République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université 8 Mai 1945 Guelma
Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Polycopié

Outils de programmation pour les mathématiques (Avec MATLAB)

Cours et Travaux Pratiques

Première année Maths et Informatique (MI)

Support de cours réalisé par :
Dr. ZEDADRA Ouarda

zedadra.ouarda@univ-guelma.dz
zedadra_nawel1@yahoo.fr

Année universitaire : 2021/2022

SYLLABUS

Unité d'enseignement : Méthodologique ;
Matière : Outils de programmation pour les mathématiques.
Domaine/Filière : LMD- 1MI
Semestre : 2, **Année universitaire :** 2021/2022
Crédit : 2, **Coefficient :** 1
Volume Horaire Hebdomadaire : 3 H

- Cours Magistral (1 H 30)
- Travaux Pratique - TP (1 H 30)

Langue d'enseignement : Français
Enseignant responsable de la matière : Dr. Ouarda ZEDADRA, **Grade :** M.C.A
Bureau : E8.2 (département d'Informatique) ou Laboratoire d'Informatique **LabSTIC**
Courriel : zedadra.ouarda@univ-guelma.dz ou
zedadra_nawel1@yahoo.fr
Heure de consultation :

Objectifs :

Ce cours s'adresse aux étudiants en formation de Licence (1MI). Il fait partie des modules de l'unité d'enseignement méthodologique. Ce cours sert à initier l'étudiant aux concepts de base du langage de programmation MATLAB pour lui permettre d'acquérir rapidement une autonomie d'utilisation du logiciel.
Il permet de : (1) Acquérir les connaissances qui permettront d'utiliser de manière efficace le logiciel Matlab, et (2) Se faire une idée précise de l'apport potentiel de ce logiciel pour la résolution des problèmes issues des mathématiques. Le cours se trouve partagé en trois chapitres :

Chapitre 1 : introduction à MATLAB

- Interface de MATLAB et outils de base ;
- Vecteurs ;
- Matrices ;
- fonctions spéciales.

Chapitre 2 : programmation avec MATLAB

- Généralités ;
- entrées-sorties ;
- structures de contrôle ;
- les fonctions et les scripts MATLAB ;
- les graphiques et la visualisation de données sous MATLAB.

Chapitre 3 : applications des méthodes numériques avec MATLAB

- Les polynômes sous MATLAB ;
- Résolution des systèmes particuliers : systèmes à matrice diagonale, systèmes à matrice triangulaire supérieure, systèmes à matrice triangulaire inferieure ;

- Résolution des équations non linéaires : méthode de la dichotomie ;
- Résolution des systèmes d'équations linéaires : méthode Directe de Gauss.

Modalité d'évaluation des apprentissages :

Le calcul de la moyenne sera une combinaison de la note de l'examen final et de la note de TP. La pondération des notes est comme suit :

Contrôle	**Pondération (%)**
Examen final TP	60% 40%
Total	100%

Références bibliographiques :

La suite des ouvrages disponibles dans la bibliothèque centrale ou sur Internet:

Code	**Titre**	**Auteur**	**Maison d'édition**
L/004.801	Introduction à MATLAB	J. T. LAPRESTE	Ellipses
L/510.862	Débuter en algorithmique avec MATLAB et SCILAB	J. P. GRENIER	Ellipses
L/510.923	Outils mathématiques pour l'ingénieur et le chercheur avec MATLAB	J. T. LAPRESTE C. VIAL	Ellipses
L/510.1128	Algorithmes numériques : fondements théoriques et analyse pratique : cours, exercices et applications avec Matlab (Méthodes numériques)	Marie-Hélène Meurisse	ellipses
L/510.680	Introduction à L'analyse Numérique : Applications sous Matlab Cours, et exercices corrigés et travaux pratiques	Jérome Bastien Jean-Noel Martin	Dunod
L/510.908	Introduction au calcul scientifique par la pratique: 12 projets résolus avec Matlab	Ionut Danaila Pascal Joly Sidi Mahmoud Kaber Marie Postel	Dunod
A/510.219	الدليل العربي لتشغيل برنامج الهندسي العلميMatlab	مثنى عبد المجيد جميل	
Documentation en ligne de MATLAB : https://www.mathworks.com/help/matlab/index.html			

Avant-Propos

Ce cours s'adresse aux étudiants de première année maths et informatique (MI), ainsi qu'à d'autres étudiants en première année désireux de résoudre des problèmes concrets au moyen de méthodes numériques à la fois simples, précises et robustes. Leurs préoccupations sont donc de parvenir à leurs fins, tout en évitant l'apprentissage fastidieux de langages de programmation qui ne disposent pas par défauts des outils mathématiques adéquats ce qui nécessite une reprogrammation inefficace.

Depuis son apparition en 1970 MATLAB a été l'outil le plus recommandé pour répondre aux préoccupations des débutants et supporter les calculs réclamés par leurs approches. MATLAB est un acronyme de 'MATrix LABoratory' conçu par la société *MathWorks*. Il a été conçu initialement pour être un environnement de calcul scientifique et de visualisation de données, c'est aujourd'hui un langage de programmation complet qui inclut un environnement graphique simple et pratique. L'intérêt de MATLAB tient, d'une part, à sa simplicité d'utilisation : pas de compilation, pas besoin de déclaration des variables utilisées, et d'autre part, à sa richesse fonctionnelle : arithmétique matricielle et nombreuses fonctions de haut niveau qui permettent une couverture large des problèmes dans divers domaines (analyse numérique, statistique, traitement de signal, traitement d'images, ...). Il possède aussi toutes les fonctionnalités des approches récentes de la programmation comme la programmation objet et la programmation évènementielle du graphisme.

Ce cours se trouve divisé en trois chapitres. Le premier chapitre sert comme introduction à l'environnement MATLAB. Il s'agit ici d'acquérir les notions indispensables aux étudiants pour utiliser le logiciel. Nous présentons donc l'interface graphique de MATLAB, les outils de base, les vecteurs, les matrices, les entrées-sorties et quelques fonctions spéciales existantes dans MATLAB.

Dans le deuxième chapitre, nous abordons la programmation avec MATLAB. Dans ce chapitre, nous introduisons les structures de données, les structures de contrôles, les expressions, les fonctions, les graphiques et la visualisation des données et même d'autres notions dont nous jugeons utiles pour avancer dans la programmation avec MATLAB.

Le troisième chapitre se distingue des deux précédents du fait qu'il contient la résolution concrète de quelques problèmes connus de mathématiques (Analyse numérique,…) à partir des méthodes disponibles sous MATLAB.

Objectifs

La nécessité des mathématiques appliquées impose à l'étudiant la maitrise d'un outil de programmation pour les mathématiques. Le logiciel MATLAB a été l'outil le plus recommandé pour répondre aux préoccupations des débutants et supporter les calculs réclamés par leurs approches.

A l'issue de ce module, l'étudiant doit être capable de :

1. Apprendre à utiliser MATLAB et à faire de l'arithmétique de base ;
2. Programmer en environnement MATLAB ;
3. Travailler avec des variables ;
4. Créer un tracé 2D et 3D ;
5. Créer des vecteurs, des matrices et effectuer des opérations de base et avancées ;
6. Résoudre des équations et un système d'équations ;

MATLAB pour sculpter votre carrière :

MATLAB est utilisé dans de nombreuses entreprises et peut essentiellement être utilisé dans n'importe quel travail où l'analyse des données est une compétence souhaitée.

Au fur et à mesure que le monde deviendra de plus en plus axé sur les données et les statistiques. Si vous prévoyez de faire un certain niveau de recherche ou d'analyse de données dans votre carrière, l'apprentissage de MATLAB vous sera très précieux.

Vous serez en mesure de prototyper de nouvelles idées rapidement et facilement. Si vous êtes plus concentré sur le fait d'être un pur ingénieur logiciel, MATLAB n'est peut-être pas le meilleur outil pour vous. Puisqu'il est vraiment plus orienté vers l'informatique scientifique, il peut laisser beaucoup à désirer si vous souhaitez développer des applications mobiles ou web.

Prérequis

Les étudiants désirant suivre cette formation, doivent avoir des prérequis sur :

1. L'informatique de base (installer des logiciels et éditer des fichiers texte) ;
2. Les notions de base de l'algorithmiques : les variables, les structures de données, les structures de contrôles, les boucles…;

Sommaire

Liste des Figures

CHAPITRE 1 : INTRODUCTION A MATLAB

CHAPITRE 2 : PROGRAMMATION AVEC MATLAB

CHAPITRE 3 : APPLICATIONS DES METHODES NUMERIQUES AVEC MATLAB

Liste des Tableaux

CHAPITRE 1 : INTRODUCTION A MATLAB

1. Introduction

Un langage de calcul scientifique est un langage de programmation destiné à être utilisé par les chercheurs pour réaliser des calculs complexes. Il est riche en terme fonctions et de bibliothèques facilitant la tâche d'un programmeur dans un domaine de recherche, qui n'a pas nécessairement des compétences de programmation avancées. Les langages de calculs scientifiques sont classés en langages compilés et langages interprétés.

Dans un langage de programmation interprété un programme supplémentaire (l'interpréteur) est nécessaire, celui-ci va générer l'exécutable des instructions et les exécute au fur et à mesure de l'exécution du programme, donc on n'a pas dans ce cas un exécutable, et à chaque fois on a besoin du code source initial pour exécuter le programme. Des exemples de langages interprétés incluent : MATLAB, le langage R, Scilab… etc. Tandis que, un langage compilé va traduire (compiler) le programme en exécutable qui peut être utilisé ultérieurement sans avoir besoin du code source initial. FORTRAN, C et C++ sont des exemples de langages compilés.

Ce premier chapitre sert comme introduction au langage MATLAB. Dans lequel, nous introduisons les notions indispensables aux étudiants pour utiliser ce logiciel. Nous présentons donc l'interface graphique de MATLAB, les outils de base, les vecteurs, les matrices, les entrées/ sorties et les fonctions spéciales existantes dans MATLAB.

2. INTRODUCTION A L'ENVIRONNEMENT MATLAB

2.1. Introduction

MATLAB (MATrix LABoratory) est un environnement de programmation interactif pour le calcul scientifique, la programmation et la visualisation des données. Il est très utilisé dans les domaines d'ingénierie et de recherche scientifique, ainsi qu'aux établissements d'enseignement supérieur. Sa popularité est due principalement à sa forte et simple interaction avec l'utilisateur mais aussi aux points suivants :

- *Sa richesse fonctionnelle :* avec MATLAB, il est possible de réaliser des manipulations mathématiques complexes en écrivant peu d'instructions. Il peut évaluer des expressions, dessiner des graphiques et exécuter des programmes classiques. Et surtout, il permet l'utilisation directe de plusieurs milliers de fonctions prédéfinies.
- *La possibilité d'utiliser les boites à outils (toolboxes):* ce qui encourage son utilisation dans plusieurs disciplines (simulation, traitement de signal, imagerie, intelligence artificielle, ...).
- *La simplicité de son langage de programmation :* un programme écrit en MATLAB est plus facile à écrire et à lire comparé au même programme écrit en C ou en PASCAL.
- *Sa manière de tout gérer comme étant des matrices*, ce qui libère l'utilisateur de s'occuper de typage de données et ainsi de lui éviter les problèmes de transtypage.

On trouve dans la littérature des concurrents de MATLAB qui font aussi du calcul scientifique payants comme MAPLE et MATHEMATICA ou libres (freeware) comme OCTAVE et SCILAB.
Comme Windows est l'environnement le plus souvent utilisé par les débutants, nous avons décrit la version installée sous Windows. Il peut y avoir quelques modifications de détail pour les autres systèmes d'exploitation (Unix/Linux). On suppose dans ce qui suit que le logiciel est déjà installé (version 7.0).

2.2. Interface Graphique de MATLAB

L'interface de MATLAB peut changer légèrement selon la version utilisée, mais les points centraux restent identiques. La **Figure 1.1** montre la fenêtre principale par défaut de MATLAB (version 7.0.0) lors de son lancement. On peut également personnaliser l'affichage des fenêtres depuis le menu : Desktop→ Desktop Layout.

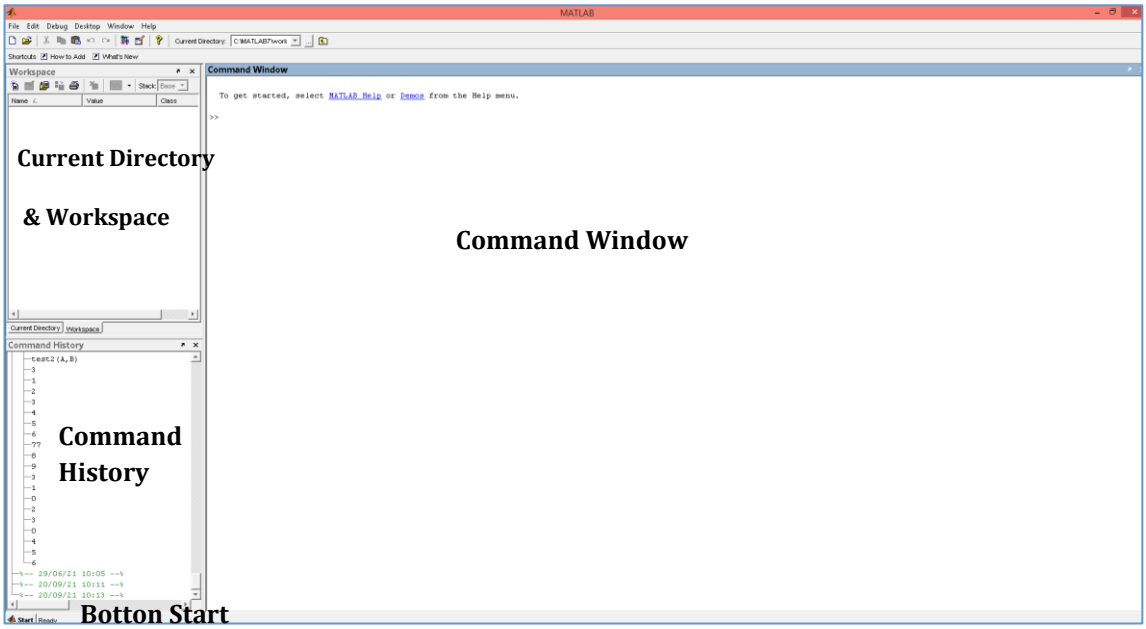


Figure 1.1 : fenêtre par défaut de MATLAB

Selon le mode d'affichage par défaut (**Figure 1.1**), on dispose de quatre fenêtres :

1) Fenêtre de commandes (command window)

C'est la fenêtre dans laquelle on tape les commandes en exécution immédiate et dans laquelle sont affichées par défaut les réponses. Une ligne commence toujours par le signe >> (prompt).

Quelques conseils pour une utilisation efficace de la fenêtre de commandes :

- La ligne de commande commence par le signe >> qu'il ne faut pas taper. Il n'apparait pas lors de l'exécution d'une instruction.
- Une instruction tapée dans la ligne de commande est immédiatement exécutée après validation par la touche *'entrée'* ;
- Le symbole point-virgule ';' est un inhibiteur d'affichage. C'est-à-dire, si une ligne de commande produisant un résultat se termine par ; ce résultat n'est pas affiché à l'écran. Tandis que, si une ligne de commande produisant un résultat ne se termine pas par ; ce résultat est affiché à l'écran.
- Toute ligne de commande est modifiable tant que on n'a pas validé par un *'entrée'.*

- On peut rappeler toute ligne dans la ligne de *la fenêtre de commandes* en utilisant les flèches du clavier ↑ et ↓. De même, les flèches du clavier → et ← permettent de se déplacer dans une ligne.
- Si une instruction produit un résultat et ne comporte pas d'indication d'affectation, le résultat est affecté par défaut à la variable *ans* (answer).

2) Espace de travail (workspace)

Contient le nom, la taille et le type des variables reconnues par *la fenêtre de commandes.* On peut charger et sauvegarder des données au moyen des icônes 'open' et 'save' du bandeau du *Workspace*. Un clic-droit sur les variables offre de nombreuses options telles que : Open (ouvrir), Duplicate (copie), Save as (enregistrer sous), Delete (supprimer)…etc.

3) Historique des commandes (command history)

Contient toute les lignes de commande validées dans la *fenêtre de commandes* depuis le précédent nettoyage*.* C'est cet historique qui nous permet de naviguer dans les commandes déjà validées. Par ailleurs, on peut parfois enregistrer dans un fichier indépendant la liste des commandes utilisées. Pour ce faire on utilise la commande : *'diary filename'*. Elle permet de créer un fichier *filename* qui enregistrera toutes les commandes entrées jusqu'à ce que l'on utilise *'diary off'*.

4) Répertoire courant (current directory)

Indique le répertoire actif, c'est-à-dire celui dans lequel toutes les fonctions de lectures/écritures ont lieu par défaut. Cette fenêtre permet de gérer les fichiers et de changer de répertoire actif.

D'autres fenêtres très intéressantes de MATLAB :

5) Editeur (editor)

La plupart de votre travail sous MATLAB va consister à créer ou modifier des fichiers .m qui est le suffixe standard pour les procédures MATLAB. On utilise souvent la *'la fenêtre de commandes*' pour des commandes simples, mais s'il s'agit de plusieurs dizaines de ligne de code on a recours à l'utilisation de l'*éditeur.* Cet éditeur nous permet de créer des fichiers *script* ou des *fonctions* selon notre choix, dont les deux sont exploitables depuis la *fenêtre de commandes.*

6) Aide (help)

C'est important d'utiliser le *help* (aide), lorsque l'on programme avec un langage de programmation de haut-niveau (comme MATLAB), où le nombre de fonctions est très important et la syntaxe est parfois complexe. La commande help, vous permet d'afficher l'aide de MATLAB. Alors que la commande '*help FunctionName'* permet d'afficher une explication et la syntaxe de la fonction *FunctionName.* Il est essentiel que vous vous familiariser avec les outils de l'aide de MATLAB pour réussir dans ce cours.

7) Commandes d'interaction

Commande	Objectif
Who	Affiche le nom des variables dans le *Workspace*.
Whos	Affiche des informations sur les variables dans le *Workspace*.
Clear x y	Supprime les variables x et y du *Workspace*.
Clear, clear all	Supprime toutes les variables dans le *Workspace.*
Clc	Efface l'écran des commandes dans la *Command Window*.
Exit, quit	Quitter MATLAB.
format	

	Définit le format de sortie pour les valeurs numériques (voir **Tableau 1.4** pour plus de détails)

Tableau 1.1. quelques commandes d'interaction dans MATLAB.

2.3. Outils de Base

Le principe de MATLAB est de considérer la plupart des objets comme des matrices. Il faut donc considérer les opérations arithmétiques +, -, *, / comme des opérations matricielles.

2.3.1. Types de variables

Il existe cinq types de variables sous MATLAB : les entiers, les réels, les complexes, les chaines de caractères et le type logique. La déclaration sous MATLAB est très simple, on utilise la ligne suivante pour déclarer une variable: '*nom variable = valeur/ expression'.* Définissons une variable de chaque type comme suit :

```
>> a=5.8; b=9+i; c='hello';
>> tr1=true(1==1);tr2=logical(1);
>> x=int8(2);
>>
```

a représente un réel, **b** un complexe, **c** une chaine de caractères, **tr1** et **tr2** des logiques définies de deux manière différentes et **x** est un entier codé sur 8 bits. Toutes les variables déclarées sont stockées dans le *espace de travail* est peuvent êtres affichées avec leurs types en utilisant la fonction : '*whos*'. Cette fonction donne une description détaillée (nom de variable, dimension, taille en octets (byte) et type). La commande '*who*' donne seulement le nom des variables dans le *espace de travail*.

```
>> whos
  Name      Size                    Bytes  Class

  a         1x1                         8  double array
  b         1x1                        16  double array (complex)
  c         1x5                        10  char array
  tr1       1x1                         1  logical array
  tr2       1x1                         1  logical array
  x         1x1                         1  int8 array
```

Il est impossible de déclarer le type d'une variable lorsque l'on crée une variable dans MATLAB. Il pourrait être utile donc de vérifier le type des variables selon les fonctions : *ischar, islogical, isreal*. MATLAB distingue les majuscules et les minuscules.

MATLAB propose une suite de constantes prédéfinies (à être utilisés sans les déclarés) (voir **Tableau 1.2**)

La constante	La valeur
pi	π=3.1416
i	= $\sqrt{-1}$
j	= $\sqrt{-1}$
inf	∞
NaN	Not a Number (pas un numéro)
eps	= 2.2204e-016 ($2x10^{-16}$)

Tableau 1.2. exemples de constantes prédéfinies dans MATLAB.

2.3.2. Les Nombres sous MATLAB

MATLAB utilise une notation décimale conventionnelle, avec un point décimal facultatif '.' et le signe '+' ou '–' pour les nombres signés. La notation scientifique utilise la lettre 'e' pour spécifier le facteur d'échelle en puissance de 10. Les nombres complexes utilisent les caractères 'i' et 'j' (indifféremment) pour designer la partie imaginaire (voir **Tableau 1.3**).

Le type	Exemples
Entier	5 - 83
Réel en notation décimale	0.0205 3.1415926
Réel en notation scientifique	1.60210e-20 6.02252e23 (1.60210×10^{-20} et 6.02252×10^{23})
Complexe	5+3i - 3.14159j

Tableau 1.3. exemples de Types de variables

MATLAB utilise toujours les nombres réels (double précision) pour faire les calculs, ce qui permet d'obtenir une précision de calcul allant jusqu'aux 16 chiffres significatifs. Le résultat d'une opération de calcul est par défaut affichée avec quatre chiffres après la virgule. Pour afficher davantage de chiffres utiliser les commandes dans le **Tableau 1.4**.

Commande	Objectif
format short	affiche les nombres avec 4 chiffres après la virgule.
format long	affiche les nombres avec 14 chiffres après la virgule.
format bank	affiche les nombres avec 2 chiffres après la virgule.
format rat	affiche les nombres sous forme d'une fraction (a/b).

Tableau 1.4. commandes pour personnaliser le nombre de chiffres après la virgule.

La fonction *vpa* peut être utilisée pour présenter le nombre de décimaux désirés après la virgule. Exemple d'utilisation : *vpa(sqrt(2),50)*. Cela affiche 50 chiffres après la virgule.

2.3.3. Les nombres aléatoires

Il est souvent utile de tester d'abord vos programmes en initialisant les variables de données à des nombres aléatoires. Les nombres aléatoires sont également utiles dans les simulations. Il existe plusieurs fonctions intégrées dans MATLAB qui génèrent des nombres aléatoires, dont certaines seront illustrées dans cette section. Ces valeurs sont pseudo-aléatoires et ne sont pas vraiment aléatoires car il existe un processus qui détermine à chaque fois la valeur suivante.

La fonction **rand** peut être utilisée pour générer des nombres réels aléatoires uniformément distribués ; l'appeler génère un nombre réel aléatoire dans l'intervalle ouvert (0,1), ce qui signifie que les extrémités de la plage ne sont pas incluses. Il n'y a pas d'arguments passés à la fonction rand dans sa forme la plus simple. Voici deux exemples d'appels de la fonction **rand** :

```
>> rand
ans =
   0.9501

>> rand
ans =
   0.2311
```

Comme rand renvoie un nombre réel dans l'intervalle ouvert (0, 1), multiplier le résultat par un entier N renverrait un nombre réel aléatoire dans l'intervalle ouvert (0, N). Par exemple, multiplier par 10 renvoie un nombre réel dans l'intervalle ouvert (0, 10), donc l'expression.

```
>> rand*10
ans =
   6.0684
```

Pour générer un nombre réel aléatoire dans la plage [valeur_initiale, valeur_finale], créez d'abord les variables : valeur_initiale, valeur_finale. Ensuite, utilisez l'expression

`rand*( valeur_finale - valeur_initiale)+ valeur_initiale.`

```
>> val_initiale=3;
>> val_finale=5;
>> rand*(val_initiale-val_finale)+val_finale
ans =
   3.2174
```

2.3.4. Fonctions et opérateurs sur les nombres

MATLAB assure les quatre opérations arithmétiques connus (+, -, *, /), les opérateurs relationnels et les opérateurs logiques (voir **Tableau 1.6**). Il peut également utiliser les fonctions trigonométriques, puissance, logarithmiques, les fonctions d'arrondis, les fonctions d'arithmétiques, et des fonctions des nombres complexes…etc. Une liste (non exhaustive) des fonctions incorporées dans MATLAB est donnée dans le **Tableau 1.5.**

Fonction	Objectif
Fonctions trigonométriques, puissance, logarithmiques	
exp(x)	exponentielle de x
log(x)	logarithme népérien de x
log10(x)	logarithme en base 10 de x
sqrt(x)	racine carré de x
abs(x)	valeur absolue de x
sign(x)	1 si x>0 et -1 si x<=0
sin(x)	sinus de x
cos(x)	cosinus de x
tan(x)	tangente de x
asin(x)	sinus inverse de x (arcsin de x)
sinh(x)	sinus hyperbolique de x
asinh(x)	sinus hyperbolique inverse de x
Fonctions d'arrondis	
round(x)	entier le plus proche a x
floor(x)	arrondi par défaut de x
ceil(x)	arrondi par excès de x
Fonctions d'arithmétiques	
rem(m,n)	reste de la division entière de m par n
lcm(m,n)	plus petit commun multiple de m et n
gcd(m,n)	plus grand commun diviseur de m et n
factor(n)	décomposition en facteurs premiers de n
Fonctions sur les nombres complexes	
conj(z)	conjugué de z
abs(z)	module de z
angle(z)	argument de z
real(z)	partie réelle de z
imag(z)	partie imaginaire de z

Tableau 1.5. fonctions prédéfinies sous MATLAB.

Opérateur	Objectif
Opérateurs arithmétiques	
+	**Addition**
-	**Soustraction**
*	**Multiplication**
/	**Division**
^	**Puissance**
Opérateurs relationnels	
==	Egalité
~=	Inégalité
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal à
<=	Inférieur ou égal à
Opérateurs logiques	
&	**Et** logique
|	**Ou** logique
~	**Non** (négation) logique

Tableau 1.6 : opérateurs arithmétiques, relationnels et logiques sous MATLAB

Le **Tableau 1.7** résume le fonctionnement des opérateurs logiques :

x	y	x & y	x|y	~x
1	**1**	1	1	0
1	**0**	0	1	0
0	**1**	0	1	1
0	**0**	1	0	1

Tableau 1.7: fonctionnement des opérateurs logiques

L'évaluation d'une expression s'exécute de gauche à droite en considérant la priorité des opérations comme suit : **Les parenthèses (), la puissance ^ et le transposé ', la multiplication * et la division /, l'addition + et la soustraction –**.

2.3.5. Exercices d'application :

Exercice 1 :

Donner le résultat des commandes suivantes :

```
>> x=15;
>> y=30;
>> x<y              %x=15 est inférieur à y=30 affiche 1 vrai
ans =
   1
>> x<=y             %x=15 est inférieur à y=30 affiche 1 vrai
ans =
   1
>> x==y             %x est différent de y donc affiche 0 faux
ans =
   0
>> (0<x)&(y<50)
ans =
   1
```

```
>> (x>15)|(y>55)
ans =
   0
>> ~(x>15)
ans =
   1
```

Exercice 2 :

Donner les commandes MATLAB permettant de calculer les expressions suivantes :

1. $x \leftarrow \frac{b}{2} \times \sqrt{c^2 - \left(\frac{b}{2,5}\right)^2}$
2. $y \leftarrow e^{2-\sqrt{b^3-\frac{1}{a}}}$

Solution :

1. $\boldsymbol{x = (b/2) * sqrt(c\text{^}2 - (b/2.5)\text{^}2)}$
2. $\boldsymbol{y = exp(2 - sqrt(b\text{^}3 - 1/a))}$

Exercice 3 :

Soit la fonction à deux variables suivante : $f(x,y) = \frac{1}{\sqrt{|x^3|+|y^3|}}$

Calculer : $f(-2,-3), f(-5,2.25), f(0.25,3.05)$

Solution :

```
>> x=-2;
>> y=-3;
>> f=1/sqrt(abs(x^3)+abs(y^3))
f =
   0.1690
>> x=-5;
>> y=2.25;
>> f=1/sqrt(abs(x^3)+abs(y^3))
f =
   0.0856
>> x=0.25;
>> y=3.05;
>> f=1/sqrt(abs(x^3)+abs(y^3))
f =
   0.1877
```

3. LES VECTEURS

Un vecteur sous MATLAB est une collection d'éléments du même type. Ils sont de deux types : ligne (une seule ligne, plusieurs colonnes) et colonne (une seule colonne, plusieurs lignes). Les différentes opérations qu'on peut réaliser sur un vecteur sont décrites par le **Tableau 1.7**.

3.1. Définition de vecteurs lignes et colonnes

Pour définir les tableaux à une ligne, ou vecteurs lignes on peut utiliser trois façons :

(1) Donner la liste entre crochets séparées par un espace:

```
>> v = [1  12.5  4  log(21)]
v =
 1.0000   12.5000    4.0000    3.0445
```

Ou séparées par des virgules :

```
>> v = [1,12.5,4,log(21)]
v =
1.0000   12.5000    4.0000    3.0445
```

(2) le définir globalement : Si h est du signe de (M-m), l'instruction X = m :h :M définit le tableau X à une ligne formé de tous les nombres de la forme $m + kh \; avec \; k \in \mathbb{N} \; et \; m + kh \; entre \; m \; et \; M$ Par contre, $si, h > 0 \; et \; M - m < 0 \; ou \; h < 0 \; et \; M - m > 0, X \; est \; vide$. Si h =1, on utilise X=m :M

```
>> X=1:0.3:2
X =
1.0000    1.3000    1.6000    1.9000
```

(3) utiliser des fonctions d'initialisation : Linspace(a,b,n) définit un vecteur ligne de n nombres régulièrement espacés entre a et b.

```
>> linspace(2,10,5)
ans =
 2    4    6    8   10
```

X= **zeros**(1,n) : définit un vecteur ligne de n valeurs égales à 0 ;

```
>> X=zeros(1,5)
X =
0    0    0    0    0
```

X= ones(1,n) : **définit** un vecteur ligne de n valeurs égales à 1.

```
>> X=ones(1,5)
X =
   1    1    1    1    1
```

Pour définir les tableaux à une colonne, ou vecteurs colonnes on peut utiliser trois façons :

(1) Donner la liste : entre crochets séparées par un des points-virgules ;

```
>> v = [1 ;12.5 ;4;log(21)]
v =
   1.0000
  12.5000
   4.0000
   3.0445
```

Ou écrivez la liste verticalement :

```
>> [1
2
3
4]
ans =
   1
   2
   3
   4
```

(2) utiliser des fonctions d'initialisation : X= zeros(n,1) : définit un vecteur colonne de n valeurs égales à 0 ;

```
>> X=zeros(5,1)
X =
   0
   0
```

```
   0
   0
   0
```

X= **ones**(n,1) : définit un vecteur colonne de n valeurs égales à 1 ;

```
>> X=ones(5,1)
X =
   1
   1
   1
   1
   1
```

3.2. Manipuler un vecteur

- **Transposé d'un vecteur :** permet de passer d'une ligne à une colonne ou le contraire. Se réalise avec le signe '.

