison_num
    case 1
        disp('Printemps');
    case 2
        disp('Été');
    case 3
        disp('Automne');
    case 4
        disp('Hiver');
    otherwise
        disp('Numéro invalide');
end
```

Dans cette structure switch, chaque case correspond à une saison. Si le nombre ne correspond à aucune des valeurs, on affiche "Numéro invalide".

TP N°06 : Fichiers des fonctions

6.1.Fichier fonction

Un fichier fonction comporte une seule fonction principale (même si on peut inclure des fonctions locales). Voici comment le structurer :

- Le nom du fichier .m doit correspondre au nom de la fonction.
- La fonction commence par le mot-clé **function**, suivi de la syntaxe de définition.

Syntaxe générale :

function [outputs] = **nom_fonction** (inputs)

% Description de la fonction (facultatif)

% Code de la fonction

end

Exemple :

Supposons que nous voulions créer une fonction appelée **sommeCarres** pour calculer la somme des carrés des éléments d'un vecteur. La fonction prendra en entrée un vecteur et renverra la somme des carrés de ses éléments.

1. Créez un fichier nommé **sommeCarres.m** qui contient le code suivant :

```
function result = sommeCarres(v)

    result = sum(v.^2);
end
```

2. Dans un script ou dans la ligne de commande MATLAB, vous pouvez appeler cette fonction :

```
clear all; close all ; clc;
v =[1:5];
result = sommeCarres(v);
disp('La somme des carrés est : '),disp(result);
```

3. résultats

```
Command Window
La somme des carrés est :
    55
```

6.2.Fonction avec Plusieurs Entrées et Sorties

Exemple :

Créons un fichier fonction qui prend un vecteur et renvoie à la fois la somme et la moyenne des éléments de vecteur v.

1. Créez un fichier fonction nommé **sommeEtMoyenne.m** qui contient le code suivant :

```
function [somme, moyenne] = sommeEtMoyenne(v)

    % Calcul de la somme
    somme = sum(v);

    % Calcul de la moyenne
    moyenne = mean(v);
end
```

2. Dans un script MATLAB, vous pouvez appeler cette fonction :

```
clear all; close all ; clc;
v =[0:25];
% SOMMEETMOYENNE Calcule la somme et la moyenne des éléments d'un vecteur
[somme, moyenne] = sommeEtMoyenne(v);
disp(['La somme est : ', num2str(somme)]);
disp(['La moyenne est : ', num2str(moyenne)]);
```

3. résultats

```
Command Window
  La somme est : 325
  La moyenne est : 12.5
```

TP N°07 : Graphisme

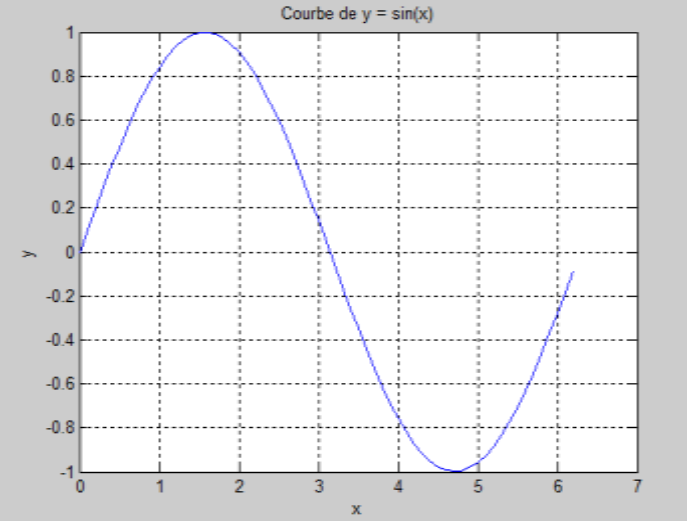
7.1.Graphisme 2D

7.1.1. Tracé d'une courbe simple

La fonction **plot** permet de de tracer des graphiques 2D ; avec **plot(x, y)** on trace y en fonction de x, x et y sont des vecteurs de données de même dimension

Exemple : tracer une fonction sinus

