REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Djilali Bounaama Khemis Miliana
Faculté des Sciences et de la Technologie
Conseil Scientifique de la Faculté

جامعة جيلالي بونعامة خميس مليانة
كليّة العلوم والتكنولوجيا
المجلس العلمي للكلية

Ref :05/ 30/CSF/2023

EXTRAIT DU PV
DE LA REUNION ORDINAIRE DU CONSEIL SCIENTIFIQUE
Du 24/10/2023

Objet : Validation des cours en ligne

L'an deux mille vingt-trois (2023) et le vingt-quatre (24) du mois d'octobre, une réunion ordinaire du Conseil Scientifique de la Faculté des Sciences et de la Technologie s'est tenue à 10h à la salle de réunion de la faculté.

Suite aux rapports favorables reçus de la part du CSD, le CSF a validé la conformité des cours en ligne comme suit :

- **Département de mathématiques et informatique**

Enseignant	Intitulé du cours	Lien du cours
Dr. MEGHATRIA Farida	Outils de programmation pour les mathématiques	http://elearning.univ-dbkm.dz/cours/view.php?id=10910
Dr. MEGHATRIA Farida	Application des mathématiques aux autres sciences.	http://elearning.univ-dbkm.dz/cours/view.php?id=10929

Président du
Conseil Scientifique de la Faculté

Dr. CHAOUCHI Belkacem

رئيس المجلس العلمي
إمضاء : د.شاوشي بلقاسم

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Djilali BOUNAAMA — Khemis Miliana

Faculté des Sciences et de la Technologie
Département des Mathématiques et Informatique

Mathématiques et Informatique · Première année · 2022–2023

جامعة الجيلالي بونعامة خميس مليانة
المجلس العلمي
كلية العلوم والتكنولوجيا

POLYCOPIÉ DE COURS

Outils de Programmation
pour les Mathématiques
'MATLAB'

Ce polycopié introduit les fondements du logiciel MATLAB et son utilisation comme outil de programmation appliqué aux mathématiques, à destination des étudiants de première année.

NIVEAU	SPÉCIALITÉ	SEMESTRE
1ère année	**Maths & Informatique**	**S2**

AUTEURE

Meghatria Farida

Enseignante — Département Maths & Info

جامعة الجيلالي بونعامة خميس مليانة
المجلس العلمي
كلية العلوم والتكنولوجيا

VALIDATION SCIENTIFIQUE

CSD : .....

CSF : .....

Année universitaire 2022–2023 · Université Djilali BOUNAAMA, Khemis Miliana, Algérie

Table des matières

Avant-propos 3

Introduction générale 4

1 Maîtrise du Logiciel MATLAB **7**

1.1 Introduction à l'environnement MATLAB 8

1.1.1 L'Interface de MATLAB 8

1.1.2 Valeurs Logiques (Logical / Boolean Values) 11

1.1.3 Constantes, Fonctions et Commandes Principales 13

1.2 Les Vecteurs et les Matrices 15

1.2.1 Les Vecteurs 16

1.2.2 Les Matrices 20

1.3 Introduction à la Programmation avec MATLAB 25

1.3.1 Généralités — Scripts et l'Éditeur 25

1.3.2 Structures de Contrôle de Flux 27

1.3.3 Les Fonctions en MATLAB 30

1.4 Graphiques et Visualisation des Données en MATLAB 33

1.4.1 La Fonction plot — Tracés 2D 34

1.4.2 Plusieurs Courbes dans une Même Figure 36

1.4.3 Plusieurs Courbes dans la Même Fenêtre — subplot 38

2 Exemples d'Application et Techniques de Résolution **42**

2.1 Applications simples sur la Command Window 44

2.1.1 Opérations arithmétiques de base 44

2.1.2 Application 1 – Commandes de base de MATLAB 45

2.1.3 Application 2 – Variables et opérateurs 47

2.1.4 Application 3 – Évaluation de formules mathématiques 49

2.1.5 Manipulation des vecteurs 51

2.1.6 Application 4 – Manipulation des vecteurs 52

2.1.7 Manipulation des matrices 55

2.1.8 Application 5 – Manipulation des matrices (I) 57

2.1.9 Application 6 – Manipulation des matrices (II) 59
2.1.10 Opérations élémentaires sur les tableaux 61
2.1.11 Application 7 – Chaînes de caractères 62
2.2 Programmation MATLAB 66
2.2.1 Les structures de contrôle 66
2.2.2 Scripts MATLAB 68
2.2.3 Application 8 – Branchement conditionnel (if/elseif/else) . . . 70
2.2.4 Application 9 – Boucle `for` : calcul de somme 72
2.2.5 Application 10 – Boucles imbriquées : table de multiplication . 73
2.2.6 Application 11 – Script : somme et produit de deux matrices . 75
2.2.7 Fonctions MATLAB 76
2.2.8 Application 12 – Fonctions MATLAB 79
2.2.9 Application 13 – Racines d'un polynôme 81
2.2.10 Exercices de synthèse 82
2.2.11 Application 14 – Représentation graphique 2D 83
2.2.12 Application 15 – Représentation graphique 3D 84
2.3 Récapitulatif des commandes MATLAB essentielles 85

جامعة يحيى فارس بالمدية
المجلس العلمي
كلية العلوم و التكنولوجيا

Avant-propos

Ce polycopié est destiné aux étudiants de première année de Licence en Mathématiques et Informatique. Il vise à initier l'apprenant à l'usage des outils de calcul scientifique, dont l'importance est devenue capitale pour tout chercheur ou ingénieur depuis les années 1970. Le cours suppose une familiarité avec les fondamentaux de l'algorithmique, de l'algèbre, de l'analyse et des statistiques.

La structure pédagogique alterne, pour chaque chapitre, une séance théorique de 1h30 et une séance de travaux pratiques (TP). Cette approche appliquée repose sur l'utilisation du langage MATLAB, développé par *MathWorks*, qui s'impose aujourd'hui comme un standard incontournable en programmation scientifique.

Pédagogiquement, ce cours privilégie une compréhension intuitive des concepts plutôt qu'un exposé formel et rigoureux. Seuls les éléments essentiels à la maîtrise technique seront présentés ; des références bibliographiques seront systématiquement indiquées pour les étudiants souhaitant approfondir les fondements mathématiques.

La particularité de MATLAB réside dans le traitement exclusif de variables numériques en double précision (codées sur 64 bits).

— **Plage numérique :** Les valeurs traitées s'étendent environ de 10^{-306} à 10^{306}. Au-delà, elles sont notées `Inf` ; en deçà, elles sont considérées comme nulles (`0`).

— **Indéterminations :** En cas d'absence de valeur ou de résultat indéfini (ex : $\frac{0}{0}$), le logiciel renvoie `NaN` (*Not a Number*).

Introduction générale

Ce document constitue une introduction à MATLAB, un logiciel de calcul scientifique. Il a pour objectif de préparer l'étudiant aux travaux pratiques d'Analyse Numérique et d'optimisation mathématique, dans lesquels cet outil est intensivement utilisé pour la mise en application et la simulation des principes théoriques présentés en cours. Par ailleurs, ce manuel offre à l'étudiant la possibilité de se former à un logiciel professionnel largement répandu.

Qu'est-ce que MATLAB ?

MATLAB est un logiciel de calcul numérique commercialisé par la société MathWorks. Il a été initialement développé à la fin des années 1970 par **Cleve Moler**, professeur de mathématiques à l'université du Nouveau-Mexique puis à Stanford, afin de permettre aux étudiants de travailler avec un outil de programmation de haut niveau sans avoir à apprendre le Fortran ou le C.

Le nom MATLAB signifie *Matrix Laboratory*. Il s'agit d'un langage dédié au calcul scientifique, à l'analyse de données, à leur visualisation et au développement d'algorithmes. Son interface propose :

— une fenêtre interactive (console) pour l'exécution de commandes ;
— un environnement de développement intégré (IDE) pour la programmation d'applications.

MATLAB trouve ses applications dans de nombreuses disciplines. Il constitue un outil numérique puissant pour la modélisation de systèmes physiques, la simulation de modèles mathématiques, la conception et la validation d'applications. Le logiciel de base peut être enrichi par de multiples *toolboxes* (boîtes à outils), c'est-à-dire des bibliothèques de fonctions dédiées à des domaines particuliers, tels que le traitement du signal, l'analyse statistique ou l'optimisation.

Aperçu des *toolboxes* disponibles

Le tableau suivant présente une sélection non exhaustive des *toolboxes* disponibles, illustrant la diversité des fonctionnalités offertes par MATLAB :

Toolbox	Toolbox	Toolbox
Control System Toolbox	Symbolic Math Toolbox	Signal Processing Toolbox
Neural Network Toolbox	Optimization Toolbox	Parallel Computing Toolbox
Statistics Toolbox	Fuzzy Logic Toolbox	Image Processing Toolbox
System Identification Toolbox	Data Acquisition Toolbox	Bioinformatics Toolbox
MATLAB Compiler	Financial Toolbox	Model-Based Calibration

Objectifs du document

Ce document propose une introduction à MATLAB et développe quelques exemples d'applications. Il ne constitue en aucun cas une documentation exhaustive du logiciel ; toutefois, les principales notions sont présentées et invitent l'étudiant à rechercher par lui-même les informations complémentaires nécessaires à la réalisation de ses projets.

L'objectif est double :

1. Acquérir une connaissance suffisante du logiciel pour pouvoir travailler efficacement et comprendre les notions abordées dans les travaux pratiques d'Analyse Numérique et d'optimisation.
2. Offrir une initiation consistante à un logiciel puissant, très largement utilisé dans le monde industriel et académique (laboratoires, universités, écoles).

Ressources complémentaires

En plus de l'aide intégrée à l'environnement et des nombreux ouvrages dédiés, une abondante documentation est disponible en ligne :

— **Documentation officielle MathWorks** :
http://www.mathworks.com/help/techdoc/
— **Espaces d'entraide** :
  — Developpez.com : http://matlab.developpez.com/
  — Matlab Central : http://www.mathworks.com/matlabcentral/

Logiciels alternatifs

Le coût d'une licence MATLAB de type éducation (*classroom*) est relativement accessible (environ 100 €), mais celui d'une version industrielle est nettement plus élevé (environ 2 500 €). À cela s'ajoute un coût supplémentaire pour chaque *toolbox* commandée. [1]

Il existe cependant plusieurs alternatives libres et gratuites :

1. Les valeurs données ne sont que des ordres de grandeur et peuvent changer. Les prix dépendent du type de licence (éducation, industriel, recherche) et du format (individuel, groupe, concurrent).

Scilab

Scilab est un logiciel open-source sous licence GPL, développé depuis 1990 par des chercheurs de l'INRIA (*Institut National de Recherche en Informatique et en Automatique*) et maintenu par la fondation de coopération scientifique Digiteo. Il est disponible sur les plateformes Windows, Mac OS X, Linux et BSD.

— Site officiel : http://www.scilab.org/

GNU Octave

Octave est également un logiciel open-source sous licence GPL, dont le développement a débuté au début des années 1990 par **John W. Eaton** dans le cadre du projet GNU. Sa syntaxe est très proche de celle de MATLAB, ce qui en fait un substitut pratique. Il est disponible sur les plateformes Windows, Mac OS X, Linux et BSD.

— Site officiel : http://www.gnu.org/software/octave/

Organisation du cours

Ce cours est divisé en deux parties :

1. **Chapitre 1** — Introduction à l'interface de MATLAB : écriture du premier programme, structure des données et opérations de base ($+$, $-$, $\times$, $\div$). Les différents types de variables sont présentés et exercés selon différents opérateurs (rationnels et relationnels), sans négliger les vecteurs et les matrices qui constituent la base du logiciel. Une part importante de ce chapitre est consacrée à la présentation de données sous forme de graphiques en 2D.
2. **Chapitre 2** — Maîtrise de MATLAB avec ses applications dans la résolution de problèmes de mathématiques appliquées : syntaxe, opérations sur les vecteurs et les matrices, scripts et fonctions.

Une bonne connaissance pratique du langage MATLAB permettra non seulement de comprendre en profondeur les concepts et les algorithmes de recherche, mais aussi d'augmenter la capacité à réaliser un travail créatif et à s'attaquer activement aux problèmes dans les cours connexes.

Chapitre 1

Maîtrise du Logiciel MATLAB

Contenu du chapitre

- Introduction à l'envirenement Matlab
- Les vecteurs et les matrices
- Introduction à la programmation avec Matlab
- Graphique et visualisation en Matlab

MATLAB (MATrix LABoratory) est un environnement de calcul numérique haute performance et un langage de programmation développé par MathWorks. Conçu à l'origine pour les calculs matriciels, il est devenu une plateforme polyvalente utilisée dans les domaines de l'ingénierie, des mathématiques appliquées, de la physique, des statistiques et de l'informatique. Ce chapitre offre une base complète — de la navigation dans l'interface à l'écriture de programmes complets.

1.1 Introduction à l'environnement MATLAB

Objectifs d'apprentissage

- Identifier et utiliser tous les panneaux principaux de l'interface MATLAB.
- Exécuter des expressions arithmétiques et symboliques dans la Fenêtre de Commandes.
- Comprendre les types de données fondamentaux de MATLAB : scalaires, chaînes de caractères et valeurs logiques.
- Mémoriser et appliquer les constantes et fonctions intégrées les plus importantes.
- Utiliser avec assurance les commandes de gestion de l'espace de travail.

1.1.1 L'Interface de MATLAB

Lors du lancement de MATLAB, un environnement de développement intégré (IDE) composé de plusieurs panneaux fonctionnels s'affiche. Comprendre le rôle de chaque panneau est la première étape vers une utilisation productive du logiciel.

Panneaux principaux de l'interface

Panneau	Rôle principal	Actions clés
Fenêtre de Commandes	Console interactive — exécution immédiate des commandes	Taper l'expression + Entrée ; utiliser `clc` pour effacer
Espace de travail	Affiche toutes les variables actuellement en mémoire	Double-clic pour inspecter ; clic droit pour supprimer
Historique des commandes	Journal des commandes exécutées précédemment	Double-clic pour ré-exécuter ; Ctrl+R pour rechercher
Éditeur	Rédiger, enregistrer et déboguer des fichiers script/fonction `.m`	Ctrl+N nouveau fichier ; F5 pour exécuter ; F9 pour exécuter la sélection
Dossier courant	Explorateur de fichiers pour le répertoire de travail actif	Double-clic pour ouvrir ; clic droit pour importer des données
Navigateur d'aide	Documentation complète et recherchable	Taper `doc nomFonction` dans la Fenêtre de Commandes

CONSEIL

- Appuyez sur **Tab** dans la Fenêtre de Commandes pour l'auto-complétion des noms de variables et de fonctions.
- Appuyez sur la touche **Flèche vers le haut** pour rappeler les commandes précédentes — très utile pour les tests itératifs.
- Tapez `help sin` pour une description concise ; tapez `doc sin` pour la documentation complète avec des exemples.

— Gardez votre Dossier courant pointé vers votre répertoire de travail pour éviter les erreurs `'fichier introuvable'`.

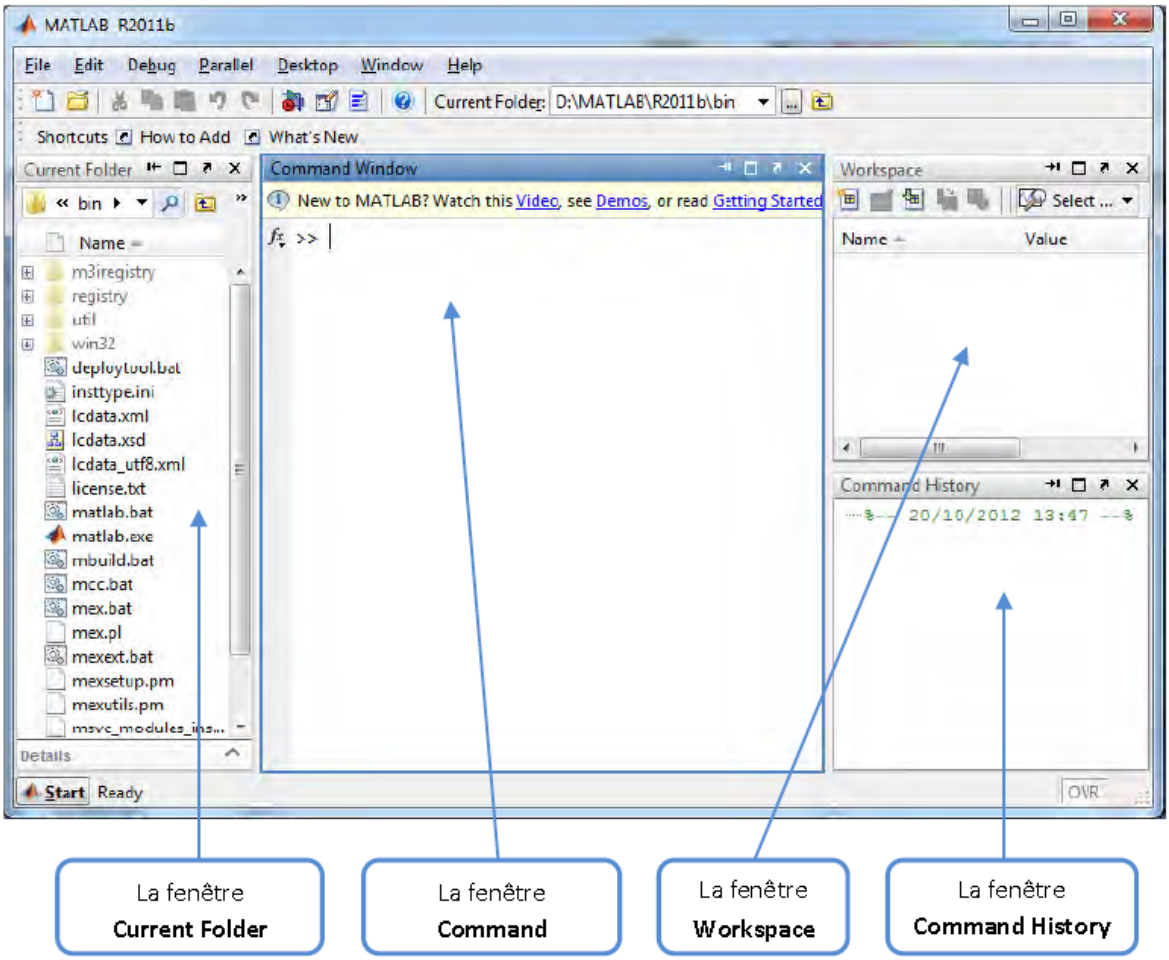


FIGURE 1.1 – L'environnement MATLAB (Version 2010b)

La Fenêtre de Commandes — Calcul interactif

La Fenêtre de Commandes est l'interface principale de MATLAB pour le calcul direct. Considérez-la comme une puissante calculatrice scientifique : vous saisissez une expression et appuyez sur Entrée pour obtenir un résultat immédiat. Chaque résultat est automatiquement stocké dans la variable `ans` s'il n'est pas explicitement affecté.

```
% -- Arithm tique de base
   ------------------------------------------------
>> 3 + 7
ans = 10
>> 2^10            % exponentiation
ans = 1024
>> sqrt(144)       % racine carr e
ans = 12
```

```
% -- Affectation de variables
   ----------------------------------------
>> x = 15
x = 15
>> y = x * 2 + 1
y = 31
>> z = x + y;     % le point-virgule supprime l'affichage

% -- Chaînes de caractères (tableau de caractères)
   ----------------------
>> name = 'MATLAB'
name = MATLAB
>> greeting = ['Bonjour, ' name ' !']
greeting = Bonjour, MATLAB !
```

NOTE

— L'invite » indique que MATLAB est prêt pour la saisie.
— Le point-virgule ; en fin de ligne supprime l'affichage — indispensable à l'intérieur des boucles.
— Les noms de variables sont sensibles à la casse : `Temperature`, `temperature` et `TEMPERATURE` sont trois variables distinctes.
— `ans` est écrasé à chaque fois qu'un résultat n'est pas explicitement affecté — ne vous y fiez pas entre plusieurs opérations successives.