Exemple :

```
>> v=[1 2 3]
v =
   1    2    3
>> v'
ans =
   1
   2
   3
```

- **Concaténer deux vecteurs :** permet de coller deux vecteurs.

Exemple :

```
>> v1= [1 3 5];
>> v2= [9 10 11];
>> v3= [v1 v2]
v3 =
   1    3    5    9   10   11
```

- **Extraction des sous-ensembles et accès aux éléments d'un vecteur :** permet de définir de nouveaux vecteurs extraits des vecteurs d'origines à l'aide des indices

Exemple :

```
>> v=[5 -1 13 -6 7]       %création du vecteur v de 5 valeurs
v =
   5   -1   13   -6    7
>> v(3)                   % la 3eme position
ans =
   13
>> v(2:4)                 % de la deuxième position jusqu'au quatrième
ans =
   -1   13   -6
>> v(4:-2:1)              % de la 4eme position jusqu'à la 1ere avec un pas = -2
ans =
   -6   -1
>> v(3:end)               % de la 3eme position jusqu'à la dernière
ans =
   13   -6    7
```

```
>> v([1,3,4])              % la 1ere, 3eme et 4eme position uniquement
ans =
   5   13   -6
>> v(1)=8                  % modifier la valeur du premier élément par la valeur 8
v =
   8   -1   13   -6   7
>> v(6)=-3                 % ajouter un 6ème élément avec la valeur -3
v =
   8   -1   13   -6   7   -3
>> v(9)=5                  % ajouter un 9eme élément avec la valeur 5
v =
   8   -1   13   -6   7   -3   0   0   5
```

3.3. Fonctions communes aux vecteurs lignes et colonnes

Des fonctions sur les vecteurs sont montrées par le **Tableau 1.8**.

Fonction	Objectif
sum(x)	Somme des éléments du vecteur x
prod(x)	Produit des éléments du vecteur x
max (x)	Plus grand élément dans x
min(x)	Plus petit élément dans x
mean(x)	Moyenne des éléments de x
sort(x)	Ordonne les éléments du vecteur x par ordre croissant
fliplr(x)	Renverse l'ordre des éléments du vecteur x
length(x)	Retourne la taille d'un vecteur
find(x)	Retourne un vecteur des indices des éléments non-nul de x
find(condition sur A)	Retourne les indices des valeurs satisfaisants une condition sur x

Tableau 1.8. fonctions à appliquer sur les vecteurs.

3.4. Opérations vectorielles

Il est possible de réaliser les opérations algébriques usuelles +, -, *, / pour les vecteurs. La somme et la soustraction sont des opérations termes à termes, donc les vecteurs doivent être de même dimension. Pour le produit et la division et la puissance termes à termes on doit remplacer *, / et ^ par .* et ./ et .^ (voir **Tableau 1.9**).

Opération	Exemple d'application :
	`>> u=[-2 6 1];  % création de u` `>> v=[3 -1 4];  % création de v`

Addition (+)	`>> u+2  % Addition d'un scalaire à un vecteur`
	`ans =`
	`  0  8  3`
	`>> u+v  % Addition de deux vecteurs`
	`ans =`
	`  1  5  5`

Soustraction (-)	`>> u-2  % soustraction d'un scalaire d'un vecteur`
	`ans =`
	`  -4  4  -1`
	`>> u-v  % soustraction élément par élément de deux vecteurs`
	`ans =`
	`  -5  7  -3`

Multiplication élément par élément (.*)	>> u*2 % Produit d'un scalaire par un vecteur ans = -4 12 2 >> u.*2 % produit élément par élément d'un scalaire par un vecteur ans = -4 12 2 >> u.*v % produit élément par élément de deux vecteurs ans = -6 -6 4 >> v.*u % produit élément par élément de deux vecteurs ans = -6 -6 4
Division élément par élément (./)	>> u/2 % Division d'un vecteur par un scalaire ans = -1.0000 3.0000 0.5000 >> u./2 % Division élément par élément d'un vecteur par un scalaire ans = -1.0000 3.0000 0.5000 >> u./v % Division élément par élément de deux vecteurs ans = -0.6667 -6.0000 0.2500
Puissance élément par élément (.^)	>> u.^2 % puissance élément par élément d'un vecteur à un scalaire ans = 4 36 1 >> u.^v % puissance élément par élément de deux vecteurs ans = -8.0000 0.1667 1.0000

Tableau 1.9. opérations vectorielles.

3.5. Exercice d'application :

- Définir le vecteur $v = (\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3})$
- Calculer $y1 = \sin(v)$ et $y2 = \cos(v)$
- Définir le vecteur v défini dans l'intervalle $[0 - 2\pi]$ $avec\ un\ pas\ de$ 0.1.
- Combien y a-t-il de valeurs dans ce vecteur ?
- Définir le vecteur x qui contient les éléments de v de la colonne 20 à la colonne 38 dans l'ordre inverse (c'est-à-dire de 38 à 20).
- Ordonner les éléments de x par ordre croissant.
- Supprimer tous les éléments de x.

Solution :

```
>> v=[pi/6,pi/4,pi/3]
v =
   0.5236   0.7854   1.0472

>> y1=sin(v)
y1 =
   0.5000   0.7071   0.8660
```

```
>> y2=cos(v)
y2 =
  0.8660   0.7071   0.5000
>> v=[0:0.1:2*pi]

v =
 Columns 1 through 19
     0   0.1000   0.2000   0.3000   0.4000   0.5000   0.6000   0.7000   0.8000   0.9000   1.0000   1.1000   1.2000
          1.3000   1.4000   1.5000   1.6000   1.7000   1.8000

 Columns 20 through 38
  1.9000   2.0000   2.1000   2.2000   2.3000   2.4000   2.5000   2.6000   2.7000   2.8000   2.9000   3.0000   3.1000
          3.2000   3.3000   3.4000   3.5000   3.6000   3.7000

 Columns 39 through 57
  3.8000   3.9000   4.0000   4.1000   4.2000   4.3000   4.4000   4.5000   4.6000   4.7000   4.8000   4.9000   5.0000
          5.1000   5.2000   5.3000   5.4000   5.5000   5.6000

 Columns 58 through 63
  5.7000   5.8000   5.9000   6.0000   6.1000   6.2000
>> length(v)
ans =
  63
>> size(v)
ans =
   1   63
>> x=v([38:-1:20])
x =3.7000   3.6000   3.5000   3.4000   3.3000   3.2000   3.1000   3.0000   2.9000   2.8000   2.7000   2.6000
          2.5000   2.4000   2.3000   2.2000   2.1000   2.0000   1.9000
>> sort(x)
ans =
  1.9000   2.0000   2.1000   2.2000   2.3000   2.4000   2.5000   2.6000   2.7000   2.8000   2.9000   3.0000   3.1000
          3.2000   3.3000   3.4000   3.5000   3.6000   3.7000
>> x=[]
x =
   []
```

4. LES MATRICES

Les matrices sont considérées comme les piliers de MATLAB. Un tableau de n lignes et p colonnes est une matrice de taille $n \times p$.

4.1. Définir une matrice

On peut définir une matrice de taille $n \times p$ de diverses manières :

1) **Le définir en extension :** on décrit la matrice ligne par ligne. Les éléments de la même ligne sont séparés par des espaces ou de virgules et les lignes sont séparées par des points-virgules.

```
>> D=[1 2 4;2 3 5]
D =
   1   2   4
   2   3   5

>> D=[1,2,4;2,3,5]
```

```
D =
   1    2    4
   2    3    5

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

2) **Définir une matrice depuis des vecteurs :** une matrice peut être définie par la concaténation de plusieurs vecteurs lignes ou colonne.

Exemple :

```
>> v1=1:4                    %création du premier vecteur v1
v1 =
   1    2    3    4

>> v2=5:5:20                 %création du premier vecteur v2
v2 =
   5   10   15   20

>> v3=4:4:16                 %création du premier vecteur v3
v3 =
   4    8   12   16

>> M=[v1;v2;v3]              % définition de M à partir de v1,v2 et v3 (ces vecteurs forment les lignes de M)
M =
   1    2    3    4
   5   10   15   20
   4    8   12   16
>> M=[v1' v2' v3'] % définition de M à partir du transposé de v1, v2 et v3 (ce vecteurs forment les colonnes de M)
M =
   1    5    4
   2   10    8
   3   15   12
   4   20   16
>> M=[v1;v1]              %M est définie par v1 dans sa 1ère et la 2ème ligne
M =
   1    2    3    4
   1    2    3    4
```

3) **Utiliser des fonctions d'initialisation :** si n et p sont des entiers strictement positifs, on peut utiliser les fonctions dans le **Tableau 1.10** pour initialiser ou générer automatiquement une matrice de dimension $n \times p$.

Fonction	**Objectif**
zeros(n,p)	Crée une matrice de taille nxp ne contenant que des 0
zeros(n)	Crée une matrice de taille nxn ne contenant que des 0
ones(n,p)	Crée une matrice de taille nxp ne contenant que des 1
ones(n)	Crée une matrice de taille nxn ne contenant que des 1

eye(n,p)	Crée une matrice de taille nxp contenant des 0 partout sauf sur la diagonale où il y a des 1
eye(n)	Crée une matrice de taille nxn contenant des 0 partout sauf sur la diagonale où il y a des 1
rand(n,p)	Crée une matrice de taille nxp contenant des éléments générés de manière aléatoire entre 0 et 1
magic(n)	Crée une matrice de taille nxn dont la somme des diagonales, des lignes et des colonnes sont égaux.

Tableau 1.10. fonctions prédéfinies de MATLAB utilisées pour initialiser une matrice.

4.2. Manipuler une matrice :

1) Extraction d'un élément : Pour extraire un élément de la matrice on indique la ligne et la colonne de celle-ci. Si M est une matrice de dimension $L \times C$ L'élément qui est à la $l - iéme$ ligne et la $c - iéme$ colonne de la matrice M est $M(l, c)$.

Attention :

- pour MATLAB, les indices sont toujours des entiers strictement positifs car ce sont des numéros de ligne et de colonnes. Par exemple, l'instruction $M(0,1)$ =50 ; provoque une erreur.
- si $l > L \; ou \; c > C$ alors :
  - Invoquer $M(l, c)$ à droite de $' = '$ ou dans une expression (logique ou algébrique) provoque une erreur.
  - Invoquer $M(l, c)$ à gauche de $' = '$ agrandit le tableau, les termes manquants sont initialisés à 0.

Exemple :

```
>> M=[3 5 6; 7 9 8]          % définition de la matrice M
M =
   3    5    6
   7    9    8
>> M(2,1)                    % extraction de l'élément dans la 2éme ligne et la 1ère colonne de M
ans =
   7
```

Lorsque l'on souhaite extraire une colonne ou une ligne entière on utilise le symbole (:) . donc l'accès à la $l - iéme$ ligne se fait par : $M(l, :)$ et l'accès à la $c - iéme$ colonne se fait par : $M(:, c)$

Exemple :

```
>> M(2,:)                    % extraction de la 2ème ligne de M
ans =
   7    9    8
>> M(:,1)                    % extraction de la 1ère colonne de M
ans =
   3
   7
```

Plusieurs combinaisons sont alors possibles. On peut extraire 2 colonnes par exemple en faisant :

```
M =
   3    5    6
   7    9    8
>> M(:,[1,2])
ans =
   3    5
   7    9
```

2) Suppression d'un élément : on peut supprimer des éléments de la matrice sans la supprimer en utilisant la commande : $M(:,c) = []$ pour les supprimer la colonne dans la $c-$ ème colonne, et $M(l,:) = []$ pour la $l-$ ème ligne.

Exemple :

```
>> M=[1 2 3; 4 5 6; 7 8 9]                    % définition de la matrice M
M =
   1    2    3
   4    5    6
   7    8    9

>> M(:,3)=[]                                  % supprimer tous les éléments de la 3 ème colonne
M =
   1    2
   4    5
   7    8

>> M(3,:)=[]                                  % supprimer tous les éléments de la 3 ème ligne
M =
   1    2
   4    5
```

3) Ajout de lignes ou de colonnes: Pour ajouter une nouvelle colonne on utilise la commande : $M = \left[M, [la\ nouvelle\ colonne]\right]$ avec la condition que le nombre des éléments dans la nouvelle colonne = l $(nombre\ de\ lignes\ de\ M)$. Aussi, pour ajouter une nouvelle ligne on utilise la commande : $M = [M; [la\ nouvelle\ ligne]]$ avec la condition que le nombre des éléments de la nouvelle ligne = c $(nombre\ de\ colonnes\ de\ M)$.

Exemple :

```
>> M=[1 2 3; 4 5 6;7 8 9]                     % définition de M
M =
   1    2    3
   4    5    6
   7    8    9
>> M=[M,[5 ;66; 99]]                          % ajout de la nouvelle colonne [5 ;66 ;99]
M =
   1    2    3    5
   4    5    6   66
   7    8    9   99
>> M=[M;[5 66 99 100]]                        % ajout d'une nouvelle ligne à M [5 66 99 100]
M =
   1    2    3    5
   4    5    6   66
   7    8    9   99
   5   66   99  100
```

4.3. Opérations matricielles :

Les opérations matricielles sont à la base du calcul sous MATLAB.

1) **Addition et soustraction :** sont possibles uniquement sur des matrices de taille identique (même nombre de lignes et même nombre de colonnes). Ce sont des opérations termes à termes, similaires aux opérations scalaires.

```
>> A=[1 2 3;4 5 6]              % définition de A
A =
   1   2   3
   4   5   6
>> B=[1 2 3; 4 5 6]             % définition de B
B =
   1   2   3
   4   5   6
>> A+B                          % calcul de la somme de A et B
ans =
   2   4   6
   8  10  12
>> A-B                          % calcul de la soustraction de B de A
ans =
   0   0   0
   0   0   0
```

2) **Multiplication** : il existe deux types de multiplication: la multiplication dite matricielle et la multiplication termes à termes. Cette dernière est l'analogue de l'addition et de la soustraction. Elle est notée de façon spécifique pour la distinguer de la vraie multiplication ($A.* B$) avec $A\ et B$ sont deux matrices de dimension identiques.

```
>> A=[1 2 3;4 5 6]              % définition de la matrice A
A =
   1   2   3
   4   5   6
>> B=[1 2 3; 4 5 6]             % définition de la matrice B
B =
   1   2   3
   4   5   6
>> A.*B                         % calcul du produit élément par élément de A par B
ans =
   1   4   9
  16  25  36
```

Le véritable produit matriciel entre la matrice A de dimension $m \times n$ et de la matrice B de dimension $n \times p$ est une matrice $M = AB$ de taille $m \times p$. Pour que ce produit soit réalisé, il est nécessaire que le nombre de colonnes de A soit égal aux nombre de lignes de B. Soit les éléments de A : a_{ij} et ceux de b_{ij}, alors les éléments de la matrice M sont définies par :

$$m_{i,j} = \sum_{0<k\leq n} a_{ik}\, b_{kj} \ldots\ldots\ldots\ldots (1)$$

Le principe de la formule est décrit de façon simple par la **Figue 1.2** :

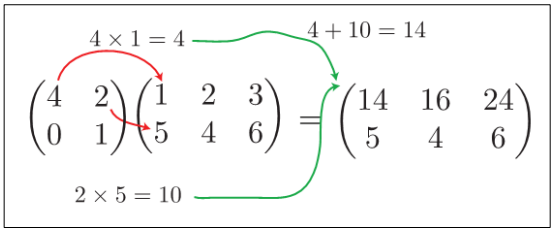


Figure 1.2: représentation graphique de la multiplication matricielle

Sous MATLAB, on calcule le produit en utilisant le signe * :

```
>> A=[1 2 3;4 5 6]                  % définition de A de taille 2x3
A =
   1   2   3
   4   5   6
>> B=[1 2;3 4;5 6]                  % définition de B de taille 3x2
B =
   1   2
   3   4
   5   6
>> A*B                              % calcule de A*B, la matrice résultante est de taille 2x2
ans =
  22   28
  49   64
>> C=[1 2 9;3 4 5]                  % définissons une nouvelle matrice C de taille 2x3
C =
   1   2   9
   3   4   5

>> A*C          % le calcul du produit A*C ne se réalise pas, puisque les dimensions ne respectent pas les conditions.
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

3) **Puissance (carré):** De même, si l'on souhaite obtenir le carré d'une matrice (au sens du produit termes à termes de cette matrice par elle-même).

```
>> A=[1 2 3;4 5 6]                  % définition de la matrice A
A =
   1   2   3
   4   5   6
>> A.^2                             % calcul de son carré A²
ans =
   1   4   9
  16  25  36
```

4) **Inverse:** puisque on dispose du produit matriciel, on peut définir l'inversion. En effet, on note A^{-1}, l'inverse de la matrice A (quand elle existe) et on définit A^{-1} par : $A^{-1}A = AA^{-1} = I$ où I est la matrice identité. Cette notion d'inverse est très importante pour la résolution d'un système d'équations. MATLAB calcule l'inverse par la fonction $inv(M)$.

```
>> M=[4 2;0 1];                     % définition de la matrice M
>> A=inv(M)                         % Calcule de l'inverse de M la matrice A
A =
  0.2500  -0.5000
       0   1.0000
```

5) **Division :** si A et B sont deux matrices de dimensions compatibles et B est inversible. la division se définit à partir de l'inverse. $A/B = AB^{-1}$.

```
>> A=[4 2;0 1];                    % définition de la matrice A
>> B=[1 2;5 4];                    % définition de la matrice B
>> M=A/B                           % calcule de la matrice M la division de la matrice A par B
M =
  -1.0000    1.0000
   0.8333   -0.1667
```

6) **Transposé :** lorsque A est une matrice à coefficient réels, la matrice A' est la transposée de A. C'est-à-dire que si $A = (a_{i,j})_{i,j}$ est une matrice de taille $n \times p$ a coefficients réels, alors $A' = (a'_{i,j})_{i,j}$ est la matrice de taille $p \times n$ definie par : $a'_{i,j} = a_{j,i}$ pour tout couple (i, j).

```
>> A=[4 2;0 1]                     % définition de la matrice A
A =
   4    2
   0    1
>> A'                              % calcule de transpose de A
ans =
   4    0
   2    1
```

Tableau 1.11 est un résume des différentes opérations matricielles :

Opération	Objectif
+	Addition
-	Soustraction
.*	Multiplication terme par terme
./	Division terme par terme
.\	Division inverse terme par terme
.^	Puissance (carre) terme par terme
*	Produit matriciel
/	Division matriciel

Tableau 1.11. opérations matriciel sous MATLAB.

4.4. Fonctions prédéfinies pour le traitement des matrices:

MATLAB inclut une liste très importante de fonctions prédéfinies qui nous permet de manipuler de grandes matrices d'une façon très simple et efficace. Nous regroupons dans le **Tableau 1.12** une liste non exhaustive des fonctions utiles pour le traitement des matrices on les clarifie par des exemples exécutés sous MATLAB.

Fonction	Objectif	Exemple
Size	Calcule la taille d'une matrice	>> A=[4 2;0 1] A = 4 2 0 1 >> [l,c]=size(A) l = 2 c = 2
det	Calcule le déterminant d'une matrice	>> A=[1 2;3 4]; >> det(A) ans = -2
rank	Calcule le rang d'une matrice	>> rank(A) ans = 2
trace	Calcule la trace d'une matrice	>> trace(A) ans = 5
eig	Calcule les valeurs propres d'une matrice	>> eig(A) ans = -0.3723 5.3723
dot	Calcule le produit scalaire de deux vecteurs	>> v1=[1 5 -3]; >> v2=[3 2 5]; >> dot(v1,v2) ans = -2
norm	Calcule la norme d'un vecteur	>> norm(v1) ans = 5.9161
cross	Calcule le produit vectoriel de deux vecteurs	>> a = [1 2 3]; b = [4 5 6]; c = cross(a,b) c = -3 6 -3
diag	Renvoie la diagonal d'une matrice	>> A=[1 2 3; 4 5 6;4 8 9] A = 1 2 3 4 5 6 4 8 9 >> v=diag(A) v = 1 5 9
diag(v)	Crée une matrice ayant le vecteur v dans la diagonale et 0 ailleurs	>> M=diag(v) M = 1 0 0 0 5 0 0 0 9

tril	Renvoie la partie triangulaire inférieure d'une matrice	>> tril(A) ans = 1 0 0 4 5 0 4 8 9
triu	Renvoie la partie triangulaire supérieure d'une matrice	>> triu(A) ans = 1 2 3 0 5 6 0 0 9

Tableau 1.12. fonctions prédéfinies pour le traitement matriciel sous MATLAB.

4.5. Résoudre un système linéaire (équations linéaires):

Soit A une matrice carrée de taille n et $B = (b_i)_i$une matrice colonne $n \times 1$,

Si le système d'équations linéaires $\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n = b_2 \\ \quad \cdots \quad \cdots \quad \cdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases}$

a une et une seule solution $(x_1, \ldots, x_n)$ alors la colonne $X = \begin{pmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix}$ est obtenue par la commande $X = A\backslash B$ connue sous le nom de division à gauche.

Exemple :

Résoudre $\begin{cases} 2x + 3x = 1 \\ 4x + 5y = 3 \end{cases}$

```
>> A=[2 3;4 5];
>> B=[1;3];
>> X=A\B
X =
   2
  -1
```

La réponse signifie $x = 2, y = -1$

4.6. Exercice d'application :

Créer une matrice ***M*** aléatoire carrée de taille$\mathbf{8 \times 8}$, contenant des valeurs comprises entre 0 et 8.

1) Calculer la moyenne des colonnes ; la moyenne des lignes puis la moyenne de la matrice.
2) Extraire la diagonale de la matrice.
3) Afficher la partie triangulaire.

```
>>A= 8*rand(8)        % matrice 8X8 avec des valeurs aléatoires entre 0 et 8

A =
   6.7578   1.8753   3.1775   5.7323   5.6285   5.3916   0.3942   6.4242
   2.9420   4.3903   3.3090   4.0905   3.8797   7.9956   4.5685   0.6710
   4.9664   7.4527   5.2417   6.2112   0.9169   7.6931   5.6069   7.5637
   5.8502   2.6816   6.7007   3.9148   5.3188   0.4709   7.6983   7.3275
```

```
   1.5511   5.2442   2.9729   1.4872   2.9230   2.8825   6.0041   4.8159
   7.2385   3.1352   3.4020   5.6051   1.1204   4.3881   5.9199   2.0285
   4.5536   5.0185   4.7573   7.8617   4.5342   2.0942   3.4550   6.9876
   5.0543   5.5926   4.5259   6.4531   6.5841   4.7788   5.0741   4.1072
>>mean(A,1)            %moyenne des colonnes
ans =
 4.2274   3.3155   5.0291   3.3450   3.9757   4.0157   5.0982   3.7430
>>mean(A,2)            %moyenne des lignes
ans =
 5.7794
   3.4697
   4.7007
   2.4101
   4.3791
   4.1211
   4.2718
   3.6176
>>mean(mean(A))        %moyenne de la matrice
ans =
   4.0937
>> diag(A)             %diagonale de la matrice
ans =
   5.8612
   0.4276
   6.3290
   2.1705
   6.6358
   3.0046
   3.5533
   3.7082
>> tril(A)             % la matrice triangulaire inférieure
ans =
   5.8612    0         0         0         0         0         0         0
   3.3778   0.4276     0         0         0         0         0         0
   7.6910   2.8532   6.3290      0         0         0         0         0
   0.5765   3.9864   6.5196   2.1705       0         0         0         0
   4.4273   3.4755   5.3600   3.2726    6.6358       0         0         0
   2.3359   4.4997   1.6070   3.7923    7.6490    3.0046       0         0
   6.8637   4.9330   2.1847   7.2719    4.7644    1.3292    3.5533       0
   2.6860   0.9067   5.0099   4.7700    0.2300    6.6652    4.9650    3.7082
>> triu(A)             % la matrice triangulaire supérieure
ans =
   5.8612   5.4416   7.1860   4.2948   2.6316   6.4969   6.7091   7.6135
      0     0.4276   6.0364   0.4760   3.8256   4.8809   3.6129   5.1201
```

0	0	6.3290	0.7117	4.7774	5.6119	7.6528	1.9786
0	0	0	2.1705	1.2916	0.7376	1.1772	2.8216
0	0	0	0	6.6358	3.3991	6.9595	1.5029
0	0	0	0	0	3.0046	6.1555	3.9252
0	0	0	0	0	0	3.5533	3.2742
0	0	0	0	0	0	0	3.7082

4.7. Conclusion :

Ce premier chapitre a été décomposé en trois parties principales. Dans la première, nous avons présenté l'interface graphique de MATLAB avec une explication de ses principales parties (Command Window, history, work space et current directory). Dans la même partie, nous sommes passés par les expressions arithmétiques, les types, les constantes prédéfinies, les nombres et les fonctions prédéfinies de MATLAB. Cette première partie permet aux étudiants d'apprendre à écrire une expression mathématique sous MATLAB. Tous les concepts présentés dans cette partie ont été abordés dans la série de TP1. Dans la deuxième partie, nous avons introduit le calcul vectoriel. Les étudiants apprennent ici à comment définir et manipuler un vecteur et connaitre les fonctions prédéfinis nécessaires pour une manipulation rapide des vecteurs. Les connaissances acquises dans cette partie ont été validé par la série de TP2. La troisième partie, traite le calcul matriciel. Les étudiants apprennent alors comment définir, et manipuler les matrices. Une liste de fonctions prédéfinies pour le calcul matriciel a été aussi présentée dans cette partie. La série de TP3 vient alors pour renforcer les notions théoriques présentées dans cette partie.

CHAPITRE 2 : PROGRAMMATION AVEC MATLAB

1. Introduction

Dans le premier chapitre, nous avons vu comment utiliser la *fenêtre de commandes* pour exécuter des commandes ou pour évaluer des expressions. Les commandes utilisées s'écrivent généralement sous forme d'une seule instructions (voir plusieurs dans une même ligne). Cependant, lorsque la tâche à réaliser devient plus complexe (plusieurs dizaines de ligne de code) ou que l'on souhaite pouvoir la transmettre à quelqu'un d'autre simplement, on utilise la *fenêtre Editor*. On a recours à l'utilisation des structures de programmation. Ces structures se résument en deux outils incontournables de MATLAB : les scripts et les fonctions. Pour faire des scripts complexes il est souvent nécessaire de faire appel aux structures de programmation qui font l'objectif de notre deuxième chapitre.

Pour visualiser et interpréter des résultats, il est souvent recommandé d'utiliser les graphiques et les outils de visualisation 2D et 3D de MATLAB. On détaillera à la fin de ce chapitre certains de ces outils de visualisation.

Les interfaces graphiques sont des outils efficaces qui vous permettent une interaction conviviale avec les applications développées, que ce soit sur le plan introduction des entrées ou affichage des résultats. Dans la dernière partie de ce chapitre, on apprend à comment réaliser des interfaces graphiques avec MATLAB.

2. Généralités

2.1. Variables, affectation :

L'instruction d'affectation $nom \leftarrow valeur$ de code : $nom = valeur$. S'il n'existe pas de variable nommée nom, cette instruction crée une variable appelée nom et lui affecte $valeur$. Sinon, sa valeur sera modifiée (remplacée) par la nouvelle $valeur$. Lorsqu'une variable est créée, son nom apparait dans l'espace de travail un icone qui indique la nature de son contenu, et la description de son encombrement mémoire.

Les noms de variables valides commencent par une lettre et sont en un seul mot. MATLAB distingue les majuscules et les minuscules.

Il existe cinq types de variables sous MATLAB : les entiers, les réels, les complexes, les chaines de caractères et le type logique.

2.2. Entrées/Sorties :

Sorties :

1. L'absence du ; à la fin d'une instruction provoque l'affichage du résultat de l'instruction.

```
>> y=3+5
y =
   8
```

2. Invoquer le nom d'une variable (qui existe) sans ; permet l'affichage de son nom et de sa valeur :

```
>> x=3+9;
>> x
```

```
>>x =
      12
```

3. L'instruction *disp* : l'instruction $disp(nombre)$ provoque l'affichage du nombre qu'elle reçoit en argument.

```
>> disp(2*15)
 30
```

L'instruction $disp(texte)$ provoque l'affichage du texte qu'elle reçoit en argument.

```
>> disp('hello world!!');
hello world!!
```

L'instruction $disp(nom\ de\ variable)$ provoque l'affichage du contenu de la variable qu'elle reçoit en argument.

```
>> x=(6+4)*15;
>> disp(x);
 150
```

4. Message d'erreur : la commande $error('message')$ permet d'afficher en rouge le texte $message$ et arrête l'exécution en indiquant l'endroit de l'erreur.

```
>> error('message')
??? message
```

Entrées :

1. La commande $x = input('saisir\ une\ valeur')$ affiche à l'écran *saisir une valeur* et attend qu'une réponse soit saisie (tapée) au clavier et validée par un *entrée*. La réponse sera affectée à x après la validation.

```
>> x=input('saisir une valeur');          % saisir une valeur au clavier
saisir une valeur 5                       % l'utilisateur tape 5
>> x                                      % 5 est affecté a x
x =
   5
```

2.3. Commentaires :

Les caractères qui suivent % ne sont pas lus par MATLAB. Cette fonctionnalité sert à écrire des commentaires (explications) à l'usage des utilisateurs.

```
>> x=input('saisir une valeur');          %la commande input permet de saisir une valeur au clavier
saisir une valeur 6
```

2.4. Les expressions logiques :

Le langage MATLAB dispose en standard de deux valeurs logiques : **0** représente **Faux/ False** et **1** qui représente **Vrai/ True**.

Un test ou expression logique est une expression qui retourne une valeur logique. Le **Tableau 1.6 (chapitre 1)** montre les opérateurs de comparaison qui s'appliquent sur des variables numériques réelles et entre caractères.

Notez bien :

- Si x et y sont de type caractère (l'affectation est un caractère), alors, les commandes :
  1) $x == y$ retourne **1** si la valeur affectée à la variable x est égale à celle affectée à y et **0** sinon ;
  2) $x < y$ renvoie **1** si la valeur de x précède celle de y dans l'ordre alphabétique conventionnel utilisé par MATLAB, **0** sinon. L'ordre utilisé par MATLAB utilise

les conventions suivantes : $' ' <' \cdots <' Z \ldots <' a' <' b' <' \cdots <' z'$ *et* $'0' <' 1' <' \cdots <' 9'$. Les caractères accentués sont plus loin dans la table.