```
clear all; close all ; clc;
% Définition de l'intervalle de x
x = 0:0.1:2*pi;
% Calculer les valeurs de y
y = sin(x);
% Tracer la courbe
plot(x, y);
% Documentation du graphisme
title('Courbe de y = sin(x)');
xlabel('x');
ylabel('y');
grid ;
```

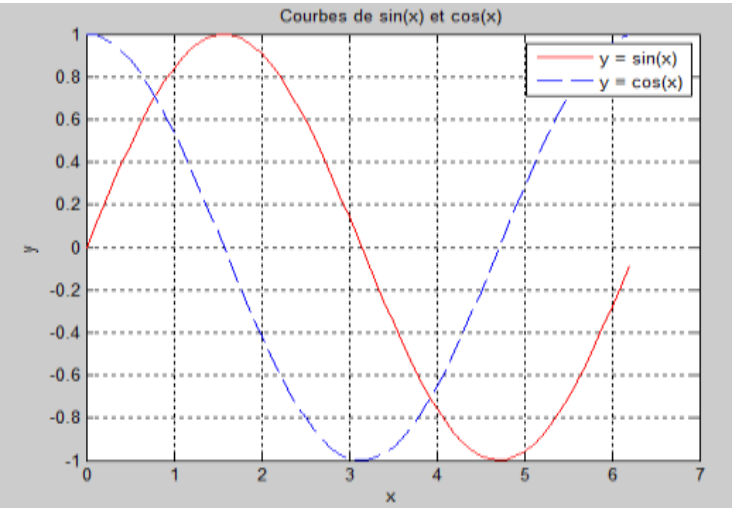


7.1.2. Tracé de Plusieurs Courbes sur un seul graphique

Pour afficher plusieurs courbes sur un même graphique, on peut utiliser plot plusieurs fois on activant la fonction **hold on** , ou bien combiner les données.

Exemple : tracer y = sin(x) et y = cos(x)

```
clear all; close all ; clc;
% Définition de l'intervalle de x
x = 0:0.1:2*pi;
% Calculer les valeurs de y
y1 = sin(x);
y2 = cos(x);
% Courbes avec des styles différents
plot(x, y1, '-r', x, y2, '--b');
legend('y = sin(x)', 'y = cos(x)');
title('Courbes de sin(x) et cos(x)');
xlabel('x');
ylabel('y');
grid ;
```



La fonction **hold on** est utilisée pour superposer plusieurs tracés sur un même graphique sans effacer les tracés précédents.

7.1.3. Modification de l'apparence d'une courbe :

Vous pouvez modifier la couleur de la courbe, la forme des points de coordonnées et le type de trais, pour faire ça vous ajoutez un nouvel argument de type chaine de caractère à la fonction plot comme ceci : plot (x,y,'marqeur')

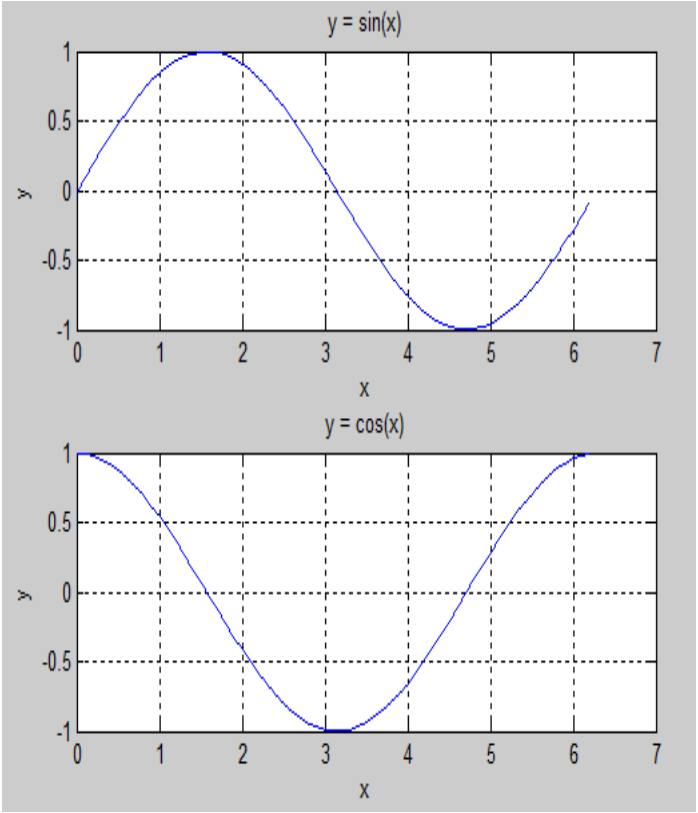
Le contenu du marqueur est une combinaison d'un ensemble de caractères spéciaux rassemblés dans le tableau suivant :

Couleur	code	Style	code	Symbole	code
Blanc	**w**	Trait plein	**-**	Point	**.**
Noir	**k**	Pointillé court	**:**	Cercles	**o**
Blue	**b**	Pointillé long	**_**	Croix	**x**
Rouge	**r**	Pointillé mixte	**-.**	Plus	**+**
Cyan	**c**	Pas de ligne	**none**	Etoile	*****
Vert	**g**			Carré	**s**
Magenta	**m**			Losange	**d**
Jaune	**y**			Triangle (bas)	**v**
				Triangle (gauche)	**<**
				Triangle (droite)	**>**
				Pentagone	**p**
				Hexagone	**h**
				aucun	**none**

7.1.4. Affichage de plusieurs graphiques dans une figure : subplot

Exemple