Aperçu des types de données

Type de données	Description & Exemple
`double` (par défaut)	Nombre en virgule flottante : `x = 3.14`
Types entiers	`int8, int16, int32, int64, uint8` ... : `n = int32(100)`
`char`	Chaîne de caractères : `s = 'bonjour'`
`string`	Objet chaîne (R2016b+) : `s = "bonjour"`
`logical`	Booléen vrai/faux : `flag = true`
Tableau de cellules	Conteneur pour types mixtes : `c = {1, 'a', [1 2]}`
`struct`	Champs nommés : `s.x = 1; s.nom = 'A'`
Descripteur de fonction	Référence à une fonction : `f = @(x) x.^2`

EXERCICE

Exercice 1.1 — Exploration de la Fenêtre de Commandes

1. Calculer `(7^3 - 4*9) / sqrt(25)` et stocker le résultat dans une variable appelée `result`.
2. Créer une variable chaîne `universite = 'Mon Université'`.

3. Afficher le type de `result` en utilisant la fonction `class()`.
4. Utiliser `whos` pour inspecter toutes les variables actuelles dans l'espace de travail.

SOLUTION

```
>> result = (7^3 - 4*9) / sqrt(25)  % = (343-36)/5 = 61.4
result = 61.4000

>> universite = 'Mon Universit '
universite = Mon Universit

>> class(result)
ans = double

>> whos
  Nom           Taille   Octets  Classe
  result        1x1           8  double
  universite    1x14         28  char
```

1.1.2 Valeurs Logiques (Logical / Boolean Values)

Les valeurs logiques constituent le fondement de la prise de décision dans les programmes MATLAB. Une valeur logique est soit `true` (représentée en interne par 1) soit `false` (représentée par 0). Elles sont produites par des opérateurs relationnels et logiques, et utilisées par les structures de contrôle telles que les instructions `if` et les boucles `while`.

Opérateurs relationnels

Opérateur	Signification	Exemple (a=5, b=3)
==	Égal à	`a == b` → 0 (faux)
~=	Différent de	`a ~= b` → 1 (vrai)
<	Inférieur à	`a < b` → 0 (faux)
>	Supérieur à	`a > b` → 1 (vrai)
<=	Inférieur ou égal	`b <= 5` → 1 (vrai)
>=	Supérieur ou égal	`a >= 5` → 1 (vrai)

Opérateurs logiques

Opérateur	Comportement & Exemple
`&&` (ET court-circuit)	`(a>0) && (b>0)` → 1 ; s'arrête au premier faux
`||` (OU court-circuit)	`(a>10)||(b>0)` → 1 ; s'arrête au premier vrai

Opérateur	Comportement & Exemple
`&` (ET élément par élément)	`[1 0 1] & [1 1 0]` → `[1 0 0]`
`|` (OU élément par élément)	`[1 0 1] | [0 1 0]` → `[1 1 1]`
`~` (NON)	`~(a == b)` → 1
`xor(A,B)`	`xor(1,0)` → 1 ; `xor(1,1)` → 0

```
>> a = 5; b = 3;
>> a > b
ans = 1
>> a == b
ans = 0
>> (a > 0) && (b > 0)    % les deux sont positifs ?
ans = 1
>> ~(a == b)              % a est-il DIFF RENT de b ?
ans = 1

% Valeurs logiques appliqu es aux vecteurs
>> v = [1 -2 3 -4 5];
>> v > 0                  % comparaison    lment    par
      lment
ans = 1   0   1   0   1
>> v(v > 0)               % extraire uniquement les    lments
    positifs
ans = 1   3   5
```

AVERTISSEMENT

- Utilisez `==` pour la comparaison, jamais `=` (qui est une affectation). Écrire `if x = 5` provoque une erreur de syntaxe.
- Utilisez `&&` et `||` pour les conditions scalaires dans `if`/`while` ; utilisez `&` et `|` pour les opérations élément par élément sur les tableaux.
- Les comparaisons avec NaN retournent toujours faux : `NaN == NaN` donne 0. Utilisez `isnan()` à la place.

Fonctions logiques utiles

Fonction	Rôle & Exemple
`any(v)`	Vrai si au moins un élément est non nul : `any([0 0 1])` → 1
`all(v)`	Vrai si tous les éléments sont non nuls : `all([1 1 1])` → 1
`isnan(x)`	Détecte les valeurs NaN
`isinf(x)`	Détecte les valeurs Inf
`isempty(x)`	Vrai si le tableau ne contient aucun élément
`isnumeric(x)`	Vrai si le type est numérique

Fonction	Rôle & Exemple
`ischar(x)`	Vrai si c'est un tableau de caractères
`logical(x)`	Convertit un tableau numérique en type logique

EXERCICE

Exercice 1.2 — Expressions logiques

Avec `x = 12` et `y = -5` :

1. Déterminer si `x` est positif ET `y` est négatif.
2. Déterminer si au moins l'un de `x` ou `y` est supérieur à 10.
3. Créer un vecteur `v = [-3 0 4 -1 7]` et extraire uniquement les éléments strictement positifs.

SOLUTION

```
>> x = 12; y = -5;
>> (x > 0) && (y < 0)
ans = 1     % vrai : x>0 et y<0

>> (x > 10) || (y > 10)
ans = 1     % vrai : x>10

>> v = [-3 0 4 -1 7];
>> v(v > 0)
ans = 4   7
```

1.1.3 Constantes, Fonctions et Commandes Principales

MATLAB fournit un riche ensemble de constantes mathématiques prédéfinies, de fonctions intégrées et de commandes de gestion de l'environnement. La maîtrise de ces éléments accélère considérablement la résolution de problèmes.

Constantes mathématiques

Constante	Valeur & Usage
`pi`	$\pi \approx 3{,}14159265358979$ — utilisé en trigonométrie, géométrie
`exp(1)`	$e \approx 2{,}71828$ — nombre d'Euler ; base du logarithme naturel
`Inf`/`inf`	Infini positif — résultat par exemple de `1/0`
`NaN`	Not a Number (pas un nombre) — résultat de `0/0` ou `Inf-Inf`
`eps`	Epsilon machine $\approx 2{,}22 \times 10^{-16}$ — plus petite différence représentable
`true`/`false`	1 / 0 logique — plus clair que 1/0 numérique pour les booléens

Constante	Valeur & Usage
`i` ou `j`	Unité imaginaire $\sqrt{-1}$ — pour les nombres complexes

Fonctions mathématiques essentielles

Fonction	Description	Exemple
`abs(x)`	Valeur absolue	`abs(-8) = 8`
`sqrt(x)`	Racine carrée	`sqrt(49) = 7`
`exp(x)`	e à la puissance x	`exp(2)` $\approx 7{,}389$
`log(x)`	Logarithme naturel ln	`log(exp(3)) = 3`
`log2(x)`	Logarithme en base 2	`log2(8) = 3`
`log10(x)`	Logarithme en base 10	`log10(1000) = 3`
`sin/cos/tan(x)`	Trigo (radians)	`sin(pi/2) = 1`
`asin/acos/atan(x)`	Trigo inverse	`asin(1) = pi/2`
`atan2(y,x)`	Arctangente quatre cadrans	`atan2(1,1) = pi/4`
`floor(x)`	Arrondi vers $-\infty$	`floor(3.9) = 3`
`ceil(x)`	Arrondi vers $+\infty$	`ceil(3.1) = 4`
`round(x)`	Arrondi à l'entier le plus proche	`round(3.5) = 4`
`mod(a,n)`	Reste de la division modulo	`mod(17,5) = 2`
`rem(a,n)`	Reste (signe suit `a`)	`rem(-7,3) = -1`
`sign(x)`	Signe : -1, 0 ou $+1$	`sign(-4) = -1`
`factorial(n)`	$n!$ factorielle	`factorial(5) = 120`
`nchoosek(n,k)`	Coefficient binomial $\binom{n}{k}$	`nchoosek(5,2) = 10`

Commandes de gestion de l'espace de travail

```
>> who            % liste les noms de variables de l'espace de
    travail
>> whos           % liste les variables avec taille, type,
   m moire
>> clear x y      % supprimer les variables x et y
>> clear          % supprimer TOUTES les variables (
   utiliser avec pr caution !)
>> clc            % effacer l'affichage de la Fen tre de
   Commandes
>> close all      % fermer toutes les fen tres graphiques
>> pwd            % afficher le r pertoire de travail courant
>> ls             % lister les fichiers dans le dossier
   courant
>> cd 'chemin'    % changer de r pertoire courant
>> save('donn es.mat', 'x', 'y')  % sauvegarder les
   variables dans un fichier .mat
>> load('donn es.mat')              % charger les variables
   depuis un fichier .mat
```

CONSEIL

— Commencez toujours les scripts par `clc; clear; close all;` — cela garantit un état propre et des résultats reproductibles.
— Utilisez `diary('fichier.txt')` pour enregistrer votre session de Fenêtre de Commandes dans un fichier texte.
— Utilisez `format long` pour afficher plus de décimales ; `format short` (par défaut) pour une sortie compacte.

Formatage des nombres

```
>> format short       % d faut : 4 d cimales
>> pi
ans = 3.1416

>> format long        % 15 chiffres significatifs
>> pi
ans = 3.141592653589793

>> format rat         % approximation rationnelle
>> pi
ans = 355/113

>> format short e     % notation scientifique
>> 0.000123
ans = 1.2300e-04
```

RÉSUMÉ DE SECTION

— L'IDE MATLAB contient : Fenêtre de Commandes, Espace de travail, Éditeur, Historique des commandes et Dossier courant.
— Les noms de variables sont sensibles à la casse ; le point-virgule supprime l'affichage ; `ans` contient le dernier résultat non affecté.
— Valeurs logiques : 1 = vrai, 0 = faux. Utilisez `==` pour la comparaison, pas `=` (qui affecte).
— Utilisez `&&` / `||` pour la logique scalaire, `&` / `|` pour la logique élément par élément sur les tableaux.
— MATLAB fournit `pi`, `Inf`, `NaN`, `eps` comme constantes intégrées.
— Commandes clés de l'espace de travail : `clc`, `clear`, `who`, `whos`, `save`, `load`.

1.2 Les Vecteurs et les Matrices

Objectifs d'apprentissage

— Créer des vecteurs ligne et colonne en utilisant plusieurs méthodes.
— Appliquer des opérations arithmétiques élément par élément et sur des vecteurs entiers.
— Construire des matrices et accéder aux éléments par indexation.

— Distinguer les opérateurs élément par élément (`.* ./ .^`) des opérateurs matriciels (`* / ^`).
— Appliquer les opérations matricielles fondamentales : transposition, inversion, déterminant, résolution de systèmes linéaires.

MATLAB signifie MATrix LABoratory — les matrices constituent sa structure de données native. Toute valeur dans MATLAB est fondamentalement une matrice : un scalaire est une matrice 1×1, un vecteur ligne est une matrice $1 \times n$, et un vecteur colonne est une matrice $n \times 1$. La maîtrise des opérations sur les vecteurs et les matrices est la compétence centrale de la programmation MATLAB.

1.2.1 Les Vecteurs

Un vecteur est une collection ordonnée unidimensionnelle de nombres. Il en existe deux orientations : les vecteurs ligne (1 ligne, n colonnes) et les vecteurs colonne (n lignes, 1 colonne).

√ Création de vecteurs ligne

```
% Méthode 1 : énumérer les éléments entre crochets
>> v = [3 7 2 9 1]
v = 3   7   2   9   1

% Méthode 2 : Opérateur deux-points début : pas : fin
>> x = 1:5            % pas par défaut = 1
x = 1   2   3   4   5
>> x = 0:0.25:1       % pas personnalisé de 0.25
x = 0   0.2500   0.5000   0.7500   1.0000
>> x = 10:-2:0        % décroissant par 2
x = 10   8   6   4   2   0

% Méthode 3 : linspace --- contrôle du nombre de points
>> x = linspace(0, 1, 6)   % 6 points équidistants
x = 0   0.2000   0.4000   0.6000   0.8000   1.0000
```

CONSEIL

— Utilisez l'opérateur deux-points lorsque le pas importe (ex. : pas de temps dans une simulation).
— Utilisez `linspace` lorsque le nombre de points de données importe (ex. : tracé avec 1000 points lisses).
— `linspace(a, b, n)` inclut toujours les deux extrémités a et b.

√ Création de vecteurs colonne

```
% Séparer les éléments par des points-virgules
>> w = [10; 20; 30; 40]
w =
    10
    20
    30
```

```
    40

% Transposer un vecteur ligne avec l'op rateur apostrophe '
>> v = [1 2 3 4 5];
>> col = v'
col =
    1
    2
    3
    4
    5
```

NOTE

— L'apostrophe ' calcule la transposée conjuguée complexe. Pour les vecteurs réels, les deux sont équivalents.
— Pour les vecteurs complexes, utilisez .' (point-apostrophe) pour la transposée simple sans conjugaison.

√ Indexation des vecteurs — Accès et modification des éléments

MATLAB utilise une indexation basée sur 1 : le premier élément a l'indice 1 (et non 0 comme en C, Java ou Python). C'est l'une des sources les plus fréquentes d'erreurs de décalage pour les étudiants venant d'autres langages.

```
>> v = [10 20 30 40 50 60];
>> v(1)          % premier    lment
ans = 10
>> v(4)          % quatri me    lment
ans = 40
>> v(end)        % dernier    lment    --- 'end' est un mot-cl
ans = 60
>> v(end-1)      % avant-dernier
ans = 50
>> v(2:4)        %    lments    2, 3, 4 (sous-vecteur)
ans = 20   30   40
>> v([1 3 5])    %    lments       des indices sp cifiques
ans = 10   30   50

% Modifier un    lment
>> v(3) = 99
v = 10   20   99   40   50   60

%   tendre   un vecteur en affectant au-del   de sa longueur
>> v(8) = 100   % remplit le vide avec des z ros
v = 10   20   99   40   50   60   0   100
```

AVERTISSEMENT

— `v(0)` génère une erreur — les indices MATLAB commencent à 1, jamais à 0.
— Accéder à `v(10)` quand `v` n'a que 6 éléments produit l'erreur `'Index dépasse les dimensions de la matrice'`.

— Affecter à `v(8)` quand `v` a 6 éléments remplit silencieusement le vide avec des zéros — faites-le délibérément.

√ Opérations arithmétiques sur les vecteurs

Quand deux vecteurs ont la même taille, les opérations élément par élément appliquent l'opérateur à chaque paire d'éléments correspondants. Les opérateurs préfixés par un point (`.*`, `./`, `.^`) sont les versions élément par élément.

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

% Addition et soustraction (toujours lment par lment)
>> a + b
ans = 6   8   10   12
>> b - a
ans = 4   4   4   4

% Multiplication lment par lment (noter le point)
>> a .* b
ans = 5   12   21   32

% Division lment par lment
>> b ./ a
ans = 5.0000   3.0000   2.3333   2.0000

% Puissance lment par lment
>> a .^ 2
ans = 1   4   9   16

% Scalaire appliqu tout le vecteur
>> 3 * a       % multiplier chaque lment par 3
ans = 3   6   9   12
>> a + 10      % ajouter 10 chaque lment
ans = 11   12   13   14

% Fonctions appliqu es automatiquement lment par
      lment
>> sqrt(a)
ans = 1.0000   1.4142   1.7321   2.0000
```

IMPORTANT

— `.*` est la multiplication élément par élément ; `*` est la multiplication matricielle (règles du produit scalaire).
— Pour deux vecteurs ligne `a` et `b` de même longueur, `a * b` génère une erreur ; `a .* b` fonctionne élément par élément.
— Pour calculer le produit scalaire de deux vecteurs colonne `u` et `v`, utilisez `u' * v` ou `dot(u, v)`.

√ Fonctions utiles pour les vecteurs

Fonction	Description	Exemple
`length(v)`	Nombre d'éléments	`length([1 2 3]) = 3`
`numel(v)`	Total des éléments	`numel([1 2 3]) = 3`
`sum(v)`	Somme de tous les éléments	`sum([1 2 3 4]) = 10`
`prod(v)`	Produit de tous les éléments	`prod([1 2 3 4]) = 24`
`min(v)`	Valeur minimale	`min([4 2 7]) = 2`
`max(v)`	Valeur maximale	`max([4 2 7]) = 7`
`mean(v)`	Moyenne arithmétique	`mean([1 2 3]) = 2`
`std(v)`	Écart-type	`std([1 2 3])` ≈ 1
`sort(v)`	Tri croissant	`sort([3 1 2]) = [1 2 3]`
`flip(v)`	Ordre inverse	`flip([1 2 3]) = [3 2 1]`
`cumsum(v)`	Somme cumulée	`cumsum([1 2 3]) = [1 3 6]`
`dot(a,b)`	Produit scalaire	`dot([1 2],[3 4]) = 11`
`norm(v)`	Norme euclidienne $\|v\|$	`norm([3 4]) = 5`
`unique(v)`	Éléments uniques triés	`unique([3 1 3 2]) = [1 2 3]`

EXERCICE

Exercice 1.3 — Opérations sur les vecteurs

1. Créer un vecteur `t` contenant 50 valeurs équidistantes de 0 à 2π.
2. Calculer `y = sin(t)` et `z = cos(t)` élément par élément.
3. Vérifier l'identité de Pythagore : confirmer que `y.^2 + z.^2` est égal à 1 pour chaque élément.
4. Trouver la valeur maximale de `y` et l'indice auquel elle se produit.

SOLUTION

```
>> t = linspace(0, 2*pi, 50);
>> y = sin(t);
>> z = cos(t);

% V rifier l'identit  --- doit  tre  ~1 partout
>> check = y.^2 + z.^2;
>> max(abs(check - 1))       %  cart  maximal par rapport
      1
ans = 2.2204e-16             % essentiellement z ro (
   pr cision machine)