- On peut combiner les expressions logiques à l'aide des trois opérateurs logiques (&, |, ~). Par exemple l'expression $(N > 0)\&(N == round(N))$ retourne **1** (vrai) si la variable réelle N contient un entier strictement positif et **0** (faux) sinon.
- La comparaison des vecteurs et des scalaires diffère des scalaires. Il existe des fonctions prédéfinies qui font ces comparaisons comme : $isequal, isempty$. $isequal$ Teste si deux (voir plusieurs) matrices sont égales (ayant les mêmes éléments partout, elle renvoie 1 si oui sinon 0). $isempty$ teste si une matrice est vide (ne contient pas d'éléments), elle renvoie 1 si oui sinon 0.

```
>> A=[1 2 3;4 5 6]          %définition de A
A =
   1   2   3
   4   5   6
>> B=[1 2 3;4 5 6]          %définition de A
B =
   1   2   3
   4   5   6
>> A==B                  % tester si A et B sont égales (leurs éléments sont-ils égaux ou non ?)
ans =
   1   1   1
   1   1   1
>> isequal(A,B)           % tester si A et B sont égales aussi (équivalente a A==B)
ans =
   1
>> M=[];                  % définition d'une matrice M vide
>> isequal(M,A)
ans =
   0
>> isempty(M)             % tester si la matrice M est vide ?
ans =
   1
```

3. Les fonctions sous MATLAB

3.1. Définitions

Une fonction c'est un script qui commence par le mot ***function.*** Une fonction permet d'étendre les possibilités au-delà des fonctions préprogrammées par les développeurs de MATLAB. Elle reçoit en entrée $zéro, un, deux\ ou\ plusieurs$ variables d'entrées appelées aussi $paramètres\ données$. Elle donne en sortie $zéro, une\ ou\ plusiuers$ variables appelées aussi $paramètres\ résultats$. Les variables d'entrées et de sorties peuvent être des matrices. Attention, lors de l'appel d'une fonction, c'est l'ordre des variables d'entrée et non pas leur nom qui détermine leur utilisation.

On utilise la syntaxe suivante pour créer une fonction dont :

- le nom est $calcul$;
- qui prend en variables d'entrée $x_1, x_2, \ldots, x_n$;
- qui a comme résultat $sortie1, sortie2 \ldots sortien$.

$$function\ [sortie1, sortie2 \ldots sortien] = calcul(x_1, x_2, \ldots, x_n)$$

$$description\ du\ procédé\ de\ calcul\ de\ Y\ au\ moyen\ de\ x_1, x_2, \ldots, x_n$$

Exemples de fonctions :

1. écrivez la fonction fonc1 qui permet d'afficher à l'écran: *Hello Nawel*

```
function [str]=fonc1(prenom)
str=['hello ', prenom];
end
```

2. on sauvegarde la fonction sous le nom *hello.m*. puis on l'appelle depuis la *Command Window* pour son exécution, mais cette fois avec un paramètre en entré de type chaine de caractère:

```
>> hello('Nawel')
ans =
hello Nawel
```

3. Écrivez la fonction permettant de calculer le résultat de la division de **a** par **b**:

```
function[c]=qot(a,b)
c=a/b;
end
```

4. en *Command Window* en exécute la fonction en donnant les deux valeurs de a et b :

```
>> qot(9,3)
ans =
   3
```

3.2. Variable formelle et variable effective, variable locale et globale

1. **Variable formelle :** dans la fonction *qot*, les variables a, b, c sont des *variables formelles*, elle n'ont pas d'existence effective. Même après appel de *qot*, en regardant *Workspace*, on voit qu'elles ne sont pas dans la liste des variables reconnues par CW. L'exemple montre ceci :

```
>> qot(9,3)
ans =
   3
>> a
??? Undefined function or variable 'a'.
```

Retenir: *les variables formelles d'entrée et de sortie ne servent qu'à indiquer le procédé de calcul. Elles ne sont reconnues que dans le script de la fonction.*

2. **Variable effective :** c'est une variable reconnue par *Command Window*. Dans l'exemple suivant les variables x, y et z sont effectives. Elles sont dans la liste des variables fournie par *Workspace*.

```
>> x=9;
>> y=3;
>> z=pgm2(x,y)
z =
   3
>> x
x =
   9
>> y
y =
   3
```

1) Variable locale / globale : outre les variables formelles et effectives, toutes les autres variables utilisées par une fonction sont classées en deux types : *locales* ou *globales*.

Dans MATLAB une variable qui n'est pas explicitement déclarée globale est une variable locale.

On déclare une variable globale comme suit : ***global x***

Lorsqu'une variable est locale à une fonction, elle n'est reconnue que par les actions de cette fonction. Lorsqu'une variable est déclarée globale dans plusieurs scripts, elle est partagée par tous les scripts.

Notez bien :

- Le **Tableau 2.3** montre une comparaison entre une fonction et un script MATLAB. Cette comparaison concerne la structure du programme et la façon de l'exécuter ou de l'invoquer dans la *Command Window*.

Un programme (script)	**Une fonction**
Structure	

x=input('donner un nombre positif: ');
y=input('donner un nombre positif: ');
z=x+y;
disp('la somme = ');
disp(z);
	function z= somme(x,y)
	z=x+y;
	end

Exécution	
>> **pgm2** %le nom du programme suffit donner un nombre positif: 5 donner un nombre positif: 15 la somme = 20	>> **somme(3,7) %** il faut les paramètres d'entrées ans = 10
Utilisation	
>> 2*pgm2+4 %utilisation interdite ??? Attempt to execute SCRIPT pgm2 as a function.	>> 2***somme(5,4)**+5 %utilisation permise ans = 23

Tableau 2.1: comparaison entre un programme et une fonction MATLAB

4. Création de programmes (scripts MATLAB)

Un programme MATLAB est un fichier stocké avec l'extension . ***m*** ou . ***M***. Un fichier programme est appelé le script de ce programme. Les noms de fichiers doivent être en un seul mot (pas d'espace) et sans accent, ni tiret ni point (sauf le .M). On peut utiliser le caractère de

soulignement ' _'. On utilise la fenêtre *Editor* pour créer un script. Un script est une suite de commande que l'on aurait tout aussi bien pu taper dans la *Command Window.*

Pour créer un nouveau script, suivez les étapes suivantes :

1. Ouvrir Editor-Debugger par : $file \rightarrow new \rightarrow M.file$ ($voir\ Figure$ 2.1)

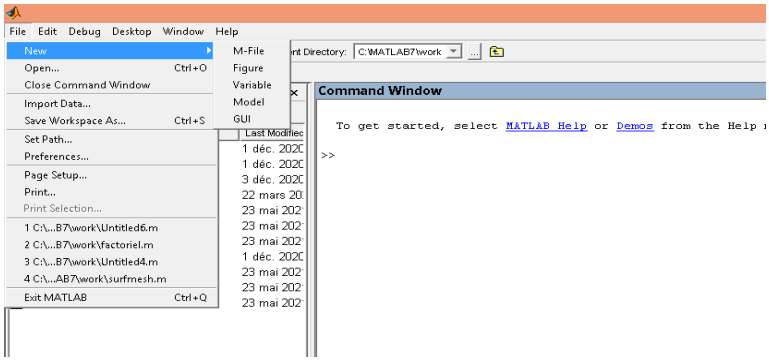


Figure 2.1 : créer un nouveau script MATLAB (méthode 1)

où cliquer sur l'icône page blanche (voir Figure 2.2):

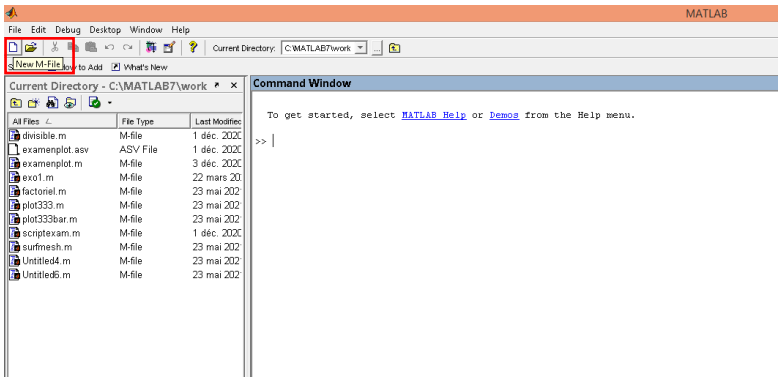


Figure 2.2 : créer un nouveau script MATLAB (méthode 2)

Après l'utilisation de l'une des deux méthodes précédentes, vous aurez une fenêtre de l'editor MATLAB similaire à celle dans la Figure 2.3. C'est là où vous devez écrire vos scripts et fonctions MATLAB.

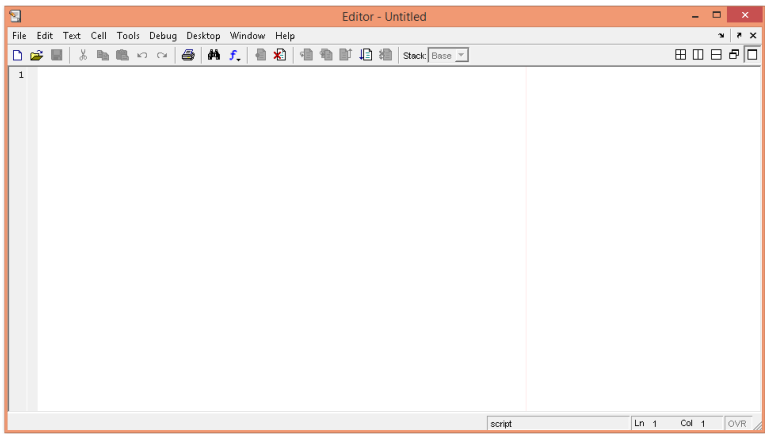


Figure 2.3 : editor MATLAB

2. Tapez une suite d'instructions dans l'éditeur ;

3. Sauvegarder par l'un des procédés suivants : $file \rightarrow save\ as$ ou $file \rightarrow save$ ou $icone\ disquette$.

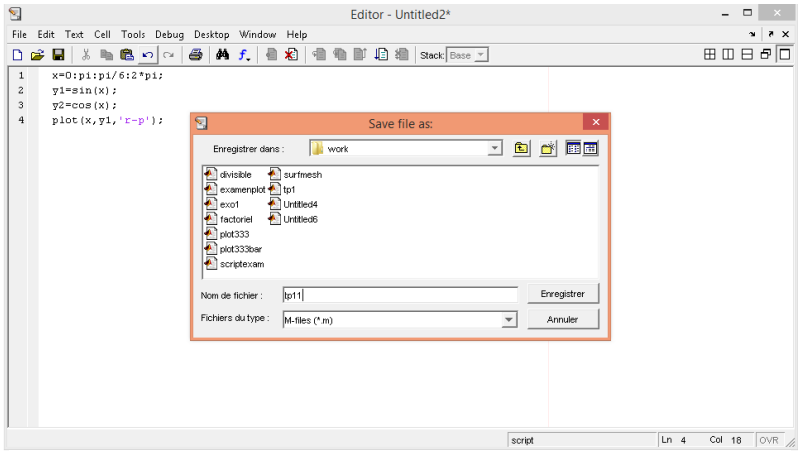


Figure 2.4 : enregistrer un script MATLAB

C'est possible d'exécuter un script MATLAB de deux façons différentes :

1. Invoquer son nom (le nom sous lequel il a été enregistré) sans l'extension $.m$ dans la $Command\ Window$ (voir Figure 2.5 où tp11 c'est le nom du script à exécuter).

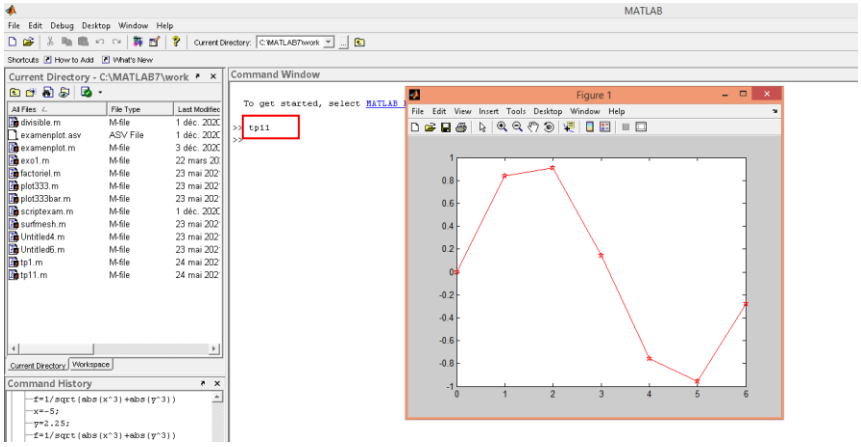


Figure 2.5 : exécuter un script MATLAB depuis la CW

2. Utiliser le bouton run dans l'éditeur (voir Figure 2.6).

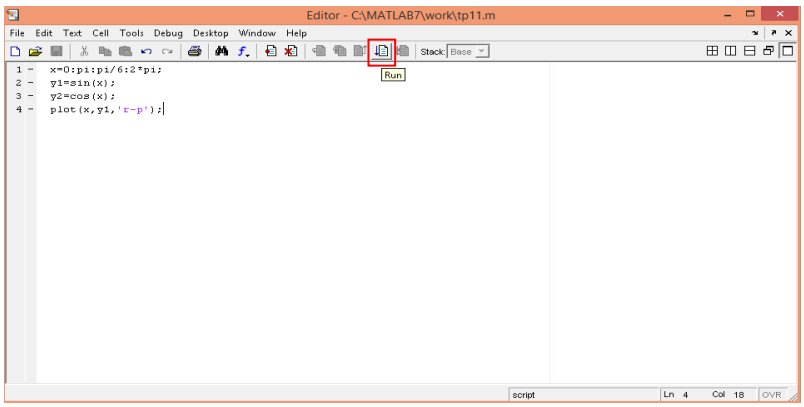


Figure 2.6 : exécuter un script MATLAB depuis le bouton run

Pour modifier un programme suivre le schéma :

1. Ouvrir le fichier : $file \rightarrow open \rightarrow double\ clique\ sur\ le\ fichier\ concerné$ ou $icone\ open$

2. Effectuer les corrections ou les modifications voulus, puis sauvegarder.

Notez bien :

- Un programme peut appeler un autre programme ou s'appeler lui-même ;
- Lorsqu'un programme ne s'arrête pas, ($plantage$). On quitte un plantage avec MATLAB en maintenant pressées les deux touches $CTRL$ et C, ou en provoquant une erreur d'entrée (en tapant une lettre accentuée) sans apostrophes lors de l'exécution d'une instruction input, sinon il faut fermer la session ou même éteindre l'ordinateur.

Exemples d'un script :

1. Programme qui affiche Hello World !! à l'écran. Le code MATLAB équivalent est le suivant sauvegardé sous le nom $pgm1.m$

```
str='Hello World !!';     %variable chaine de caractères qui contient hello World !!
str                       % affichage de la variable str
```

pour exécuter aller sur le bouton run, ou invoquer le nom du programme (nom de sauvegarde)

```
>> pgm1
str =
Hello World !!
```

2. écrire un programme qui demande deux nombres a et b, puis calcule et affiche le quotient a/b.

On peut déduire l'algorithme équivalent : $\begin{cases} lire(a,b) \\ c \leftarrow a/b \\ afficher(c) \end{cases}$

Le code MATLAB (à taper dans $Editor - Debugger - Window$) est :

```
disp('donner deux nombres pour calculer le quotient a/b');
A=input('tapez a=  ');
B=input('     b=  ');
C=A/B;
disp('le quotient = ');
disp(C);
```

On exécute dans la $Command\ Window$ de la même façon que l'exemple 1 :

```
>> pgm2
donner deux nombres pour calculer le quotient a/b
tapez a=  9
    b=  3
le quotient =
    3
```

Un script fait souvent appelle à des fonctions définies par le programmeur (voir même des fonctions prédéfinies). La forme générale d'un programme simple, consistant en un script qui appelle une fonction pour calculer et renvoyer une valeur, ressemble au diagramme illustré à la Figure 2.7.

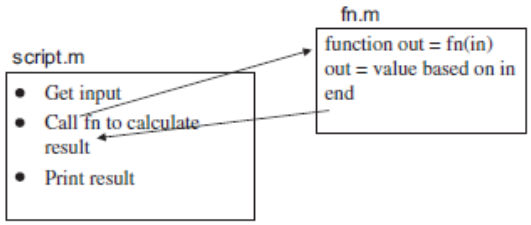


Figure 2.7. forme générale d'un script appelant une fonction

Exemple :

1. écrire la fonction factoriel permettant de calculer la factorielle d'un nombre entier x ;

```
function f=factoriel(x)
f=1;
for i=2:x
   f=f*i;
end
```

2. écrire le script MATLAB permettant de saisir un nombre entier et affiche sa factorielle (le script doit faire appel à la fonction factoriel.m).

```
x=input('donner une valeur');
disp('la factorielle de x est: ');
factoriel(x)    %%appel de la fonction factoriel
```

5. Les structures de contrôle

5.1. Alternative if :

Les instructions $si\ (test) alors < instruction >$ se traduit comme suit sous MATLAB, mais aucun mot ne traduit $alors$. Il y a trois versions qui traduit cette instruction :

1) **Version courte (conditionnelle simple):** Si test est une expression logique, le code de la structure :

$$\begin{cases} si\ test\ alors \\ instruction \\ fin \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test \\ instruction \\ end \end{cases}$$

Exemple : La fonction suivante affiche $Bonjour$!! si le nom passé en argument ='Nawel', ne fait rien autrement.

```
function nom(x)
if x=='nawel'
disp('Bonjour!!');
end
```

L'exécution dans la $Command\ Window$ donne:

```
>> pgm2('nawel')                %si le nom = nawel le programme affiche Bonjour
Bonjour!!

>> pgm2('laval')       %si le nom est different de nawel, le programme n'affiche rien
```

2) **Version normale (conditionnelle alternative):** Si test est une expression logique, le code de la structure :

$$\begin{cases} si\ test\ alors \\ instruction1 \\ sinon \\ instrcution2 \\ fin\ (si) \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test \\ instruction1 \\ else \\ instruction2 \\ end \end{cases}$$

Exemple : le programme suivant permet d'afficher si un nombre est positif ou négatif

```
x=input('donner nombre');
if x<0
   disp('nombre négatif');
else
   disp('nombre positif');
end
```

En exécutant dans la *Command Window*, on aura :

```
>>donner nombre-5
nombre négatif
>>donner nombre6
nombre positif
>>donner nombre0
nombre positif
```

3) Version compliqué (conditionnelle imbriquée): quand on dispose de trois conditions (cases) ou plus, on a recours à l'utilisation des si dans les rubriques sinon pour achever tous les conditions. La structure est la suivante :

$$\begin{cases} si\ test1\ alors \\ instruction1 \\ sinonsi\ test2\ alors \\ instruction2 \\ sinon \\ instrcution3 \\ fin\ (si) \end{cases} \quad \text{Devient} \quad \begin{cases} if\ test1 \\ instruction1 \\ elseif\ test2 \\ instruction2 \\ else \\ instruction3 \\ end \end{cases}$$

Exemple : reprenant le même exemple dans 2 mais on ajoutant le cas où le nombre est nul.

```
x=input('donner nombre');
if x<0
   disp('nombre négatif');
elseif x>0
   disp('nombre positif');
else
   disp('nombre nul');
end
```

En exécutant dans la *Command Window*, on aura :

```
>>donner nombre9
nombre positif
>>donner nombre-9
nombre négatif
>>donner nombre0
nombre nul
```

5.2. Instruction de Switch :

Exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé à une clause **case** qui définit si ce groupe doit être exécuté ou non. Si tous les

cas n'ont pas été acceptés, il est possible d'ajouter une clause **otherwise** qui sera exécutée dans ce cas. La structure de cette instruction est :

$$\begin{array}{|c} switch\ (expression) \\ case\ valeur\_1 \\ instructions\ 1 \\ case\ valeur\_2 \\ instrcution\ 2 \\ \dots \\ case\ valeur\_n \\ instrcution\ n \\ otherwise \\ autre\ instruction \end{array}$$

Exemple : prenant un exemple qui affiche plusieurs cas selon la valeur de x saisie au clavier :

```
x=input('donner nombre');
switch(x)
   case 0
      disp('x = 0');
   case 1000
      disp('x = 1000');
   case 2000
      disp('x = 2000');
   otherwise
      disp('x n''est pas 0 ni 1000 ni 2000');
end
```

L'exécution dans la *Command Window* donne comme résultat:

```
>>donner nombre0
x = 0
>>donner nombre1000
x = 1000
>>donner nombre2000
x = 2000
>>donner nombre5000
x n'est pas 0 ni 1000 ni 2000
```

5.3. Les boucles (répétitions) :

Une boucle permet de répéter la même commande un grand nombre de fois en faisant varier un paramètre. On verra deux types de boucles : boucles **while (tantque)** et **for (pour).**

1) La boucle while

Répète l'exécution d'un ensemble d'instructions un nombre indéterminé de fois, selon l'évaluation d'une condition logique. Elle s'écrit syntaxiquement comme suit :

$$\begin{array}{|c} \boldsymbol{tant\ que\ test\ faire} \\ \boldsymbol{instructions} \\ \boldsymbol{fin\ (tantque)} \end{array} \quad \text{Devient en MATLAB :} \quad \begin{array}{|c} \boldsymbol{while\ (condition)} \\ \boldsymbol{instructions} \\ \boldsymbol{end} \end{array}$$

Tant que la condition du **while** (expression) est correcte (évaluée par un **true**) la partie **instructions** sera exécutée encore.

Exemple : écrire une fonction somme qui prend en entrée un réel $\in]0, 20]$, donne un message d'erreur si $x \notin]0, 20]$, sinon calcule et donne en sortie le plus petit n tel que la somme $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ est supérieure à x.

Analyse : La structure algorithmique de la fonction est :

$$\left|\begin{array}{c} \boldsymbol{si\ x \le 0\ ou\ x \ge 20\ alors\ erreur} \\ \boldsymbol{sinon}\begin{cases} \boldsymbol{initialiser\ n\ a\ 0\ et\ S\ \text{à}\ 0\ (somme\ de\ 0\ termes)} \\ \boldsymbol{tantque\ S_n \le x\ (c'est\ \text{à}\ dire\ S < x)\ faire} \\ \begin{cases} \boldsymbol{n \leftarrow n+1} \\ \boldsymbol{S \leftarrow S + 1/n} \end{cases} \end{cases} \end{array}\right.$$

L'arrêt de la répétition a lieu pour la première valeur n telle que $S_n > x$.

Le code MATLAB équivalent est :

```
function N=somme(x)
if(x<=0)|(x>=20)
   error('x non conforme');
else
   N=0; S=0;              %initialisation de N et S
   while S<=x
      N=N+1;              %N est incrémenté (augmente) de 1
      S=S+1/N;            % S est remplacée par S+1
   end                    %end while
end                       %end if
```

2) La boucle for

La boucle for répète l'exécution d'un groupe d'instructions un nombre défini de fois (le nombre de répétitions voir itération est connu à l'avance). Une boucle for se définie syntaxiquement comme suit :

$$\left|\begin{array}{c} \boldsymbol{pour\ i = m1 : h : m2\ faire} \\ \boldsymbol{instrcution} \\ \boldsymbol{fin\ (pour)} \end{array}\right. \quad \text{Devient en MATLAB :} \quad \left|\begin{array}{c} \boldsymbol{for\ i = m1 : h : m2} \\ \boldsymbol{instructions} \\ \boldsymbol{end} \end{array}\right.$$

- La boucle effectue *instructions* pour les valeurs i comprises entre $m1$ et $m2$ suivantes : $m1, m1+h, m1+2h$ et ainsi de suite. Par contre si $h > 0\ et\ m1 > m2$ alors *instructions* n'est pas effectuée ; de même si $h < 0\ et\ m1 < m2$.
- Si h=1, on ne peut pas l'écrire, donc les écritures $for\ i = m1 : 1 : m2$ et $for\ i = m1 : m2$ sont équivalentes.

Exemple : écrire une fonction $factorielle(n)$ qui prend comme entrée un entier $n \ge 0$, et donne en sortie n !

Analyse : La structure algorithmique de la fonction est :

$$\left|\begin{array}{c} \boldsymbol{donner\ n\ comme\ param\text{è}tre\ d'entr\text{é}ede\ la\ fonction} \\ \boldsymbol{initialiser\ f\ \text{à}\ 1\ (f \leftarrow 1)} \\ \boldsymbol{pour\ i\ allant\ de\ 1\ \text{à}\ n\ faire} \\ \boldsymbol{f \leftarrow f * i} \\ \boldsymbol{afficher\ f} \end{array}\right.$$

Le code MATLAB équivalent est :

```
function f=factorielle(n)
f=1;
for i=1:n
   f=f*i;
end
```

3) Sorties continue et break en cours d'instruction

Dans les structures de boucles, un bloc d'instructions est exécuté un certain nombre de fois, et ce nombre de fois peut même être infini dans le cas de boucles while.

Deux instructions permettent d'interrompre l'exécution d'un bloc d'instructions d'une boucle.

1. **Continue:** interrompt l'exécution du bloc d'instructions en cours d'exécution, et passe à l'itération suivante de la boucle for ou while.
2. **Break:** interrompt l'exécution du bloc d'instructions en cours d'exécution, et sort totalement de la boucle for ou while, en ignorant les itérations suivantes

Exemple 1 (continue): le programme MATLAB qui permet d'afficher les nombres divisibles par 7 compris entre 7 et 50. L'instruction continue permet si un nombre non divisible par 7 d'ignorer son affichage avec l'instruction disp et passe au nombre suivant.

```
for n = 1:50
   if mod(n,7)
      continue
   end
   disp(['Divisible by 7: ' num2str(n)])
end
```

L'exécution sur CW donne :

Divisible by 7: 7
Divisible by 7: 14
Divisible by 7: 21
Divisible by 7: 28
Divisible by 7: 35
Divisible by 7: 42
Divisible by 7: 49

Exemple 2 (break):

```
c=input('donner un nombre des nombres')
n=0;
while n<=c
   x=input('siaisr un nombre ');
   if x==0
      break
   end
   disp(['le nombre saisie est',num2str(x)])
   n=n+1;
end
```

L'exécution sur CW donne :

```
donner un nombre des nombres5
c =5
siaisr un nombre 3
le nombre saisie est3
siaisr un nombre 4
le nombre saisie est4
siaisr un nombre 0
>>
```

6. Les graphiques et la visualisation de données sous MATLAB

Une compétence importante pour utiliser MATLAB dans la rédaction des documents scientifiques est la capacité de réaliser des graphs de qualité présentant clairement vos données. Les récentes versions de MATLAB ont fait d'énormes progrès dans ce domaine et il est maintenant possible d'obtenir des rendus très corrects avec MATLAB. Cette Section présente les fonctions graphiques de haut niveau, c'est-à-dire celles qui permettent de manière simple et rapide d'avoir un graphe satisfaisant.

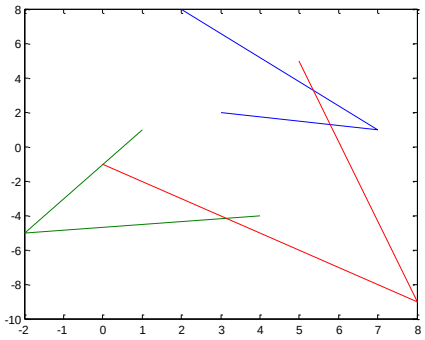
6.1. Tracés de fonctions et de données 2D

La fonction plot est la fonction de base pour tracer des fonctions. Elle est utilisable par des vecteurs ou des matrices. Elle trace des lignes en reliant des points de coordonnées définies dans ses arguments. Elle se présente sous plusieurs formes :

- La forme de base est $plot(x, y)$ où x et y sont des matrices de même dimension. Elle considère les valeurs de chaque colonne de la première matrice comme les éléments de l'axe X, et les valeurs de chaque colonne de la deuxième matrice comme les valeurs de l'axe Y.

  Exemple :

```
>> A=[3 4 5; 7 -2 8; 2 1 0]
A =
   3   4   5
   7  -2   8
   2   1   0
>> B=[2 -4 5; 1 -5 -9; 8 1 -1]
B =
   2  -4   5
   1  -5  -9
   8   1  -1
>> plot(A,B)
```



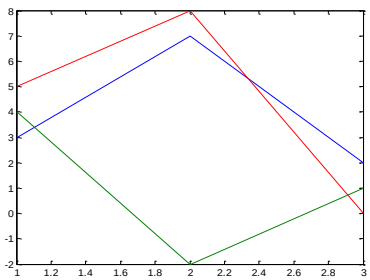
- S'il n'y a qu'une seule matrice comme argument, elle considère les valeurs de chaque colonne comme les éléments de l'axe Y, et leurs positions relatives (le numéro de ligne) comme les valeurs de l'axe X. Donc elle donne plusieurs courbes, une pour chaque colonne.

  Exemple :

```
>> A=[3 4 5; 7 -2 8; 2 1 0]

A =


   3   4   5
   7  -2   8
   2   1   0
>>plot(A)
```



- S'il y a un seul vecteur comme paramètre, alors elle considère les valeurs du vecteur comme les éléments de l'axe Y (les ordonnées), et leurs positions relatives définissons l'axe X (les abscisses).

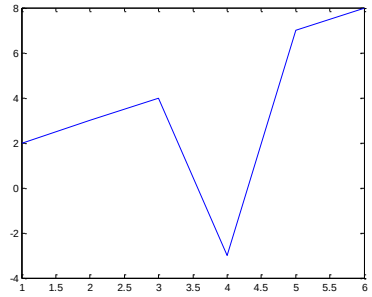
Exemple :

```
>> v=[2 3 4 -3 7 8]

v =

   2    3    4   -3    7    8

>> plot(v)
```

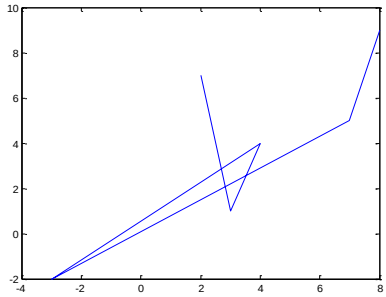


- Si x et y sont des vecteurs, alors elle considère les valeurs du premier vecteur comme les éléments de l'axe des abscisses et les éléments du deuxième comme ceux des ordonnées.

```
>> v=[2 3 4 -3 7 8]
v =
   2    3    4   -3    7    8

>> z=[7 1 4 -2 5 9]
z =
   7    1    4   -2    5    9

>> plot(v,z)
```



- Un troisième paramètre de type chaine peut être présent. Il autorise des couleurs et des formes variées pour les points de la courbe et le style du tracé. Cette chaine de caractères peut contenir à la fois une couleur, un style de ligne et un marqueur de point. Si le style de ligne est absent, seuls les marqueurs sont tracés (voir **Tableau 2.2**). La commande devient alors : plot (x,y,'marqueur').