```
clear all; close all ; clc;
x = 0:0.1:2*pi;
y1 = sin(x);
y2 = cos(x);
% 2 lignes, 1 colonne, premier sous-graphe
subplot(2, 1, 1);
plot(x, y1);
title('y = sin(x)');
xlabel('x');
ylabel('y');
grid;
% 2 lignes, 1 colonne, deuxième sous-graphe
subplot(2, 1, 2);
plot(x, y2);
title('y = cos(x)');
xlabel('x');
ylabel('y');
grid;
```



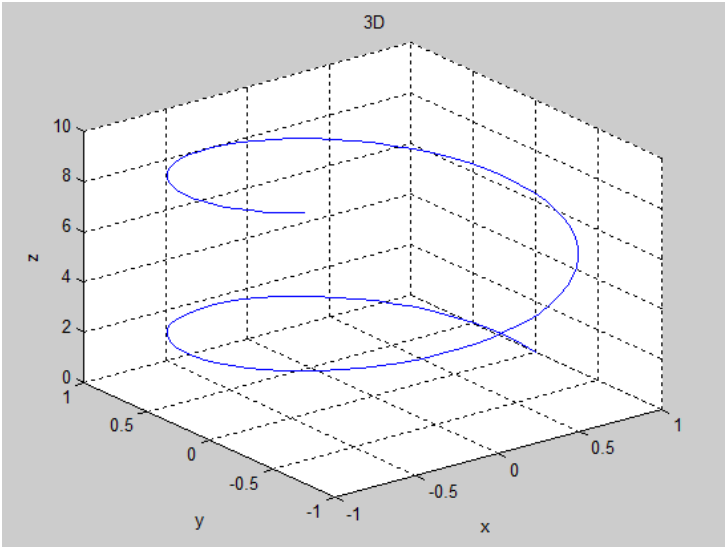
7.2.Tracé en 3D

7.2.1. La fonction **plot3**

La fonction **plot3** permet de tracer des courbes en 3 dimensions.

Exemple :