>> [maxVal, idx] = max(y)
maxVal = 1.0000
idx = 13                     % indice du maximum
```

```
>> t(idx)                          % valeur de t correspondante
ans = 1.5394   (approximativement pi/2)
```

1.2.2 Les Matrices

Une matrice est un tableau rectangulaire bidimensionnel de nombres à m lignes et n colonnes. Les matrices codent des transformations linéaires, stockent des données tabulaires et apparaissent dans quasiment tous les domaines des mathématiques appliquées — des systèmes linéaires au traitement d'images.

√ Création de matrices

```
% Saisie directe : lignes s par es par ;  colonnes par
   espaces ou virgules
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
   1   2   3
   4   5   6
   7   8   9

% Matrices sp ciales
>> zeros(3, 4)        % matrice 3x4 de z ros
>> ones(2, 5)         % matrice 2x5 de uns
>> eye(4)             % matrice identit  4x4
>> rand(3)            % valeurs al atoires uniformes dans
   [0,1] (3x3)
>> randn(3)           % valeurs al atoires normales standard
   (3x3)
>> diag([1 2 3])      % matrice diagonale 3x3

% Construire une matrice    partir de vecteurs
>> x = 1:3; y = 4:6;
>> M = [x; y]
M =
   1   2   3
   4   5   6
```

√ Taille, dimensions et forme

```
>> A = [1 2 3; 4 5 6];
>> size(A)            % renvoie [lignes, cols]
ans = 2   3
>> [m, n] = size(A)      % affecter s par ment
m = 2,   n = 3
>> size(A, 1)         % nombre de lignes
ans = 2
>> size(A, 2)         % nombre de colonnes
ans = 3
>> numel(A)           % nombre total d'  lments
ans = 6
```

```
>> A(:)              % tous les    lments    en vecteur colonne
   (ordre column-major)
```

√ Indexation des matrices — A(ligne, col)

```
>> A = [10 20 30; 40 50 60; 70 80 90];
>> A(2, 3)           % ligne 2, colonne 3
ans = 60
>> A(1, :)           % toute la premi re ligne
ans = 10   20   30
>> A(:, 2)           % toute la deuxi me colonne
ans =
   20
   50
   80
>> A(1:2, 2:3)       % sous-matrice : lignes 1-2, cols 2-3
ans =
   20   30
   50   60
>> A(end, :)         % derni re ligne
ans = 70   80   90

% Modifier un bloc
>> A(2:3, 1:2) = zeros(2,2)
A =
   10   20   30
    0    0   60
    0    0   90
```

√ Opérations arithmétiques sur les matrices

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

% Addition et soustraction
>> A + B
ans =
    6    8
   10   12

% Multiplication matricielle (ligne x colonne)
>> A * B
ans =
   19   22
   43   50

% Multiplication    lment    par    lment
>> A .* B
ans =
    5   12
   21   32
```

```
% Multiplication par un scalaire
>> 3 * A
ans =
   3   6
   9   12

% Transpos e
>> A'
ans =
   1   3
   2   4

% Puissance matricielle (A^2 = A*A)
>> A^2
ans =
    7   10
   15   22

% Puissance    lment    par    lment
>> A.^2
ans =
   1   4
   9   16
```

IMPORTANT

- `A * B` est la multiplication matricielle : le nombre de colonnes de `A` doit égaler le nombre de lignes de `B`.
- `A .* B` est élément par élément : `A` et `B` doivent avoir des dimensions identiques.
- `A^n` élève la matrice à la puissance entière n (multiplication matricielle répétée).
- `A.^n` élève chaque élément à la puissance n indépendamment.

√ Fonctions matricielles

Fonction	Description	Exemple
`det(A)`	Déterminant d'une matrice carrée	`det([1 2;3 4]) = -2`
`inv(A)`	Inverse de la matrice (si det $\neq 0$)	`inv(A) * A` $\approx$ `eye(n)`
`A \ b`	Résoudre $Ax = b$ (barre oblique — recommandé)	`x = A \ b`
`rank(A)`	Rang de la matrice	`rank(eye(3)) = 3`
`trace(A)`	Somme des éléments diagonaux	`trace(eye(4)) = 4`
`eig(A)`	Valeurs propres (et vecteurs)	`[V,D] = eig(A)`
`norm(A)`	Norme matricielle (défaut : 2-norme)	`norm(A)`

Fonction	Description	Exemple
`lu(A)`	Factorisation LU	`[L,U,P] = lu(A)`
`[Q,R] = qr(A)`	Décomposition QR	`Q*R = A`
`svd(A)`	Décomposition en valeurs singulières	`[U,S,V] = svd(A)`
`diag(A)`	Extraire la diagonale en vecteur	`diag([1 2;3 4]) = [1;4]`
`triu(A)`	Partie triangulaire supérieure	zéros dans la moitié inférieure
`tril(A)`	Partie triangulaire inférieure	zéros dans la moitié supérieure
`reshape(A,m,n)`	Redimensionner sans modifier les données	`reshape(1:6,2,3)`

√ Résolution de systèmes linéaires — $Ax = b$

L'une des capacités les plus puissantes de MATLAB est la résolution du système linéaire $Ax = b$. L'opérateur barre oblique inversée (\) implémente cela efficacement en utilisant la factorisation LU — il est numériquement supérieur au calcul de `inv(A) * b`.

```
% R soudre le syst me :  2x + y  = 8
%                        x + 3y = 13
>> A = [2 1; 1 3];
>> b = [8; 13];

% M thode 1 : Barre oblique invers e (recommand e)
>> x = A \ b
x =
   2.2000
   3.6000

% V rification
>> A * x
ans =
    8.0000
   13.0000     % correspond exactement     b

% M thode 2 : Inverse (plus lent,      viter  pour les grands
    syst mes)
>> x2 = inv(A) * b
x2 =
   2.2000
   3.6000
```

CONSEIL

— Privilégiez toujours `A \ b` plutôt que `inv(A) * b` : c'est plus rapide et plus stable numériquement.

— Vérifiez le nombre de condition avec `cond(A)` : une valeur très grande indique un système mal conditionné.
— Pour les systèmes surdéterminés (plus d'équations que d'inconnues), \ calcule la solution aux moindres carrés.

EXERCICE

Exercice 1.4 — Opérations matricielles

Considérez la matrice $A = \begin{bmatrix} 4 & 7 \\ 2 & 6 \end{bmatrix}$ et le vecteur $b = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.

1. Calculer le déterminant et l'inverse de A.
2. Résoudre le système linéaire $Ax = b$.
3. Vérifier la solution en calculant $A \cdot x$ et en comparant avec b.
4. Calculer les valeurs propres de A.

SOLUTION

```
>> A = [4 7; 2 6];
>> b = [1; 3];

>> det(A)          % 4*6 - 7*2 = 24-14 = 10
ans = 10

>> inv(A)
ans =
    0.6000   -0.7000
   -0.2000    0.4000

>> x = A \ b
x =
   -1.5000
    1.0000

>> A * x           % doit tre gal b
ans =
   1.0000
   3.0000      % confirm !

>> eig(A)          % valeurs propres
ans =
   0.7639
   9.2361
```

RÉSUMÉ DE SECTION

— Vecteurs : créés avec `[]`, l'opérateur deux-points `(:)`, ou `linspace()`. Indexés à partir de 1 (pas 0).
— Opérateurs élément par élément : `.* ./ .^` — exigent des tableaux de même taille.
— Opérateurs matriciels : `* / ^` — suivent les règles de l'algèbre linéaire.

— Fonctions matricielles clés : `det`, `inv`, `size`, `rank`, `eig`, `norm`.
— Résoudre $Ax = b$ avec la barre oblique : `x = A \ b` (préférable à `inv(A)*b`).
— Utilisez `end` comme dernier indice valide : `v(end)`, `A(end,:)`, `A(:,end-1)`.

1.3 Introduction à la Programmation avec MATLAB

Objectifs d'apprentissage

— Créer et exécuter des scripts MATLAB (fichiers `.m`) dans l'Éditeur.
— Utiliser `if` / `elseif` / `else` pour le branchement conditionnel.
— Implémenter des boucles `for` et `while` pour les calculs répétitifs.
— Sortir prématurément des boucles avec `break` et `continue`.
— Écrire des fonctions MATLAB modulaires et réutilisables.
— Appliquer de bonnes pratiques de style de codage et de débogage.

1.3.1 Généralités — Scripts et l'Éditeur

Un script MATLAB est un fichier texte brut avec l'extension `.m` contenant une séquence de commandes MATLAB. Les scripts s'exécutent dans l'espace de travail de base et constituent l'outil principal pour organiser des calculs en plusieurs étapes. L'Éditeur MATLAB offre la coloration syntaxique, l'indentation, les points d'arrêt et les contrôles de débogage.

Anatomie d'un script bien rédigé

```
%
   ================================================================

% Fichier  : analyse_cercle.m
% Objectif : Calculer l'aire et le p rim tre de cercles
% Auteur   : Votre Nom
% Date     : 2024
%
   ================================================================

clc;        % effacer la Fen tre de Commandes
clear;      % effacer toutes les variables
close all; % fermer toutes les fen tres graphiques

% --- Param tres d'entr e
   ----------------------------------------
rayons = [1, 2, 3, 5, 10];    % vecteur de rayons

% --- Calcul
   ------------------------------------------------------------
aires  = pi * rayons.^2;
```

```
perims = 2 * pi * rayons;

% --- Affichage des r sultats
    ------------------------------------
fprintf('%-10s %-12s %-12s\n', 'Rayon', 'Aire', 'P rim tre'
    );
fprintf('%s\n', repmat('-', 1, 36));
for k = 1:length(rayons)
    fprintf('%-10.1f %-12.4f %-12.4f\n', rayons(k), aires(k),
        perims(k));
end
```

CONSEIL

- Commencez toujours les scripts par `clc; clear; close all;` pour la reproductibilité.
- Utilisez `fprintf` pour la sortie formatée plutôt que `disp` — cela donne un contrôle total sur la mise en page.
- Organisez les scripts en sections clairement commentées : Entrée, Calcul, Sortie.
- Sauvegardez fréquemment (Ctrl+S) ; MATLAB ne sauvegarde pas automatiquement.

Fonctions d'entrée et de sortie

Fonction	Rôle & Syntaxe
`disp(x)`	Afficher une valeur sans le nom de la variable
`fprintf(fmt, v1, v2, ...)`	Affichage formaté dans la Fenêtre de Commandes (comme `printf` en C)
`input('invite')`	Lire une valeur de l'utilisateur à l'exécution
`input('invite','s')`	Lire une chaîne de l'utilisateur
`sprintf(fmt, v, ...)`	Formater en chaîne (n'affiche pas — retourne une chaîne)

```
% Sp cificateurs de format fprintf :
%   %d  -> entier      %f -> virgule flottante
%   %e  -> scientif.   %s -> cha ne
%   %g  -> auto        \n -> nouvelle ligne    \t ->
    tabulation
>> fprintf('Pi    6 d cimales : %.6f\n', pi)
Pi    6 d cimales : 3.141593

>> fprintf('Valeur : %e\n', 0.000123)
Valeur : 1.230000e-04

>> nom = input('Entrez votre nom : ', 's');
>> fprintf('Bienvenue, %s !\n', nom)
```

1.3.2 Structures de Contrôle de Flux

Les structures de contrôle de flux modifient l'exécution séquentielle d'un programme. MATLAB fournit trois familles : le branchement conditionnel (`if`), les boucles définies (`for`) et les boucles indéfinies (`while`), ainsi que `switch` pour le branchement multi-voies.

L'instruction if — elseif — else

```
% Syntaxe :
%   if condition
%     instructions
%   elseif condition
%     instructions
%   else
%     instructions
%   end

% Exemple : classification par note
note = 73;
if note >= 90
    grade = 'A'; disp('Excellent !');
elseif note >= 80
    grade = 'B'; disp('Tr s bien');
elseif note >= 70
    grade = 'C'; disp('Satisfaisant');
elseif note >= 60
    grade = 'D'; disp('   am liorer');
else
    grade = 'F'; disp('Insuffisant');
end
fprintf('Note : %d -> Grade : %s\n', note, grade);
```

AVERTISSEMENT

— Chaque `if` doit être fermé par `end` — oublier `end` est l'une des erreurs de syntaxe les plus fréquentes.
— Utilisez == pour les tests d'égalité dans les conditions — jamais = (qui affecterait, et non comparerait).
— L'indentation n'est pas requise par MATLAB mais est indispensable pour la lisibilité — utilisez-la systématiquement.

L'instruction switch — Branchement multi-voies

```
jour = 'Lundi';
switch jour
    case {'Samedi', 'Dimanche'}
        disp('Week-end --- repos !');
    case {'Lundi','Mardi','Mercredi','Jeudi','Vendredi'}
        disp('Jour de semaine --- travail !');
    otherwise
        disp('Jour inconnu');
```

```
end
```

La boucle for — Itération définie

Une boucle `for` exécute un bloc de code un nombre prédéterminé de fois, en itérant sur chaque valeur d'un vecteur ou d'une plage.

```
% Exemple 1 : Somme des n premiers carr s
n = 10; total = 0;
for k = 1:n
    total = total + k^2;
end
fprintf('Somme des carr s 1    %d = %d\n', n, total);
% Sortie : Somme des carr s 1    10 = 385

% Exemple 2 : Suite de Fibonacci
fib = zeros(1, 10);
fib(1) = 1; fib(2) = 1;
for k = 3:10
    fib(k) = fib(k-1) + fib(k-2);
end
disp(fib)
% Sortie : 1  1  2  3  5  8  13  21  34  55

% Exemple 3 : It ration sur les colonnes d'une matrice
A = [1 2 3; 4 5 6; 7 8 9];
for col = A         % col prend chaque colonne comme vecteur
    fprintf('Somme de colonne : %d\n', sum(col));
end
```

CONSEIL

- Pré-allouez les tableaux avant une boucle : `result = zeros(1, n);` — ne jamais faire croître des tableaux dans des boucles (très lent).
- Préférez les opérations vectorisées aux boucles explicites quand c'est possible — elles sont bien plus rapides dans MATLAB.
- Utilisez `break` pour sortir d'une boucle prématurément ; utilisez `continue` pour passer à l'itération suivante.

La boucle while — Itération indéfinie

Une boucle `while` se répète tant que sa condition reste vraie. Elle convient quand le nombre d'itérations n'est pas connu à l'avance.

```
% Exemple 1 :  tapes  de la conjecture de Collatz
n = 27; etapes = 0;
while n ~= 1
    if mod(n, 2) == 0
        n = n / 2;
    else
        n = 3*n + 1;
    end
    etapes = etapes + 1;
```

```
end
fprintf('Atteint 1 en %d  tapes \n', etapes);

% Exemple 2 : M thode de Newton pour sqrt(a)
a = 2; x = 1; tol = 1e-10; iter = 0;
while abs(x^2 - a) > tol
    x = (x + a/x) / 2;     % it ration de Newton
    iter = iter + 1;
    if iter > 1000
        warning('Non converg '); break;
    end
end
fprintf('sqrt(2) ~= %.15f (converg  en %d iter.)\n', x, iter
   );
```

AVERTISSEMENT

- Assurez-vous toujours que la condition de `while` peut éventuellement devenir fausse — les boucles infinies bloquent MATLAB.
- Incluez un compteur d'itérations maximum et un `break` comme garde-fou contre la non-convergence.
- Appuyez sur **Ctrl+C** pour interrompre une boucle en cours d'exécution (potentiellement infinie).

break et continue

```
% break : quitter la boucle imm diatement
for k = 1:20
    if k^2 > 50
        fprintf('Premier k tel que k^2 > 50 : k = %d\n', k);
        break;
    end
end
% Sortie : Premier k tel que k^2 > 50 : k = 8

% continue : sauter le reste de cette it ration, passer
   la suivante
for k = 1:10
    if mod(k, 2) == 0
        continue;     % ignorer les nombres pairs
    end
    fprintf('%d est impair\n', k);
end
```

EXERCICE

Exercice 1.5 — Structures de contrôle

1. Écrire un script qui calcule la factorielle de $n = 8$ en utilisant une boucle `for`.
2. Écrire une boucle `while` qui trouve le plus petit entier n tel que $2^n > 1000$.
3. Écrire une boucle qui affiche les entiers de 1 à 30, en remplaçant les multiples de 3 par `'Fizz'`, les multiples de 5 par `'Buzz'` et les multiples des deux par

'FizzBuzz'.

SOLUTION

```
% Partie 1 : Factorielle
n = 8; resultat = 1;
for k = 1:n
    resultat = resultat * k;
end
fprintf('%d! = %d\n', n, resultat)    % 8! = 40320

% Partie 2 : Puissance de 2
n = 0;
while 2^n <= 1000
    n = n + 1;
end
fprintf('2^%d = %d > 1000\n', n, 2^n)  % 2^10 = 1024

% Partie 3 : FizzBuzz
for k = 1:30
    if mod(k,15) == 0
        fprintf('FizzBuzz\n');
    elseif mod(k,3) == 0
        fprintf('Fizz\n');
    elseif mod(k,5) == 0
        fprintf('Buzz\n');
    else
        fprintf('%d\n', k);
    end
end
```

1.3.3 Les Fonctions en MATLAB

Les fonctions sont la pierre angulaire de la programmation modulaire. Une fonction MATLAB est un fichier `.m` (ou une fonction locale dans un fichier script) qui accepte des arguments d'entrée, effectue un calcul et retourne des arguments de sortie. Les fonctions opèrent dans leur propre espace de travail — elles ne partagent pas de variables avec le script appelant.

Syntaxe et structure d'une fonction

```
%
    ============================================================

% Fichier : calcul_stats.m
%
    ============================================================

function [mn, et, mn_val, mx_val] = calcul_stats(donnees)
% CALCUL_STATS  Calcule les statistiques descriptives d'un
```

```
    vecteur.
%
%    Entr es   : donnees   - vecteur num rique ligne ou
    colonne
%    Sorties   : mn        - moyenne arithm tique
%                et        -   cart -type (  chantillon  )
%                mn_val    - valeur minimale
%                mx_val    - valeur maximale
%    Exemple   : [m, s, lo, hi] = calcul_stats([4 7 2 9 1]);

if isempty(donnees)
    error('calcul_stats : le vecteur de donn es ne peut pas
        tre  vide.');
end
mn     = mean(donnees);
et     = std(donnees);
mn_val = min(donnees);
mx_val = max(donnees);
end  % function calcul_stats

% Appel de la fonction :
>> v = [4 7 2 9 1 5 8];
>> [m, s, lo, hi] = calcul_stats(v)
m  = 5.1429
s  = 2.7948
lo = 1
hi = 9
```

IMPORTANT

- Le fichier de la fonction doit avoir le **même nom** que la fonction elle-même : `calcul_stats.m`.
- Les fonctions ont leur propre espace de travail — elles ne peuvent pas voir les variables de l'espace de travail de base sauf si elles leur sont passées en arguments.
- Validez toujours les entrées au début de la fonction (vérifiez la taille, le type, les bornes).
- La ligne H1 (premier commentaire après la déclaration de la fonction) apparaît dans les recherches `lookfor`.