Symbol	signification	Symbol	signification	Symbol	signification
y	Jaune (yellow)	**:**	Pointillé	**s**	Carré (square)
m	Magenta	**-.**	Tirets-points	**d**	Diamant (diamond)
c	Cyan	**_**	Tirets	**^**	Triangle supérieur
r	Rouge (red)	**.**	Point	**v**	Triangle inférieur
g	Vert (green)	◦	Cercle	**>**	Triangle droit
b	Bleu (blue)	**+**	Plus	**<**	Triangle gauche
w	Blanc (white)	**-**	Trait	**p**	Pentagramme
k	Noir (black)	*****	étoile	**h**	hexagramme
				×	Croix

Tableau 2.2: chaines permettant la modification de la couleur, style de courbe et la représentation des points

Exemple : tracer la fonction $y = sin(x)$ avec $x \in [0 \dots 2\pi]$ avec un $pas = \pi/6$. En changeant le marqueur on obtient différentes résultats (voir **Tableau 2.3**)

```
>> x=0:pi/6:2*pi;
>> y=sin(x);
```

plot(x,y,'r:*')	plot(x,y,'black -.^')	plot(x,y,'pb-')

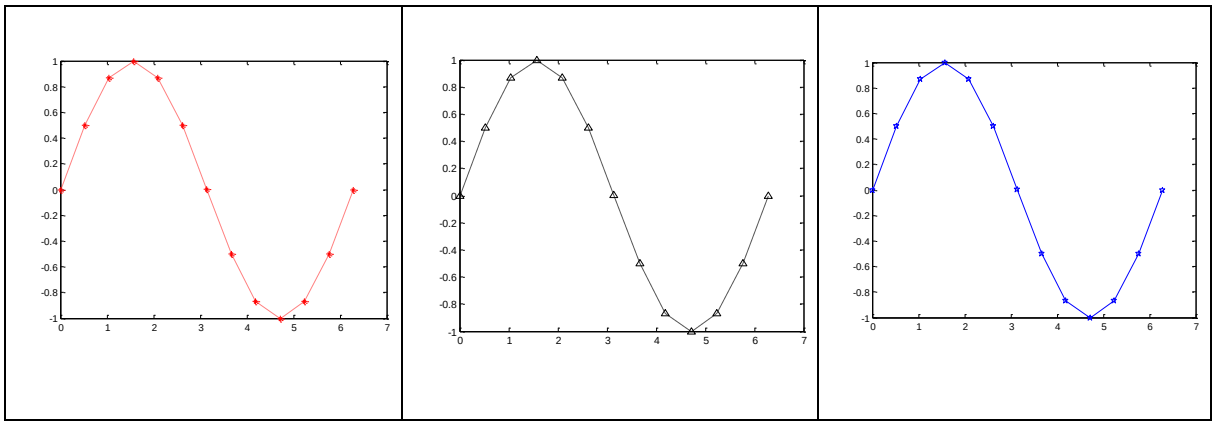


Tableau 2.3: résultats de graphs avec des marqueurs différents

6.2. Tracés de fonctions et de données 3D

MATLAB a des fonctions qui afficheront des courbes 3D (sur 3 axes). Pour tracer des courbes 3D on utilise la fonction prédéfinie de MATLAB plot3(X,Y,Z) où X, Y et Z sont soit des vecteurs soit des matrices. Ces fonctions affichent les points dans l'espace 3D. Cliquer sur l'icône de rotation 3D puis dans la courbe permet à l'utilisateur de faire pivoter pour voir la courbe sous différents angles. De plus, l'utilisation de la fonction de **grid** facilite la visualisation, comme le montre la Figure 2.8.

Le code MATLAB permettant de tracer la courbe 3D de la Figure 2.8 est le suivant ; où la fonction **zlabel** est utilisée pour étiqueter l'axe z.

```
>> x = 1:5;
>> y = [0 -2 4 11 3];
>> z = 2:2:10;
>> plot3(x,y,z,'k*')
>> grid
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('3D Plot')
```

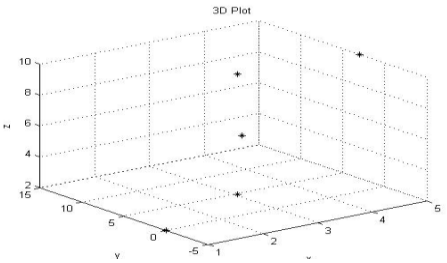


Figure 2.8 : courbe 3D avec grilles

Pour la fonction bar3, les vecteurs y et z sont passés et la fonction affiche des barres 3D comme illustré, dans l'exemple suivant et la Figure 2.9.

```
y = 1:6;
z = [33 11 5 9 22 30];
bar3(y,z)
xlabel('x')
ylabel('y')
zlabel('z')
title('3D Bar')
```

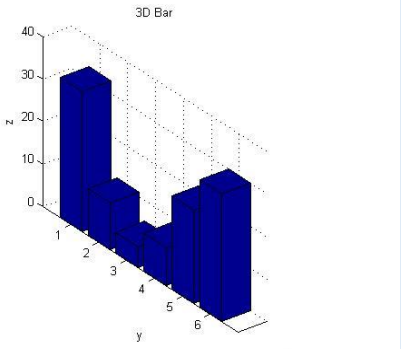


Figure 2.9 : barres 3D avec grilles

D'autres types de courbes 3D intéressants incluent **mesh** et **surf**. La fonction **mesh** dessine un maillage filaire de points 3D, tandis que la fonction **surf** crée un tracé de surface en utilisant la couleur pour afficher les surfaces paramétriques définies par les points. MATLAB a plusieurs

fonctions qui créeront les matrices utilisées pour les coordonnées (x, y, z) pour les formes spécifiées (par exemple, sphère (voir exemple 1 et Figure 2.10) et cylindre (voir exemple 2 et Figure 2.11)).

Exemple 1 :

```
[x,y,z] = sphere(15);
size(x)
mesh(x,y,z)
title('Mesh of sphere')
```

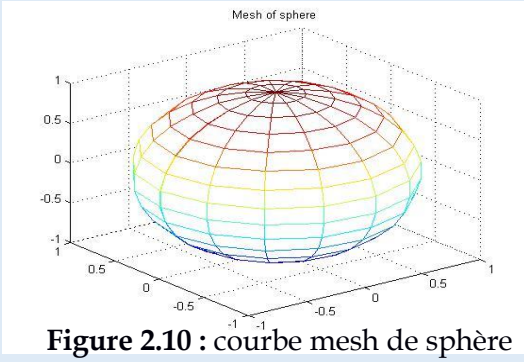


Figure 2.10 : courbe mesh de sphère

Exemple 2:

```
[x,y,z] = sphere(15);
surf(x,y,z)
title('Surf of sphere')
colorbar
```

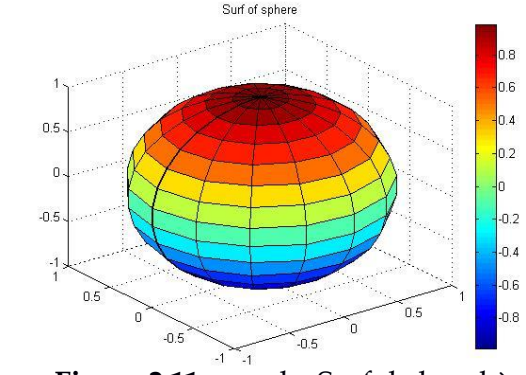


Figure 2.11 : courbe Surf de la sphère

6.3. Graphes spécifiques :

MATLAB dispose de plusieurs routines permettant de tracer des graphes spécifiques (voir **Tableau 2.4**) :

Fonction	signification
bar	barres
errorbar	Barre d'erreur
hist	Histogramme
polar	Coordonnées polaires
rose	Histogramme angulaire
stairs	Marche d'escalier

Tableau 2.4: fonctions de graphes spécifiques sous MATLAB

Exemple : voici un exemple de tracé avec les différentes fonctions dans le **Tableau 2.5.**

x=linspace(0,1,1000); **>> polar(sin(8.*x.*pi), cos(12.*x.*pi),'oy')**	**>> x=linspace(0,8*pi,1000);** **>> rose(x.*x,20)**

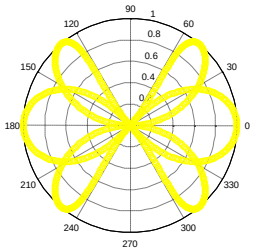
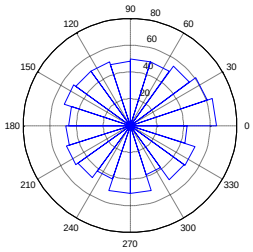
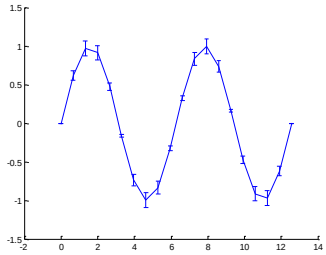
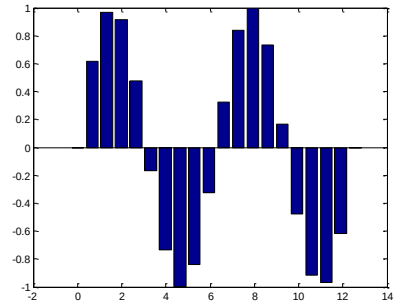
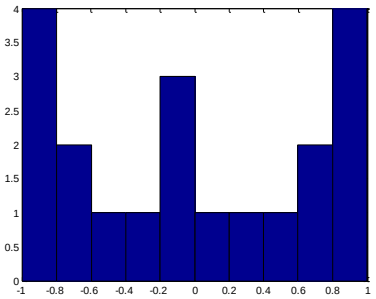
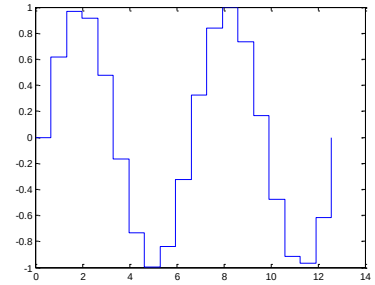
	
>> x=linspace(0,4*pi,20); **>> errorbar(x,sin(x),0.1*sin(x))**	**>> x=linspace(0,4*pi,20);** **>> bar(x,sin(x))**
	
>> x=linspace(0,4*pi,20); **>> hist(sin(x))**	**>> x=linspace(0,4*pi,20);** **>> stairs(x,sin(x))**
	

Tableau 2.5: exemples de tracé avec fonctions spécifiques MATLAB

6.4. Organiser les graphes

Les trois outils de haut niveau pour organiser les graphes sont :

- ***Hold*** (***hold on***)**:** conservation/effacement des tracés précédents ;
- ***Figure*** : nouvelle fenêtre ;
- ***Subfigure*** : axes multiples dans une même fenêtre.

Si l'on désire tracer plusieurs graphes, trois possibilités existent :

- Les tracer tous dans les mêmes axes ;
- Les tracer tous la même fenêtre, dans des axes distincts ;
- Les tracer dans des fenêtres différentes.

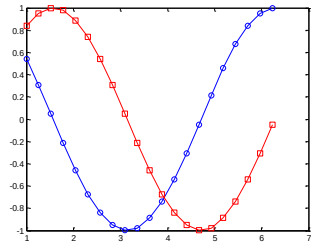
Et on peut évidemment mélanger ces options.

1) **Graphes dans les mêmes axes :** Pour pouvoir tracer plusieurs graphes successivement et en les superposant, il faut, après avoir effectué un premier tracé, exécuter au moins une

fois la commande : *hold on*. Pour annuler son effet, il suffit de taper la commande *hold off*.

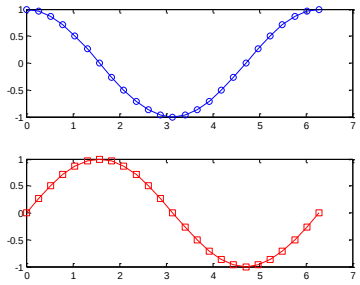
Exemple :

```
>> x=01:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>> plot(x,y1,'b-o')
>> hold on
>> plot(x,y2,'r-s')
```



2) **Graphes dans la même fenêtre :** ceci signifie qu'une fenêtre peut être découpée en plusieurs zones, en fait en plusieurs axes. La fonction qui réalise ce découpage c'est : *subplot*. Avec la syntaxe $Subplot(m, n, p)$, le *subplot* permet de decouper la fenêtre en $m \times n$ axis et place la figure le p *axis*

```
>> subplot(2,1,1);      %subdivisé la fenêtre en 2 lignes, 1 colonne et placé la 1ere figure dans l'axe 1.
>> plot(x,y1,'b-o')   %tracer la 1ère figure
>> subplot(2,1,2);     % se placer au 2ème axe
>> plot(x,y2,'r-s')     % tracer la 2ème figure
```

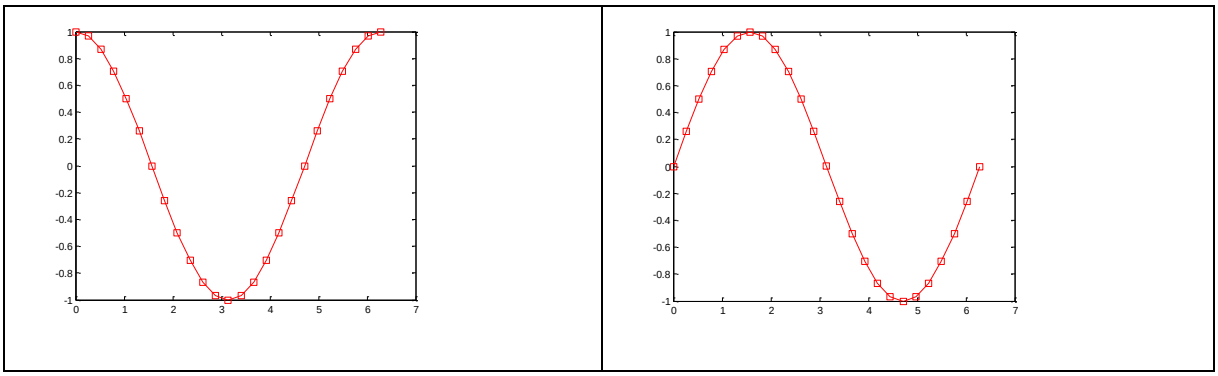


3) **Graphes dans des fenêtres différentes :** c'est la fonction $figure$, qui permet d'ouvrir de nouvelles fenêtres graphiques. Chaque fenêtre possède un numéro unique (la première étant numéro 1). ***Figure*(*n*)** où n est un entier, permet soit de créer une fenêtre de numéro n, ou bien si elle existe déjà, de la rendre active. Le numéro de la fenêtre active peut être obtenu par la fonction ***gcf*** (***Get handle to current figure***). Il faut faire attention au fait qu'une fenêtre graphique, même si elle est active, n'est pas forcément visible, étant recouverte par d'autres. La commande ***shg*** (***Show graph window***) peut la faire émerger.

Exemple : avec la commande figure on crée à chaque appel une fenêtre séparée, puis on trace la fonction sur la figure active.

```
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>>figure
>> plot(x,y1,'r-s')
```

```
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>> figure
>> plot(x,y2,'r-s')
```



6.5. Placement de textes dans les graphiques :

Partout où existent des axes, un texte peut être inséré à l'aide des fonctions suivantes (**Tableau 2.6**):

Fonction	Objectif
gtext	Placement de texte à l'aide de la souris
legend	Légendes
text	Annotation
title	Titre du graphe
xlabel	Label de l'axe des x
ylabel	Label de l'axe des y
zlabel	Label de l'axe z

Tableau 2.6: fonctions pour insérer du texte dans un graphique

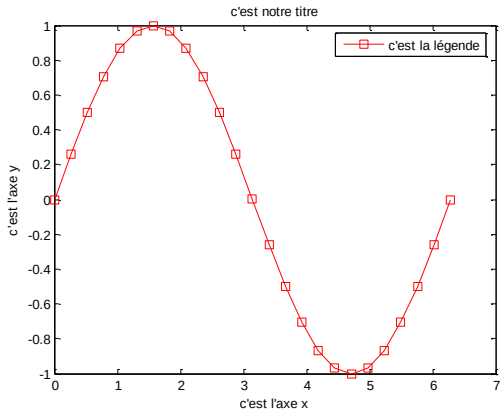
Exemple : on peut ajouter des informations qui expliquent le graphe. Le titre, la légende, l'axe des x et des y à l'aide du Tableau 2.8 comme suit :

```
>> legend('c''est la légende')

>> title('c''est notre titre')

>> ylabel('c''est l''axe y')

>> xlabel('c''est l''axe x')
```



6.6. Modification des axes :

Elle peuvent être réalisées par la fonction/commande *axis*. A l'aide de cette fonction, on peut d'une part fixer le domaine des axes indépendamment de ce que l'on y trace, et d'autre part indiquer leurs tailles respectives :

- ***axis***([***xmin xmax ymin ymax***]) **:** permet de fixer les limites des axes 2D ;
- ***axis***([***xmin xmax ymin ymax zmin zmax***]) **:** celles des axes 3D.

Les axes ont un comportement par défaut leur permet de s'adapter automatiquement aux données qui y sont tracées. Ceci peut être modifié (voir **Tableau 2.7**):

Fonction	Objectif
axis manual	Règle les axes qui restent constants indépendamment des données qui y sont tracées, si le graphe est en mode conservation (hold on)
axis tight	Règle le rapport d'aspect de sorte que toutes les unités soient identiques dans les différentes directions ;
axis fill	Règle les limites sur les intervalles des données ;
axis off	Cache les lignes, les graduations et les labels des axes
axis on	On rétablit leur visibilité.

Tableau 2.7: fonctions pour régler le comportement des axes dans un graphe.

6.7. Les interfaces utilisateurs graphiques sous MATLAB

Les interfaces utilisateur graphiques, ou GUI (Graphical user interfaces) sont essentiellement des objets qui permettent aux utilisateurs d'avoir des entrées à l'aide d'interfaces graphiques, telles que : les boutons, les radio-boutons, menus contextuels…etc. Les interfaces graphiques permettent de contrôler des applications logicielles avec des commandes de type pointer-cliquer.

Dans MATLAB, il existe deux méthodes de base pour créer des interfaces graphiques :

1. **Créer une interface graphique MATLAB de manière interactive :** utiliser l'environnement de développement d'interface utilisateur graphique (GUIDE) intégré. Le GUIDE permet à l'utilisateur de présenter graphiquement l'interface graphique et MATLAB génère automatiquement le code correspondant.
2. **Créer une interface graphique MATLAB de manière programmatique :** pour pouvoir comprendre et modifier ce code, il est important de comprendre les concepts de programmation sous-jacents.

Comme premier démarrage avec les GUI on se focalise sur la première méthode (utilisation du GUIDE MATLAB).
Le GUIDE est un constructeur d'interface graphique qui regroupe tous les outils dont le programmeur a besoin pour créer une interface graphique de façon intuitive. Il s'ouvre, soit en cliquant sur l'icône  , soit en tapant guide dans le Command Window de MATLAB. Soit en choisissant file→ new→ GUI (voir Figure 2.12)

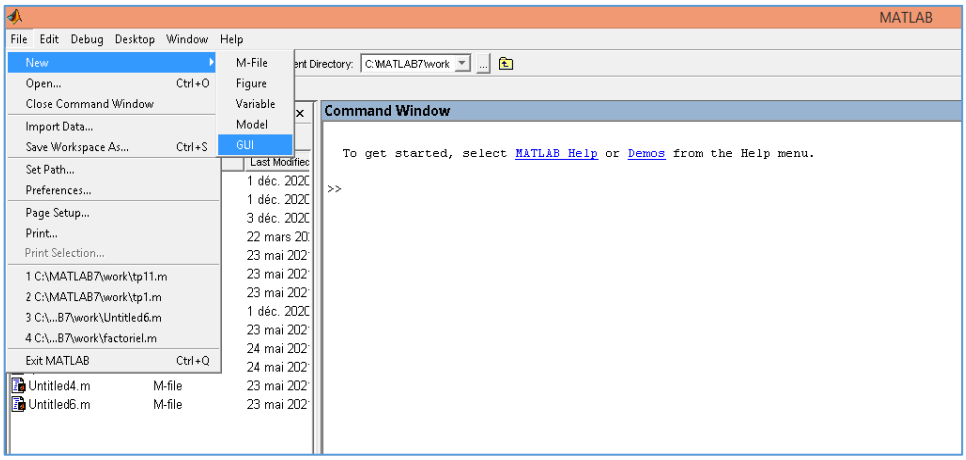


Figure 2.12 : créer une nouvelle GUI via le menu

On choisissant new GUI depuis le menu file, la fenêtre suivante s'affiche (voir Figure 2.13):

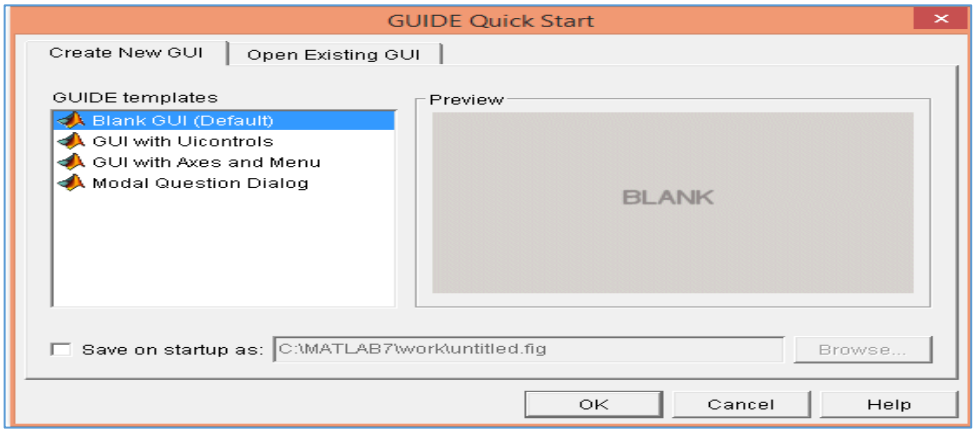


Figure 2.13 : la fenêtre de création du GUI

Le placement des objets est réalisé par sélection dans une boite à outils. Leur mise en place et leur dimensionnement se font à l'aide de la souris. Dans la Figure 2.14, la GUI est créée et la boite à outils se trouve à gauche (celle encadrée en rouge).

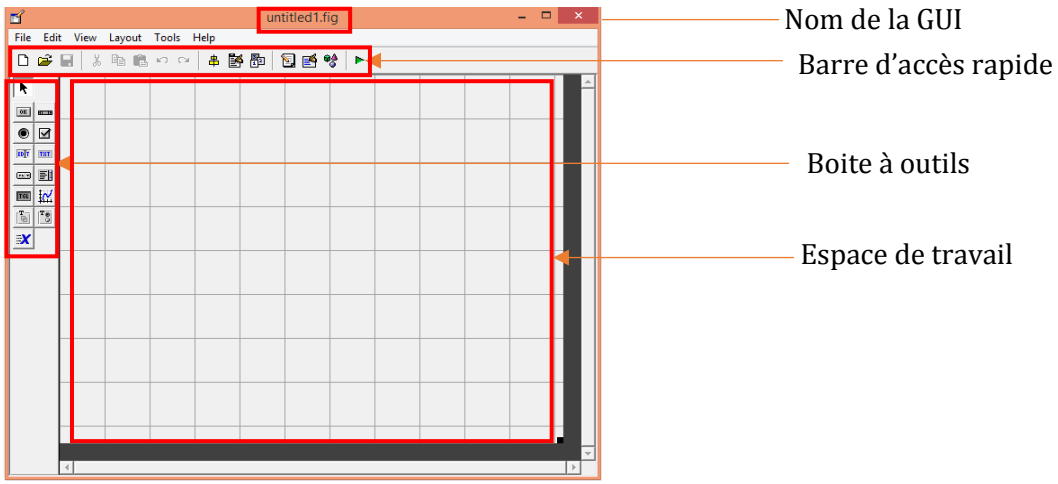


Figure 2.14 : boite à outils.

Un double-clique sur un objet permet de faire apparaître le Property Inspector (voir Figure 2.15) où les propriétés des objets sont facilement éditables. Leurs modifications et la visualisation de ces modifications sont immédiates.

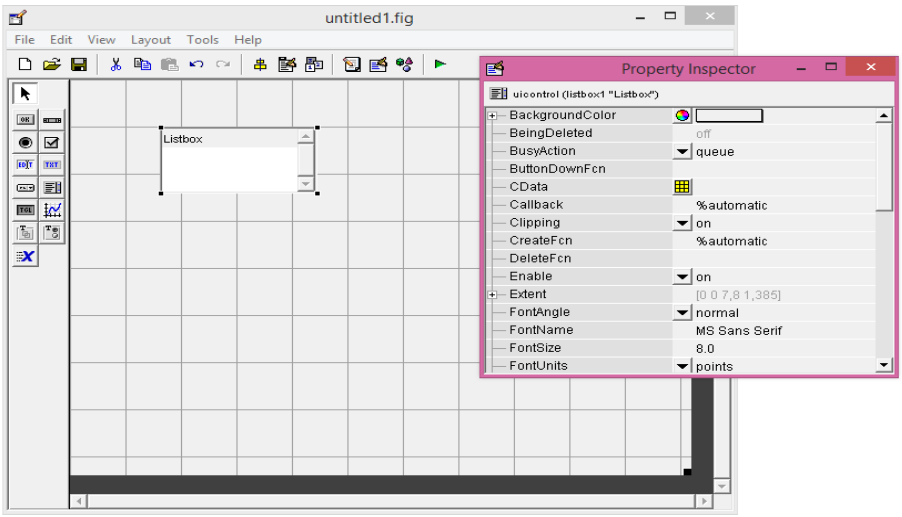


Figure 2.15 : property Inspector

Le GUIDE possède également des outils pour gérer l'alignement des objets et pour créer des barres d'outils ou des menus. Une fois l'interface graphique terminée, son enregistrement donne deux fichiers portant le même nom mais dont les deux extensions sont .fig et .m.

Le fichier .fig contient la définition des objets graphiques (positions et propriétés). Ce fichier peut être ouvert ultérieurement avec le GUIDE pour modifier les objets graphiques.

Le fichier .m contient les lignes de code qui assurent le fonctionnement de l'interface graphique (actions des objets). Ce fichier peut être édité dans le MATLAB Editor pour y ajouter des actions à la main. C'est ce fichier qui doit être lancé pour utiliser l'interface graphique.

6.8. Conclusion

MATLAB permet trois objectifs : (1) calcul scientifique, (2) programmation (scripts et fonctions) et (3) la visualisation de données. L'objectif 1 a été abordé en détails dans le premier chapitre. Dans ce chapitre, nous avons abordés les objectifs 2 et 3.

La programmation sous MATLAB est aussi facile par rapport aux autres langages de programmation du fait qu'il y a pas de déclaration. La programmation sous MATLAB est réalisée à l'aide de fonctions ou de scripts. Ces derniers sont des portions de code qui peuvent s'exécuter en isolation et qui font souvent appel aux fonctions. Tandis que les fonctions sont des portions de codes qui s'exécute à leur appel avec des paramètres effectifs. Nous avons présenté dans ce chapitre les structures de contrôle utilisable sous MATLAB. La série de TP4, permet un renforcement et un côté pratique de ce qui a été appris ici.

Dans des travaux de recherche c'est toujours préférable de présenter les résultats sous forme graphique et de les analyser par la suite. MATLAB offre des outils de visualisation graphique 2D et 3D. Nous avons présenté dans ce chapitre aussi les fonctions permettant de tracer une courbe 2D, dans une figure indépendante des autres courbes, dans la même figure ou même dans le même axe sur la même figure. D'autres fonctions viennent pour compléter ou expliquer une courbe comme le titre des abscisses, des ordonnées, la légende et le titre du graphe. Les notions théoriques abordées dans cette partie du chapitre sont validées par la série de TP5.

CHAPITRE 3 : APPLICATIONS DES METHODES NUMERIQUES AVEC MATLAB

1. Introduction

L'analyse numérique est l'étude des algorithmes permettant de résoudre numériquement par discrétisation les problèmes de mathématiques continues. Cela signifie qu'elle s'occupe principalement de répondre de façon numérique à des questions à variable réelle ou complexe comme l'algèbre linéaire numérique sur les champs réels ou complexes, la recherche de solution numérique d'équations différentielles et d'autres problèmes liés survenant dans les sciences physiques et l'ingénierie. C'est une branche des mathématiques appliquées dont son développement est étroitement lié à celui des outils informatiques.

L'utilisation de l'analyse numérique est grandement facilitée par les ordinateurs. La disponibilité et la puissance des ordinateurs a permis l'application de l'analyse numérique dans de nombreux domaines scientifiques, techniques et économiques, avec souvent des effets importants.

Dans ce chapitre, nous présentons quelques applications des méthodes numériques avec MATLAB. On commence par présenter les polynômes et leur résolution avec MATLAB. Nous montrons par la suite, la résolution de certains systèmes particuliers sous MATLAB (à matrice diagonale, à matrice triangulaire inférieure, à matrice triangulaire supérieure). Puis, nous présentons une méthode de résolution des équations non linéaires (méthode de la dichotomie). Enfin, nous présentons la méthode directe de Gauss, une méthode de résolution des systèmes d'équations linéaires.

2. Les polynômes sous MATLAB

Les principales fonctions liées aux polynômes sont : la représentation d'un polynôme, la recherche de racines; l'évaluation et l'adaptation à des données.

2.1. Représentation d'un polynôme :

MATLAB représente un polynôme sous la forme d'un vecteur ligne de ses coefficients classés dans l'ordre des puissances décroissantes. Le polynôme P d'expression : $P\ (x)\ =\ x^2\ -\ 6x\ +\ 9$, est représenté par le vecteur suivant : P = [1 – 6 9] . Le nombre d'éléments du vecteur est égal au degré du polynôme + 1.

2.2. Racines d'un polynôme :

On peut déterminer les racines du polynôme $P\ (x)\ =\ 2\ x^2\ -\ 3x\ +\ 1$ à l'aide de la fonction ***roots***.

```
>> P = [2 -3 1]
>> roots (P)
ans =
   1.0000
   0.5000
```

2.3. Évaluation de polynômes :

Pour évaluer un polynôme en un point, on utilise la fonction ***polyval***. Essayons de trouver la valeur du polynôme P en 2.

```
>> polyval (P, 2)
ans =
   3
```

2.4. Détermination d'un polynôme à partir de ces racines :

On peut déterminer les coefficients d'un polynôme à partir de ses racines en utilisant la fonction ***poly***. On cherche, par exemple, le polynôme qui a pour racines : 1, 2 et 3.

Celles-ci peuvent être définies comme les éléments d'un vecteur r.

```
>> r = [1 2 3]
>> K = poly (r)
K =
   1   -6   11   -6
```

Qui correspond à : $\boldsymbol{K\,(x)\;=\;x^3-6x^2+11x-6}$. En multipliant par un réel non nul, tous les coefficients de K, on trouve un autre polynôme ayant les mêmes racines que K.

On vérifie bien que les racines du polynôme K sont 1, 2 et 3.

```
>> racines = roots (K)
racines =
3.0000
2.0000
1.0000
```

2.5. Dérivée d'un polynôme :

Une autre commande utile est **polyder**, qui permet à partir d'un vecteur interprété comme polynôme, de calculer le vecteur représentant la dérivée de ce polynôme.

Calculons la dérivée du polynôme représenté par le vecteur [1 2 1].

```
>> r=[1 2 1] ;

>> polyder(r)
ans =
   2    2
```

2.6. Représentation graphique :

Pour représenter graphiquement (tracer la courbe) un polynôme $\boldsymbol{K\,(x)\;=\;x^3-6x^2+11x-6}$, définissons un domaine pour la variable x qui contient les racines de K.Domaine des valeurs de la variable x et évaluation du polynôme K:

```
>> x = 0:0.1:4;
>> y = polyval (K, x);
```

Tracé la fonction y = K (x) :

```
>> plot (x, y)
>> grid on
>> title ('tracé de y = x^3 - 6x^2 + 11x -6')

>> xlabel ('x')
>> ylabel ('y')
```

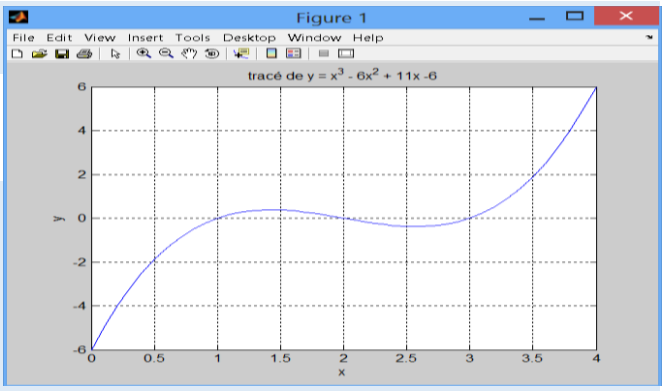


Figure 3.1 : représentation graphique de y

Quelques autres fonctions utiles pour la manipulation des polynômes sont citées dans le **Tableau 3.1**.