```
clear all; close all ; clc;
t = 0:0.1:10;
x = cos(t);
y = sin(t);
z = t;
plot3(x, y, z);
title(' 3D');
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
```



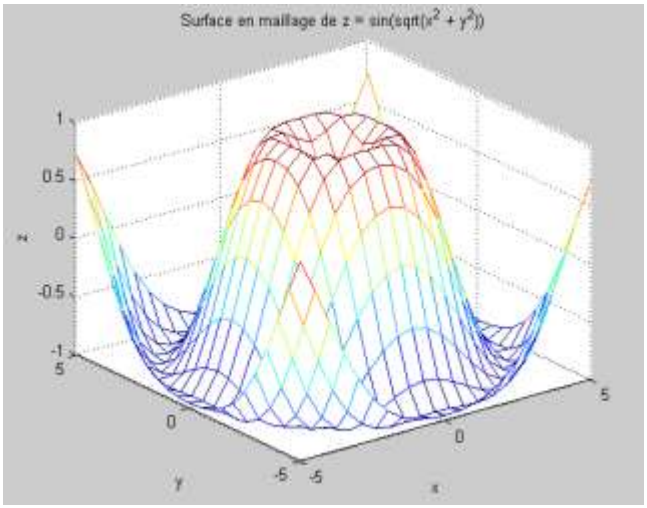
7.2.2. Surface 3D : mesh et surf

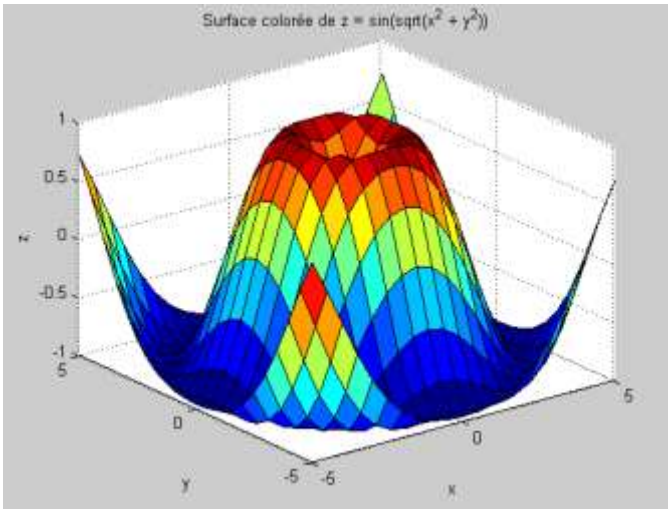
Les fonctions **mesh** et `surf` sont utiles pour tracer des surfaces en 3D.

Exemple : Surface de z = sin (sqrt(x^2 + y^2))

```
clear all; close all ; clc;
[X, Y] = meshgrid(-5:0.5:5, -5:0.5:5);
Z = sin(sqrt(X.^2 + Y.^2));
figure;
% Tracé en maillage
mesh(X, Y, Z);
title('Surface en maillage de z = sin(sqrt(x^2 + y^2))');
xlabel('x');
ylabel('y');
zlabel('z');

figure;
% Tracé en surface colorée
surf(X, Y, Z);
title('Surface colorée de z = sin(sqrt(x^2 + y^2))');
xlabel('x');
ylabel('y');
zlabel('z');
```





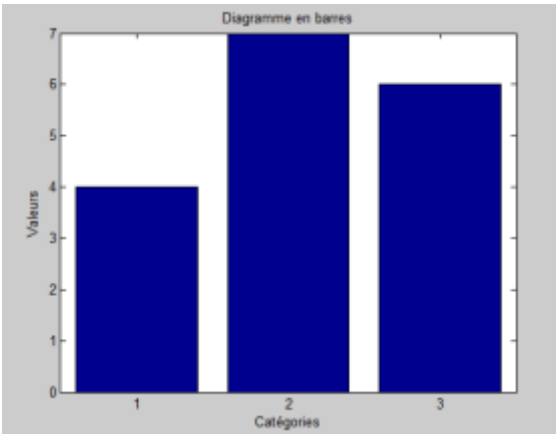
7.2.3. Histogrammes et diagrammes en barres

Histogrammes : utilisez la fonction ***hist*** pour afficher un histogramme de données

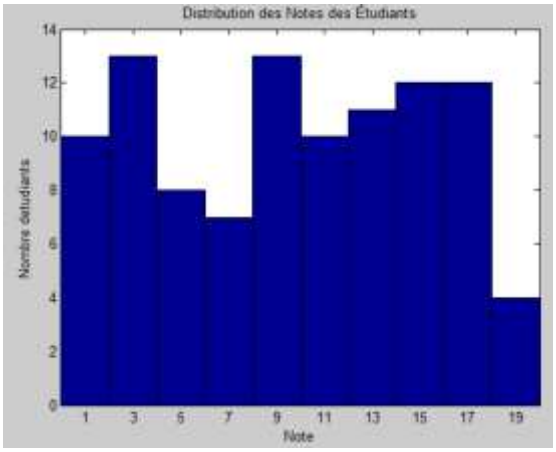
Diagrammes : utilisez la fonction ***bar*** pour afficher un graphique en barres

Exemple :