Fonctions anonymes — Définitions en ligne

Pour les fonctions simples sur une seule ligne, MATLAB fournit des fonctions anonymes utilisant l'opérateur `@`.

```
% Syntaxe : f = @(args) expression
>> carre = @(x) x.^2;
>> carre(5)
ans = 25
>> carre([1 2 3 4])
ans = 1   4   9   16
```

```
>> gaussienne = @(x, mu, sigma) exp(-((x-mu).^2) / (2*sigma
   ^2));
>> gaussienne(0, 0, 1)
ans = 1

% Passer une fonction comme argument     une autre fonction
>> integral(@(x) x.^2, 0, 1)    % int grer x^2 de 0     1
ans = 0.3333
```

Fonctions récursives

```
% Fichier : factorielle_rec.m
function resultat = factorielle_rec(n)
% Calcul r cursif de n!
    if n < 0
        error('La factorielle est ind finie pour les entiers
            n gatifs.');
    elseif n == 0 || n == 1
        resultat = 1;
    else
        resultat = n * factorielle_rec(n - 1);
    end
end

>> factorielle_rec(7)
ans = 5040
```

Nombre variable d'arguments

```
function resultat = puissance_flexible(base, exposant)
% nargin : nombre d'arguments d'entr e effectivement fournis
    if nargin < 2
        exposant = 2;     % par d faut : au carr
    end
    resultat = base .^ exposant;
end

>> puissance_flexible(3)       % utilise l'exposant par
   d faut 2
ans = 9
>> puissance_flexible(3, 4)    % explicitement 3^4
ans = 81
```

Bonnes pratiques de programmation MATLAB

Pratique	Pourquoi c'est important	Exemple
Pré-allouer les tableaux	Évite le redimensionnement dynamique lent dans les boucles	`result = zeros(1, n);`

Pratique	**Pourquoi c'est important**	**Exemple**
Vectoriser les boucles	10x à 1000x plus rapide que les boucles `for` explicites	`y = x.^2` au lieu d'une boucle
Valider les entrées	Évite les erreurs cryptiques au cœur du code	`if ~isnumeric(x) error(...)`
Commenter généreusement	Garantit que le code reste compréhensible	`% Normaliser à l'intervalle unitaire`
Utiliser des noms parlants	Code auto-documenté	`temperature` et non `t1`
Éviter les variables globales	Prévient les couplages cachés	Passer les valeurs en arguments
Utiliser `tic`/`toc`	Identifier les goulots d'étranglement	`tic; code; elapsed=toc`

RÉSUMÉ DE SECTION

- Les scripts (fichiers `.m`) s'exécutent séquentiellement dans l'espace de travail de base ; commencez par `clc; clear; close all`.
- `if`/`elseif`/`else`/`end` : branchement conditionnel ; `switch`/`case` : branchement multi-voies.
- `for` : boucles définies sur une plage ou un vecteur ; `while` : boucles indéfinies avec condition.
- `break` sort d'une boucle ; `continue` passe à l'itération suivante.
- Les fonctions ont des espaces de travail isolés ; elles acceptent des arguments et retournent des valeurs.
- Fonctions anonymes : `f = @(x) expression` — idéales pour les expressions mathématiques courtes.

1.4 Graphiques et Visualisation des Données en MATLAB

Objectifs d'apprentissage

- Tracer des courbes 2D avec la fonction `plot` et un contrôle complet de la mise en forme.
- Ajouter des étiquettes, des titres, des légendes et des grilles aux figures.
- Superposer plusieurs courbes sur les mêmes axes.
- Disposer plusieurs sous-graphiques dans une seule fenêtre graphique.
- Sauvegarder et exporter des figures vers des fichiers image et PDF.

La visualisation des données est l'une des fonctionnalités les plus appréciées de MATLAB. Le système graphique produit des tracés de qualité publication avec un code minimal. Chaque figure MATLAB fonctionne via un système de graphiques à descripteurs : chaque figure, axe et série de données est un objet dont les propriétés

peuvent être inspectées et modifiées.

1.4.1 La Fonction plot — Tracés 2D

La fonction `plot` est l'outil principal de la visualisation 2D.

```
% Syntaxe de base :
%   plot(x, y)                --- courbe simple
%   plot(x, y, 'spec')        --- avec cha ne de
    sp cification
%   plot(x, y, 'spec', Nom, Valeur, ...) --- propri t s
    suppl mentaires

% Exemple : Tracer y = sin(x) sur [0, 2*pi]
x = linspace(0, 2*pi, 200);
y = sin(x);

figure;
plot(x, y);
title('Graphe de y = sin(x)');
xlabel('x (radians)');
ylabel('y = sin(x)');
grid on;
xlim([0 2*pi]);
ylim([-1.2 1.2]);
```

la suppression du point-virgule dans la commande plot permet d'afficher la figure suivante.

Chaînes de spécification de ligne

Une chaîne de format `'CouleurMarqueurLigne'` contrôle l'apparence de chaque courbe tracée :

Catégorie	Code	Signification
Couleur	`b`	Bleu (défaut)
Couleur	`r`	Rouge
Couleur	`g`	Vert
Couleur	`k`	Noir
Couleur	`m`	Magenta
Couleur	`c`	Cyan
Marqueur	`o`	Cercle
Marqueur	`s`	Carré
Marqueur	`*`	Étoile
Marqueur	`^`	Triangle
Marqueur	`.`	Point

Catégorie	Code	Signification
Marqueur	`d`	Losange
Style de ligne	`-`	Continu (défaut)
Style de ligne	`--`	Tirets
Style de ligne	`:`	Pointillés
Style de ligne	`-.`	Tiret-point

```
% Combinaison couleur + marqueur + style de ligne
x = 0:0.5:4*pi;
figure;
plot(x, sin(x), 'r--o', 'LineWidth', 2, 'MarkerSize', 8);
title('Courbe Sinus Mise en Forme');
xlabel('x'); ylabel('sin(x)');
grid on;
```

Référence complète de mise en forme

Propriété / Commande	Effet
`title('texte')`	Définir le titre de la figure
`xlabel('texte')`	Étiqueter l'axe des x
`ylabel('texte')`	Étiqueter l'axe des y
`zlabel('texte')`	Étiqueter l'axe des z (tracés 3D)
`grid on` / `off`	Afficher ou masquer la grille
`xlim([a b])`	Définir la plage de l'axe des x
`ylim([a b])`	Définir la plage de l'axe des y
`axis equal`	Même échelle sur les deux axes
`axis tight`	Ajuster les axes aux données
`'LineWidth', n`	Épaisseur du tracé (défaut $\approx 0{,}5$)
`'MarkerSize', n`	Diamètre du marqueur en points
`'Color', [r g b]`	Couleur via triplet RGB $[0 \dots 1]$
`'DisplayName', 'txt'`	Nom utilisé dans la légende
`legend`	Afficher la légende (utilise DisplayName)
`legend('a','b',...)`	Entrées de légende explicites
`set(gca,'FontSize',14)`	Augmenter la taille de police des axes
`saveas(gcf,'fig.png')`	Sauvegarder la figure en PNG

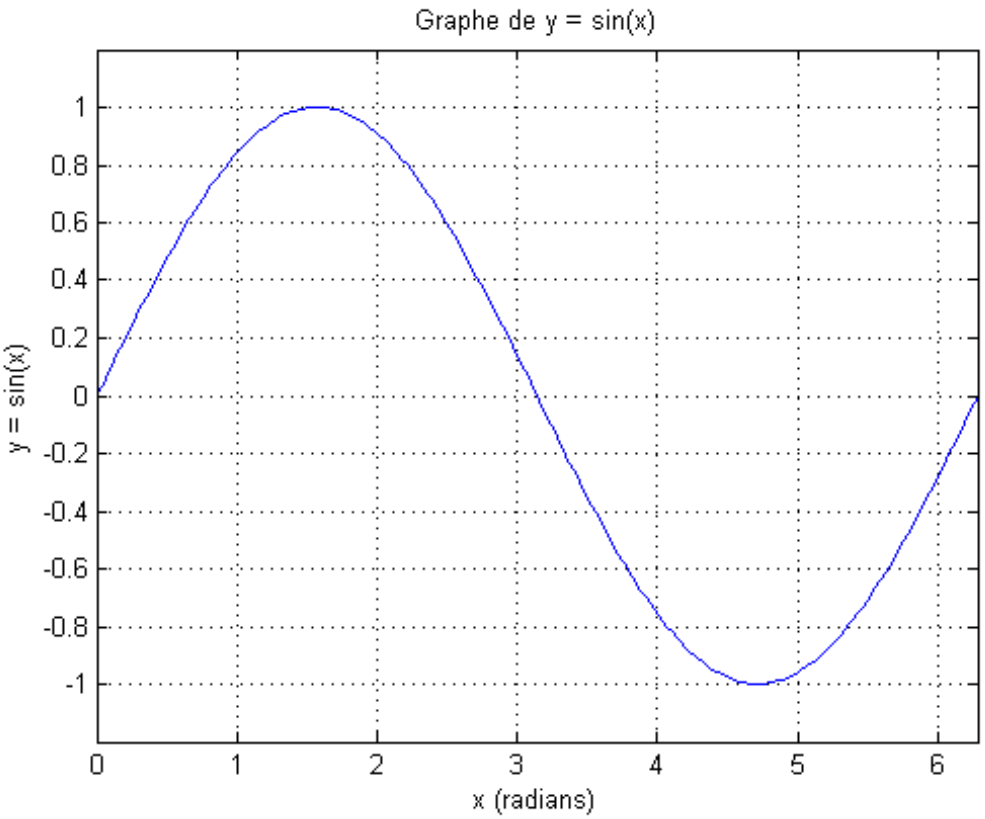
EXERCICE

Exercice 1.6 — Tracé de base

Tracer la fonction $f(x) = x^3 - 6x^2 + 11x - 6$ sur l'intervalle $[-0{,}5;\ 3{,}5]$.

— Utiliser une courbe lisse (au moins 200 points).

— Ajouter une ligne horizontale rouge en tirets à $y = 0$ (l'axe des x).



— Ajouter un titre, des étiquettes d'axes et une grille.

SOLUTION

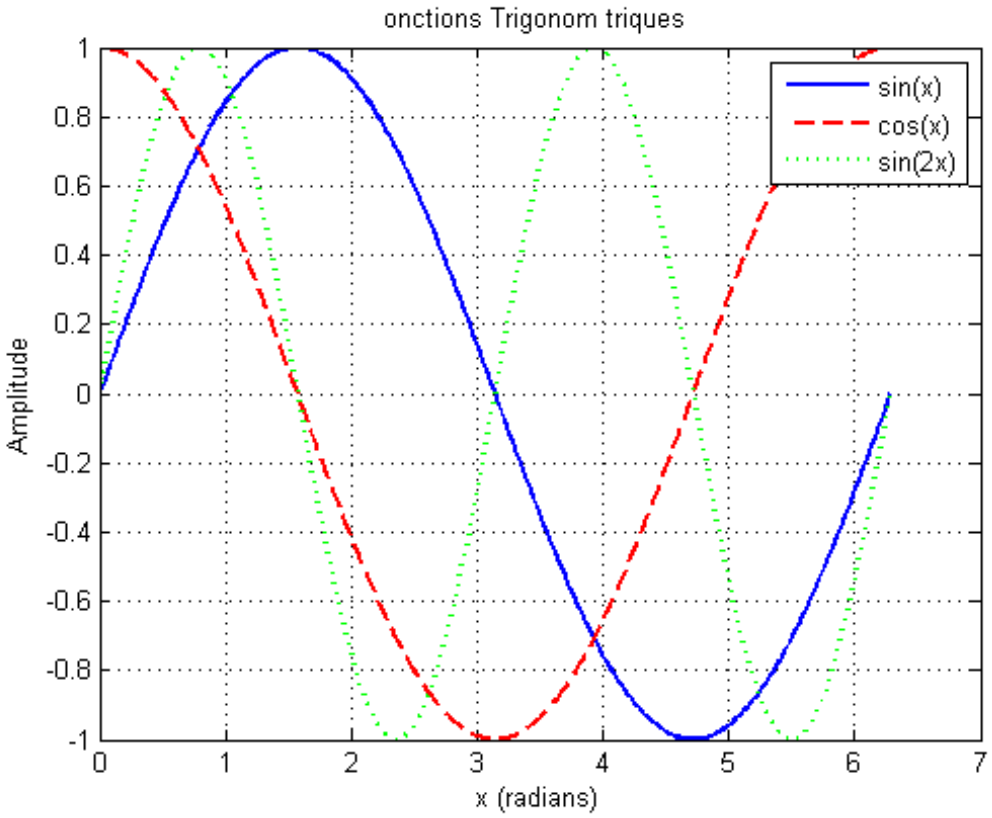
```
x = linspace(-0.5, 3.5, 300);
f = x.^3 - 6*x.^2 + 11*x - 6;

figure;
plot(x, f, 'b-', 'LineWidth', 2);
hold on;
plot(xlim, [0 0], 'r--', 'LineWidth', 1);     % ligne
   horizontale     y=0
hold off;
title('f(x) = x^3 - 6x^2 + 11x - 6');
xlabel('x'); ylabel('f(x)');
grid on;
xlim([-0.5 3.5]);
```

1.4.2 Plusieurs Courbes dans une Même Figure

Il est souvent nécessaire de superposer deux ou plusieurs courbes sur les mêmes axes pour une comparaison directe.

Méthode 1 : Plusieurs paires dans un seul appel à plot

```
x = linspace(0, 2*pi, 300);
figure;
plot(x, sin(x), 'b-', x, cos(x), 'r--', x, sin(2*x), 'g:',
    ...
      'LineWidth', 2)
title('Fonctions Trigonom triques');
xlabel('x (radians)'); ylabel('Amplitude');
legend('sin(x)', 'cos(x)', 'sin(2x)');
grid on;
```

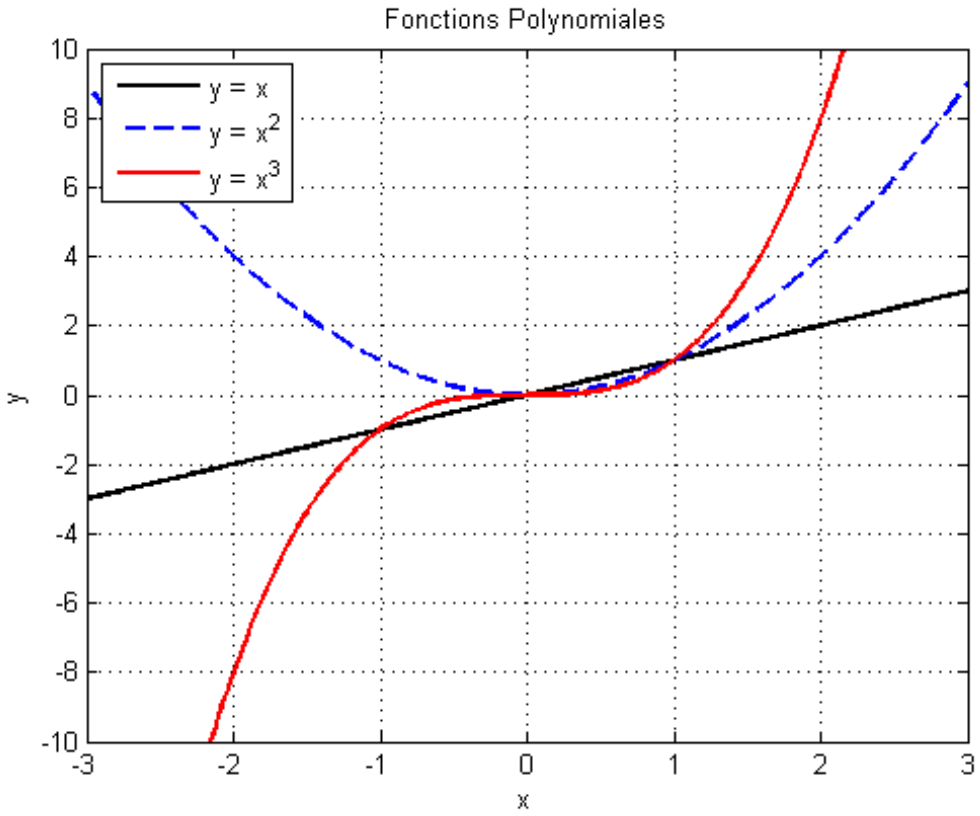


Méthode 2 : hold on / hold off

`hold on` fige les axes courants de sorte que les commandes `plot` suivantes s'ajoutent à la figure existante. `hold off` libère cet état.

```
x = linspace(-3, 3, 300);
figure;
plot(x, x,    'k-',  'LineWidth', 1.5, 'DisplayName', 'y = x'
   )
hold on;
plot(x, x.^2, 'b--', 'LineWidth', 1.5, 'DisplayName', 'y = x
   ^2');
plot(x, x.^3, 'r-',  'LineWidth', 1.5, 'DisplayName', 'y = x
   ^3');
```

```
hold off;
title('Fonctions Polynomiales');
xlabel('x'); ylabel('y');
legend('Location', 'northwest');
grid on;
ylim([-10 10]);
```