Fonction	Opération à effectuer
conv(p, q)	Multiplication de p par q
deonv(p, q)	Effectuer une division de p par q.
polyint(p)	Retourne l'intégrale de p

Tableau 3.1: certaines fonctions pour manipuler un polynôme sous MATLAB

Exercices d'application :

Soit les polynômes : $A = x^3 + 2x^2 + 3x + 4,\ B = x^3 + 4x^2 + 9x + 16$

1. Calculer le produit des deux polynômes ;
2. Calculer la somme des deux polynômes ;
3. Soit **C** le polynôme obtenu après la multiplication de : **A** et **B**, faire la division de **C** par **B** ;
4. Tracer la courbe du polynôme $P = x^3 + 4x^2 - 7x - 10$, pour des valeurs de x entre -1 et 100 espacées par un pas de 3.

Solution sous MATLAB :

```
>> A=[1 2 3 4];  %écrivant d'abord les vecteurs lignes représentant les polynômes A et B
>> B=[1 4 9 16];
>> C=conv(A,B)  %appliquant ensuite la fonction conv qui calcul le produit
C =
   1    6   20   50   75   84   64
>> D=A+B        %appliquant l'opérateur + pour réaliser la somme
D =
   2    6   12   20                 %D correspond a D = 2x^3 + 6x^2 + 12x + 20
>> [Q,R]=deconv(C,B)          %appliquant la fonction deconv qui permet de calculer la division de deux polynômes
Q =
   1    2    3    4                 %Q représente le quotient de la division

R =
   0    0    0    0    0    0    0      %R représente le reste de la division
>> x=linspace(-1,3,100);      %définition de x
>> P=[1 4 -7 -10];             %définissons P
>> V=polyval(P,x);             %polyval évalue le polynôme P(x) aux différentes valeurs de x et place le résultat dans
V
>> plot(x,V)                   %les valeurs de V en fonction de x constituent la courbe qui représente le polynôme P(x)
>> title('x^3+4x^2-7x-10'), grid
```

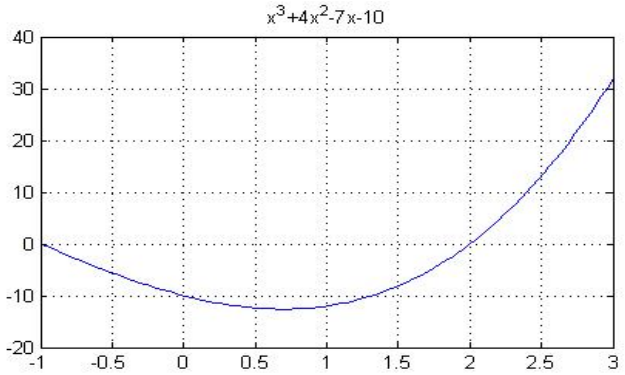


Figure 3.2 : représentation graphique de V

3. Résolution des systèmes particuliers

3.1. Systèmes à matrice diagonale

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ . & . & . & . \\ 0 & 0 & a_{ii} & 0 \\ . & . & . & . \\ 0 & 0 & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_i \\ . \\ y_n \end{bmatrix}$$

La solution de ce type de systèmes résulte directement de :

$$x_i = y_i / a_{ii} \;\; avec \; 1 \leq i \leq n$$

3.2. Systèmes à matrice triangulaire supérieure

$$\begin{bmatrix} a_{11} & a_{12} & . & a_{1j} & . & a_{1n} \\ 0 & a_{22} & . & a_{2j} & . & a_{2n} \\ . & . & . & . & . & . \\ 0 & 0 & . & a_{ij} & . & a_{in} \\ . & . & . & . & . & . \\ 0 & 0 & . & 0 & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_i \\ . \\ y_n \end{bmatrix}$$

Le calcul de la solution se fera par substitution arrière suivant les étapes suivantes :

- Commençons par calculer : $x_n = y_n / a_{nn}$
- Substituer x_n sa valeur dans l'équation $n-1$ et calculer x_{n-1}, nous obtenons : $x_{n-1} = [y_{n-1} - a_{n-1,n}\, x_n] / a_{nn}$;
- De la même façon, on calcule le reste des inconnus. Nous pouvons écrire la formule de récurrence suivante : $x_n = y_n / a_{nn}$, $x_i = [y_i - \sum_{k=i+1}^{n} a_{ik} x_k] / a_{ii}$ $\;i = n-1$ à 1

3.3. Systèmes à matrice triangulaire inferieure

$$\begin{bmatrix} a_{11} & 0 & . & 0 & . & 0 \\ a_{12} & a_{22} & . & 0 & . & 0 \\ . & . & . & . & . & . \\ a_{i1} & a_{i2} & . & a_{ij} & . & 0 \\ . & . & . & . & . & . \\ a_{n1} & a_{n2} & . & a_{nj} & . & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_i \\ . \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ y_i \\ . \\ y_n \end{bmatrix}$$

Le calcul de la solution se fera par substitution avant suivant les étapes suivantes :

- Commençons par calculerx_1 à l'aide de la 1ere équation, nous obtenons : $x_1 = y_1 / a_{11}$
- Substituer à x_1 sa valeur dans la 2eme équation et calculer x_2, nous obtenons : $x_2 = [y_2 - a_{21}\, x_1] / a_{22}$;
- De la même façon, on calcule le reste des inconnus. Nous pouvons écrire la formule de récurrence suivante : $x_1 = y_1 / a_{11}$, $x_i = [y_i - \sum_{k=0}^{n} a_{ik} x_k] / a_{ii}$ $\;i = 2$ à n

4. Résolution des équations non linéaires

Nous présentons dans cette section une méthode de résolution numérique des équations non linéaires à une seule variable. L'extraction des racines d'une équation non linéaire $f(x) = 0$ passe nécessairement par deux étapes :

- Situer la racine (séparation des racines), en l'encadrant par un intervalle $[x_1, x_2]$ ne contenant que celle-ci ;
- Améliorer ensuite la valeur de cette racine, pour atteindre la précision voulue en l'une des méthodes de résolution.

4.1. Méthode de la dichotomie :

Soit $f(x) = 0$ une équation à résoudre, continue sur un segment d'intervalle $[x_1, x_2]$. Si $f(x_1)f(x_2) < 0$, il existe alors avec certitude au moins une racine dans cet intervalle.

Le principe de cette méthode est :

- Diviser le segment en deux parties, le milieu de l'intervalle étant $x_3 = 0.5(x_1 + x_2)$. Si $f(x_3) = 0$ alors x_3 est une racine, sinon nous devrons situer le nouvel intervalle $[x_1, x_3]$ ou $[x_3, x_2]$ qui contient la racine recherche.
- Le nouvel intervalle est divisé à son tour en deux segments pour situer un nouveau domaine contenant la racine. Il est procédé ainsi jusqu'à ce que la différence entre les limites du domaine qui encadrent la racine soit inferieure a la précision recherchée (E). si f(x1)f(x3) < 0 -> choisir [x1, x3], sinon choisir [x3, x2].

La fonction est évaluée $n + 1$ fois pendant chacune des n itérations. Le nombre n d'itérations nécessaires pour obtenir un niveau de précision donné de la racine, peut être calculé au préalable à l'aide de l'expression suivante : $n = log\left[\frac{x_2 - x_1}{E}\right]/log2 + 1$

Algorithme de la Dichotomie :

Données : $f(x), x_1, x_2, e$

$y_1 = f(x_1)$

$$Tant\ que\ (y_m \neq 0) \begin{cases} x_m = (x_1 + x_2)/2 \\ y_m = f(x_m) \\ si \quad [y_1 . y_m] < 0 \quad alors\ (x_2 = x_m) \\ sinon \qquad (x_1 = x_m) \end{cases}$$

Racine recherchée: x_m

Exercice d'application :

Calculer la racine qui se trouve dans l'intervalle $[x_1 = 1, x_2 = 2]$ et la racine qui se trouve dans l'intervalle $[x_1 = 4, x_2 = 5]$ pour l'équation $f(x) = exp^x - x^3$, en utilisant la méthode de la dichotomie avec une précision de $e = 10^{-5}$.

Sur l'intervalle $[\boldsymbol{x_1 = 1, x_2 = 2}]$,

1ere étape : nous trouvons $f(x_1) = 1.718281$ et $f(x_2) = -0.610943$, cela vérifie que $f(x_1)f(x_2) < 0$,

2eme étape : le calcul itératif

itération	x_1	x_2	x_m	$f(x_m)$
1	1	2	1.5	1.106689
2	1.5	2	1.75	0.3952228
3	1.75	2	1.875	-0.070978
4	1.75	1.875	1.8125	0.171397
5	1.8125	1.875	1.84375	0.052525
6	1.84375	1.875	1.859375	-0.008648
7	1.84375	1.859375	1.851562	0.022083
8	1.851562	1.859375	1.855469	0.006754
9	1.855469	1.859375	1.857422	0.000938
10	1. 855469	1.857422	1.856445	0.002910
11	1.856445	1.857422	1.856934	0.000986
12	1.856934	1.857422	1.857178	0.000024
13	1.857178	1.857422	1.857300	-0.000457
14	1.857178	1.857300	1.857239	-0.000216
15	1.857178	1.857239	1.857208	-0.000096
16	1.857178	1.857208	1.857193	-0.000036
17	1.857178	1.857193	1.857185	-0.000006

Apres 17 itérations nous obtenons la racine $x_r = 1.857185$

Sur l'intervalle $[\boldsymbol{x_1 = 4, x_2 = 5}]$, nous trouvons que $(x_1) = -9.401849$ et $f(x_2) = 23.413159$, cela vérifie que $f(x_1)f(x_2) < 0$. Après 18 itérations, le calcul itératif donne a racine $x_r = 4.536404$

5. Résolution des systèmes d'équations linéaires

Un système d'équations linéaires est un ensemble d'équations portant sur les mêmes inconnues. Un système de m équations linéaires à n inconnus peut être écrit sous la forme suivante:

$$\begin{pmatrix} \boldsymbol{a_{11}x_1 - a_{12}x_2 +} & \boldsymbol{\cdots + a_{1n}x_n =} & \boldsymbol{y_1} \\ \boldsymbol{a_{21}x_1 - a_{22}x_2 +} & \boldsymbol{\cdots + a_{2n}x_n =} & \boldsymbol{y_2} \\ \boldsymbol{\vdots} & \boldsymbol{\ddots} & \boldsymbol{\vdots} \\ \boldsymbol{a_{n1}x_1 - a_{n2}x_2 +} & \boldsymbol{\cdots + a_{nn}x_n =} & \boldsymbol{y_n} \end{pmatrix} \boldsymbol{o}\text{ù: } x_1, x_2 \ldots, x_n \textit{ sont des inconnus}$$

Un système d'équations linéaires peut aussi s'écrire sous la forme matricielle : $Ax = y$ tels que :

$$A = \begin{bmatrix} a_{11}\, a_{12}x_2 & \cdots\, a_{1n} = & y_1 \\ a_{21}\, a_{22}x_2 & \cdots\, a_{2n} = & y_2 \\ \vdots & \ddots & \vdots \\ a_{n1}\, a_{n2} & \cdots\, a_{nn} = & y_n \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \; et \; x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Si la matrice est inversible, il est possible de calculer $x = A^{-1} \times y$. Plusieurs méthodes directes et itératives permettent de résoudre un système linéaire comme ceux de Gauss (méthode directe) et de Gauss-Seidel (méthode itérative).

5.1. Méthode Directe de Gauss

La méthode de Gauss consiste en premier lieu à réaliser des transformations linéaires sur le système pour obtenir un nouveau système équivalent : $[a'_{i,j}]\{x_j\} = \{y'_j\}$ à matrice $A' triangulaire$, et en second lieu a résoudre ce système équivalent par substitution arrière (ou avant).

$$\begin{bmatrix} a_{11}\, a_{12}x_2 & \cdots\, a_{1n} \\ a_{21}\, a_{22}x_2 & \cdots\, a_{2n} \\ \vdots & \ddots \\ a_{n1}\, a_{n2} & \cdots\, a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \rightarrow \begin{bmatrix} a'_{11}\, a'_{12}x_2 & \cdots\, a'_{1n} \\ 0 \quad a'_{22}x_2 & \cdots\, a'_{2n} \\ \vdots & \ddots \\ 0 \quad 0 & \cdots\, a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y'_1 \\ y'_2 \\ \vdots \\ y'_n \end{bmatrix}$$

Système initial **Système à matrice A′ triangulaire**

Pour expliquer le passage du système initial au système à matrice triangulaire, nous nous limitons à l'étude du système a quatre équations suivant :

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= y_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= y_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= y_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= y_4 \end{aligned}$$

Qui s'écrit sous la forme matricielle :

$$\begin{bmatrix} a_{11}\, a_{12}\, a_{13}\, a_{14} \\ a_{21}\, a_{22}\, a_{23}\, a_{24} \\ a_{31}\, a_{32}\, a_{33}\, a_{34} \\ a_{41}\, a_{42}\, a_{43}\, a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Nous procédons en deux étapes :

1. **Elimination de Gauss**
   - **a)** Commençons par diviser par le pivot a_{11} tous les éléments de la 1ere équation, qui prendra alors la forme suivante :

$$x_1 + a^1_{12}x_2 + a^1_{13}x_3 + a^1_{14}x_4 = y^1_1$$

$$\text{Où: } a^1_{1j} = a_{1j}/a_{11} \;\; (2 < j \leq 4)$$

$$et \; y^1_1 = a_1/a_{11}$$

Eliminons ensuite, l'inconnue x_1 du reste des équations. Pour cela, il suffira de soustraire de la 2eme équation le produit par a_{21}de la 1ere équation sous sa dernière forme, soustraire de la 3eme équation le produit de la 1ere par a_{31}et enfin soustraire de la 4eme équation le produit de la 1ere par a_{41}. Le système devient :

$$\begin{aligned} x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + a_{14}^1 x_4 &= y_1^1 \\ 0 + a_{22}^1 x_2 + a_{23}^1 x_3 + a_{24}^1 x_4 &= y_2^1 \\ 0 + a_{32}^1 x_2 + a_{33}^1 x_3 + a_{34}^1 x_4 &= y_3^1 \\ 0 + a_{42}^1 x_2 + a_{43}^1 x_3 + a_{44}^1 x_4 &= y_4^1 \end{aligned}$$

$$\textbf{Où: } a_{ij}^1 = a_{ij} - a_{i1} \times a_{1j}^1 \ (2 < i \leq 4 \ et \ 2 < j \leq 4)$$

$$et \ y_1^1 = y_i - a_{i1} \times y_1^1 \ (2 < i \leq 4)$$

Et sous forme matricielle :

$$\begin{bmatrix} a_{11}^1 & a_{12}^1 & a_{13}^1 & a_{14}^1 \\ 0 & a_{22}^1 & a_{23}^1 & a_{24}^1 \\ 0 & a_{32}^1 & a_{33}^1 & a_{34}^1 \\ 0 & a_{42}^1 & a_{43}^1 & a_{44}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{bmatrix}$$

b) Divisons les éléments de la 2eme équation par le pivot a_{22}^1, celle-ci prend la forme suivante:

$$x_2 + a_{23}^2 x_3 + a_{24}^2 x_4 = y_2^2$$

$$\textbf{Où: } a_{2j}^2 = a_{2j}^1 / a_{22}^1 \ (2 < j \leq 4)$$

$$et \ y_2^2 = y_2^1 / a_{22}^1$$

Procédons a l'élimination de l'inconnue $x_2$des 3eme et 4eme équations de la même façon que pour x_1. On soustrait de la 3eme équation le produit de la 2eme équation par a_{32}^1 et de la 4eme le produit de la 2eme par a_{42}^1, pour aboutir au système suivant :

$$\begin{aligned} x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + a_{14}^1 x_4 &= y_1^1 \\ 0 + x_2 + a_{23}^2 x_3 + a_{24}^2 x_4 &= y_2^2 \\ 0 + 0 + a_{33}^2 x_3 + a_{34}^2 x_4 &= y_3^2 \\ 0 + 0 + a_{43}^2 x_3 + a_{44}^2 x_4 &= y_4^2 \end{aligned}$$

$$\textbf{Où: } a_{ij}^2 = a_{ij}^1 - a_{i2}^1 \times a_{2j}^2 \ (3 < i \leq 4 \ et \ 3 < j \leq 4)$$

$$et \ y_i^2 = y_i^1 - a_{i2}^1 \times y_2^2 \ (3 < i \leq 4)$$

Et sous forme matricielle :

$$\begin{bmatrix} 1 & a_{12}^1 & a_{13}^1 & a_{14}^1 \\ 0 & 1 & a_{23}^2 & a_{24}^2 \\ 0 & 0 & a_{33}^2 & a_{34}^2 \\ 0 & 0 & a_{43}^2 & a_{44}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1^1 \\ y_2^2 \\ y_3^2 \\ y_4^2 \end{bmatrix}$$

c) Divisons les éléments de la 3eme équation par le pivot a_{33}^2, nous obtenons :

$$x_3 + a_{34}^3 x_4 = y_3^3$$

Où: $a_{3j}^3 = a_{3j}^2 / a_{33}^2 \ (j = 4)$

$$et\ y_3^3 = y_3^2 / a_{33}^2$$

Et éliminons $x_3$de la 4eme équation de la même façon que pour x_1 et x_2. Elle devient :

$$\begin{aligned} x_1 + a_{12}^1 x_2 + a_{13}^1 x_3 + a_{14}^1 x_4 &= y_1^1 \\ 0 + x_2 + a_{23}^2 x_3 + a_{24}^2 x_4 &= y_2^2 \\ 0 + 0 + x_3 + a_{34}^3 x_4 &= y_3^3 \\ 0 + 0 + 0 + a_{44}^3 x_4 &= y_4^3 \end{aligned}$$

Où: $a_{ij}^3 = a_{ij}^2 - a_{i3}^2 \times a_{3j}^3 \ (i = 4\ et\ j = 4)$

$$et\ y_i^3 = y_i^2 - a_{i3}^2 \times y_2^3 \ (i = 4)$$

Et sous forme matricielle :

$$\begin{bmatrix} 1 & a_{12}^1 & a_{13}^1 & a_{14}^1 \\ 0 & 1 & a_{23}^2 & a_{24}^2 \\ 0 & 0 & 1 & a_{34}^3 \\ 0 & 0 & 0 & a_{44}^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1^1 \\ y_2^2 \\ y_3^3 \\ y_4^3 \end{bmatrix}$$

d) Divisons les éléments de la 4eme équation par le pivot a_{44}^3, nous obtenons :

$$x_4 = y_4^4$$

Où: $y_4^4 = y_4^3 / a_{44}^3$

Il en résulte finalement la forme matricielle suivante :

$$\begin{bmatrix} 1 & a_{12}^1 & a_{13}^1 & a_{14}^1 \\ 0 & 1 & a_{23}^2 & a_{24}^2 \\ 0 & 0 & 1 & a_{34}^3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1^1 \\ y_2^2 \\ y_3^3 \\ y_4^4 \end{bmatrix}$$

2. Résolution par substitution arrière

Du système ainsi transforme, nous pouvons déduire les inconnues $x_i$par substitution arrière :

$$x_4 = y_4^4$$

$$x_3 = y_3^3 - \mathrm{a}_{34}^3 x_4$$

$$x_2 = y_2^2 - \mathrm{a}_{23}^2 x_3 - \mathrm{a}_{24}^2 x_4$$

$$x_1 = y_1^1 - \mathrm{a}_{12}^1 x_2 - \mathrm{a}_{13}^1 x_3 - \mathrm{a}_{14}^1 x_4$$

Remarque : la méthode ne peut fonctionner que si les pivots a_{ii} ne s'annulent pas pendant le processus d'élimination. La division des éléments a_{ij} par le pivot serait dans le cas contraire infinie.

Exemple de résolution :

Considérons le système suivant :

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_1 + 3x_2 - 2x_3 = -1 & L_2 \\ 3x_1 + 5x_2 + 8x_3 = 8 & L_3 \end{cases}$$

$$\text{Avec} \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & -2 \\ 3 & 5 & 8 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, y = \begin{pmatrix} 2 \\ -1 \\ 8 \end{pmatrix}$$

Première étape du pivot pour éliminer la variable x_1 des lignes L_2 et L_3 :

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & (L_{1)} \\ x_2 - 4x_3 = -3 & (L_2 \leftarrow L_2 - L_1) \\ -x_2 + 2x_3 = +2 & (L_3 \leftarrow L_3 - 3L_1) \end{cases}$$

Deuxième étape du pivot pour éliminer la variable x_2 de la ligne L_3 :

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_2 - 4x_3 = -3 & L_2 \\ -2x_3 = -1 & L_3 \leftarrow L_3 - (-L_2) \end{cases}$$

En remontant dans le système, on obtient aisément la solution x du système :

$$x = \begin{pmatrix} 3 \\ -1 \\ 1/2 \end{pmatrix}$$

Algorithme Gauss

Données : n, a_{ij}, y_i $(0 <= i < n, 0 <= j < n)$

$$Pour\ (0 \le l < n) \begin{cases} d_0 = 1/a_{ll} \\ pour\ (l \le j < n) \quad a_{lj} = a_{lj} \times d_0 \\ y_l = y_l \times d_0 \\ pour\ (l+1 \le i < n) \begin{cases} d_1 = a_{il} \\ pour\ (l \le j < n) \quad a_{ij} = a_{ij} - d_1 \times a_{lj} \\ y_i = y_i - d_1 \times y_l \end{cases} \end{cases}$$

$$Pour(\text{n} > \text{i} >= 0) \qquad x_i = y_i - \sum_{j=n-1}^{j<i} a_{ij} \times x_j$$

Exercice d'application :

Résoudre le système d'équations suivant à la main, en utilisant la méthode de GAUSS.

$$\begin{bmatrix} 2 & 3 & 8 \\ 2 & 1 & 4 \\ 6 & 4 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ 1 \end{Bmatrix}$$

1ere étape : Triangularistaion de la matrice $[a_{ij}]$

- **Faisons apparaitre un 1 a la place de a_{11}, ensuite des 0 dans la 1ere colonne sous a_{11}**

$$\frac{l_1}{(-2)} \rightarrow \begin{bmatrix} 1 & \frac{3}{2} & 4 \\ 2 & 1 & 4 \\ 6 & 4 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 1 \\ 1 \end{Bmatrix} \quad l_2 - 2l_1 et\ l_3 - 6l_1 \rightarrow \begin{bmatrix} 1 & 3/2 & 4 \\ 0 & -2 & -4 \\ 0 & -5 & -22 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1/2 \\ 0 \\ -2 \end{Bmatrix}$$

- **Faisons apparaitre un 1 a la place de a_{22}, ensuite des 0 dans la 1ere colonne sous a_{22}**

$$\frac{l_2}{2} \rightarrow \begin{bmatrix} 1 & \frac{3}{2} & 4 \\ 0 & 1 & 2 \\ 0 & -5 & -22 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 0 \\ -2 \end{Bmatrix} \qquad l_3 + 5l_2 \rightarrow \begin{bmatrix} 1 & 3/2 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & -12 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1/2 \\ 0 \\ -2 \end{Bmatrix}$$

- **Faisons apparaitre un 1 a la place de a_{33}**

$$\frac{l_3}{-12} \rightarrow \begin{bmatrix} 1 & \frac{3}{2} & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} \frac{1}{2} \\ 0 \\ 1/6 \end{Bmatrix}$$

2eme étape : Résolution par substitution arrière

$0x_1 + 0x_2 + 1\,x_3 = \frac{1}{6} \quad \rightarrow\ x_3 = \frac{1}{6}$

$0x_1 + 1\,x_2 + 2\,x_3 = 0 \quad \rightarrow x_2 = -1/3$

$1\,x_1 + \frac{3x_2}{2} + 4\,x_3 = \frac{1}{2} \quad \rightarrow\ x_1 = 1/3$

6. Conclusion

Pour montrer les potentielles utilisations de MATLAB dans les mathématiques, nous avons présenté dans ce chapitre plusieurs applications sur des problèmes réels. Nous avons commencé par la résolution des polynômes et nous avons présenté comment représenter un polynôme sous MATLAB, les différente fonctions prédéfinies permettant de calculer les racines, et d'évaluer un polynôme, comment calculer la dérivée d'un polynôme et comment tracer graphiquement les racines d'un polynôme. Nous avons vu aussi comment déterminer un polynôme depuis ses racines.

De l'autre côté, nous avons présenté la résolution de certains systèmes particuliers sous MATLAB, à savoir : les systèmes à matrice diagonale, les systèmes à matrice triangulaire inférieure et les systèmes à matrice triangulaire supérieure. Aussi, nous avons présenté une des méthodes de résolution des équations non linéaires qui est la méthode de dichotomie. Et enfin, nous avons présenté la méthode directe de Gauss, une méthode de résolution de systèmes d'équations linéaires.

Séries de TP

TP1- Premiers pas avec MATLAB

Objectif : *L'objectif de ce premier TP est de se familiariser avec l'interface graphique de MATLAB et d'apprendre comment définir les expressions de calcul et comment utiliser certaines fonctions prédéfinies sous la ligne de commande (command window) de MATLAB.*

Partie 1. L'interface graphique de MATLAB

- La méthode la plus intuitive pour lancer MATLAB c'est de cliquer sur le raccourcis ou d'aller au menu Démarrer/ MATLAB 7.0. La fenêtre suivante sera affichée :

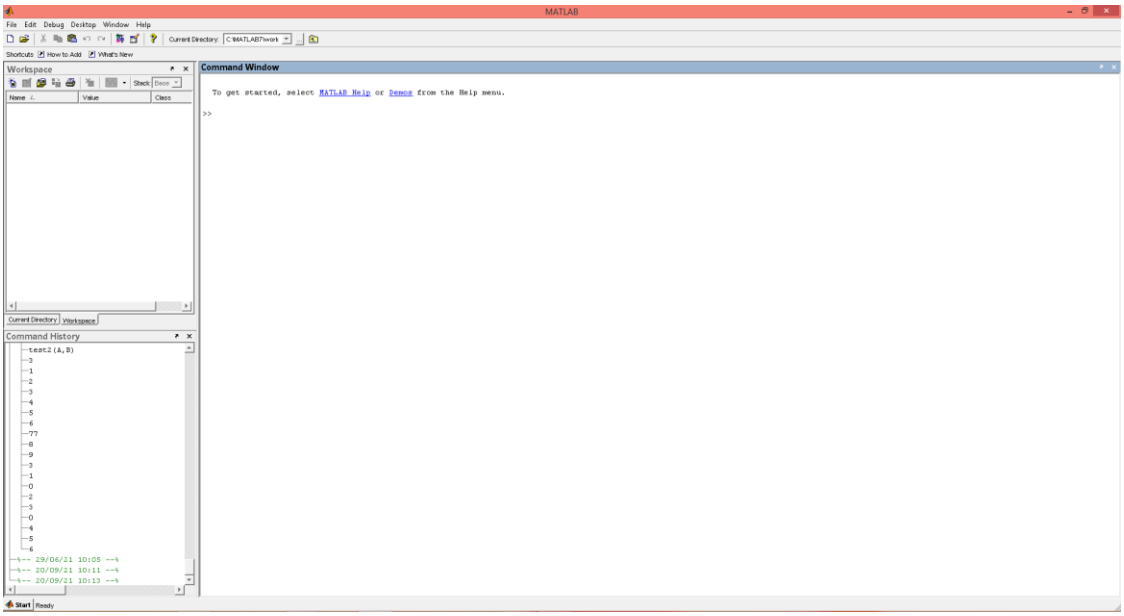


Figure 1. Fenêtre principale de MATLAB 7.0

D'après la fenêtre principale de MATLAB dans la **Figure 1**, Il y a quatre parties principales qui s'affichent par défaut (aller sur le menu *Desktop Layout,* si vous voulez ajouter ou supprimer certaines parties…):

Questions :

1. Observez cette fenêtre principale et citez les quatre fenêtres qui la composent.
2. A quoi sert chaque fenêtre de cette interface ?
3. Laquelle de ces parties vous permet d'écrire des codes MATLAB ?
4. Utiliser les commandes vues en cours pour connaitre le répertoire courant de travail ? est-il possible de modifier ce répertoire ? avec quelle commande ?

Partie 2. Se familiariser avec la Command Window de MATLAB

1. **Operations arithmétiques sur les scalaires :** Essayer les opérations arithmétiques suivantes dans la *command window* et noter ce qu'elles affichent. L'affectation se fait avec l'opérateur **=** sous MATLAB:

```
>>x = 2; y = 1.5 ;
>>somme = x+y
>>difference = x-y
>>produit = x*y
>>division = x/y
>>x^2+ 1
```

- La variable **ans** n'a pas été déclarée mais elle stocke le résultat de x^2+1. A quoi sert la variable **ans**?

- Quelle est l'effet du ; dans la première ligne de commande et le reste des commandes ?

2. **Types de variables :**

```
>>a=5.8
>>b=9+i
>>c= 'hello'
>>tr1=true(1==1)
>>tr2=logical(1)
>>z=int8(2)
```

- Quel est le type de chacune des variables ? Est-il important de déclarer le type d'une variable sous MATLAB ? **(voir également la fenêtre workspace, onglet class)**
- Tester les fonctions ***ischar, islogical, isreal*** sur les différents variables ? À quoi servent ces fonctions ? (**utiliser le help en cas de nécessité**)

3. **Commandes de base** : Evaluez les quantités y et z dans MATLAB. Différencier entre **who** et **whos,** entre **clear** et **clear all,** entre **xx** et **XX**:

```
>>xx=8;
>>y=x+10;
>>z=y-2;
>>xx
>>XX
>>clc
>>clear x;
>>who
>>whos
>>clear all
```

4. **Priorité des opérateurs:** les priorités sont comme suit: (), ^ et ', * et /, + et –

   Evaluer les expressions suivantes :

```
>>3 + 2 * 4 ^ 2
>> ((3 + 2) * 4) ^ 2
```

>> $\frac{123+456}{123+456}$ est-ce que ça donne le même résultat que : (123+456)/(123+456). Qu'est-ce que vous conclure ?

>>$\frac{1}{12-6^2}+\frac{2}{3}$. Mettez les parenthèses dans l'expression pour donner le même résultat.