```
clear all; close all ; clc;
% Graphique en barres
bar([1, 2, 3], [4, 7, 6]);
title('Diagramme en barres');
xlabel('Catégories');
ylabel('Valeurs');
```




```
clear all; close all ; clc;
% un vecteur 1x100 de nombres entiers entre 0 et 20
notes = randi([0, 20], 1, 100);
% l'histogramme de la distribution des notes
hist(notes, 10);  % 10 bacs
title('Distribution des Notes des Étudiants');
xlabel('Note');
ylabel('Nombre detudiants');
```



La fonction ***randi*** génère des nombres entiers aléatoires.

7.3.Travail demandé

Exercice N°1

On donne le polynôme

- **$P(x)=3x^3 - 2x^2 - 4x + 2$**

Créez un fichier nommé **derive_integral.m** pour calculer la dérivée et l'intégrale de P
Créez un fichier nommé **polynome.m** pour tracer : le polynôme P, la dérivée de P (DP), l'intégrale de P (IP) En fonction de x, avec x = [-10 : 1 : 10]

- dans la même figure. (utilisez la fonction **hold on**)
- dans la même figure avec des sous-graphes (utilisez la fonction **subplot**)

Exercice N°2

- Créez deux fichiers nommés ***E.m*** et ***L.m*** contenant les fonctions suivantes :
  E(x)= exp(x)
  L(x)=log(x)
- Créez un fichier nommé ***log_exp.m*** qui trace les fonctions E(x) et L(x) dans la même figure avec x varie entre 0.1 et 5 avec un pas de 0.1.
- Donner un titre à la figure ***exp(x) _log(x)***, avec des grilles, et écrire le libeller des axes.

Exercice N°3

- Créez deux fichiers nommés ***f.m*** et ***g.m*** contenant les fonctions suivantes :
  f(x)= *sin* (x^2+1)
  g(x)= *cos* (x^2+1)

Créez un fichier nommé ***prog.m*** qui trace les fonctions f(x)et g(x) dans deux figures avec x varie entre π et 10π avec un pas de $\pi/10$.

- Donner un titre à la figure avec des grilles, et écrire le libeller des axes.

References bibliographies

Gilat, A. *(2013). MATLAB: An Introduction with Applications (5e éd.). Wiley. ISBN: 9781118629864.*

Hunt, B., Lipsman, R., Rosenberg, J., Coombes, K., Osborn, J., & Stuck, G. (2001). *A Guide to MATLAB for Beginners and Experienced Users*. Cambridge University Press.

Kattan, P. I. (2008). *MATLAB for Beginners: A Gentle Approach*. Petra Books, Smashwords Edition. ISBN: 978-1438203096.

Attaway, S. (2016). *MATLAB: A Practical Introduction to Programming and Problem Solving*. Butterworth-Heinemann. ISBN: 978-0128045251.

Valentine, D. T., & Hahn, B. D. (2022). *Essential MATLAB for Engineers and Scientists*. Academic Press. ISBN: 9780323995481.

Moore, H. (2018). *MATLAB® for Engineers* (5e éd.). Pearson Education. ISBN: 978-0-13-458964-0.

Chapman, S. J. (2020). *MATLAB Programming for Engineers* (6e éd.). Cengage Learning. ISBN: 9780357676054.