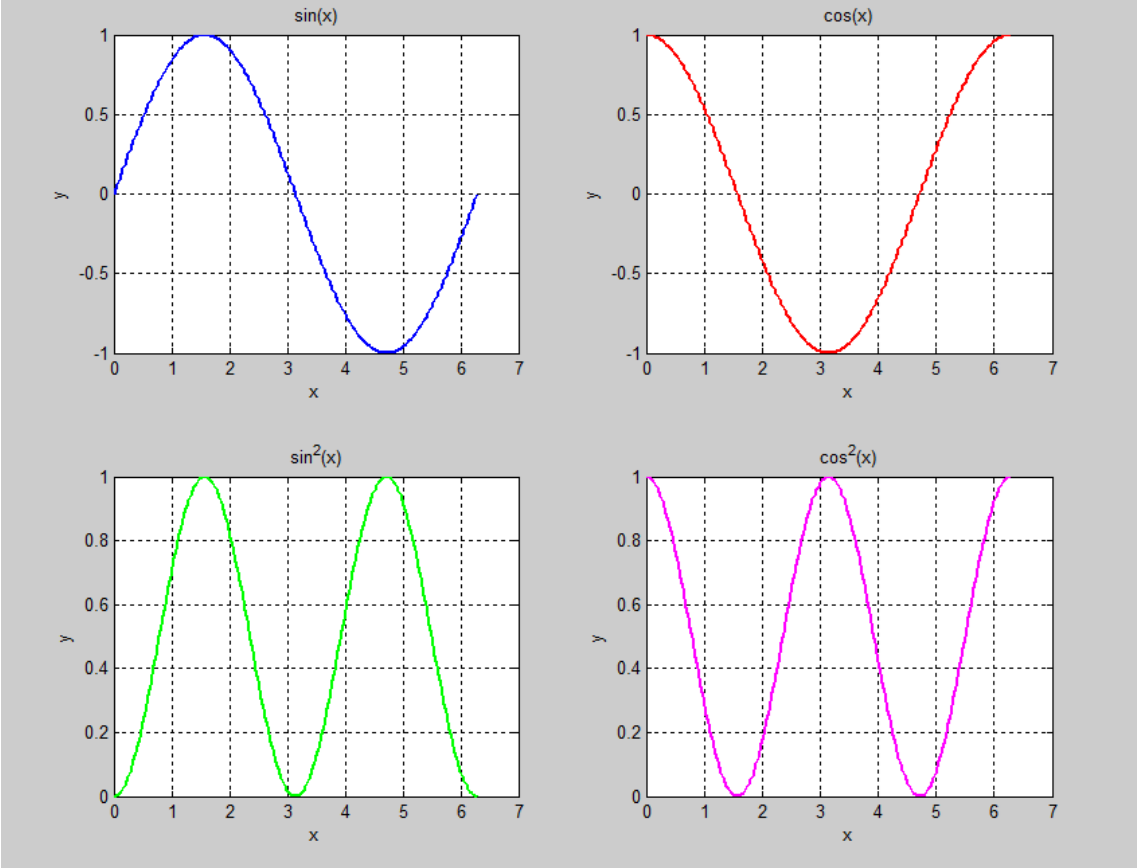


1.4.3 Plusieurs Courbes dans la Même Fenêtre — subplot

La fonction `subplot` divise une fenêtre graphique en une grille de m lignes et n colonnes, plaçant chaque tracé dans une cellule spécifiée.

```
% subplot(m, n, p)
%   m = lignes,  n = colonnes,  p = position (1-index ,
    gauche->droite, haut->bas)

x = linspace(0, 2*pi, 200);
figure('Position', [100 100 900 700]);    % taille de la
    figure en pixels

subplot(2, 2, 1);
plot(x, sin(x), 'b-', 'LineWidth', 2);
```

```
title('sin(x)'); xlabel('x'); ylabel('y'); grid on;

subplot(2, 2, 2);
plot(x, cos(x), 'r-', 'LineWidth', 2);
title('cos(x)'); xlabel('x'); ylabel('y'); grid on;

subplot(2, 2, 3);
plot(x, sin(x).^2, 'g-', 'LineWidth', 2);
title('sin^2(x)'); xlabel('x'); ylabel('y'); grid on;

subplot(2, 2, 4);
plot(x, cos(x).^2, 'm-', 'LineWidth', 2);
title('cos^2(x)'); xlabel('x'); ylabel('y'); grid on;

sgtitle('Fonctions Trigonom triques et Leurs Carr s', '
   FontSize', 14);
```



Autres types de graphiques

Fonction	Type de graphique	Usage typique
`bar(x, y)`	Diagramme en barres	Données par catégories discrètes
`stem(x, y)`	Graphique en tiges	Visualisation de signaux discrets
`scatter(x, y)`	Nuage de points	Nuages de données bivariées
`histogram(data)`	Histogramme	Distribution des données
`semilogy(x, y)`	Axe y en échelle log	Données exponentielles
`loglog(x, y)`	Les deux axes en log	Données en loi de puissance
`fill(x, y, c)`	Polygone rempli	Régions ombrées
`errorbar(x,y,e)`	Barres d'erreur	Moyenne ± écart-type
`plot3(x,y,z)`	Courbe 3D	Trajectoires 3D
`surf(X,Y,Z)`	Surface 3D	Fonctions de deux variables
`contour(X,Y,Z)`	Courbes de niveau	Lignes de niveau
`imagesc(M)`	Image couleur	Données matricielles / image

EXERCICE

Exercice 1.7 — Sous-graphiques et courbes multiples

Créer une figure avec une disposition de sous-graphiques 1×2 :

— **Panneau gauche** : Tracer e^x, e^{2x} et $e^{0,5x}$ sur les mêmes axes pour $x \in [0, 2]$. Ajouter légende, titre, étiquettes et grille.

— **Panneau droit** : Tracer les trois mêmes fonctions en échelle semi-logarithmique (`semilogy`). Observer comment la croissance exponentielle devient linéaire sur une échelle logarithmique.

SOLUTION

```
x = linspace(0, 2, 200);
figure('Position', [100 100 1100 450]);

subplot(1, 2, 1);
plot(x, exp(x),     'b-',  'LineWidth', 2, 'DisplayName',
   'e^x');
hold on;
plot(x, exp(2*x),   'r--', 'LineWidth', 2, 'DisplayName',
   'e^{2x}');
plot(x, exp(0.5*x), 'g:',  'LineWidth', 2, 'DisplayName',
   'e^{0.5x}');
hold off;
title('Fonctions Exponentielles ( chelle  Lin aire)');
xlabel('x'); ylabel('y');
legend('Location', 'northwest'); grid on;

subplot(1, 2, 2);
```

```
semilogy(x, exp(x),     'b-',  'LineWidth', 2, '
   DisplayName', 'e^x');
hold on;
semilogy(x, exp(2*x),   'r--', 'LineWidth', 2, '
   DisplayName', 'e^{2x}');
semilogy(x, exp(0.5*x), 'g:',  'LineWidth', 2, '
   DisplayName', 'e^{0.5x}');
hold off;
title('Fonctions Exponentielles ( chelle  Log-y)');
xlabel('x'); ylabel('y ( chelle  log)');
legend('Location', 'northwest'); grid on;

sgtitle('Comparaison de la Croissance Exponentielle');
```

RÉSUMÉ DE SECTION

- `plot(x, y, 'spec')` trace des courbes 2D avec couleur, marqueur et style de ligne personnalisés.
- Mise en forme : `title`, `xlabel`, `ylabel`, `grid on`, `xlim`, `ylim`, `legend`.
- Superposer des courbes : plusieurs séries dans un appel `plot()`, ou utiliser `hold on` / `hold off`.
- `subplot(m, n, p)` divise la figure en grille $m \times n$ et active le panneau p.
- Autres types de graphiques : `bar`, `scatter`, `histogram`, `semilogy`, `surf`, `contour`.
- Sauvegarder les figures : `saveas(gcf, 'fichier.png')` ou Fichier → Enregistrer sous dans la fenêtre graphique.

Chapitre 2

Exemples d'Application et Techniques de Résolution

Ce chapitre consolide les techniques du Chapitre 1 à travers des exemples courts et ciblés exécutés de manière interactive dans la Fenêtre de Commandes. La deuxième section présente des scripts complets dans l'éditeur, organisés selon les standards professionnels de codage.

Contenu du chapitre

PARTIE 1 — Fenêtre de commande

- Application 1 : Les commandes
- Application 2 : Les opérateurs
- Application 3 : Les formules
- Application 4 : Les vecteurs
- Application 5 : Les matrices
- Application 6 : Les chaînes de caractères

PARTIE 2 — L'Éditeur

- Application 7 : Structures de contrôle
- Application 8 : Scripts
- Application 9 : Fonctions
- Application 10 : Représentation graphique

À l'issue de ce chapitre de TP, l'étudiant sera capable de :

— utiliser la *Command Window* de MATLAB pour réaliser des calculs élémentaires ;

— manipuler des vecteurs et des matrices (création, indexation, opérations) ;

— écrire et exécuter un script MATLAB simple ;

— définir et appeler une fonction MATLAB ;

— utiliser les structures de contrôle de base (boucles, conditions).

Tous les codes présentés dans ce chapitre sont directement exécutables dans MATLAB. Le symbole >> représente l'invite de commande de la *Command Window*. Les lignes commençant par % sont des commentaires et ne sont pas exécutées.

Fenêtre de Commande (Command Window)

2.1 Applications simples sur la Command Window

La *Command Window* est la fenêtre principale de MATLAB. Elle permet d'entrer des commandes et d'obtenir immédiatement les résultats. C'est l'environnement idéal pour tester rapidement des calculs avant de les intégrer dans un script.

2.1.1 Opérations arithmétiques de base

MATLAB peut être utilisé comme une calculatrice scientifique avancée. Le tableau suivant rappelle les opérateurs de base :

Opérateur	Signification	Exemple
`+`	Addition	`3 + 5` → `8`
`-`	Soustraction	`9 - 4` → `5`
`*`	Multiplication	`6 * 7` → `42`
`/`	Division	`10 / 4` → `2.5`
`^`	Puissance	`2^8` → `256`
`sqrt()`	Racine carrée	`sqrt(49)` → `7`

[Calculs arithmétiques simples] Calculer dans la *Command Window* les expressions suivantes :

1. $3 + 5 \times 2$
2. $\dfrac{10^3 - 125}{5^2}$
3. $\sqrt{144} + \ln(e^3)$
4. $\cos\left(\frac{\pi}{4}\right)^2 + \sin\left(\frac{\pi}{4}\right)^2$

```
>> 3 + 5 * 2
ans = 13

>> (10^3 - 125) / 5^2
ans = 35

>> sqrt(144) + log(exp(3))    % log() = logarithme naturel en
   MATLAB
ans = 15

>> cos(pi/4)^2 + sin(pi/4)^2
ans = 1
```

En MATLAB, log() désigne le logarithme naturel (ln) et log10() le logarithme décimal. La constante π est accessible via pi.

[Variables et affectation]

1. Affecter $a = 4$, $b = 7$, puis calculer $c = a^2 + 2ab - b^2$.

2. Calculer le discriminant $\Delta = b^2 - 4ac$ pour $a = 2$, $b = -5$, $c = 3$.
3. Afficher le résultat avec un message grâce à fprintf.

```
% Question 1
a = 4;
b = 7;
c = a^2 + 2*a*b - b^2
% c = 16 + 56 - 49 = 23

% Question 2
a = 2; b = -5; c = 3;
delta = b^2 - 4*a*c
% delta = 25 - 24 = 1

% Question 3
fprintf('Le discriminant est : delta = %d\n', delta)
```

Résultat attendu :

```
c = 23
delta = 1
Le discriminant est : delta = 1
```

Le point-virgule ; en fin de ligne supprime l'affichage automatique du résultat. Sans lui, MATLAB affiche le résultat de chaque opération.

2.1.2 Application 1 – Commandes de base de MATLAB

Objectif

Découvrir les commandes fondamentales de gestion de l'environnement MATLAB ainsi que les constantes et fonctions prédéfinies.

Exercice 1

Énoncé : Pour chacune des commandes MATLAB suivantes, indiquez ce qu'elle affiche ou effectue, puis vérifiez votre réponse en l'exécutant dans la Command Window.

1. clear all
2. clc
3. help abs
4. clock
5. date
6. pi
7. eps
8. 0/0
9. realmin
10. realmax

SOLUTION

1. `clear all`
   Supprime toutes les variables de l'espace de travail (Workspace). Après cette commande, la liste des variables est vide.
2. `clc`
   Efface entièrement le contenu affiché dans la Command Window (équivalent à *clear screen*). Les variables en mémoire **ne sont pas** supprimées.
3. `help abs`
   Affiche l'aide intégrée de la fonction `abs`. On obtient notamment :

```
ABS(X) is the absolute value of the elements of X.
When X is complex, ABS(X) is the complex modulus (
   magnitude).
```

4. `clock`
   Retourne un vecteur ligne de 6 éléments représentant la date et l'heure courantes :

   [année, mois, jour, heure, minutes, secondes]

   Exemple : `[2025.0000, 5.0000, 15.0000, 10.0000, 30.0000, 47.3210]`
5. `date`
   Retourne la date du jour sous forme de chaîne de caractères, par exemple : `'15-May-2025'`.
6. `pi`
   Affiche la valeur de la constante mathématique π :

```
ans = 3.1416
```

   (En format `long` : `3.141592653589793`)
7. `eps`
   Affiche le *machine epsilon* : le plus petit nombre flottant tel que $1+\varepsilon \neq 1$:

```
ans = 2.2204e-16
```

8. `0/0`
   Produit un *Not a Number* (**NaN**) accompagné d'un avertissement :

```
Warning: Divide by zero.
ans = NaN
```

9. `realmin`
   Le plus petit nombre réel positif normalisé en double précision :

```
ans = 2.2251e-308
```

10. `realmax`
    Le plus grand nombre réel représentable en double précision :

```
ans = 1.7977e+308
```

Remarques / Extensions

- `clear` (sans argument) supprime toutes les variables ; `clear x` ne supprime que la variable `x`.
- `clc` et `clear` sont souvent utilisés en début de script pour repartir d'un état propre.
- `Inf` est le résultat de `1/0` (infini) ; `NaN` (Not a Number) est le résultat de formes indéterminées comme `0/0`.
- Tapez `format long` puis `pi` pour voir plus de décimales.

2.1.3 Application 2 – Variables et opérateurs

Objectif

Comprendre la manipulation des variables scalaires, le rôle du point-virgule, les formats d'affichage et les commandes d'inspection de l'espace de travail.

Exercice 2

Énoncé : Exécutez les instructions suivantes dans l'ordre et, pour chacune, expliquez ce qui se passe et indiquez la sortie éventuelle.

```
clear all
clc
a = 75.125
b = 43.125;
b
a + b
format long ; a, b
format short ; a, b
a = 25
44 + a
clear a
a
clear
a = 9, b = pi/2, e = eps, c = a + 3*i
who
whos
clear b,
who
```

SOLUTION

Ligne 1. `clear all` → Vide l'espace de travail. Aucune sortie.

Ligne 2. `clc` → Efface la fenêtre de commande. Aucune sortie.

Ligne 3. `a = 75.125` → Crée et affiche `a` (pas de `;`) :

```
a = 75.1250
```

Ligne 4. `b = 43.125;` → Crée `b` **sans affichage** (le `;` supprime l'écho).

Ligne 5. `b` → Affiche la valeur de `b` :

```
b = 43.1250
```

Ligne 6. `a + b` → Calcule et affiche la somme :

```
ans = 118.2500
```

Ligne 7. `format long ; a, b` → Passe au format longue précision, puis affiche :

```
a = 75.125000000000000
b = 43.125000000000000
```

Ligne 8. `format short ; a, b` → Revient au format court :

```
a = 75.1250
b = 43.1250
```

Ligne 9. `a = 25` → Réaffecte `a` à 25 et l'affiche :

```
a = 25
```

Ligne 10. `44 + a` → Calcule `44 + 25` :

```
ans = 69
```

Ligne 11. `clear a` → Supprime uniquement la variable `a`.

Ligne 12. `a` → Erreur car `a` n'existe plus :

```
Undefined function or variable 'a'.
```

Ligne 13. `clear` → Supprime toutes les variables restantes.

Ligne 14. `a = 9, b = pi/2, e = eps, c = a + 3*i` → Crée 4 variables et les affiche :

```
a = 9
b = 1.5708
e = 2.2204e-16
c = 9.0000 + 3.0000i
```

Ligne 15. `who` → Liste les noms des variables de l'espace de travail :

```
Your variables are:  a  b  c  e
```

Ligne 16. `whos` → Affiche les noms, tailles, types et mémoire occupée :

```
  Name        Size            Bytes  Class
  a           1x1                 8  double
  b           1x1                 8  double
  c           1x1                16  double  complex
  e           1x1                 8  double
```

Ligne 17. `clear b,` → Supprime `b`. Le paramètre après la virgule est ignoré s'il est absent.

Ligne 18. `who` → Liste les variables restantes :

```
Your variables are:  a  c  e
```

Remarques / Extensions

- Le **point-virgule** ; est essentiel dans les scripts pour éviter des affichages parasites.
- La variable `ans` est automatiquement créée pour stocker le résultat d'une expression non affectée.
- `i` (ou `j`) désigne l'unité imaginaire $\sqrt{-1}$ dans MATLAB.
- `who` donne une liste simple ; `whos` fournit des détails complets (très utile pour déboguer les problèmes de mémoire).

2.1.4 Application 3 – Évaluation de formules mathématiques

Objectif

Traduire des expressions mathématiques en instructions MATLAB et comprendre les opérateurs arithmétiques et logiques.

Exercice 3

Énoncé : Définissez les variables suivantes :

$$A = 1, \quad B = 2, \quad a = 3, \quad b = 2, \quad C = 3, \quad D = \text{false}, \quad E = \text{false}, \quad F = \text{true}, \quad x = 1.25$$

Puis calculez, avec MATLAB, les expressions suivantes :

$$M = \sqrt{A^2 + \left(\frac{a}{B + a^b}\right)}, \qquad P = \frac{2x}{5 + 4x^2 + 3\sqrt{x^3 + 1}}, \qquad N = \big((D{+}F){\cdot}F{+}D{\cdot}E\big){\cdot}F$$

SOLUTION

Étape 1 : Définir les variables

```
clear all; clc
A = 1;   B = 2;   a = 3;   b = 2;
C = 3;   D = false;   E = false;   F = true;
x = 1.25;
```

Étape 2 : Calculer M

La formule est $M = \sqrt{A^2 + \dfrac{a}{B + a^b}}$.

```
M = sqrt(A^2 + (a / (B + a^b)))
```

Sortie MATLAB attendue

```
M = 1.2445
```

Vérification : $A^2 = 1$, $a^b = 9$, $B + a^b = 11$, $a/11 \approx 0.2727$, $M = \sqrt{1.2727} \approx 1.2445$. ✓

Étape 3 : Calculer P

```
P = (2*x) / (5 + 4*x^2 + 3*sqrt(x^3 + 1))
```

Sortie MATLAB attendue

```
P = 0.1480
```

Vérification : $2x = 2.5$, $4x^2 = 6.25$, $x^3 = 1.9531$, $\sqrt{2.9531} \approx 1.7185$, dénominateur ≈ 16.8056. $P \approx 0.1488$. ✓

Étape 4 : Calculer N (expression logique)

$N = \big((D + F) \cdot F + D \cdot E\big) \cdot F$

Rappel : `false` $= 0$, `true` $= 1$. En logique booléenne, l'addition correspond à **OU** (`|`) et la multiplication à **ET** (`&`).

```
N = ((D | F) & F | (D & E)) & F
```

Sortie MATLAB attendue

```
N = 1    % (true)
```

Vérification : $D \vee F = \text{true}$, $\text{true} \wedge F = \text{true}$, $D \wedge E = \text{false}$, $\text{true} \vee \text{false} = \text{true}$, $\text{true} \wedge F = \text{true}$. ✓

Récapitulatif des opérateurs et priorités (ordre décroissant) :

Priorité	Opérateur	Description
1	`()`	Parenthèses
2	`^`	Puissance
3	`-` (unaire)	Négation
4	`*`, `/`	Multiplication, Division
5	`+`, `-`	Addition, Soustraction
6	`[ ]`	Concaténation

Remarques / Extensions

— En MATLAB, `D + F` sur des booléens effectue une addition numérique (0+1=1). Pour des opérations logiques strictes, utilisez `|` (OU), `&` (ET),

~ (NON).