>> $\frac{1+(\frac{2}{3})^{\frac{1}{2}}}{12-6^2}$, $\frac{(1+\left(\frac{2}{3}\right)^{\frac{1}{2}})}{(12-6^2)}$ Est-ce que le résultat est le même dans les deux expressions ?

```
>> x=20 ; y=30 ;
>>x<y
>> y<x
>> ~(x<y)
>> (x==y)
>> (x<11)&(y==3)
```

5. **Fonctions arithmétiques** : Donner le résultat des commandes suivantes. Elles font quoi ces fonctions ?

```
>> x=35;
>> y=5;
>> rem(x,y)  (Remainder after division, rem(x,y) ↔ mod(x,y))
>> lcm(x,y)  (Least common multiple of x and y)
```

```
>> gcd(x,y)   (Greatest common divisor of x and y)
>> factor(x)  (renvoie un vecteur contenant les facteurs premiers de x)
```

6. **Fonctions d'arrondis et format d'affichage**: Donner le résultat des commandes suivantes. Déduire ce que font ces fonctions en utilisant la commande **help**:

```
>>t=20.34;
>>p=18.32;
>>format short ;
>> p+t
>>floor(t)
>>ceil(p)
>>p=18.50;
>>round(p)
>>format rat ;
>>pi/6
```

7. **Fonctions trigonométriques, puissance, logarithme**: Essayer et noter le résultat des fonctions suivantes. Que font ces fonctions ? s'il y a des erreurs corrigé les…

```
>> x = 2; y = π;
>>sin(x)
>>COS(y)
>>e^x
>>n=3;
>>x^n
>>√x
>>|x|
>>Sign(x)
```

8. **Fonctions sur les nombres complexes**: Trouvez la partie réelle et imaginaire des nombres complexes suivants en utilisant les fonctions: **real** et **imag** respectivement. Utiliser **help real** et **help imag** pour rappeler la syntaxe de ces fonctions.

```
>> e^(i(3π+4))
>>1/(1+i)
>>ln(−1)
```

<u>Exercice 1:</u> écriture de fonctions

Soit la fonction à deux variables suivante :

$$f(x,y) = \frac{1}{\sqrt{|x^3| + |y^3|}}$$

1. Calculer : $f(-2,\ -3)$, $f(-5,\ 2.25)$, $f(0.25,3.05)$

<u>Exercice 2:</u> écriture d'expressions

A. Donner les commandes MATLAB pour évaluer les expressions suivantes :

1. $logarithme_{10}(2)$
2. $\frac{1}{1+\frac{1}{1+\frac{1}{2}}}$
3. $-x^6 - \frac{5}{7}x^3 + x^2 + 5$, pour $x = 1, x = 4$

4. $\frac{1}{\sqrt{8^3+2}} - \frac{2\sin(45)}{e^2} + \ln(4)$ Les angles sont donnés en degré.

5. $\frac{x^3 \sin(\frac{4\pi}{2})^2}{\cos(2\pi-1)}$, pour $x = e^3$

6. $-2\,ln(5x) + \sqrt{4x^3+1}$, pour $x = -3i$

7. $\frac{4}{3}\pi R^3$ où $R = 3cm$

8. $z \leftarrow \frac{|2n^5-3|}{\sqrt{4n^2+\ln(6n)}}$
9. $x \leftarrow \frac{e^{\sqrt{x}}}{2y-1} + |x| - \frac{1}{y^2+3}$
10. $w \leftarrow \frac{b}{2} \times \sqrt{c^2 - \left(\frac{b}{2,5}\right)^2}$;
11. $y \leftarrow e^{2-\sqrt{b^3-\frac{1}{a}}}$;

B. Donner les expressions mathématiques équivalentes aux instructions MATLAB suivantes :
1. $\exp(sqrt(x))/(2 * y - 1) + abs(x) - 1\ /(y\ \hat{}\ 2 + 3)$;
2. $2 * sind(45)/\exp(2)$;

En conclusion:

- Sous MATLAB le type des variables n'a pas besoin d'être spécifié, MATLAB infère le type d'une variable en fonction de la donnée que l'on y stocke.
- MATLAB est sensible à la casse : la variable EXO1 est différente de la variable Exo1 et exo1...
- Les commandes : **clc** pour nettoyer l'écran, **clear variable** pour supprimer la variable indiquée, **clear all, clear** pour supprimer toutes les variables créées auparavant, **who** pour afficher la liste des variables dans le *Workspace* et **whos** pour afficher des informations des variables dans le *Workspace*.

- Une liste non exhaustive de fonctions prédéfinies incorporées dans Matlab a été donnée en cours, il faut l'apprendre pour une utilisation ultérieure (pour un examen ou une validation TP par exemple !!)

TP2- Les vecteurs sous MATLAB

Objectif : *L'objectif de ce TP c'est d'apprendre à : définir, manipuler et opérer sur les vecteurs sous MATLAB.*

Exercice 1:

1. Soit **k** un vecteur défini de 3 à 5 avec un pas de 2, **N** un vecteur défini de 1 à 2 avec un pas de 1 et le vecteur **F** qui est la concaténation horizontale de **k** et **N**. Définir les vecteurs **K**, **N** et **F** sous MATLAB et afficher les résultats obtenus;
2. Définir les vecteurs t (vecteur-ligne) et v (vecteur-colonne) suivant :

$$t = (0\ 3\ 6\ 9\ 12\ 15\ 18),\ v = \begin{pmatrix} 1 \\ 3 \\ 7 \\ 9 \\ 5 \end{pmatrix}$$

3. Modifier les valeurs des éléments dans les indices : 5 et 7 par la valeur -3 du vecteur t.

$$t = (0\ 3\ 6\ 9 - 3\ 15 - 3);$$

4. Modifier les valeurs des éléments dans les indices : 1 à 3 par les valeurs 3, 5 et 9 respectivement, puis les éléments d'indices 4 et 5 par 8 et 10 du vecteur v;

$$v = \begin{pmatrix} 4 \\ 8 \\ 0 \\ 5 \\ 7 \end{pmatrix}$$

5. Soit $v1 = (1\ 2\ 3\ 4)$, $v2 = (7\ 6\ 0)$ deux vecteurs-lignes. Créer un vecteur-ligne **v3** de taille 10 en concaténant le vecteur **v1** de taille 4 puis deux vecteurs de taille 3 qui correspondent au vecteur **v2 ;**

$$v3 = (1\ 2\ 3\ 4\ 7\ 6\ 0\ 7\ 6\ 0)$$

6. Soit x et y des vecteurs-colonnes $x = \begin{pmatrix} 3 \\ 5 \\ 6 \end{pmatrix}$, $y = \begin{pmatrix} 9 \\ 6 \end{pmatrix}$. Définir z qui est une concaténation de x et y

$$z = \begin{pmatrix} 3 \\ 5 \\ 6 \\ 9 \\ 6 \end{pmatrix}$$

   - Insérer la valeur 2 en première position du vecteur z, la valeur 20 à la dernière position du vecteur et la valeur -7 en 3ème position de z ;
   - Supprimer les éléments dans les positions 2 et 4 ;
   - Ordonner les éléments du vecteur z dans l'ordre croissant ;

Exercice 2:

1. Construire les vecteurs suivants sous MATLAB:

$$u1 = (3\ 6\ 9 \ldots 27\ 30)$$
$$u2 = (-\pi/2\ \ -\pi/4\ \ 0\ \ \pi/4\ \ \pi/2\ \ \ldots 11\pi/4\ \ 3\pi)$$
$$u3 = (0\ 1\ 49 \ldots 81\ 100)$$

Exercice 3:

1. Proposer les instructions MATLAB qui permettent de définir les trois vecteurs suivants: $v1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10)$, $v2 = (-1.5, 0, 1.5, \dots, 4.5, 6)$, $v3 = (1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \dots, \frac{1}{81}, \frac{1}{100})$
2. Créer un vecteur **v** qui contient tous les éléments de v1, v2, v3 consécutivement ;
3. Afficher les éléments de v de la 11^{eme} position jusqu'à la 5^{eme} position;
4. Donner la commande MATLAB permettant d'afficher le deuxième tiers du vecteur v.

Exercice 4: opérations entre vecteurs

Soit $u1 = \begin{pmatrix}1\\2\\3\end{pmatrix}, u2 = \begin{pmatrix}-5\\2\\1\end{pmatrix}, u3 = \begin{pmatrix}-1\\-3\\7\end{pmatrix}$,

1. Définir les vecteurs u1, u2 et u3 sous MATLAB ;
2. Réaliser les opérations suivantes toute en déduisant les opérations permises :
   - $15 * u1$;
   - $u1 + 3$;
   - $u2/3$;
   - $u1 + u2$ et $u1 - u2$;
   - $u1 * u2$ et $u1.* u2$ et $u1' * u2$;
   - $u1/u2$ et $u1./u2$;
   - $u1\backslash u2$ et $u1.\backslash u2$;
   - $u1\text{^}7$ et $u1.\text{^}7$

   a. Quelle différence existe entre les opérateurs : * .* / \ ./ .\ ^ .^

   b. Citer les opérations permises entre les vecteurs

Exercice 4: fonctions prédéfinies sur les vecteurs

Soit le vecteur-ligne $v = (1\ -2\ 12\ 0\ 15\ 23\ 50\ 1\ 8)$.

1. Calculer et afficher la somme, la moyenne, le produit des éléments de v;
2. Calculer et afficher le minimum et le maximum dans v ;
3. Renverser le vecteur v (afficher les éléments du vecteur dans l'ordre inverse);
4. Chercher et afficher les valeurs supérieures à 10 dans v.

TP3- Manipulation des Matrices sous MATLAB

Objectif : *L'objectif de ce TP c'est d'apprendre à : définir, manipuler et opérer sur les matrices (vecteurs à 2 dimensions).*

Exercice 1: Manipuler une matrice

Donner le résultat de chacune des instructions MATLAB suivantes :

$\gg C = diag(diag(ones(3)))$

$\gg k = [1:1:5]$

$\gg D = \big(ones(5) + diag(k)\big) + 3 * zeros(5)$

Exercice 2: Manipuler une matrice

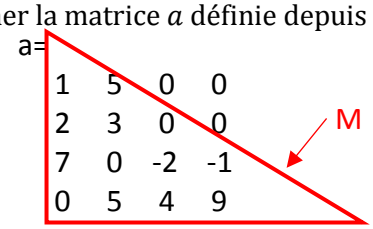
Soit les trois matrices: A, B et C définies comme suit :

$$A = \begin{pmatrix} 1 & 5 \\ 2 & 3 \end{pmatrix}, B = \begin{pmatrix} 7 & -2 \\ 0 & 5 \end{pmatrix} \text{ et } C = \begin{pmatrix} -1 & 5 \\ 0 & 3 \\ -2 & -1 \\ 4 & 9 \end{pmatrix}$$

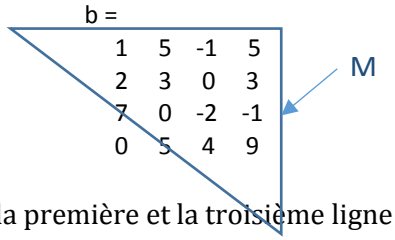
1. Donner le résultat des expressions suivantes :
   - $A \times B - 3$
   - $C \times B + 1 + zeros(4,2)$
   - $C(end:-1:1,2).\backslash 24$
2. Créer la matrice M de dimension 4×4 qui se définie par les matrices A, B et C comme suit :

$$M = \begin{pmatrix} \text{A} & \text{C} \\ \text{B} & \end{pmatrix}$$

3. Remplacer la valeur de l'élément dans la troisième ligne, deuxième colonne par la valeur 0 ;
4. Calculer et afficher la matrice a définie depuis la matrice M comme suit :



5. Calculer et afficher la matrice b définie depuis la matrice M comme suit :



6. Supprimer la première et la troisième ligne de la matrice M;
7. Supprimer la dernière colonne de M;
8. Ajouter comme dernière ligne dans la matrice M un vecteur ligne x qui a le même nombre de colonnes de *M.* Confirmer par la fonction **size** que le nombre de colonnes de x est égale au nombre de colonnes de M ;
9. Remplacer la troisième colonne de M par un vecteur colonne $v = \begin{pmatrix} 1 \\ 9 \\ 11 \end{pmatrix}$.

Exercice 3: matrices particulières

1. Définissez les matrices A, B, C et D, où : A est une matrice carrée d'ordre 3 initialisée aléatoirement, B est une matrice 3×2 avec des valeurs toutes nulles, C est une matrice de dimension 2×3 avec des

valeurs toutes égales à 1 et D est une matrice carrée d'ordre 3 avec les valeurs toutes nulles sauf la diagonale égale à 1.

$$A = \begin{pmatrix} 0{,}95 & 0.48 & 0{,}45 \\ 0{,}23 & 0{,}89 & 0{,}01 \\ 0{,}60 & 0{,}76 & 0{,}82 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, C = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \text{ et } D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Donner une nomination adéquate pour chacune de ces matrices.
- Soit x un vecteur colonne qui a pour valeurs (3 5 7). Remplacer la diagonale de la matrice D par le vecteur x. Comme résultat vous aurez: $D = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 7 \end{pmatrix}$
- Créer une matrice magique M de dimension 4×4. *Une matrice magique c'est une matrice où la somme des éléments de chaque colonne et la somme des éléments de chaque ligne et ceux des deux diagonales sont identiques.*

<u>Exercice 4:</u> opérations matricielles

1. Définissez une matrice M = [1 2 ; 3 4] puis essayez les opérations suivantes dans l'interpréteur :

```
>> 2 * M + 3
>> M ^ 3  (M *M *M)
>> M .^ 3
>> M + M
>> M - M
>> M * M   et  M .* M
>> M / M   et  M ./ M
```

2. Quelle est la différence entre les opérateurs * et .* , ^ et .^, / et ./? quelle est le résultat et la signification de chacune des commandes ?
3. Créer facilement une matrice 54 × 42 ne contenant que des 7 ?
4. En utilisant les fonctions prédéfinies de MATLAB, calculer : la matrice transposée de M (M'), l'inverse de M (M^{-1}), le déterminant de M, les valeurs et les vecteurs propres de M.

En conclusion:

- Quelques instructions utiles pour manipuler une matrice :

$size(A)$	*Nombre de lignes et de colonnes de A*
$A(i,j)$	*Coefficient d'ordre i, j de A*
$A(i1 : i2\ ,\ :)$	*Lignes i1 à i2 de A*
$A(i1 : i2\ ,\ :) = [\ ]$	*Supprimer les lignes i1 à i2 de A*
$A(:, j1 :\ j2)$	*Colonnes j1 à j2 de A*
$A(:, j1 :\ j2) = [\ ]$	*Supprimer les colonnes j1 à j2 de A*
$A(:)$	*Concaténation des vecteurs colonnes de A*
$diag(A)$	*Coefficients diagonaux de A*

- Quelques matrices particulières :

$zeros(m,n)$	*Matrice nulle de taille m, n*
$ones(m,n)$	*Matrice de taille m, n dont tous les coefficients valent 1*
$eye(n)$	*Matrice identité de taille n*
$diag(x)$	*Matrice diagonale dont la diagonale est le vecteur x*
$magic(n)$	*Carré magique de taille n*
$rand(m,n)$	*Matrice de taille m, n à coefficients respectant la loi uniforme sur [0, 1]*

- Quelques opérations matricielles :

A'	*Transposée de A*
$inv(A)$	*Inverse de A*
$det(A)$	*Déterminant de A*
$eig(A)$	*Valeurs propres de A*
$[U, D] = eig(A)$	*Vecteurs et valeurs propres de A*
$+\ -$	*Addition, soustraction matriciels*
$*\ \hat{}$	*Produit et puissance matriciels*
$.*\ .\hat{}$	*Produit et puissance terme à terme*
$./$	*Division terme à terme*

TP4- Les Scripts et les fonctions MATLAB

Scripts MATLAB: *Afin de pouvoir réutiliser les lignes de calcul, il est utile de les mettre dans un script. Un script est un fichier texte que MATLAB pourra lire et exécuter. Pour y accéder : (1) Ouvrez l'éditeur de scripts de MATLAB soit en cliquant sur la page blanche de la barre d'outils, soit en allant dans le menu "File →New →M-file".*

Exemple pour démarrer:

10. Créez le script suivant :

*% Ceci est un script MATLAB, % le signe "**pourcen**t" permet de mettre des commentaires qui ne seront pas interprètes*	
disp('Salut') ;	***disp** permet d'afficher ce que l'on veut à l'écran, les ' permettent d'indiquer que l'on veut afficher du texte.*
a = input('entrez a : ') ;	***input** demande à l'utilisateur d'entrer une valeur au clavier*
b = 6 ;	***=** assure une affectation de la valeur 6 à la variable b*
$b = b + a$;	*Affectation du résultat de l'expression $b + a$ à b*
a, b	*la **virgule** permet de mettre plusieurs commandes sur une seule ligne, elle a le même rôle que la touche entrée.*

11. Enregistrez le fichier et exécuter le soit : (1) en cliquant sur le bouton *run* de la barre d'outils, (2) appelez-le dans la *Command Window* par son nom de sauvegarde.

Exercice 1: instructions conditionnelles

1. Écrivez le script MATLAB qui demande deux valeurs x et y à l'utilisateur et qui les affiche, qui échange leurs contenus et qui les affiche à nouveau.
2. Écrivez le script MATLAB qui demande un nombre, puis affiche son signe (positif, négatif ou nul) (fait en cours).

Exercice 2: instructions répétitives (boucles)

1. Ecrire un programme MATLAB qui génère aléatoirement un code numérique et permet un nombre illimité d'essais. Un code numérique est un nombre entier $\in [0,999]$, qui sera fixé au début du programme. Si le code lu est faux, le programme affiche CODE FAUX et redemande un nouveau code. Si le code est bon, le programme affiche CODE BON puis s'arrête.

2. Écrivez un programme qui demande deux entiers **a** et **b** et qui affiche le résultat de la somme suivante : $\sum_{k=1}^{b} k^a$

3. Écrivez un programme MATLAB qui prend en entrée un réel $x \in]0,20[$, donne un message d'erreur si $x \notin]0,20[$, sinon calcule et donne en sortie le plus petit n tel que la somme $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ est supérieure a x (fait en cours).

4. Écrivez un programme qui calcule le $10^{\text{ème}}$ terme de la suite de Fibonacci (fait en cours):

$$u_0 = 0, u_1 = 1, u_{n+2} = u_{n+1} + u_n$$

5. Écrivez le script MATLAB qui prend en entrée deux matrices A et B, vérifie si leur taille est compatible. Si elles ne le sont pas, donne un message d'erreur et pas de sortie, si elles le sont, calcule le produit $A \times B$ des deux matrices, puis affiche le résultat a la matrice C.

NB : A est de taille $p \times n$ et B est de taille $n \times q$, alors la matrice C est de taille $p \times q$ et a pour terme général :

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

Fonctions MATLAB: *Il existe de nombreuses fonctions prédéfinies en MATLAB, mais il arrivera forcément un moment où vous voulezutiliser une fonction qui n'est pas définie. Heureusement, il est possible de définir ses propres fonctions et de s'en servir exactement comme les fonctions préexistantes.*

Exemple pour démarrer:

Supposons que l'on veuille définir la fonction **cos2** définie par :

$$cos2: \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \to cos^2(x) \end{cases}$$

Il faut alors créer un nouveau fichier de script (il faut un fichier séparé pour chacune des fonctions que l'on définit), puis écrire : $\begin{bmatrix} function\ r = cos2(x) \\ \quad r = \cos(x)\text{^}2 \end{bmatrix}$

Sauvegarder le script sous le nom **cos2.m**. Une fois que le script est sauvegardé, il est possible d'appeler la fonction directement dans la *Command Window* par son nom ou depuis un autre script exécutable.

Exercice 1:

1. Définissez une fonction test(x) en recopiant le code suivant dans un script nommé ***test.m***:

```
function reponse = test(x)
if x == 42
   reponse = 'C''est vrai !' ;
elseif x == 41
   reponse = 'Presque...' ;
else
   reponse = 'C''est faux !' ;
end
```

2. Le **double symbole égal** et la **double apostrophe** dans le script sont tous les deux importants... À votre avis à quoi servent-ils ?
3. Que renvoie la fonction si on l'appelle avec l'argument 12 ? Et avec 42 ? Et avec 41 ?

Exercice 2:

1. Écrivez une fonction *pair* qui peut indiquer si un nombre entier x est pair ou non ;
2. Écrivez la $fonction\ somme$ qui calcule la somme de deux matrices A et B.
3. Écrivez la $fonction\ produit$ qui calcule le produit de deux matrices A et B.

Exercice 3:

Soit la fonction *calcul* a deux boucles imbriquées suivante :

```
function M=calcul(M)
[n,m]=size(M);
for i=1:n
   v=M(i,:);
   for j=1:m
      M(i,j)=v(m-j+1)
   end
end
```

1. Donner la valeur de B après l'exécution des instructions suivantes :

   >> A=[1 2 3 4;5 6 7 8 ;9 10 11 12];

   >> B=calcul(A)

2. Déduire ce que fait cette fonction.
3. Réécrire la fonction *calcul* pour obtenir le même résultat en utilisant une seule boucle.

Utilisation des fonctions et scripts ensemble: *dans le cas de programmes complexes et difficile à résoudre d'utiliser des fonctions avec le programme MATLAB.*

Exercice 1:

1. Ecrire une fonction MATLAB *produit* qui prend comme argument deux matrices carrées de même dimensions **A** et **B** et calcule le produit $A \times B$, puis affiche le résultat a la matrice C. La matrice C a pour terme général :

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

2. Écrivez le script MATLAB qui permet de saisir deux matrices carrées de même dimensions **A** et **B**, vérifie si leur taille est compatible. Si elles ne le sont pas, donne un message d'erreur et pas de sortie, si elles le sont, calcule $AB + BA$. Le script fait appel à la fonction *produit*.

NB : On s'interdit d'utiliser les opérations du type $A \times B$ ou $A + B$, on doit plutôt utiliser une affectation élément par élément.

Exercice 2:

1. Ecrire une fonction MATLAB « *fact*» qui prend comme argument un nombre entier positif $\boldsymbol{n}$ et qui retourne $\boldsymbol{n}$! comme réponse.

   Sachant que $n! = \mathbf{1} \times \mathbf{2} \times \mathbf{3} \times \ldots \times \boldsymbol{n}$.

2. Ecrire un script MATLAB qui permet de lire deux nombre $\boldsymbol{n}$ et $\boldsymbol{p}$ et qui calcule et affiche

$$\boldsymbol{n}!/(\boldsymbol{p}! * (\boldsymbol{n} - \boldsymbol{p})!)$$

Exercice 3:

1. Ecrire la fonction *parfait* qui prend comme argument un nombre entier x, et qui retourne 1 si ce nombre est parfait sinon elle retourne 0.

2. Ecrire un *script Matlab* qui permet à l'utilisateur de saisir un nombre entier compris entre 1 et 10, puis teste et affiche si ce nombre est parfait. Le script doit faire appel à la fonction parfait.

NB : un nombre parfait c'est un nombre qui égale à la somme de ses diviseurs hormis lui.

Exemple d'exécution : x=6. La fonction retourne 1. Le script affiche : ce nombre est parfait.

Exercices supplémentaires

Exercice 1:

1. Ecrire une fonction Matlab « *facto »* qui prend comme argument un nombre entier positif $\boldsymbol{n}$ et qui retourne $\boldsymbol{n}$! comme réponse.

   Sachant que $! = 1 \times 2 \times 3 \times \ldots \times n$.

2. Ecrire un script Matlab qui permet de lire deux nombre $\boldsymbol{n}$ et $\boldsymbol{p}$ et qui calcule et affiche

$$X = n!/(n-p)!$$

NB : Le script doit faire appel à la fonction *facto*.

Exemple d'exécution : $n = 4, p = 2$. La fonction retourne $\mathbf{4!} = \mathbf{4 \times 3 \times 2 \times 1 = 24}$. Le script calcule et affiche $\boldsymbol{X = \frac{4!}{2!} = \frac{24}{2} = 12}$.

Exercice 2:

1. Ecrire une Ecrire une fonction Matlab « *premier* » qui prend comme argument un nombre **n** et qui retourne **1** (**true**) si **n** est premier sinon elle retourne **0** (**false**).
2. Ecrire le *script* Matlab qui permet de saisir deux entiers positifs **x** et **y** (x < y) et qui affiche ensuite la liste des nombres premiers compris entre **x** et **y**. Le script doit utiliser la fonction *premier*

Exemple d'exécution : $x = 3, y = 16$. Le script calcule et affiche $3, 5, 7, 11, 13$.

Exercice 3:

1. Ecrire une fonction Matlab « *divisible* » qui prend comme argument deux nombres **n** et **m** et qui retourne 1 (true) si **n** est divisible par **m** et 0 (false) sinon.
2. Ecrire le script Matlab, permettant de saisir trois entiers **n**, **a** et **b**, puis affiche la liste des nombres compris entre **a** et **b** qui sont divisibles par **n**.

Exemple d'exécution : $n = 2, a = 5, b = 25$. Le script affiche $6, 8, 10, 12, 14, 16, 18, 20, 22, 24$

Exercice 4:

1. Ecrire une fonction Matlab « *amicaux* » qui prend comme argument deux nombres entiers positifs x *et* y et qui retourne 1 s'ils sont amicaux, sinon elle retourne 0. Sachant que x *et* y sont amicaux si x est la somme des diviseurs de *y*, hormis *y*, et si y est la somme des diviseurs de *x*, hormis *x*. **Exemple :** 220 et 284 sont amicaux.
2. Ecrire un script Matlab qui permet de lire deux nombre $\boldsymbol{a}$ et $\boldsymbol{b}$ et qui affiche s'ils sont amicaux ou non. Le script doit faire appel à la fonction *amicaux*.

Exemple d'exécution : $a = 220, b = 284$. La fonction retourne **1**. Le script affiche **220** ***et*** **284** ***sont amicaux***.

TP5- Représentation graphique et GUI sous MATLAB

Objectif : *L'objectif de ce TP c'est d'apprendre à représenter et analyser les données graphiquement à l'aide de fenêtres graphique. On apprend aussi a comment construire une interface graphique utilisateur en utilisant le GUIDE MATLAB.*

Exemple pour démarrer:

Tapez le code suivant (voir les Tableaux en bas de la page). Déduire l'intérêt de chacune des fonctions dans l'exemple.

```
x=-pi :0.1 :3*pi ; y=x.*sin(x) ;
plot(x,y)
clf
plot(x,y,x,2*y)
plot(x,y,'r-',x,2*y,'g+')
xlabel('x')
ylabel('y')
title(['graphe de la fonction x sin(x) sur l''intervalle ' num2str(x(1)) ',' num2str(x(end)) ])
```

Exercice 1:

3. Tracer sur la même figure sur l'intervalle $[-\pi, \pi]$ la représentation graphique de la fonction $f(x) = \sin(x)$ et la fonction $g(x) = cos(x)$.
4. Tracer les fonctions précédentes dans deux axes différents sur la même figure.

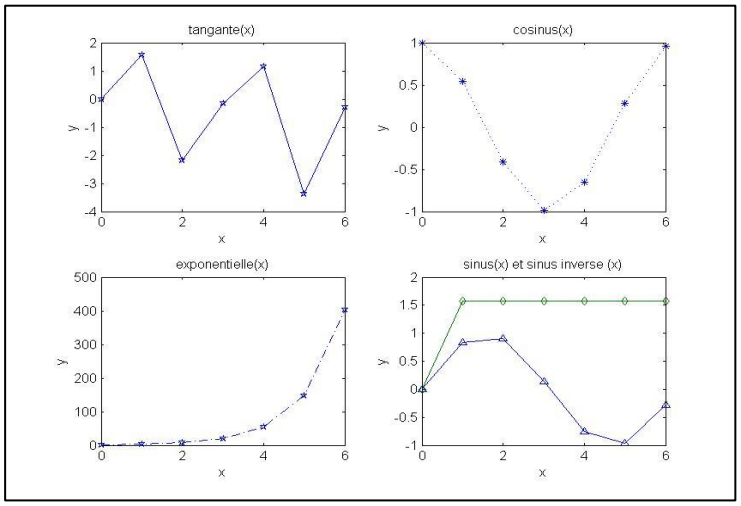
Exercice 2:

Soit les fonctions : $f(x) = \sin(x-2) + 4$ et $p(x) = -2x^3 + x^2 - 3$

1. Tracer la courbe de la fonction $f(x)$ en variant x de 0 à 2π avec un pas de $\pi/12$. Utiliser un style trait pointillé vert avec des points en forme de losange;
2. Tracer la courbe de la fonction $p(x)$ en variant x de -5 à 5 avec un pas de 0,2. Utiliser un tiret bleu avec des points en forme carrés;

Exercice 3:

Écrire l'ensemble des instructions MATLAB permettant de tracer les courbes suivantes dans la même Figure :



Solutions des séries de TP

Solution TP2- Les vecteurs sous MATLAB

Solution Exercice 1:

1.Soit k un vecteur défini de 3 à 5 avec un pas de 2, N un vecteur défini de 1 à 2 avec un pas de 1 et le vecteur F qui est la concaténation horizontale de k et N. Définir les vecteurs K, N et F sous MATLAB et afficher les résultats obtenus;

```
>> k=[3:2:5]
k =
   3   5
>> N=[1:2]
N =
   1   2
>> F=[k,N]
F =
   3   5   1   2
```

2.Définir les vecteurs t (vecteur-ligne) et v (vecteur-colonne)

1ere méthode de définition :

```
>> t=[0 3 6 9 12 15 18]
t =
   0   3   6   9   12   15   18
```

2eme méthode de définition :

```
>> t=0:3:18
t =
   0   3   6   9   12   15   18
```

3eme méthode de définition :

```
>> t=linspace(0,18,7)
t =
   0   3   6   9   12   15   18

>> v=[1;3;7;9;5]
v =
   1
   3
   7
   9
   5
```

3.Modifier les valeurs des éléments dans les indices : 5 et 7 par la valeur -3 du vecteur t.

```
>> t([5,7])=-3
t =
   0   3   6   9   -3   15   -3
```

4.Modifier les valeurs des éléments dans les indices : 1 à 3 par les valeurs 3, 5 et 9 respectivement, puis les éléments d'indices 4 et 5 par 8 et 10 du vecteur v;

```
>> v(1:3)=[3;5;9]
v =
   3
   5
   9
   9
   5
>> v(4:5)=[8;10]
v =
   3
   5
   9
   8
   10
```

Autre méthode pour modifier le tout, c'est comme une nouvelle définition du vecteur:

```
>> v(:)=[3 5 9 8 10]
v =
   3
   5
   9
   8
  10
```

5. Soit $v1 = (1\ 2\ 3\ 4)$, $v2 = (7\ 6\ 0)$ deux vecteurs-lignes. Créer un vecteur-ligne v3 de taille 10 en concaténant le vecteur v1 de taille 4 puis deux vecteurs de taille 3 qui correspond au vecteur v2 ;

```
>> v1=[1 2 3 4]
v1 =
   1    2    3    4
>> v2 =[7 6 0]
v2 =
   7    6    0
>> v3=[v1 v2 v2]
v3 =
   1    2    3    4    7    6    0    7    6    0
```

6. Soit x et y des vecteurs-colonnes

```
>> x=[3;5;6]
x =
   3
   5
   6
>> y=[9;6]
y =
   9
   6
>> z=[x;y]
z =
   3
   5
   6
   9
   6
```

6.1. Insérer la valeur 2 en première position du vecteur z, la valeur 20 à la dernière position la valeur -7 en 3ème position de z ;

```
>> z=[2;z;20]
z =
   2
   3
   5
   6
   9
   6
  20
```

z

6.2. Insérer la valeur -7 en 3ème position de z ;

```
>> z(3)=-7
z =
   2
   3
  -7
   6
   9
   6
  20
```

6.3. Supprimer les éléments dans les positions 2 et 4;

```
>> z([2,4])=[]
z =
   2
  -7
   9
   6
```

```
   20
```

6.4. Ordonner z dans l'ordre croissant

```
>> sort(z)
ans =
   -7
    2
    6
    9
   20
```

Solution Exercice 2:

Ecrire les instructions pour construire les vecteurs suivants :

$$u1 = (3\ 6\ 9 \ldots 27\ 30)$$
$$u2 = (-\pi/2\ \ -\pi/4\ \ 0\ \ \pi/4\ \ \pi/2\ \ \ldots 11\pi/4\ \ 3\pi)$$
$$u3 = (0\ 1\ 49 \ldots 81\ 100)$$

1. Définir u1

```
>> 3:3:30          %première méthode
ans =
   3    6    9   12   15   18   21   24   27   30

>> 3*[1:10]        %deuxième méthode
ans =
   3    6    9   12   15   18   21   24   27   30

>> linspace(3,30,10)  %troisième méthode
ans =
   3    6    9   12   15   18   21   24   27   30
```

2. Définir u2

```
>> -pi/2:pi/4:3*pi          %première méthode
ans =
  Columns 1 through 14
   -1.5708  -0.7854       0   0.7854   1.5708   2.3562   3.1416   3.9270   4.7124   5.4978   6.2832   7.0686   7.8540   8.6394
  Column 15   9.4248
>> pi/4*[-2:12]             %deuxième méthode
ans =
  Columns 1 through 14
   -1.5708  -0.7854       0   0.7854   1.5708   2.3562   3.1416   3.9270   4.7124   5.4978   6.2832   7.0686   7.8540   8.6394
  Column 15
    9.4248

>> linspace(-pi/2,3*pi,15)   % troisième méthode
ans =
  Columns 1 through 14
   -1.5708  -0.7854       0   0.7854   1.5708   2.3562   3.1416   3.9270   4.7124   5.4978   6.2832   7.0686   7.8540   8.6394
  Column 15
    9.4248
```

3. Définir u3

```
>> u3 = [0:10].^2
u3 =
   0    1    4    9   16   25   36   49   64   81  100
```

Solution Exercice 3:

1. Proposer les instructions MATLAB qui permettent de définir les trois vecteurs suivants:

$$v1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10),\ v2 = (-1.5, 0, 1.5, \ldots, 4.5, 6),\ v3 = (1, \tfrac{1}{4}, \tfrac{1}{9}, \tfrac{1}{16}, \tfrac{1}{25}, \ldots, \tfrac{1}{81}, \tfrac{1}{100})$$

```
>> v1 = [2:10]
v1 =
   2    3    4    5    6    7    8    9   10
>> V2 = [-1.5:1.5:6]
V2 =
-1.5000       0   1.5000   3.0000   4.5000   6.0000
```

```
>> V3 = (1./[1:10]).^2
V3 =
1.0000   0.2500   0.1111   0.0625   0.0400   0.0278   0.0204   0.0156   0.0123   0.0100
```

2. Créer un vecteur **v** qui contient tous les éléments de v1, v2, v3 consécutivement ;

```
>> V = [v1,V2,V3]
V =
Columns 1 through 14

 2.0000   3.0000   4.0000   5.0000   6.0000   7.0000   8.0000   9.0000  10.0000  -1.5000       0   1.5000   3.0000   4.5000

Columns 15 through 25

6.0000   1.0000   0.2500   0.1111   0.0625   0.0400   0.0278   0.0204   0.0156   0.0123   0.0100
```

3. Afficher les éléments de v de la 11eme position jusqu'à la 5eme position;

```
>> V(11:-1:5)
ans =  0  -1.5000  10.0000   9.0000   8.0000   7.0000   6.0000
```

4. Donner la commande MATLAB permettant d'afficher le deuxième tiers du vecteur v. Voir l'exemple en dessous pour comprendre c'est quoi le tiers

$$\underbrace{[1 \quad 2 \quad 3}_{1^{ere}\text{ tiers}} \underbrace{5 \quad 6 \quad -3}_{2^{eme}\text{ tiers}} \underbrace{4 \quad 2 \quad 9}_{3^{eme}\text{ tiers}}]$$

Selon la question, le deuxième tiers est le vecteur: [5 6 -3]. On doit calculer en premier la taille d'un tiers (tiersPosition = round(length(V)/3)), puis on extrait la deuxième tiers en spécifiant le début et la fin de ce dernier (début du deuxième tiers : tiersPosition+1 et la fin du deuxième tiers : 2*tiersPosition)

```
>> tiersPosition = round(length(V)/3)
tiersPosition = 8

>> V(tiersPosition+1 : 2*tiersPosition)
ans =
  10.0000  -1.5000       0   1.5000   3.0000   4.5000   6.0000   1.0000
```

Solution Exercice 4:

Soit $u1 = \begin{pmatrix}1\\2\\3\end{pmatrix}, u2 = \begin{pmatrix}-5\\2\\1\end{pmatrix}, u3 = \begin{pmatrix}-1\\-3\\7\end{pmatrix}$,

1. Définir les vecteurs u1, u2 et u3 sous MATLAB ;

```
>> u1=[1;2;3]
u1 =
    1
    2
    3
>> u2=[-5;2;1]
u2 =
   -5
    2
    1
>> u3=[-1;-3;7]
u3 =
   -1
   -3
    7
```

2. Réaliser les opérations suivantes toute en déduisant les opérations permises :

```
>> 15*u1   %%produit par un scalaire, opération permise ou chaque élément du vecteur est multiplié par le scalaire

ans =
   15
   30
   45

>> u1+3   %%ajout d'un scalaire, opération permise ou le scalaire est ajouté a chaque valeur du vecteur
ans =
```

```
    4
    5
    6

>> u2/3   %%division par un scalaire, opération permise ou chaque valeur du vecteur sera divisée par le scalaire
ans =
  -1.6667
   0.6667
   0.3333
>> u1+u2   %%opération permise si les vecteurs sont de même taille, on fait la somme élément par élément
ans =
   -4
    4
    4

>>  u1-u2   %%opération permise si les vecteurs sont de même dimension, opération faite élément par élément
ans =
    6
    0
    2
>> u1*u2   %%opération non permise si les vecteurs sont de même type
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> u1.*u2   %%opération permise, le produit est fait élément par élément comme la somme
ans =
   -5
    4
    3
>> u1'*u2   %% u1' est le vecteur transposé de u1, le produit de deux vecteurs de type diffèrent mais même taille est permis et sa
donne un scalaire (produit matriciel)
ans =
    2
>> u1/u2   %%l'opérateur / permet une division à droite, c'est à dire les éléments de u1 sont divisés par les éléments de u2,
opération permise qui résulte en une matrice
ans =
  -0.2000       0       0
  -0.4000       0       0
  -0.6000       0       0
>> u1./u2   %%division à droite élément par élément, opération permise et donne comme résultat un vecteur
ans =
  -0.2000
   1.0000
   3.0000
>> u1\u2   %%l'opérateur \ permet une division à gauche ou u2 est divisé par u1, opération permise qui donne un scalaire
ans =
   0.1429
>> u1.\u2   %%division à gauche élément par élément, opération permise qui donne un vecteur
ans =
  -5.0000
   1.0000
   0.3333
>> u1^7    %%opérations non permise avec les vecteurs
??? Error using ==> mpower
Matrix must be square.
>> u1.^7    %%la puissance élément par élément, opération permise qui donne un vecteur ou chaque élément est à la puissance de 7
ans =
       1
     128
    2187
```

Solution Exercice 5:

Soit le vecteur-ligne $v = (1\ -2\ 12\ 0\ 15\ 23\ 50\ 1\ 8)$.

5. Calculer et afficher la somme, la moyenne, le produit des éléments de v;
6. Calculer et afficher le min et le max de v ;
7. Renverser le vecteur v ;
8. Chercher et afficher les valeurs supérieures à 10 dans v.

```
>> v=[1-2 12 0 15 23 50 1 8]
v =
   -1   12    0   15   23   50    1    8
>> sum(v)   %%la somme des éléments de v avec la fonction sum
ans =
  108
>> prod(v)   %%le produit des éléments de v avec la fonction prod
ans =
    0
>> mean(v)   %%la moyenne des éléments de v avec la fonction mean
ans =
13.5000
>> min(v)   %%la valeur minimale dans v avec la fonction min
ans =
   -1
>> max(v)   %%la valeur maximale dans v avec la fonction max
ans =
   50
>> fliplr(v)   %%l'inversion de v avec la fonction fliplr
ans =
    8    1   50   23   15    0   12   -1
>> v
v =
   -1   12    0   15   23   50    1    8
%%première méthode
>> >> x=find(v>10)   %%chercher les positions (indices) des valeurs>10 avec la fonction find
x =
    2    4    5    6
>> v(x)        %%extraire les valeurs>10 depuis les indices dans x
ans =
   12   15   23   50
%%deuxième méthode extraire directement les valeurs >10 de v
>> v(v>10)
ans =
   12   15   23   50
```

Solution TP3- Les matrices sous MATLAB

Solution Exercice 1: manipuler une matrice

7. Donner le résultat de chacune des instructions MATLAB suivantes :

- $\boldsymbol{C = diag(diag(ones(3)))}$

```
>> x1 =
   1   1   1
   1   1   1
   1   1   1
>> x2=diag(ones(3))
x2 =
   1
   1
   1
>> C=diag(x2)
C =
   1   0   0
   0   1   0
   0   0   1
```

- $\boldsymbol{k = [1:1:5]}$

```
k =
   1   2   3   4   5
```

- $\boldsymbol{D = (ones(5) + diag(k)) + 3 * zeros(5)}$

```
>> x1=ones(5)
x1 =
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
>> x2=diag(k)
x2 =
   1   0   0   0   0
   0   2   0   0   0
   0   0   3   0   0
   0   0   0   4   0
   0   0   0   0   5
>> x3=3*zeros(5)
x3 =
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
>>  (x1+x2)+x3        % ((x1+x2)+x3 )=D = (ones(5) + diag(k)) + 3 * zeros(5)
ans =
   2   1   1   1   1
   1   3   1   1   1
   1   1   4   1   1
   1   1   1   5   1
   1   1   1   1   6
```

Solution Exercice 2: manipulation d'une matrice

1. Définir A, B et C

```
>> A=[1 5;2 3] ;
>> B=[7 -2;0 5] ;
>> C=[-1 5;0 3;-2 -1;4 9] ;
```

Calculer l'expression $A * B - 3$

```
>> A*B-3              %un produit matriciel et une soustraction…
ans =
   4   20
  11    8
```

Calculer l'expression C ∗ B + 1 + zeros(4,2)**;**

```
>> C*B+1+zeros(4,2)
ans =
   -6   28
    1   16
  -13    0
   29   38
```

Calculer l'expression $C(end:-1:1,2).\backslash 24$**;**

```
>> >> C(end:-1:1,2).\24
ans =
   2.6667
 -24.0000
   8.0000
   4.8000
```

2. Matrice M construite depuis A, B et C:

```
>> M=[[A;B] C]
M =
    1    5   -1    5
    2    3    0    3
    7   -2   -2   -1
    0    5    4    9
```

3. Remplacer M(3,2) par 0:

```
>> M(3,2)=0
M =
    1    5   -1    5
    2    3    0    3
    7    0   -2   -1
    0    5    4    9
```

4. Calcule de la matrice a:

```
>> a=tril(M,1)
a =
    1    5    0    0
    2    3    0    0
    7    0   -2   -1
    0    5    4    9
```

5. Calcule de la matrice b:

```
>> b=triu(M,-2)
b =
    1    5   -1    5
    2    3    0    3
    7    0   -2   -1
    0    5    4    9
```

6. Supprimer la 1ere et la 3eme lignes de M:

```
M =
    1    5   -1    5
    2    3    0    3
    7    0   -2   -1
    0    5    4    9
>> M([1,3],:)=[]
M =
    2    3    0    3
    0    5    4    9
```

7. Supprimer la dernière colonne de M:

```
>> M(:,end)=[]
M =
    2    3    0
    0    5    4
```

8. Ajouter un vecteur ligne x en dernière ligne de M

```
>> x=[1 2 3]
x =
```

```
   1    2    3
>> M=[M;x]
M =
   2    3    0
   0    5    4
   1    2    3
```

9.**Remplacer la colonne 3 dans M par le vecteur colonne v**

```
>> v=[1;9;11]
v =
   1
   9
  11
>> M(:,3)=v
M =
   2    3    1
   0    5    9
   1    2   11
```

Solution Exercice 3: matrices particulières

1. Définir A, B, C et D

```
>> A=rand(3,3)   %matrice aléatoire
A =
   0.4447    0.9218    0.4057
   0.6154    0.7382    0.9355
   0.7919    0.1763    0.9169
>> B=zeros(3,2)   %matrice des zeros
B =
   0    0
   0    0
   0    0
>> C=ones(2,3)    %matrice des 1
C =
   1    1    1
   1    1    1
>> D=eye(3)     %matrice identité
D =
   1    0    0
   0    1    0
   0    0    1
```

2. Remplacer diagonale de D par vecteur x

```
>> x=[3;5;7]
x =
   3
   5
   7
>> D=diag(x)
D =
   3    0    0
   0    5    0
   0    0    7
```

3. Créer une matrice magique 3X3

```
>> A=magic(3)
A =
   8    1    6
   3    5    7
   4    9    2
```

Solution Exercice 4: opérations matricielles

1. Définir M et différentes expressions

```
>> M=[1 2;3 4]
M =
```

```
   1    2
   3    4
>> 2 * M + 3    %produit et ajout scalaire a une matrice
ans =
   5    7
   9   11
>> M ^ 3      %puissance matricielle, équivalente a M*M*M
ans =
  37   54
  81  118
>> M .^ 3     %puissance terme à terme à un scalaire
ans =
   1    8
  27   64
>> M + M      %somme matricielle
ans =
   2    4
   6    8
>> M – M     %soustraction matricielle
ans =
   0    0
   0    0
>> M * M     %produit matricielle
ans =
   7   10
  15   22
>> M .* M    %produit terme à terme
ans =
   1    4
   9   16
>> M / M     %division matricielle
ans =
   1    0
   0    1
>> M ./ M     %division terme à terme
ans =
   1    1
   1    1
```

2. <u>matrice 54 × 42 ne contenant que des 7</u>

```
>> A= 7 * ones(54, 42) ;
```

3. <u>Calcules</u>

```
>> M=[1 2;3 4]
M =
   1    2
   3    4
>> M'     %transpose de M
ans =
   1    3
   2    4
>> M^ -1     %inverse de M
ans =
  -2.0000    1.0000
   1.5000   -0.5000
>> det(M)    %déterminant de M
ans =
   -2
>> [U,D]=eig(M)    %vecteurs (U) et valeurs propres (D)
U =
  -0.8246   -0.4160
   0.5658   -0.9094
D =
  -0.3723         0
        0    5.3723
>> eig(M)      %valeurs propres de M
ans =
  -0.3723
   5.3723
```

Solution TP4- Les Scripts et les fonctions MATLAB

Les scripts MATLAB

Solution Exercice 1:

1. Permutation de deux nombres

```
disp('Donner x') ;
x = input('entrez x : ') ;
disp('Donner y') ;
y = input('entrez y : ') ;
z=x;
x=y;
y=z;
x, y
```

2. Affichage de signe d'un nombre

```
x=input('donner x: ')
if x<0
   disp('nombre negatif')
elseif x>0
      disp('nombre positif')
   else
      disp('nombre null')
   end
```

Solution Exercice 2:

4. Deviner un code aléatoire

```
code= 100*rand;
code=floor(code)
trouve=1;
while trouve==1
   codelu=input('tapez un code : ');
   if codelu==code
      disp('BON CODE');
      trouve=0;
   else
      disp ('CODE FAUX');
   end
end
```

5. Somme k, a et b

```
a = input('entrez a : ');
b = input('entrez b : ');
s=0;
for k=1:b
s=s+k^a;
end
s
```

6. Le plus petit n pour la somme Sn

```
x=input('donner x');
```

```
if (x<=0)|(x>=20)
   error('x non conforme')
else
   s=0;n=0;
   while s<=x
   n=n+1;
   s=s+(1/n);
end
end
n
```

7. <u>10 eme terme de fibonnaci</u>

```
u0=0; % terme representant u_n
u1=1; % terme representant u_n+1
for k=1:10
u2 = u1 + u0;
u0 = u1;
u1 = u2;
end
disp(u0);
```

8. <u>Produit matriciel</u>

```
p=3, n=2, q=2;   %dimension des deux matrices
for i=1:p       %remplissage de A
   for j=1:n
      A(i,j)=input('donner une valeur A: ');
   end
end

for i=1:n    %remplissage de B
   for j=1:q
     B(i,j)  =input('donner une valeur B: ');
   end
end

for i = 1:p    %calcul et affichage du produit
   for j = 1:q
      x = 0
      for k = 1:n
         x = x + A(i,k)*B(k,j);
      end
      C(i,j) = x
   end
end
```

Les Fonctions MATLAB

<u>Solution Exercice 1</u>

1. **Le double symbole égal** désigne l'égalité, il test si la partie gauche est égale à la partie droite du signe. **La double apostrophe** désigne un accent, on l'utilise pour la différencier de l'accent d'un message, sinon elle va marquer la fin du message, ce qui génère une erreur.

2. Avec 12 : c'est faux.
Avec 42 : c'est vrai
Avec 41 : presque

Solution Exercice 2

1. Nombre pair ou non

```
function r=pair(x)
if mod(x,2)==0
r='vrai';
else
r='faux';
end
```

2. Somme matriciel

```
function C=sommematriciel (A,B)
[p,n]=size(A);[m,q]=size(B);
if (p==m)&(n=q)
   C=zeros(p,n);
   for i=1:p
   for j=1:n
        C(i,j)=A(i,j)+B(i,j);
      end
   end
else
   error('taille de matrices incompatibles')
end
```

3. Produit matriciel

```
function C=produitmatriciel (A,B)
[p,n]=size(A);[m,q]=size(B);

if n==m
   C=zeros(p,q);
   forli=1:p
   for co=1:q
      for k=1:n
         C(li,co)=C(li,co)+A(li,k)*B(k,co);
      end
   end
end
else
   error('taille de matrices incompatibles')
end
```

Solution Exercice 3

1. Donner la valeur de B après l'exécution des instructions suivantes :

```
>> A=[1 2 3 4;5 6 7 8 ;9 10 11 12];
>> B=calcul(A)
B =
   4    3    2    1
```

```
   8    7    6    5
  12   11   10    9
```

2. <u>Déduire ce que fait cette fonction :</u>

Renversement des lignes de la matrice.

<u>Réécrire la fonction pour obtenir le même résultat en utilisant une seule boucle</u> :

```
function M=calcul1(M)
m=size(M,2);
N=M;
  for j=1:m
    M(:,j)=N(:,m-j+1)
  end
```

Utilisation des fonctions et scripts ensemble sous MATLAB

<u>Solution Exercice 1</u>

1. <u>Ecrire une fonction MATLAB produit.</u>

```
function C=produitmatriciel (A,B)
[p,n]=size(A);

  C=zeros(p,n);
  for i=1:p
  for j=1:n
    for k=1:n
      C(i,j)=C(i,j)+A(i,k)*B(k,j);
    end
  end
  end
```

2. <u>Écrivez le script MATLAB.</u>

```
n1=input('donner nb ligne A');     %nombre de lignes de A
p1=input('donner nb colonnes A');  %nombre de colonnes de A
n2=input('donner nb ligne B');     %nombre de lignes de B
p2=input('donner nb colonnes B');  %nombre de colonnes de B

if (n1==p1 & n2==p2 & n1==n2)   %A et B sont carrées et nombre de colonnes de A doit égale au nombre de lignes de B
for i=1:n1          %remplissage de A
  for j=1:p1
    A(i,j)=input('donner une valeur A: ');
  end
end

for i=1:n2       %remplissage de B
  for j=1:p2
    B(i,j)  =input('donner une valeur B: ');
  end
end
  C1=produitmatriciel(A,B)  %appel de la fonction produitmatriciel qui donne A*B
  C2=produitmatriciel(B,A)  %appel de la fonction produitmatriciel qui donne B*A
for i=1:n1       %calcul de A*B-B*A
  for j=1:p1
    C3(i,j)=C1(i,j)+C2(i,j)
```

```
  end
end
else error('dimension non compatibles'); %affichage d'un message d'erreur
end
```

Solution Exercice 2

1. Ecrire une fonction Matlab « facto » qui prend comme argument un nombre entier positif n et qui retourne $n!$ comme réponse. Sachant que $n! = 1 \times 2 \times 3 \times \dots \times n$.

```
function r=fact(n);
r =1;
for i=2:n
r= r * i;
end
```

2. Ecrire un script Matlab qui permet de lire deux nombre $\boldsymbol{n}$ et $\boldsymbol{p}$ et qui calcule et affiche

$$X = n!/(p! * (n-p)!)$$

```
n=input('donner n');
p=input('donner p');
if n<0 | p<0 error('nombres negatifs');
else
  X=fact(n)/(fact(p)*fact(n-p));
end
disp(X);
```

Solution Exercice 3

1. Ecrire la fonction parfait qui prend comme argument un nombre entier x, et qui retourne 1 si ce nombre est parfait sinon elle retourne 0.

```
function r=parfait(x)
s=0;
for i=1:x-1
  if mod(x,i)==0 s=s+i;
  end
end
if s==x r=1;
else r=0;
end
end
```

2. Ecrire un script Matlab qui permet à l'utilisateur de saisir un nombre entier compris entre 1 et 10, puis test et affiche si ce nombre est parfait. Le script doit faire appel à la fonction parfait.

```
x=input('donner x');
if x<=10 & x>=1
if parfait(x)==1 disp('parfait');
else disp('non parfait');
end
else disp('nombre >10 ou < 1');
end
```

Solutions des exercices supplémentaires

Solution Exercice 1 : même solution que l'exercice 2 sauf qu'on doit changer l'instruction X=fact(n)/(fact(p)*fact(n-p)); par X=fact(n)/fact(n-p);

Solution Exercice 2:

1. Ecrire une Ecrire une fonction Matlab « premier » qui prend comme argument un nombre n et qui retourne 1 (true) si n est premier sinon elle retourne 0 (false).

```
function r=premier(x)
nb=0;
for i=1:x
  if mod(x,i)==0 nb=nb+1;
  end
end
if nb >2 r=0;
else r=1;
end
end
```

2. Ecrire le script Matlab qui permet de saisir deux entiers positifs x et y (x < y) et qui affiche ensuite la liste des nombres premiers compris entre x et y. Le script doit utiliser la fonction premier

```
x=input('donner x');
y=input('donner y');
if x<y
for i=x:y
  if (premier(i)==1) disp(i);
  end
end
else disp('error') ;
end
```

Solution Exercice 3:

1. Ecrire une fonction Matlab « divisible » qui prend comme argument deux nombres n et m et qui retourne 1 (true) si n est divisible par m et 0 (false) sinon.

```
function r=divisible(a,b)
if mod(a,b)==0 r=1;
else r=0;
end
end
```

2. Ecrire le script Matlab, permettant de saisir trois entiers n, a et b, puis affiche la liste des nombres compris entre a et b qui sont divisibles par n.

```
n=input('donner un nombre');
a=input('donner un nombre');
b=input('donner un nombre');
for i=a:b
  if divisible(i,n)==1 disp(i);
  end
end
```

Solution Exercice 4:

1. Ecrire une fonction Matlab « amicaux » qui prend comme argument deux nombres entiers positifs x *et* y et qui retourne 1 si ils sont amicaux, sinon elle retourne 0. Sachant que x *et* y sont amicaux si x est la somme des diviseurs de y, hormis y, et si y est la somme des diviseurs de x, hormis x. Exemple : 220 et 284 sont amicaux.

```
function r=amicaux(x,y)
s1=0;s2=0;
for i=1:x-1
```

```
   if mod(x,i)==0 s1=s1+i;
   end
end
for j=1:y-1
   if mod(y,j)==0 s2=s2+j;
   end
end
if s1==y & s2==x r=1;
else r=0;
end
end
```

2. Ecrire un script Matlab qui permet de lire deux nombre ***a*** et ***b*** et qui affiche si ils sont amicaux ou non. Le script doit faire appel à la fonction amicaux.

```
a=input('donner a');
b=input('donner b');
if amicaux(a,b)==1 disp(' a et b amicaux');
else disp('non amicaux');
end
```

Solution TP5- Représentation graphique des données sous MATLAB

Solution Exercice 1:

3. Tracer les fonctions f et g sur la même figure même axis

```
x=-pi:0.1:pi;
f=sin(x);
g=cos(x);
plot(x,f,x,g);
```

4. Tracer les fonctions f et g sur la même figure des axes différents

```
x=-pi:0.1:pi;
f=sin(x);
g=cos(x);
%plot(x,f,x,g);
subplot(221);
plot(x,f);
subplot(222);
plot(x,g);
```

9. Courbe de f(x)

```
x=0:pi/12:2*pi;
f=sin(x-2)+4;
plot(x,f,':gd')
```

Courbe de p(x)

```
x=-5:0.2:5;
p=-2*x.^3+x.^2-3;
plot(x,p,'b--s');
```

Solution Exercice 2:

1. Courbe de f(x)

```
x=0:pi/12:2*pi;
f=sin(x-2)+4;
plot(x,f,':gd')
```

2. Courbe de g(x)

```
x=-5:0.2:5;
g=-2*x.^3+x.^2-3;
plot(x,g,'b--s');
```

Solution Exercice 3:

```
x=[0:1:2*pi];
y=tan(x);
f=cos(x);
g=exp(x);
z1=sin(x);
z2=asin(x);
subplot(2,2,1), plot(x,y,'p-'),xlabel('x'), ylabel('y'),title('tangante(x)')
subplot(2,2,2), plot(x,f,':*'), xlabel('x'), ylabel('y'),title('cosinus(x)')
subplot(2,2,3), plot(x,g,'-.p'),xlabel('x'), ylabel('y'),title('exponentielle(x)')
subplot(2,2,4), plot(x,z1,'-^',x,z2,'-d'), xlabel('x'), ylabel('y'),title('sinus(x) et sinus inverse (x)')
```

Prototypes d'examens

Examen Final – Outils de Programmation pour les Mathématiques

Remarque **: Toutes les réponses doivent êtres sous le langage MATLAB**

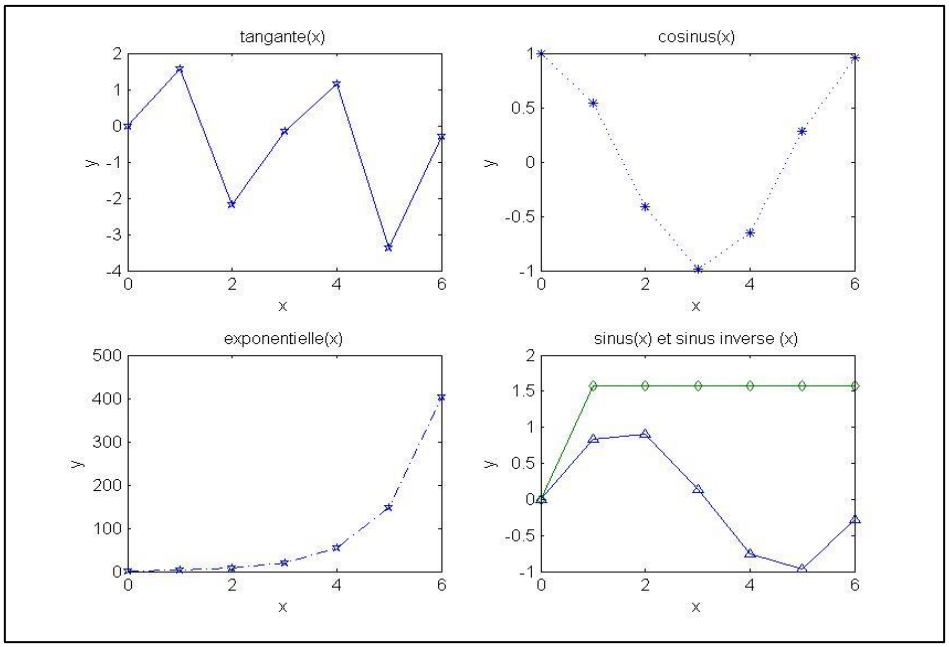
Exercice 1 :

1. Donner le résultat de chacune des instructions MATLAB suivantes :
   $\gg B = diag(diag(ones(diag(3))))$
   $\gg k = [3\colon 2\colon 5]$
   $\gg A = \big(ones(3) + diag([k\ 10])\big)' + 3 * eye(3)'$
   $\gg F = k + [1\colon 2]$

2. Donner les instructions MATLAB équivalentes aux expressions mathématiques suivantes :
   $x \leftarrow \frac{b}{2} \times \sqrt{c^2 - \left(\frac{b}{2{,}5}\right)^2}$; $y \leftarrow e^{2-\sqrt{b^3-\frac{1}{a}}}$; $z \leftarrow \frac{|2n^5-3|}{\sqrt{4n^2+\ln(6n)}}$

3. Donner les expressions mathématiques équivalentes aux instructions MATLAB suivantes :
   $\exp(sqrt(x))/(2 * y - 1) + abs(x) - 1\ /(y\ \hat{}\ 2 + 3)$; $2 * sind(45)/\exp(2)$;

4. Écrire l'ensemble des instructions MATLAB permettant de tracer les courbes suivantes dans la même Figure :



Exercice 2 :

Soit la fonction *calcul* a deux boucles imbriquées suivante :

```
function M=calcul(M)
[n,m]=size(M);
for i=1:n
  v=M(i,:);
  for j=1:m
    M(i,j)=v(m-j+1)
  end
end
```

4. Donner la valeur de B après l'exécution des instructions suivantes :

```
>> A=[1 2 3 4;5 6 7 8 ;9 10 11 12];
>> B=calcul(A)
```

5. Déduire ce que fait cette fonction.
6. Réécrire la fonction *calcul* pour obtenir le même résultat en utilisant une seule boucle.

Exercice 3 :

3. Ecrire une fonction MATLAB *produit* qui prend comme argument deux matrices carrées de même dimensions **A** et **B** et calcule le produit $A \times B$, puis affiche le résultat a la matrice C. La matrice C a pour terme général :

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

4. Écrivez le script MATLAB qui permet de saisir deux matrices carrées de même dimensions **A** et **B**, vérifie si leur taille est compatible. Si elles ne le sont pas, donne un message d'erreur et pas de sortie, si elles le sont, calcule $AB + BA$. Le script fait appel à la fonction *produit*.

NB : On s'interdit d'utiliser les opérations du type $A \times B$ ou $A + B$, on doit plutôt utiliser une affectation élément par élément.