— Utilisez toujours des parenthèses pour rendre les priorités explicites et éviter les erreurs.

— La fonction `sqrt(x)` exige $x \geq 0$ (pour des résultats réels). Pour $x < 0$, le résultat est complexe.

2.1.5 Manipulation des vecteurs

Un vecteur est un tableau à une dimension. On distingue les vecteurs lignes et les vecteurs colonnes.

Syntaxe	Description
v = [1, 2, 3, 4]	Vecteur ligne
v = [1 ; 2 ; 3 ; 4]	Vecteur colonne
v = a :pas :b	Vecteur par pas régulier
v = linspace(a,b,n)	Vecteur de n points entre a et b
v(i)	Accès à l'élément i
v(end)	Dernier élément

[Création et indexation de vecteurs]

1. Créer le vecteur $v = [3,\ 7,\ 2,\ 9,\ 5]$.
2. Afficher son troisième élément, puis les deux derniers.
3. Créer le vecteur des entiers pairs de 2 à 20.
4. Créer un vecteur de 5 points équidistants entre 0 et π.

```
% Question 1
v = [3, 7, 2, 9, 5];

% Question 2
v(3)           % 3eme element
v(end-1:end)   % deux derniers elements

% Question 3
pairs = 2:2:20

% Question 4
x = linspace(0, pi, 5)
```

Résultat attendu :

```
ans = 2
ans = 9    5
pairs = 2   4   6   8   10   12   14   16   18   20
x = 0   0.7854   1.5708   2.3562   3.1416
```

[Opérations sur les vecteurs] Soit $u = [1,\ 4,\ 9,\ 16,\ 25]$.

1. Calculer la racine carrée de chaque élément (*opération élément par élément*).
2. Calculer la somme et la moyenne des éléments de u.
3. Trouver la valeur maximale et sa position.
4. Trier u par ordre décroissant.

```
u = [1, 4, 9, 16, 25];

% Question 1 : racine element par element
r = sqrt(u)

% Question 2 : somme et moyenne
s = sum(u)
m = mean(u)

% Question 3 : maximum et sa position
[val_max, idx_max] = max(u)

% Question 4 : tri decroissant
u_tri = sort(u, 'descend')
```

Résultat attendu :

```
r       = 1   2   3   4   5
s       = 55
m       = 11
val_max = 25
idx_max = 5
u_tri   = 25   16   9   4   1
```

2.1.6 Application 4 – Manipulation des vecteurs

Objectif

Créer, indexer, extraire et effectuer des opérations élément par élément sur des vecteurs MATLAB.

Exercice 4

1. Définissez le vecteur $V = [0, 1, 2, 3, \ldots, 50]$. Quelle est sa taille ?
2. Extrayez W = les 5 premiers éléments de V, et X = les 5 premiers *et* les 5 derniers éléments. Créez $Z = [0, 2, 4, \ldots, 50]$ à partir de V.
3. Proposez une instruction pour **inverser** l'ordre des éléments de V.
4. Affichez les éléments de V de la 5^{e} à la 11^{e} position.
5. Avec $A = [1, 2, 3, 4]$ et $B = [4, 3, 2, 1]$, calculez : $A + B$, $A - B$, $A \cdot B$ (terme à terme), $A \div B$ (terme à terme), A^3 (terme à terme), $A + 3$, $2 + A$, $B - 2A$, $A/3$, A^3. Déterminez le min/max et la racine carrée de chaque élément de A et B.
6. Créez un vecteur colonne $C = [1, -2, 1]^T$.

7. Créez H de 11 éléments contenant $-5, -4, \ldots, 4, 5$.
8. Créez M avec 10 valeurs équiréparties entre 0 et π ; affichez le 5^{e} et le 7^{e} élément.

SOLUTION

```
clear all; clc

%% 1. Vecteur V et sa taille
V = 0:50;            % ou V = 0:1:50
disp(length(V))      % taille = 51 elements

%% 2. Extraction de sous-vecteurs
W = V(1:5)                        % [0 1 2 3 4]
X = [V(1:5), V(end-4:end)]        % 5 premiers + 5 derniers
Z = V(1:2:end)                    % pas de 2 : [0 2 4 ... 50]

%% 3. Inversion du vecteur V
V_inv = V(end:-1:1)               % ou fliplr(V)

%% 4. Elements de la 5eme a la 11eme position
V(5:11)              % indices MATLAB commencent a 1

%% 5. Operations sur A et B
A = [1 2 3 4];
B = [4 3 2 1];

A + B                % addition element par element
A - B                % soustraction element par element
A .* B               % produit element par element
A ./ B               % division element par element
A .^ 3               % puissance element par element
A + 3                % ajouter un scalaire
2 + A                % commutativite
B - 2*A              % combinaison lineaire
A / 3                % division par un scalaire
A .^ 3               % cube de chaque element

[min(A), max(A)]     % min et max de A
[min(B), max(B)]     % min et max de B
sqrt(A)              % racine carree de chaque element de A
abs(A)               % valeur absolue (ici identique car A >
    0)
sqrt(B)
abs(B)

%% 6. Vecteur colonne C
C = [1; -2; 1]       % point-virgule = separateur de lignes

%% 7. Vecteur H de -5 a 5
```

```
H = -5:5              % 11 elements

%% 8. Vecteur M avec 10 valeurs entre 0 et pi
M = linspace(0, pi, 10)
M(5)                  % 5eme element
M(7)                  % 7eme element
```

Sortie MATLAB attendue

Quelques sorties clés :

```
>> W = V(1:5)
W = 0   1   2   3   4

>> X = [V(1:5), V(end-4:end)]
X = 0   1   2   3   4   46   47   48   49   50

>> A + B
ans = 5   5   5   5

>> A .* B
ans = 4   6   6   4

>> A .^ 3
ans = 1   8   27   64

>> C =
    1
   -2
    1

>> H = -5:5
H = -5   -4   -3   -2   -1   0   1   2   3   4   5

>> M(5)
ans = 1.3963
>> M(7)
ans = 2.0944
```

Remarques / Extensions

- **Opérateurs élément par élément** : `.*`, `./`, `.^` sont essentiels pour les opérations terme à terme sur des vecteurs/matrices. Sans le point, `*` est un produit matriciel.
- **Indexation** : en MATLAB, les indices commencent à **1** (pas 0 comme en Python/C).
- `end` est un mot-clé pratique désignant le dernier indice.
- `linspace(a,b,n)` crée n valeurs équiréparties entre a et b (bornes in-

cluses).

— Alternativement : `a:h:b` avec le pas $h = (b-a)/(n-1)$.

2.1.7 Manipulation des matrices

Une matrice est un tableau à deux dimensions (lignes × colonnes). MATLAB est particulièrement puissant pour les opérations matricielles.

Syntaxe	**Description**
A = [1 2 ; 3 4]	Matrice 2×2
zeros(m,n)	Matrice de zéros
ones(m,n)	Matrice de uns
eye(n)	Matrice identité $n \times n$
A'	Transposée de A
A * B	Produit matriciel
A .* B	Produit élément par élément
det(A)	Déterminant
inv(A)	Inverse
A(i,j)	Élément ligne i, colonne j
A(i, :)	Ligne i entière
A(:,j)	Colonne j entière

[Création et inspection de matrices]

1. Créer la matrice $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.
2. Afficher la taille de A, sa deuxième ligne et sa troisième colonne.
3. Extraire la sous-matrice des deux premières lignes et deux premières colonnes.
4. Créer une matrice identité 4×4 et une matrice de zéros 3×2.

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

% Question 2
size(A)          % taille
A(2, :)          % 2e ligne
A(:, 3)          % 3e colonne

% Question 3
B = A(1:2, 1:2)

% Question 4
I4 = eye(4)
Z  = zeros(3, 2)
```

Résultat attendu :

```
ans    = 3    3
A(2,:) = 4    5    6
A(:,3) = 3; 6; 9
B   = 1   2
      4   5
I4  = (matrice identite 4x4)
Z   = (matrice 3x2 de zeros)
```

[Opérations matricielles] Soient $A = \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & -1 \\ -2 & 4 \end{pmatrix}$.

1. Calculer $A + B$, $A \times B$ et A^3.
2. Calculer $\det(A)$ et A^{-1}.
3. Vérifier que $A \times A^{-1} = I$.
4. Résoudre le système linéaire $A\mathbf{x} = \mathbf{b}$ avec $\mathbf{b} = \begin{pmatrix} 4 \\ 11 \end{pmatrix}$.

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

% Question 1
S  = A + B
P  = A * B
A3 = A^3

% Question 2
d    = det(A)
Ainv = inv(A)

% Question 3
verif = A * Ainv    % doit donner l'identite (a la precision
   machine)

% Question 4
b = [4; 11];
x = A \ b           % solution par division matricielle gauche
```

Résultat attendu :

```
S     = 3    0
        3    7
P     = 0    2
        1    7
A3    = 61   35
       175  101
d     = 1
Ainv  = 3   -1
       -5    2
verif = 1   0
         0   1
x     = 1
```

```
2
```

L'opérateur \ (backslash) résout le système $Ax = b$ de manière plus efficace et plus stable numériquement que inv(A)*b.

2.1.8 Application 5 – Manipulation des matrices (I)

Objectif

Créer et manipuler des matrices : accès aux éléments, opérations matricielles, matrices spéciales.

Exercice 5

Définissez la matrice :

$$A = \begin{pmatrix} 3 & 1 & 0 \\ 2 & 2 & 4 \end{pmatrix}$$

Puis donnez le résultat de chaque instruction ci-dessous :

1. size(A)
2. 2 * A + 4
3. b = A(2, :)
4. exp(A)
5. Y = (A > 0)
6. n = 4 ; B = ones(n)
7. C = zeros(n)
8. D = eye(n)

SOLUTION

```
clear all; clc
A = [3 1 0; 2 2 4];     % matrice 2x3

size(A)             % dimensions
2*A + 4             % operation sur tous les elements
b = A(2,:)          % 2eme ligne entiere (: = toutes colonnes
   )
exp(A)              % exponentielle element par element
Y = (A > 0)         % tableau booleen

n = 4;
B = ones(n)         % matrice 4x4 remplie de 1
C = zeros(n)        % matrice 4x4 remplie de 0
D = eye(n)          % matrice identite 4x4
```

Sortie MATLAB attendue

```
>> size(A)
ans = 2  3          % 2 lignes, 3 colonnes

>> 2*A + 4
ans =
   10    6    4
    8    8   12

>> b = A(2,:)
b = 2  2  4         % 2eme ligne de A

>> exp(A)
ans =
   20.0855    2.7183    1.0000
    7.3891    7.3891   54.5982

>> Y = (A > 0)
ans =
   1  1  0
   1  1  1

>> B = ones(4)
B =
   1 1 1 1
   1 1 1 1
   1 1 1 1
   1 1 1 1

>> D = eye(4)
D =
   1 0 0 0
   0 1 0 0
   0 0 1 0
   0 0 0 1
```

Remarques / Extensions

- `A(i,j)` accède à l'élément ligne i, colonne j. `A(i,:)` = ligne i entière ; `A(:,j)` = colonne j entière.
- `exp(A)`, `sqrt(A)`, `abs(A)` s'appliquent élément par élément.
- `(A > 0)` retourne un tableau de `0` et `1` : utile pour le filtrage conditionnel.
- `ones(m,n)`, `zeros(m,n)`, `eye(n)` sont des constructeurs de matrices spéciales.

2.1.9 Application 6 – Manipulation des matrices (II)

Objectif

Extraire des sous-matrices, inverser des lignes/colonnes, effectuer des opérations avancées sur les matrices.

Exercice 6

Définissez la matrice 3×10 :

$$M = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \end{pmatrix}$$

1. Extrayez la 2e colonne, la 2e ligne et la sous-matrice 3×4 centrale (colonnes 4 à 7).
2. Inversez les 2 colonnes centrales (colonnes 5 et 6), puis inversez les deux premières lignes.
3. Extrayez les sous-matrices suivantes :

$$M_1 = \begin{pmatrix} 1 & 2 \\ 11 & 12 \\ 21 & 22 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 8 & 9 & 10 \\ 18 & 19 & 20 \\ 28 & 29 & 30 \end{pmatrix}, \quad Q = \begin{pmatrix} 3 & 7 \\ 23 & 27 \end{pmatrix}$$

4. Avec $N = M$, calculez : $P = N$, N^3 (puissance matricielle), $N+3$, $2+N$, $2N$, $N/3$, $P > 15$.
5. Déterminez le min et le max des éléments de N et P.
6. Calculez la racine carrée et la valeur absolue élément par élément de N et P.
7. Donnez les instructions pour : la taille de P, la transposée de P, l'inverse de P (si carré), les signes des éléments de P.

SOLUTION

```
clear all; clc

%% Definition de M
M = [1:10; 11:20; 21:30];

%% 1. Extraction
M(:,2)          % 2eme colonne
M(2,:)          % 2eme ligne
M(:,4:7)        % sous-matrice centrale 3x4

%% 2. Inversion de colonnes et de lignes
M(:,[5 6]) = M(:,[6 5])    % echanger colonnes 5 et 6
M([1 2],:) = M([2 1],:)    % echanger lignes 1 et 2
```

```
%% 3. Extraction de sous-matrices
N = [1:10; 11:20; 21:30];   % matrice originale

M1 = N(:, 1:2)              % colonnes 1 et 2
M2 = N(:, 8:10)             % colonnes 8 a 10
Q  = N([1 3], [3 7])        % lignes 1,3 et colonnes 3,7

%% 4. Calculs (N est 3x10, non carree)
N = [1:10; 11:20; 21:30];
P = N;
N_cube  = N .^ 3            % puissance element par element
N_plus3 = N + 3
N_fois2 = 2 * N
N_div3  = N / 3
P_sup15 = (P > 15)          % tableau booleen

%% 5. Min et max
min(N(:))    % minimum de tous les elements de N
max(N(:))    % maximum de tous les elements de N

%% 6. Racine carree et valeur absolue
sqrt(N)
abs(N)

%% 7. Operations supplementaires (sous-matrice carree)
P_sq = N(1:3, 1:3);
size(P_sq)              % taille
P_sq'                   % transposee
inv(P_sq)               % inverse (si det != 0)
sign(P_sq)              % signe de chaque element
```

Sortie MATLAB attendue

```
>> M1 = N(:,1:2)
M1 =
     1    2
    11   12
    21   22

>> M2 = N(:,8:10)
M2 =
     8    9   10
    18   19   20
    28   29   30

>> Q = N([1 3],[3 7])
Q =
     3    7
    23   27

>> min(N(:))
ans = 1

>> max(N(:))
ans = 30
```

Remarques / Extensions

- `A(:)` transforme toute matrice en vecteur colonne, utile pour `min(A(:))` et `max(A(:))`.
- `A^n` est la puissance **matricielle** (produit matriciel répété) ; `A.^n` est la puissance **élément par élément**.
- `inv(A)` n'est défini que pour les matrices carrées non singulières. Pour les systèmes linéaires, préférez `A\b`.
- `sign(x)` retourne -1 si $x < 0$, 0 si $x = 0$, $+1$ si $x > 0$.

2.1.10 Opérations élémentaires sur les tableaux

Les opérations *élément par élément* s'effectuent avec un point devant l'opérateur : .*, ./, .

.

[Opérations élément par élément] Soit $x = [1,\ 2,\ 3,\ 4,\ 5]$.

1. Calculer x^2 (chaque élément au carré) et e^x.
2. Construire le vecteur $y_i = \dfrac{x_i^2 + 1}{2x_i - 1}$ pour chaque élément.
3. Trouver les indices des éléments de x strictement supérieurs à 3.

```
x = [1, 2, 3, 4, 5];

% Question 1
x2 = x .^ 2
ex = exp(x)

% Question 2
y = (x.^2 + 1) ./ (2*x - 1)

% Question 3
idx = find(x > 3)
val = x(x > 3)        % valeurs correspondantes
```

Résultat attendu :

```
x2  =  1   4   9  16  25
ex  =  2.7183  7.3891  20.0855  54.5982  148.4132
y   =  2.0000  1.6667  2.0000  2.3333  2.6000
idx =  4  5
val =  4  5
```

2.1.11 Application 7 – Chaînes de caractères

Objectif

Manipuler les chaînes de caractères : création, concaténation, indexation, tests (`ischar`, `isempty`, `strcmp`).

Exercice 7

1. Créez et concaténez des chaînes pour former la phrase : *"Le résultat de la multiplication est donnée ci-dessous."*
2. En utilisant le vecteur `alphabet = 'abcdefghijklmnopqrstuwxyz'`, extrayez la 1re lettre et les lettres de la 5e à la 10e position.
3. Analysez les résultats de `ischar` et `isempty` pour les variables suivantes :
   - `a = '1'` puis `a = 1`
   - `a = 'abcd'` ; `a = ''` ; `a = []` ; `a = [1,2]`
4. Utilisez `strcmp` pour comparer : `'toto'` vs `'toto'`, `'toto'` vs `'titi'`, puis deux variables `str1='toto'` et `str2='tutu'`.

SOLUTION

```
clear all; clc

%% 1. Concatenation de chaines
phrase = ['Le resultat de la multiplication ', ...
```

```
          'est donnee ci-dessous.']

%% 2. Indexation dans une chaine (comme un vecteur)
alphabet = 'abcdefghijklmnopqrstuwxyz';
alphabet(1)          % premiere lettre : 'a'
alphabet(5:10)       % lettres 5 a 10 : 'efghij'

%% 3. ischar et isempty
a = '1';
ischar(a)            % 1 (vrai, c'est une chaine)
a = 1;
ischar(a)            % 0 (faux, c'est un nombre)

a = 'abcd';
ischar(a)            % 1
isempty(a)           % 0 (non vide)

a = '';              % chaine vide
ischar(a)            % 1 (c'est toujours une chaine)
isempty(a)           % 1 (vide !)

a = [];              % tableau vide (PAS une chaine)
isempty(a)           % 1
ischar(a)            % 0

a = [1, 2];
ischar(a)            % 0 (tableau numerique)
isempty(a)           % 0

%% 4. Comparaison de chaines avec strcmp
strcmp('toto', 'toto')     % 1 : egal
strcmp('toto', 'titi')     % 0 : different

str1 = 'toto'; str2 = 'tutu';
strcmp(str1, str2)          % 0 : different
strcmp(str1, 'toto')        % 1 : egal
```

Sortie MATLAB attendue

```
>> alphabet(1)
ans = a

>> alphabet(5:10)
ans = efghij

>> ischar('1')
ans = 1

>> ischar(1)
ans = 0

>> strcmp('toto','toto')
ans = 1

>> strcmp('toto','titi')
ans = 0
```

Remarques / Extensions

— En MATLAB, une chaîne de caractères est un vecteur de caractères : `'abc'` est équivalent à un vecteur de longueur 3.

— Pour comparer des chaînes, **ne jamais utiliser** `==` (compare les codes ASCII caractère par caractère) ; utilisez toujours `strcmp`.

— `[]` est un tableau vide numérique ; `''` est une chaîne vide. Les deux sont vides (`isempty = 1`) mais `ischar` diffère.

— D'autres fonctions utiles : `upper`, `lower`, `strtrim`, `num2str`, `str2num`.

Techniques de résolution (Éditeur)

2.2 Programmation MATLAB

Cette deuxième partie est consacrée à la programmation MATLAB dans l'**Éditeur** : structures de contrôle (`if`, `for`, `while`), scripts et fonctions. Chaque exercice doit être sauvegardé sous forme de fichier `.m`.

2.2.1 Les structures de contrôle

Avant d'écrire des scripts complets, il est essentiel de maîtriser les structures de contrôle qui permettent de répéter des actions ou de prendre des décisions.

La boucle `for`

```
for variable = debut : pas : fin
    % instructions
end
```

[Boucle `for` : sommes et suites]

1. Calculer la somme $S = \sum_{k=1}^{10} k^2$ à l'aide d'une boucle for.
2. Afficher les 8 premiers termes de la suite de Fibonacci : $F_1 = 1,\ F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$.
3. Construire le vecteur $v = [n^3 - n \ : \ n = 1, \ldots, 6]$ par boucle.

```
% Question 1 : somme des carres
S = 0;
for k = 1:10
    S = S + k^2;
end
fprintf('Somme = %d\n', S)

% Question 2 : suite de Fibonacci
F = zeros(1, 8);
F(1) = 1; F(2) = 1;
for n = 3:8
    F(n) = F(n-1) + F(n-2);
end
disp(F)

% Question 3 : construction d'un vecteur
v = zeros(1, 6);
for n = 1:6
    v(n) = n^3 - n;
end
disp(v)
```

Résultat attendu :

```
Somme = 385
F =   1   1   2   3   5   8   13   21
v =   0   6   24   60   120   210
```

La boucle `while`

```
while condition
    % instructions
end
```

[Boucle `while` : convergence]

1. Trouver le plus petit entier n tel que $2^n > 1000$.
2. Calculer la somme $S = \sum_{k=1}^{\infty} \frac{1}{k^2}$ en s'arrêtant quand le terme ajouté est inférieur à 10^{-6}.

```
% Question 1
n = 0;
while 2^n <= 1000
    n = n + 1;
end
fprintf('Plus petit n : %d  (2^n = %d)\n', n, 2^n)

% Question 2 : serie convergente
S = 0;
k = 1;
terme = 1/k^2;
while terme >= 1e-6
    S = S + terme;
    k = k + 1;
    terme = 1/k^2;
end
fprintf('S     %.6f  (apres %d termes)\n', S, k-1)
fprintf('Valeur exacte pi^2/6     %.6f\n', pi^2/6)
```

Résultat attendu :

```
Plus petit n : 10  (2^n = 1024)
S     1.644934  (apres 999999 termes)
Valeur exacte pi^2/6     1.644934
```

La structure conditionnelle `if`

```
if condition1
    % instructions
elseif condition2
    % instructions
```

```
else
    % instructions
end
```

[Conditions : classification] Écrire un code qui, pour un entier n saisi par l'utilisateur :

1. détermine si n est pair ou impair ;
2. affiche si n est positif, négatif ou nul ;
3. teste si n est divisible par 3 et par 5 simultanément (FizzBuzz).

```
n = input('Entrez un entier n : ');

% Question 1 : pair/impair
if mod(n, 2) == 0
    fprintf('%d est pair.\n', n)
else
    fprintf('%d est impair.\n', n)
end

% Question 2 : signe
if n > 0
    disp('Positif')
elseif n < 0
    disp('Negatif')
else
    disp('Nul')
end

% Question 3 : FizzBuzz
if mod(n, 3) == 0 && mod(n, 5) == 0
    disp('FizzBuzz')
elseif mod(n, 3) == 0
    disp('Fizz')
elseif mod(n, 5) == 0
    disp('Buzz')
else
    fprintf('%d n''est pas divisible par 3 ni par 5.\n', n)
end
```

2.2.2 Scripts MATLAB

Un **script** est un fichier texte avec l'extension `.m` qui contient une séquence de commandes MATLAB. Il s'exécute en entier, dans l'ordre, en tapant son nom dans la *Command Window*.

Pour créer un script : *Home* → *New Script* (ou `Ctrl+N`). Le fichier doit être sauvegardé dans le répertoire courant de MATLAB.

[Script : table de multiplication] Écrire un script `table_mult.m` qui :

1. demande à l'utilisateur un entier n entre 1 et 10 ;

2. affiche la table de multiplication de n (de $1 \times n$ à $10 \times n$) ;
3. signale une erreur si n est hors de l'intervalle $[1, 10]$.

Fichier : `table_mult.m`

```
% table_mult.m
% Affiche la table de multiplication d'un entier n

n = input('Entrez un entier entre 1 et 10 : ');

if n < 1 || n > 10
    error('Erreur : n doit etre entre 1 et 10.')
end

fprintf('\n--- Table de multiplication de %d ---\n', n)
for k = 1:10
    fprintf('%2d x %2d = %3d\n', k, n, k*n)
end
```

Résultat attendu (pour $n = 7$) :

```
--- Table de multiplication de 7 ---
 1 x  7 =   7
 2 x  7 =  14
 3 x  7 =  21
 4 x  7 =  28
 5 x  7 =  35
 6 x  7 =  42
 7 x  7 =  49
 8 x  7 =  56
 9 x  7 =  63
10 x  7 =  70
```

[Script : statistiques descriptives] Écrire un script `stats.m` qui :

1. définit le vecteur de notes $v = [12,\ 15,\ 8,\ 17,\ 10,\ 14,\ 11,\ 9,\ 16,\ 13]$;
2. calcule et affiche : moyenne, médiane, écart-type, note max et note min ;
3. compte et affiche le nombre d'étudiants ayant une note ≥ 10.

Fichier : `stats.m`

```
% stats.m
% Calcul de statistiques descriptives sur un vecteur de notes

v = [12, 15, 8, 17, 10, 14, 11, 9, 16, 13];

fprintf('=== Statistiques des notes ===\n')
fprintf('Moyenne    : %.2f\n', mean(v))
fprintf('Mediane    : %.2f\n', median(v))
fprintf('Ecart-type : %.2f\n', std(v))
fprintf('Note max   : %d\n',    max(v))
fprintf('Note min   : %d\n',    min(v))

nb_reussi = sum(v >= 10);
```

```
fprintf('Etudiants >= 10 : %d sur %d\n', nb_reussi, length(v)
    )
```

Résultat attendu :

```
=== Statistiques des notes ===
Moyenne     : 12.50
Mediane     : 12.50
Ecart-type  : 2.87
Note max    : 17
Note min    : 8
Etudiants >= 10 : 8 sur 10
```

2.2.3 Application 8 – Branchement conditionnel (if/elseif/else)

Objectif

Implémenter une fonction définie par morceaux à l'aide de la structure `if/elseif/else/end`.

Exercice 8

Écrivez un script MATLAB qui demande à l'utilisateur une valeur de x et évalue la fonction définie par morceaux :

$$f(x) = \begin{cases} x^2 + 4 & \text{si } x < 2 \\ \sqrt{x-2} & \text{si } 2 \leq x < 20 \\ 2x + 1 & \text{si } x \geq 20 \end{cases}$$

Le programme doit afficher le nom de la branche choisie ainsi que la valeur calculée.

SOLUTION

Sauvegardez le script sous le nom `fonction_morceaux.m` :

```
%% fonction_morceaux.m
%  Evaluation d'une fonction definie par morceaux
%  ------------------------------------------------
clear all
clc

x = input('Entrez la valeur de x : ');

if x < 2
    disp('Branche 1 : f(x) = x^2 + 4')
    f = x^2 + 4;

elseif (x >= 2) && (x < 20)
```

```
    disp('Branche 2 : f(x) = sqrt(x - 2)')
    f = sqrt(x - 2);

else    % x >= 20
    disp('Branche 3 : f(x) = 2x + 1')
    f = 2*x + 1;
end

fprintf('f(%g) = %g\n', x, f)
```

Explication ligne par ligne :

- `input(...)` : lit une valeur au clavier et la stocke dans `x`.
- `if x < 2` : première condition ; si vraie, calcule $x^2 + 4$.
- `elseif (x >= 2) && (x < 20)` : deuxième condition (en employant `&&` pour les scalaires).
- `else` : cas restant, obligatoirement $x \geq 20$.
- `fprintf` : affichage formaté, plus souple que `disp`.

Sortie MATLAB attendue

Test avec $x = 7$ (branche 2) :

```
Entrez la valeur de x : 7
Branche 2 : f(x) = sqrt(x - 2)
f(7) = 2.2361
```

Test avec $x = -3$ (branche 1) :

```
Entrez la valeur de x : -3
Branche 1 : f(x) = x^2 + 4
f(-3) = 13
```

Test avec $x = 25$ (branche 3) :

```
Entrez la valeur de x : 25
Branche 3 : f(x) = 2x + 1
f(25) = 51
```

Remarques / Extensions

— Utilisez `&&` et `||` (court-circuit) pour les conditions scalaires, et `&` / `|` pour les tableaux.

— `fprintf('%g', x)` affiche x sans zéros superflus. `%f` affiche toujours 6 décimales.

— **Extension :** Vectorisez la fonction : `f = (x<2).*(x.^2+4) + (x>=2 & x<20).*sqrt(max(x-2,0)) + (x>=20).*(2*x+1).`

2.2.4 Application 9 – Boucle `for` : calcul de somme

Objectif

Utiliser une boucle `for` pour calculer la somme $S = 1 + 2 + \cdots + n$ et vérifier le résultat par la formule de Gauss.

Exercice 9

1. Écrivez un script `somme.m` qui lit un entier n et calcule $S_n = \sum_{i=1}^{n} i$ à l'aide d'une boucle `for`.
2. Vérifiez le résultat en utilisant la formule de Gauss : $S_n = \dfrac{n(n+1)}{2}$.
3. Comparez les deux résultats et affichez un message indiquant s'ils coïncident.

SOLUTION

```
%% somme.m
%  Calcul de la somme 1 + 2 + ... + n par boucle et par
   formule
%
   ------------------------------------------------------------

clear all
clc

n = input('Entrez un entier naturel n : ');

if n <= 0 || floor(n) ~= n
    error('n doit etre un entier strictement positif.')
end

% Methode 1 : boucle for
S_boucle = 0;
for i = 1:n
    S_boucle = S_boucle + i;
end

% Methode 2 : formule de Gauss
S_gauss = n * (n + 1) / 2;

fprintf('Methode boucle  : S(%d) = %d\n', n, S_boucle)
fprintf('Formule de Gauss: S(%d) = %d\n', n, S_gauss)

if S_boucle == S_gauss
    disp('Les deux methodes donnent le meme resultat.')
else
    disp('Erreur : les resultats different !')
```

```
end
```

Algorithme détaillé (trace pour $n = 7$) :

Itération i	S avant	S après
1	0	1
2	1	3
3	3	6
4	6	10
5	10	15
6	15	21
7	21	28

Sortie MATLAB attendue

```
Entrez un entier naturel n : 7
Methode boucle  : S(7) = 28
Formule de Gauss: S(7) = 28
Les deux methodes donnent le meme resultat.
```

Solution MATLAB vectorisée (sans boucle) :

```
S_vect = sum(1:n);    % sum() sur le vecteur 1:n
```

Remarques / Extensions

- La boucle `for` a ici un intérêt **pédagogique**. En pratique, MATLAB est optimisé pour les opérations vectorisées : `sum(1:n)` est beaucoup plus rapide.
- `floor(n) ~= n` teste si n n'est pas un entier (comparaison après troncature).
- **Extension :** Calculez la somme des carrés $\sum i^2 = \frac{n(n+1)(2n+1)}{6}$ et des cubes $\sum i^3 = \left(\frac{n(n+1)}{2}\right)^2$.

2.2.5 Application 10 – Boucles imbriquées : table de multiplication

Objectif

Utiliser des boucles `for` imbriquées pour construire et afficher une table de multiplication 10×10.

Exercice 10

Écrivez un script MATLAB qui construit et affiche la table des produits $x \times y$ pour x et y variant de 0 à 10.

SOLUTION

```
%% table_multiplication.m
clear all; close all; clc

x = 0:10;
y = 0:10;

% Construction de la matrice produit via boucles
P = zeros(length(x), length(y));

for i = 1:length(x)
    for j = 1:length(y)
        P(i,j) = x(i) * y(j);
    end
end

disp('Table de multiplication (0 a 10) :')
disp(P)
```

Solution vectorisée (sans boucle) :

```
P = (0:10)' * (0:10);    % produit externe : matrice 11x11
disp(P)
```

Sortie MATLAB attendue

```
P =
     0     0     0     0     0     0     0     0     0     0
     0
     0     1     2     3     4     5     6     7     8     9
    10
     0     2     4     6     8    10    12    14    16    18
    20
   ...
     0    10    20    30    40    50    60    70    80    90
   100
```

Remarques / Extensions

— L'expression `(0:10)' * (0:10)` exploite le produit externe (vecteur colonne × vecteur ligne) pour construire la matrice en une seule ligne : élégant et rapide.

— **Erreur classique :** Confondre `*` (produit matriciel) et `.*` (produit élé-

ment par élément) pour des vecteurs de même dimension.

2.2.6 Application 11 – Script : somme et produit de deux matrices

Objectif

Écrire un script permettant de calculer la somme et le produit de deux matrices $n \times m$ saisies par l'utilisateur.

Exercice 11

Écrivez un script MATLAB qui :

1. Demande à l'utilisateur de saisir deux matrices A et B de mêmes dimensions $n \times m$.
2. Calcule et affiche leur somme $S = A + B$ (terme à terme).
3. Calcule et affiche leur produit matriciel $P = A \times B^T$ (si les dimensions sont compatibles).

SOLUTION

```
%% matrices_somme_produit.m
clear all; close all; clc

A = input('Entrez la matrice A (ex: [1 2 3; 4 5 6; 1 3 4])
    : ');
B = input('Entrez la matrice B (meme taille que A) : ');

[n, m] = size(A);

if ~isequal(size(A), size(B))
    error('A et B doivent avoir les memes dimensions !')
end

% Calcul de la somme via boucle (illustration pedagogique)
S = zeros(n, m);
P = 0;

for i = 1:n
    for j = 1:m
        S(i,j) = A(i,j) + B(i,j);
        P = P + A(i,j) * B(j,i);
    end
end

fprintf('Dimensions : n = %d, m = %d\n', n, m)
disp('Somme S = A + B :')
```

```
disp(S)
fprintf('Produit scalaire total P = %g\n', P)

% Version vectorisee
disp('Verification (version vectorisee) :')
disp(A + B)
```

Sortie MATLAB attendue

```
Entrez la matrice A : [1 2 3; 4 5 6; 1 3 4]
Entrez la matrice B : [1 0 3; 4 1 6; 4 3 4]
Dimensions : n = 3, m = 3
Somme S = A + B :
     2     2     6
     8     6    12
     5     6     8
```

2.2.7 Fonctions MATLAB

Une **fonction** est un programme réutilisable qui reçoit des *arguments en entrée* et retourne des *valeurs en sortie*. Elle est définie dans un fichier `.m` portant le même nom qu'elle.

```
function [sortie1, sortie2] = nom_fonction(entree1, entree2)
%NOM_FONCTION  Description courte
%   Description detaillee si necessaire

    % corps de la fonction
    sortie1 = ...;
    sortie2 = ...;

end
```

La première ligne de commentaire après la déclaration de la fonction constitue l'aide accessible via help nom_fonction.

[Fonction : conversion d'unités] Écrire une fonction `conv_temp.m` qui convertit une température :

1. de Celsius en Fahrenheit : $F = \frac{9}{5}C + 32$;
2. la fonction prend en argument la température en Celsius et retourne la valeur en Fahrenheit ;
3. tester avec $0C$, $100C$ et $-40C$.

Fichier : `conv_temp.m`

```
function F = conv_temp(C)
%CONV_TEMP  Convertit une temperature de Celsius en
   Fahrenheit
```

```
%   F = conv_temp(C) retourne la temperature F en Fahrenheit
%   correspondant a C degres Celsius.

    F = (9/5) * C + 32;

end
```

Appel depuis la Command Window :

```
>> conv_temp(0)
ans = 32

>> conv_temp(100)
ans = 212

>> conv_temp(-40)
ans = -40
```

[Fonction : factorielle et combinaisons]

1. Écrire une fonction `fact.m` qui calcule $n!$ (sans utiliser factorial).
2. Écrire une fonction `combinaison.m` qui calcule $\binom{n}{k} = \dfrac{n!}{k!\,(n-k)!}$ en appelant `fact`.
3. Calculer $\binom{10}{3}$ et $\binom{6}{2}$.

Fichier : `fact.m`

```
function f = fact(n)
%FACT  Calcule n! (factorielle de n)

    if n == 0 || n == 1
        f = 1;
    else
        f = 1;
        for k = 2:n
            f = f * k;
        end
    end

end
```

Fichier : `combinaison.m`

```
function c = combinaison(n, k)
%COMBINAISON  Calcule le coefficient binomial C(n,k)

    if k < 0 || k > n
        error('On doit avoir 0 <= k <= n.')
    end
    c = fact(n) / (fact(k) * fact(n - k));

end
```

Appel :

```
>> combinaison(10, 3)
ans = 120

>> combinaison(6, 2)
ans = 15
```

[Fonction avancée : résolution d'équation du second degré] Écrire une fonction `solve_quad.m` qui :

1. prend en entrée les coefficients a, b, c d'un trinôme $ax^2 + bx + c = 0$;
2. retourne les racines réelles si $\Delta \geq 0$, ou un message si $\Delta < 0$;
3. teste la fonction sur : $2x^2 - 5x + 3 = 0$ et $x^2 + x + 1 = 0$.

Fichier : `solve_quad.m`

```
function [x1, x2] = solve_quad(a, b, c)
%SOLVE_QUAD  Resout l'equation ax^2 + bx + c = 0
%   [x1, x2] = solve_quad(a, b, c)

    delta = b^2 - 4*a*c;

    if delta > 0
        x1 = (-b + sqrt(delta)) / (2*a);
        x2 = (-b - sqrt(delta)) / (2*a);
        fprintf('Deux racines reelles : x1=%.4f, x2=%.4f\n',
           x1, x2)

    elseif delta == 0
        x1 = -b / (2*a);
        x2 = x1;
        fprintf('Racine double : x = %.4f\n', x1)

    else
        x1 = NaN; x2 = NaN;
        fprintf('Pas de racine reelle (delta = %.4f < 0)\n',
           delta)
    end

end
```

Tests :

```
>> [x1, x2] = solve_quad(2, -5, 3)
>> [x1, x2] = solve_quad(1,  1, 1)
```

Résultat attendu :

```
Deux racines reelles : x1=1.5000, x2=1.0000
Pas de racine reelle (delta = -3.0000 < 0)
```

2.2.8 Application 12 – Fonctions MATLAB

Objectif

Créer des fonctions MATLAB réutilisables : factorielle, somme répétée, périmètre/surface d'un cercle, somme et produit des N premiers entiers.