Examen Final – Outils de Programmation pour les Mathématiques

Remarque **: Toutes les réponses doivent êtres sous le langage MATLAB**

Exercice 1 :

5. Donner le résultat de chacune des instructions MATLAB suivantes **:**

   $\gg k = 20 : -3 : 1$

   $\gg b = [1 \quad -3 \quad 2 \quad 10] * (5 * eye(4))$

   $\gg a = b - [4 \quad 5 \quad 9 \quad -1]$

   $\gg S = [k(5) \quad k(1) \quad (k(5) - 1) \quad (k(5) + 3) ; \quad a \quad ; \quad b \,]$

6. Traduire les expressions mathématiques suivantes en instructions MATLAB :

   – $w \leftarrow -2\, ln(5x) + \sqrt{4x^3 + 1},$ pour $x = -3i$

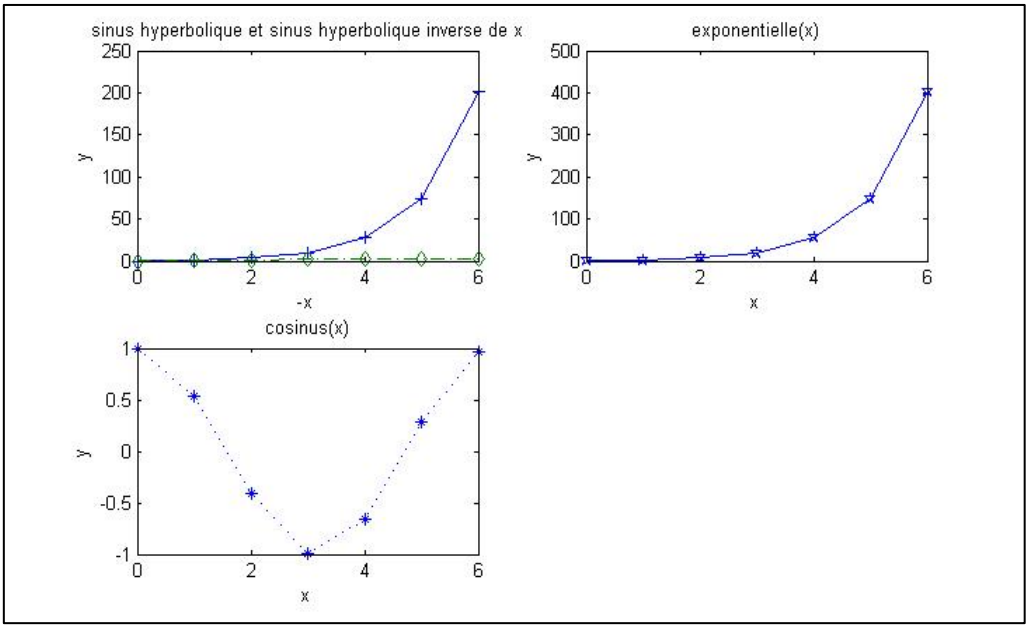
   – $y \leftarrow \frac{|2n^5 - 3|}{\sqrt{4n^2 + \ln(6n)}}$;

   – $z \leftarrow \frac{e^{\sqrt{x}}}{2y-1} + |x| - \frac{1}{y^2+3}$

7. Soit **n** un entier naturel défini à l'avance. Donnez les commandes MATLAB permettant de calculer les expressions suivantes de la manière la plus simple (sans utiliser de boucle ::

   f = n! ; $z = n + \frac{n-1}{2} + \frac{n-2}{3} + \cdots + \frac{1}{n}$

8. Écrire l'ensemble des instructions MATLAB permettant de tracer les courbes suivantes **(2**:



Exercice 2 :

Soit le script suivant:

```
x=input('donner une valeur de x') ;
n=input('donner une valeur de n');
z=0 ;  v=1;  i=1 ;
while i<=n
  z=z+v*(x^i)/i ;
  v=-v ;
  i=i+2 ;
```

```
end
z
```

7. Donner la valeur de z après l'exécution du script avec x=5 et n=7;
8. Déduire ce que fait ce script.
9. Réécrire le script pour obtenir le même résultat en utilisant une boucle for.

Exercice 3 :

5. Ecrire une fonction MATLAB *parfait* qui prend comme argument un nombre entier x, et qui retourne 1 si ce nombre est parfait sinon elle retourne 0. Un nombre parfait c'est un nombre qui égale à la somme de ses diviseurs stricts.
6. Ecrire un script Matlab qui permet à l'utilisateur de saisir une matrice A, puis qui affiche les nombres parfaits de cette matrice. Le script fait appel à la fonction précédente.

Corrigée type des examens

Correction Examen Final – Outils de Programmation pour les Mathématiques

Remarque **: Toutes les réponses doivent êtres sous le langage MATLAB**

Exercice 1 :

1. Donner le résultat de chacune des instructions MATLAB suivantes :

$\gg k = [3\colon 2\colon 5]$	**k =** **3 5**
$\gg A = \left(ones(3) + diag([k\ 10])\right)' + 3 * eye(3)'$	**A =** **7 1 1** **1 9 1** **1 1 14**
$\gg F = k + [1\colon 2]$	**F =** **4 7**

2. Donner les instructions MATLAB équivalentes aux expressions mathématiques suivantes :

$x \leftarrow \frac{b}{2} \times \sqrt{c^2 - \left(\frac{b}{2,5}\right)^2}$	$\boldsymbol{x = (b/2) * sqrt(c\text{\^{}}2 - (b/2.5)\text{\^{}}2)}$
$y \leftarrow e^{2-\sqrt{b^3-\frac{1}{a}}}$	$\boldsymbol{y = exp(2 - sqrt(b\text{\^{}}3 - 1/a))}$
$z \leftarrow \frac{\lvert 2n^5-3\rvert}{\sqrt{4n^2+\ln(6n)}}$	$\boldsymbol{z = abs(2 * n\text{\^{}}5 - 3)/sqrt(4 * n\text{\^{}}2 + log(6 * n))}$

3. Donner les expressions mathématiques équivalentes aux instructions MATLAB suivantes :

$\exp(sqrt(x))/(2 * y - 1) + abs(x) - 1\ /(y\ \text{\^{}}\ 2 + 3)$	$\boldsymbol{\frac{e^{\sqrt{x}}}{2y-1} + \lvert x\rvert - \frac{1}{y^2+3}}$
$2 * sind(45)/\exp(2)$	$\boldsymbol{\frac{2\sin(45)}{e^2}}$**, les angles sont données en dégrée**

4. Écrire l'ensemble des instructions permettant de tracer les courbes suivantes :

```
x=[0:1:2*pi];
y=tan(x);
f=cos(x);
g=exp(x);
z1=sin(x);
z2=asin(x);
subplot(2,2,1), plot(x,y,'p-'),xlabel('x'), ylabel('y'),title('tangante(x)')
subplot(2,2,2), plot(x,f,':*'), xlabel('x'), ylabel('y'),title('cosinus(x)')
subplot(2,2,3), plot(x,g,'-.p'),xlabel('x'), ylabel('y'),title('exponentielle(x)')
subplot(2,2,4), plot(x,z1,'-^',x,z2,'-d'), xlabel('x'), ylabel('y'),title('sinus(x) et sinus inverse (x)')
```

Exercice 2 :

1. Donner la valeur de B après l'exécution des instructions suivantes :

```
>> A=[1 2 3 4;5 6 7 8 ;9 10 11 12];
>> B=calcul(A)
```

B =

```
 4   3   2   1
 8   7   6   5
12  11  10   9
```

2. Déduire ce que fait cette fonction .

   Renversement des lignes de la matrice.

3. Réécrire la fonction pour obtenir le même résultat en utilisant une seule boucle .

```
function M=calcul1(M)
m=size(M,2);
N=M;
  for j=1:m
    M(:,j)=N(:,m-j+1)
  end
```

Exercice 3 :

7. Ecrire une fonction MATLAB *produit* .

```
function C=produitmatriciel (A,B)
[p,n]=size(A);
  C=zeros(p,n);
  for i=1:p
  for j=1:n
    for k=1:n
      C(i,j)=C(i,j)+A(i,k)*B(k,j);
    end
  end
  end
```

8. Écrivez le script MATLAB.

```
n1=input('donner nb ligne A');
p1=input('donner nb colonnes A');
n2=input('donner nb ligne B');
p2=input('donner nb colonnes B');
if (n1==p1 & n2==p2 & n1==n2)
for i=1:n1      %remplissage de A
  for j=1:p1
    A(i,j)=input('donner une valeur A: ');
  end
end
for i=1:n2      %remplissage de B
  for j=1:p2
    B(i,j)  =input('donner une valeur B: ');
  end
end
  C1=produitmatriciel(A,B)
  C2=produitmatriciel(B,A)
for i=1:n1      %calcul de A*B-B*A
  for j=1:p1
    C3(i,j)=C1(i,j)+C2(i,j)
  end
end
else error('dimension non compatibles');
end
```

Correction Examen Final – Outils de Programmation pour les Mathématiques

Remarque **: Toutes les réponses doivent êtres sous le langage MATLAB**

Exercice 1 :

1. Donner le résultat de chacune des instructions MATLAB suivantes :

$\gg k = 20 : -3 : 1$	**k =** **20 17 14 11 8 5 2**
$\gg b = [1 \quad -3 \quad 2 \quad 10] * (5 * eye(4))$	**b =** **5 -15 10 50**
$\gg a = b - [4 \quad 5 \quad 9 \quad -1]$	**a =** **1 -20 1 51**
$\gg S = [k(5) \quad k(1) \quad (k(5)-1) \quad (k(5)+3) ; \quad a \quad ; \quad b]$	**S =** **8 20 7 11** **1 -20 1 51** **5 -15 10 50**

2. Traduire les expressions mathématiques suivantes en instructions MATLAB :

$w \leftarrow -2\,ln(5x) + \sqrt{4x^3+1}$, pour $x = -3i$	$\boldsymbol{x = -3 * i;}$ $\boldsymbol{w = -2 * log(5 * x) + sqrt((4 * x \ \hat{}\ 3) + 1)}$
$y \leftarrow \frac{\lvert 2n^5 - 3 \rvert}{\sqrt{4n^2 + \ln(6n)}}$;	$\boldsymbol{y = abs(2 * n\hat{}5 - 3)/sqrt(4 * n\hat{}2 + log(6 * n))}$
$z \leftarrow \frac{e^{\sqrt{x}}}{2y-1} + \lvert x \rvert - \frac{1}{y^2+3}$	$\boldsymbol{z = \exp(sqrt(x))/(2 * y - 1) + abs(x) - 1\ /(y\ \hat{}\ 2 + 3)}$

3. Soit n un entier naturel défini à l'avance. Donnez les commandes MATLAB permettant de calculer les expressions suivantes de la manière la plus simple (sans utiliser de boucle):

F = n!	$\boldsymbol{prod(1:n)}$
$Z = n + \frac{n-1}{2} + \frac{n-2}{3} + \cdots + \frac{1}{n}$	$\boldsymbol{z = sum((n:-1:1)./(1:n))}$

4. Écrire l'ensemble des instructions MATLAB permettant de tracer les courbes suivantes :

```
x=[0:1:6];
f=sinh(x);
g=asinh(x);
z1=exp(x);
z2=cos(x);
subplot(2,2,1), plot(x,f,'+-',x,g,'-.d'), xlabel('-x'), ylabel('y'),title('sinus hyperbolique et sinus hyperbolique inverse de x')
subplot(2,2,2), plot(x,z1,'-p'),xlabel('x'), ylabel('y'),title('exponentielle(x)')
subplot(2,2,3), plot(x,z2,':*'), xlabel('x'), ylabel('y'),title('cosinus(x)')
```

Exercice 2 :

1. Donner la valeur de z après l'exécution du script avec x=5 et n=7 ;

   $\boldsymbol{z = 5 - \frac{5^3}{3} + \frac{5^5}{5} - \frac{5^7}{7} = -1.0572 + e004}$

2. Déduire ce que fait ce script

   Le script calcul la somme $\boldsymbol{z = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots \mp \frac{x^n}{n}}$

3. Réécrire le script pour obtenir le même résultat en utilisant une boucle for.

```
x=input('donner une valeur de x') ;
n=input('donner une valeur de e');
z=0 ;  v=1;
for i=1:2:n
  z=z+v*(x^i)/i ;
  v=-v ;
end
z
```

Exercice 3 :

1. Ecrire une fonction MATLAB *parfait* qui prend comme argument un nombre entier x, et qui retourne 1 si ce nombre est parfait sinon elle retourne 0. Un nombre parfait c'est un nombre qui égale à la somme de ses diviseurs stricts.

```
function r=parfait(x)
s=0;
for i=1:x-1
  if mod(x,i)==0 s=s+i;   %somme des diviseurs de x
  end
end
if s==x r=1;
else r=0;
end
end
```

2. Ecrire un script Matlab qui permet à l'utilisateur de saisir une matrice A, puis qui affiche les nombres parfaits de cette matrice. Le script fait appel à la fonction précédente.

```
n1=input('donner nb ligne A');        %nombre de lignes de A
p1=input('donner nb colonnes A');        %nombre de colonnes de A

for i=1:n1          %remplissage de A
  for j=1:p1
    A(i,j)=input('donner une valeur A: ');
  end
end

for i=1:n1          %Affichage des nombres parfaits de A
  for j=1:p1
    if parfait (A(i,j))==1 disp(A(i,j))
    end
  end
end
```

Conclusion générale

Matlab est l'un des langages scientifiques les plus recommandés grâce à sa préoccupation des débutants et son support des calculs réclamés par leurs approches. Son intérêt tient d'une part à sa simplicité d'utilisation et d'autre part sa richesse fonctionnelle qui couvre une large catégorie de problèmes dans divers domaines. Il permet : (1) le calcul scientifique, (2) la programmation et (3) la visualisation graphique de données.

Nous avons abordé dans ce cours les trois objectifs de MATLAB. Le cours a été divisé en trois chapitres et cinq séries de TP permettant aux étudiants de renforcer les notions théoriques prise dans le cours. Le premier chapitre se focalise sur le calcul scientifique sous MATLAB. Nous avons présenté dans ce chapitre certains outils permettant le calcul scientifique avancé, le calcul vectoriel et le calcul matriciel. Le deuxième chapitre montre la programmation sous MATLAB à travers la compréhension et l'écriture des scripts et des fonctions. D'autres informations complémentaires comme les structures de contrôles, et les instructions break et continue ont été aussi présentées. Comme dernier point dans ce chapitre, nous avons présenté la visualisation graphique 2D, ainsi les étudiants apprennent à comment représenter leurs données graphiquement. Le troisième chapitre, montre l'utilisation de MATLAB pour la résolution de problèmes réels de mathématiques.

A la fin de ce cours, un guide d'installation de MATLAB 7.0 a été donné dans l'annexe 1.

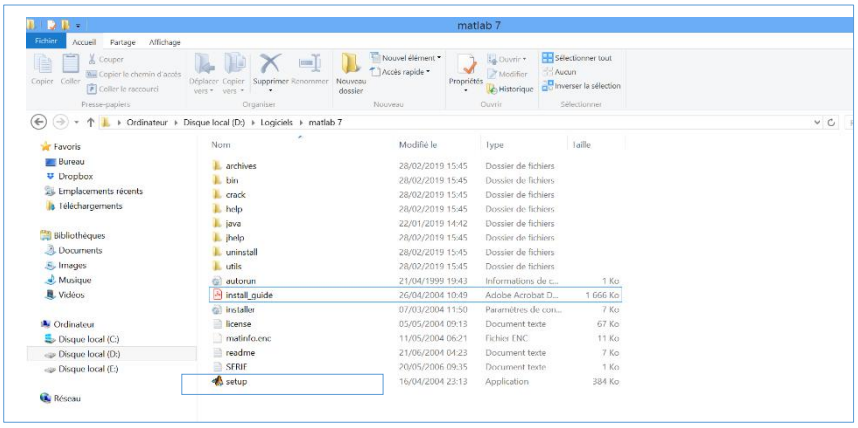
Annexes

Annexe 1 : Guide d'installation de MATLAB

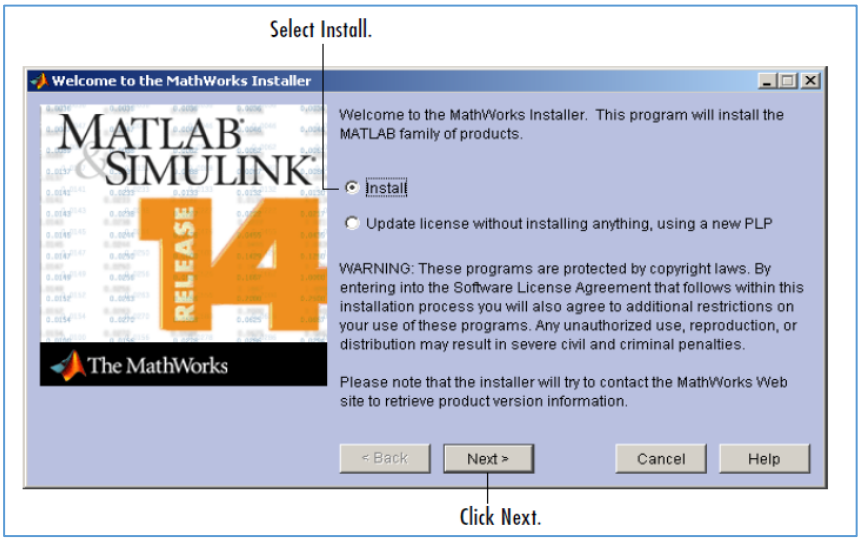
(Extrait du guide d'installation de MATLAB)

Par le guide suivant, on va apprendre à comment installer la version 7 de MALAB. Pour une installation réussite, suivez les étapes suivantes :

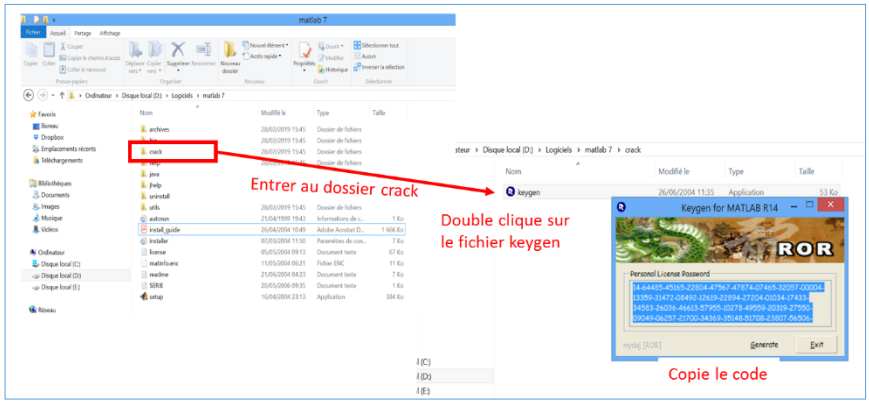
1. Une première étape consiste à lancer l'installation par un double clic sur le fichier setup (le fichier d'installation de MATLAB).



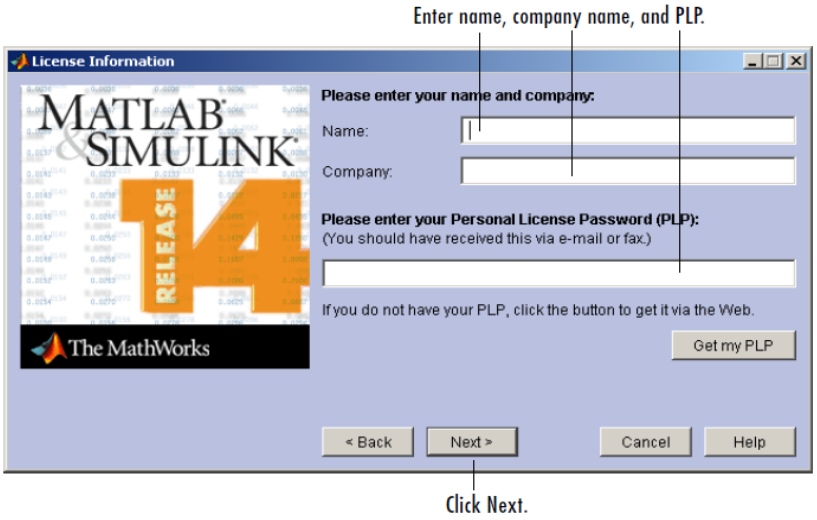
2. Cocher le radio bouton Install et cliquer sur le bouton Next



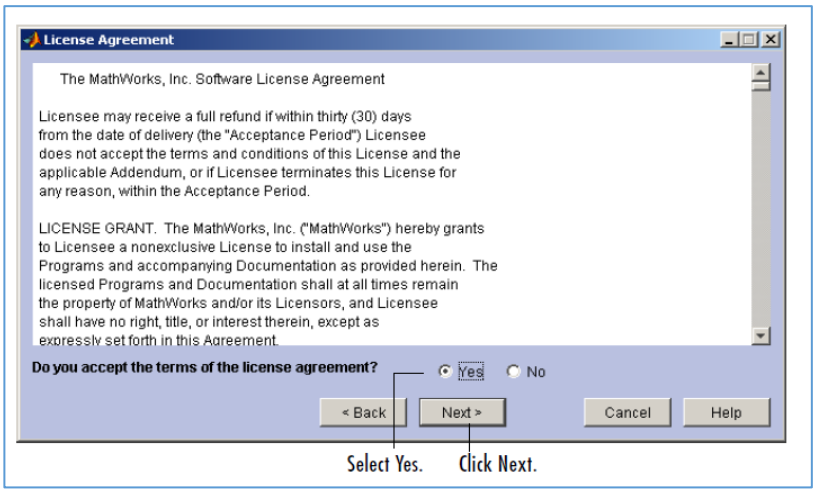
3. Copie le code d'activation.



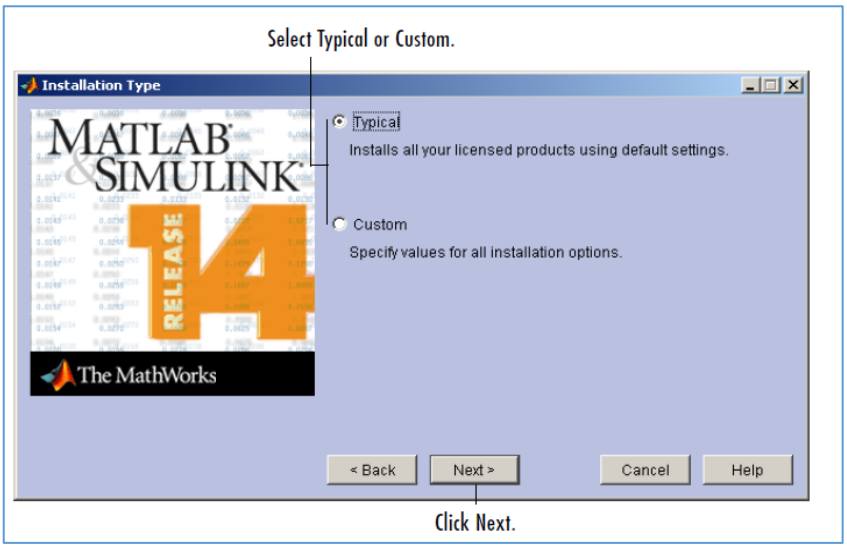
4. Remplir les champs Name et company par des caractères. Puis, copie le code d'activation depuis le fichier crack (voir le diapo précèdent) et le coller dans la case PLP;



5. Cocher le bouton Yes et cliquer sur Next

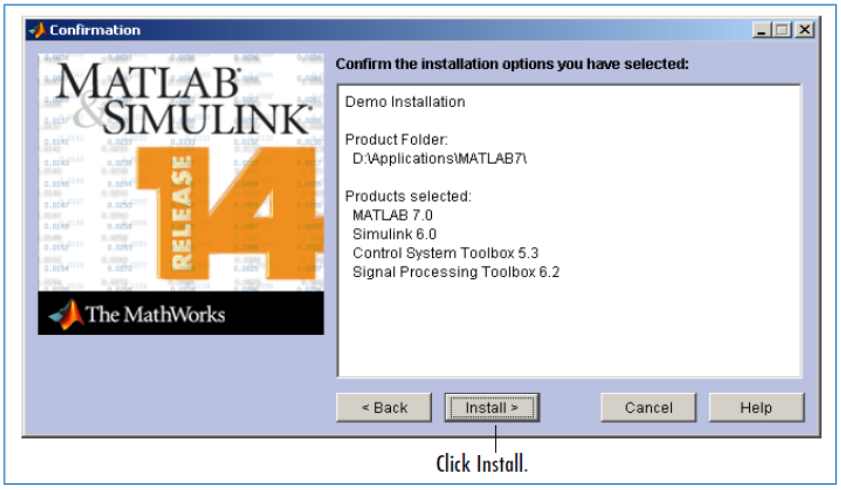


6. Cocher le type d'installation Typical et cliquer sur Next

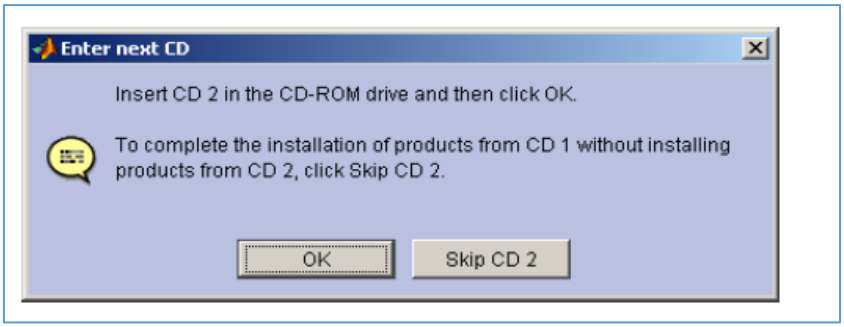


7. Spécifier le ficher d'installation puis Cliquer sur Next;

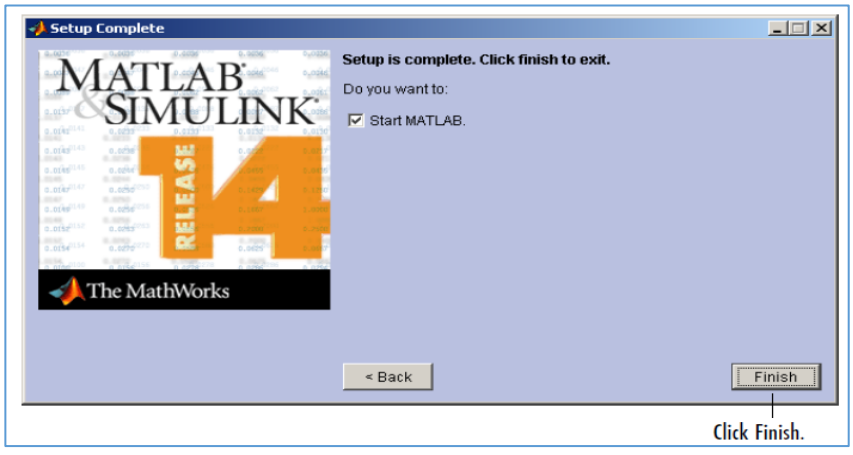
8. Confirmer l'installation et Cliquer sur Next;



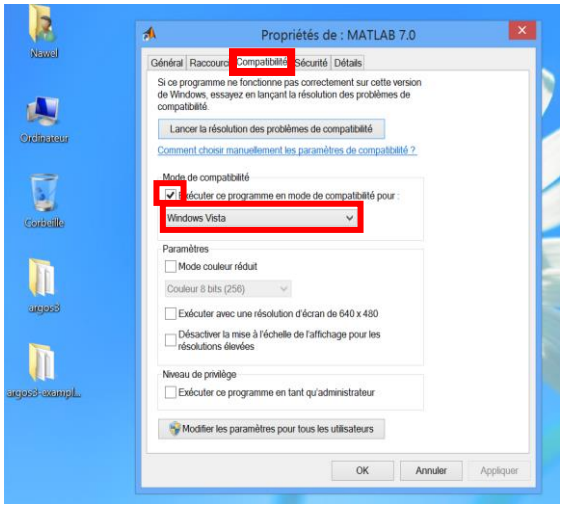
9. Cliquer sur Skip CD 2. Puis sur Next sur la nouvelle fenêtre qui s'affiche ;



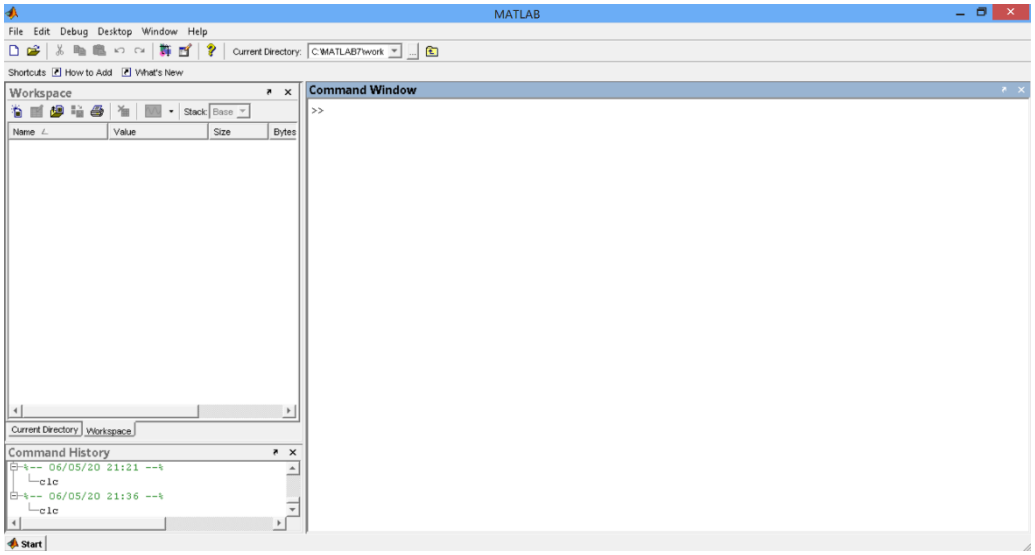
10. Compléter l'installation en Cliquant sur Finish;



11. Apres l'installation, si MATLAB s'affiche correctement c'est bien, sinon, aller sur l'icône de MATLAB dans le bureau et cliquer à **droite** vous aurez le menu dans la figure, puis cliquer sur propriétés. Aller sur l'onglet compatibilité, puis choisir Windows Vista comme mode de compatibilité ;



12. Double clic sur l'icône MATLAB pour lancer, et voilà MATLAB est bien installé sur votre machine ☺



Bibliographie

1. Attaway, S. (2013). *Matlab: a practical introduction to programming and problem solving*. Butterworth-Heinemann.
2. Koko, J. (2009). *Calcul scientifique avec MATLAB: outils MATLAB spécifiques, équations aux dérivées partielles*. Ellipses.
3. Sizemore, J., & Mueller, J. P. (2014). *MATLAB for Dummies*. John Wiley & Sons.
4. Biran, A., & Breiner, M. (2004). *MATLAB pour l'ingenieur*. Pearson Education.
5. Merrien, J. L. (2007). *Exercices et problèmes d'Analyse numérique avec Matlab: Rappels de cours, corrigés détaillés, méthodes*. Dunod.
6. Grenier, J. P. (2007). *Débuter en algorithmique avec Matlab et Scilab*. Ellipses.
7. Bastien, J., & Martin, J. N. (2003). *Introduction à l'analyse numérique: applications sous Matlab: cours et exercices corrigés*. Dunod.
8. Mokhtari, M., & Mesbah, A. (1997). *Apprendre et maîtriser MATLAB* (No. BOOK). Springer-Verlag.