Exercice 12

1. Écrivez la fonction `factoriel(n)` qui calcule $n!$.
2. Écrivez la fonction `somme_rep(x, n)` qui calcule $x+x+\cdots+x$ (n fois), c'est-à-dire $n \cdot x$.
3. Écrivez la fonction `cercle(R)` qui retourne le périmètre *et* la surface d'un cercle de rayon R.
4. Écrivez la fonction `somme_produit(N)` qui retourne la somme *et* le produit des N premiers entiers non nuls.

SOLUTION

Fonction 1 : `factoriel.m`

```
function f = factoriel(n)
% FACTORIEL(n) : calcule n! = 1 * 2 * ... * n
    if n < 0 || floor(n) ~= n
        error('n doit etre un entier positif ou nul.')
    end
    f = 1;
    for i = 2:n
        f = f * i;
    end
end
```

Sortie MATLAB attendue

```
>> factoriel(4)
ans = 24
>> factoriel(0)
ans = 1
>> factoriel(7)
ans = 5040
```

Fonction 2 : `somme_rep.m`

```
function s = somme_rep(x, n)
% SOMME_REP(x,n) : calcule x + x + ... + x  (n fois)
    s = 0;
    for i = 1:n
        s = s + x;
    end
```

```
end
```

Fonction 3 : `cercle.m`

```
function [perimetre, surface] = cercle(R)
% CERCLE(R) : perimetre et surface d'un cercle de rayon R
    if R < 0
        error('Le rayon doit etre positif.')
    end
    perimetre = 2 * pi * R;
    surface   = pi * R^2;
end
```

Sortie MATLAB attendue

```
>> [p, s] = cercle(5)
p = 31.4159
s = 78.5398
```

Fonction 4 : `somme_produit.m`

```
function [S, P] = somme_produit(N)
% SOMME_PRODUIT(N) : somme et produit des N premiers
   entiers
    S = 0;  P = 1;
    for k = 1:N
        S = S + k;
        P = P * k;
    end
end
```

Sortie MATLAB attendue

```
>> [S, P] = somme_produit(5)
S = 15      % 1+2+3+4+5
P = 120     % 5!
```

Remarques / Extensions

- Une fonction MATLAB se définit dans un fichier `.m` portant le **même nom** que la fonction.
- Les sorties multiples se déclarent sous forme de vecteur : `function [out1, out2] = nom(...)`.
- MATLAB dispose de `factorial(n)` comme fonction intégrée ; la réécrire ici est un exercice pédagogique.
- L'instruction `help nomFonction` affiche les commentaires situés juste après la ligne `function` : documentez toujours vos fonctions !

2.2.9 Application 13 – Racines d'un polynôme

Objectif

Utiliser la fonction `roots` de MATLAB pour trouver les racines d'un polynôme de degré quelconque.

Exercice 13

Trouvez les racines du polynôme :

$$p(x) = 4x^5 - x^2 + 4x - 1$$

1. Encodez le polynôme sous forme de vecteur de coefficients.
2. Calculez ses racines avec `roots`.
3. Vérifiez une racine réelle par substitution.
4. Tracez le polynôme pour visualiser l'emplacement des racines.

SOLUTION

```
%% racines_polynome.m
clear all; close all; clc

% Vecteur des coefficients (du degre le plus haut au plus
    bas)
% p(x) = 4x^5 + 0*x^4 + 0*x^3 - x^2 + 4x - 1
coeffs = [4, 0, 0, -1, 4, -1];

r = roots(coeffs)

% Verification : substituer une racine reelle dans p(x)
r_reelle = r(imag(r) == 0);
if ~isempty(r_reelle)
    val = polyval(coeffs, r_reelle(1));
    fprintf('p(%g) = %g (doit etre ~0)\n', r_reelle(1),
        val)
end

% Trace du polynome
x = linspace(-2, 2, 500);
y = polyval(coeffs, x);

figure
plot(x, y, 'b-', 'LineWidth', 2)
hold on
yline(0, 'k--')
plot(real(r_reelle), zeros(size(r_reelle)), 'ro', ...
     'MarkerSize', 8, 'MarkerFaceColor', 'r')
grid on
xlabel('x')
```

```
ylabel('p(x)')
title('Polynome p(x) = 4x^5 - x^2 + 4x - 1')
legend('p(x)', 'y=0', 'Racines reelles')
```

Sortie MATLAB attendue

```
r =
   -0.0000 + 1.0000i
   -0.0000 - 1.0000i
    0.2500 + 0.0000i
   -0.9270 + 0.0000i    (approx.)
    0.6770 + 0.0000i    (approx.)
```

Le polynôme a 2 racines complexes conjuguées et 3 racines réelles.

Remarques / Extensions

- Le vecteur de coefficients de `roots` va **du degré le plus élevé au plus bas** : pour $4x^5 - x^2 + 4x - 1$, on entre `[4, 0, 0, -1, 4, -1]`.
- `polyval(p, x)` évalue le polynôme p au point (ou vecteur) x.
- Les racines complexes d'un polynôme réel apparaissent toujours par paires conjuguées.
- **Extension :** utilisez `poly(r)` pour retrouver les coefficients à partir des racines.

2.2.10 Exercices de synthèse

[Synthèse : table de vérité et opérateurs logiques] Écrire un script qui génère et affiche la table de vérité de $(A \wedge B) \vee (\neg A)$ pour toutes les combinaisons booléennes de A et B.

```
% table_verite.m
fprintf('  A   |   B   | (A AND B) OR (NOT A)\n')
fprintf('------|-------|----------------------\n')
for A = [0, 1]
    for B = [0, 1]
        resultat = (A && B) || (~A);
        fprintf('  %d   |   %d   |          %d\n', A, B, resultat
            )
    end
end
```

2.2.11 Application 14 – Représentation graphique 2D

Objectif

Tracer des courbes 2D avec différents marqueurs, gérer la fenêtre graphique, ajouter titres, axes et légendes.

Exercice 14

1. Tracez sur une même figure la fonction $f(x) = \dfrac{(x+1)\sin(x)}{x^2+1}$ et $z(x) = \sin(x)$ pour $x \in [-1, 3]$ avec un pas de 0.05. Ajoutez une grille, un titre, des labels et une légende.
2. Tracez $y = \sin(x)$ pour $x \in [0, 2\pi]$ avec un pas $\pi/6$ en utilisant trois marqueurs différents dans trois sous-graphiques.

SOLUTION

```
%% graphique_2D.m
clear all; close all; clc

%% --- Partie 1 : f(x) et sin(x) ---
x = -1:0.05:3;
f = ((x+1) .* sin(x)) ./ (x.^2 + 1);
z = sin(x);

figure(1)
hold on
plot(x, f, 'b-',  'LineWidth', 2, 'DisplayName', 'f(x)')
plot(x, z, 'r--', 'LineWidth', 2, 'DisplayName', 'sin(x)')
grid on
xlabel('x');  ylabel('y')
title('La fonction f et sin(x)')
legend('show')
hold off

%% --- Partie 2 : sin(x) avec marqueurs differents ---
x2 = 0:pi/6:2*pi;
y2 = sin(x2);

figure(2)

subplot(1,3,1)
plot(x2, y2, 'r:*', 'LineWidth', 1.5)
grid on;  title("plot(x,y,'r:*')")
xlabel('x');  ylabel('sin(x)')

subplot(1,3,2)
plot(x2, y2, 'k-.^', 'LineWidth', 1.5)
grid on;  title("plot(x,y,'k-.^')")
```

```
xlabel('x')

subplot(1,3,3)
plot(x2, y2, 'bs-', 'LineWidth', 1.5)
grid on;  title("plot(x,y,'bs-')")
xlabel('x')
```

Résumé des spécificateurs de tracé `plot` :

Couleur	Style de ligne	Marqueur
`b` bleu	`-` continu	`.` point
`r` rouge	`--` tirets	`o` cercle
`k` noir	`:` pointillés	`*` étoile
`g` vert	`-.` tiret-point	`^` triangle
`m` magenta		`s` carré

Remarques / Extensions

- `hold on` superpose les tracés ; `hold off` désactive la superposition.
- `subplot(n,m,p)` divise la figure en $n \times m$ zones et sélectionne la p-ième.
- `close all` ferme toutes les fenêtres graphiques ouvertes.
- Utilisez `'LineWidth', 2` pour des tracés plus lisibles lors des présentations.

2.2.12 Application 15 – Représentation graphique 3D

Objectif

Tracer des courbes et des histogrammes 3D avec `plot3` et `bar3`.

Exercice 15

1. Tracez une courbe 3D en utilisant $x = 1:5$, $y = [0, 2, 4, 1, 1, 3]$ et $z = 2:2:10$ avec des croix noires. Ajoutez la grille et les labels des axes.
2. Tracez un histogramme 3D avec $y = 1:6$ et $z = [33, 11, 5, 9, 22, 30]$.

SOLUTION

```
%% graphique_3D.m
clear all; close all; clc

%% --- Partie 1 : Courbe 3D ---
x = 1:5;
y = [0 2 4 1 3];    % 5 valeurs pour correspondre a x
```

```
z = 2:2:10;

figure(1)
plot3(x, y, z, 'k*', 'MarkerSize', 10)
grid on
xlabel('x'); ylabel('y'); zlabel('z')
title('Courbe 3D avec grilles')

%% --- Partie 2 : Histogramme 3D ---
y2 = 1:6;
z2 = [33 11 5 9 22 30];

figure(2)
bar3(y2, z2)
xlabel('x'); ylabel('y'); zlabel('z')
title('3D Bar')
```

Fonctions de visualisation 3D utiles :

Fonction	Description
`plot3(x,y,z)`	Courbe dans l'espace 3D
`bar3(y,z)`	Histogramme 3D
`surf(X,Y,Z)`	Surface colorée
`mesh(X,Y,Z)`	Surface en grille (maillage)
`contour(X,Y,Z)`	Courbes de niveau
`zlabel('z')`	Label de l'axe z

Remarques / Extensions

- Pour les surfaces, utilisez `meshgrid` pour créer les matrices X et Y : `[X,Y] = meshgrid(x,y)`.
- L'icône de rotation 3D dans la fenêtre graphique permet de faire pivoter la vue interactivement.
- `view(az, el)` règle l'azimut et l'élévation du point de vue par programme.

2.3 Récapitulatif des commandes MATLAB essentielles

Commande	**Description**
Environnement	
`clear all`	Supprime toutes les variables
`clc`	Efface la Command Window
`who / whos`	Liste les variables (simple / détaillée)
Constantes et fonctions scalaires	
`pi, eps, Inf, NaN`	π, epsilon machine, infini, indéfini
`realmin, realmax`	Plus petit / plus grand réel normalisé
`abs, sqrt, exp, log`	Valeur absolue, racine, exponentielle, logarithme
Vecteurs et matrices	
`a:h:b`	Vecteur de a à b avec pas h
`linspace(a,b,n)`	n valeurs équiréparties entre a et b
`zeros(m,n), ones(m,n), eye(n)`	Matrices spéciales
`size, length, numel`	Dimensions, longueur, nombre d'éléments
`.*, ./, .^`	Opérations élément par élément
`sum, prod, min, max`	Fonctions de réduction
`A'`	Transposée
`inv(A)`	Inverse (matrice carrée)
Chaînes de caractères	
`ischar, isempty, strcmp`	Tests sur chaînes
`length, upper, lower`	Longueur, casse
Structures de contrôle	
`if / elseif / else / end`	Branchement conditionnel
`for i = ... / end`	Boucle déterministe
`while / end`	Boucle conditionnelle
`function [out] = nom(in)`	Déclaration de fonction
Graphiques	
`plot, plot3, bar, bar3`	Tracés 2D et 3D
`hold on/off, grid, subplot`	Gestion de la figure
`xlabel, ylabel, zlabel, title, legend`	Étiquettes
`roots, polyval`	Racines et évaluation de polynômes

Fin du cahier d'exercices – Chapitre 2 MATLAB

Applications avancés

Les exemples suivants sont conçus pour être exécutés de manière interactive. Chacun illustre un concept mathématique ou informatique spécifique en utilisant la Fenêtre de Commandes de MATLAB.

Script 2.1— Détection des nombres premiers

```
% D terminer si un nombre est premier en utilisant isprime()
n = 97;
if isprime(n)
    fprintf('%d est premier.\n', n);
else
    fprintf('%d n''est pas premier.\n', n);
end

% Lister tous les premiers jusqu'   100
premiers_100 = primes(100)
fprintf('Il y a %d nombres premiers jusqu''   100.\n', numel(
   premiers_100));
```

Script 2.2 — Approximation par série de Taylor de e^x

```
% Approximer e^x avec N termes de la s rie de Taylor :
% e^x ~= 1 + x + x^2/2! + x^3/3! + ... + x^N/N!

x = 1;      % approximer e^1 = e
N = 15;     % nombre de termes
approx = 0;
for k = 0:N
    approx = approx + x^k / factorial(k);
end
exact = exp(x);
fprintf('Approx. Taylor (%d termes) : %.15f\n', N+1, approx);
fprintf('Valeur exacte              : %.15f\n', exact);
fprintf('Erreur absolue             : %.2e\n',  abs(approx -
   exact));
```

Script 2.3 — Statistiques sur un jeu de données

```
% Notes simul es d'un examen (30   tudiants )
notes = [72 85 91 68 78 55 88 92 74 65 ...
         80 83 77 90 62 71 84 79 93 70 ...
         58 87 75 81 67 89 73 76 82 69];

fprintf('--- Statistiques descriptives ---\n');
fprintf('Nb   tudiants   : %d\n',   numel(notes));
fprintf('Moyenne       : %.2f\n', mean(notes));
fprintf('M diane       : %.2f\n', median(notes));
fprintf(' cart -type   : %.2f\n', std(notes));
fprintf('Min / Max     : %d / %d\n', min(notes), max(notes));

% Compter les   tudiants   dans chaque tranche de note
```

```
tranches  = {'A (>=90)','B (80-89)','C (70-79)','D (60-69)','
   F (<60)'};
effectifs = [sum(notes>=90), sum(notes>=80 & notes<90), ...
             sum(notes>=70 & notes<80), sum(notes>=60 & notes
                <70), ...
             sum(notes<60)];
fprintf('\n--- R partition des notes ---\n');
for k = 1:5
    fprintf('  %s : %d  tudiants \n', tranches{k}, effectifs(
       k));
end

% Histogramme
figure;
histogram(notes, 10, 'FaceColor', [0.2 0.4 0.8], 'EdgeColor',
    'white');
title('Distribution des notes');
xlabel('Note'); ylabel('Nombre d'' tudiants ');
grid on;
```

Script 2.4 — Recherche des racines d'un polynôme

```
% Polyn me : p(x) = x^4 - 8x^3 + 19x^2 - 12x
% Vecteur des coefficients (puissances d croissantes) :
p = [1 -8 19 -12 0];

% Trouver les racines
r = roots(p);
fprintf('Racines du polyn me :\n');
disp(r);

%  valuer  et tracer
x = linspace(-0.5, 5.5, 400);
y = polyval(p, x);

figure;
plot(x, y, 'b-', 'LineWidth', 2);
hold on;
racines_reelles = r(imag(r)==0);
plot(racines_reelles, zeros(size(racines_reelles)), 'ro', ...
     'MarkerSize', 10, 'MarkerFaceColor', 'r');
hold off;
yline(0, 'k--');
title('Polyn me et ses racines');
xlabel('x'); ylabel('p(x)');
grid on;
```

Script 2.5 — Intégration numérique

```
% Comparer trois m thodes d'int gration pour l'int grale
   de sin(x) de 0    pi
% Valeur exacte = 2
a = 0; b = pi;
f = @(x) sin(x);
exact = 2;

% M thode 1 : integral() int gr
I1 = integral(f, a, b);
fprintf('integral()        : %.10f    erreur : %.2e\n', I1, abs
   (I1-exact));

% M thode 2 : R gle des trap zes
n = 1000;
x = linspace(a, b, n+1);
I2 = trapz(x, f(x));
fprintf('trapz (n=%d)     : %.10f    erreur : %.2e\n', n, I2,
   abs(I2-exact));

% M thode 3 : R gle de Simpson (manuelle)
h = (b-a)/n;
x = linspace(a, b, n+1);
w = [1 repmat([4 2], 1, n/2-1) 4 1];    % pond rations de
   Simpson
I3 = (h/3) * dot(w, f(x));
fprintf('Simpson (n=%d) : %.10f    erreur : %.2e\n', n, I3,
   abs(I3-exact));
```

Les scripts complets et bien documentés suivants illustrent une programmation MATLAB de niveau professionnel.

Script 2.6 — Méthode de Bissection pour la Recherche de Racines

La méthode de bissection est un algorithme d'encadrement fiable pour trouver une racine d'une fonction continue $f(x) = 0$ sur un intervalle $[a, b]$ où $f(a)$ et $f(b)$ ont des signes opposés.

```
%
   ============================================================

% Script  : methode_bissection.m
% Objectif: Trouver la racine de f(x) = x^3 - 2x - 5 sur [2,
   3]
% M thode : Bissection
%
   ============================================================
```

```
clc; clear; close all;

% --- D finir la fonction et l'intervalle
    ----------------------
f = @(x) x.^3 - 2*x - 5;
a = 2;   b = 3;
tol = 1e-8;    maxIter = 100;

% --- Valider l'encadrement
    ------------------------------------
if f(a) * f(b) > 0
    error('f(a) et f(b) doivent avoir des signes oppos s.');
end

% --- It rations de bissection
    --------------------------------
fprintf('%-6s %-12s %-12s %-12s %-12s\n', ...
        'Iter', 'a', 'b', 'milieu', '|f(mil)|');
for k = 1:maxIter
    mil   = (a + b) / 2;
    f_mil = f(mil);
    err   = abs(f_mil);
    fprintf('%-6d %-12.8f %-12.8f %-12.8f %-12.2e\n', k, a, b
       , mil, err);
    if err < tol || (b - a)/2 < tol
        fprintf('\nConverg  en %d it rations.\n', k);
        break;
    end
    if f(a) * f_mil < 0
        b = mil;    % la racine est dans la moiti  gauche
    else
        a = mil;    % la racine est dans la moiti  droite
    end
end
racine = mil;
fprintf('Racine ~= %.10f\n', racine);
fprintf('f(racine) = %.2e\n', f(racine));

% --- Visualisation
    -------------------------------------------
x = linspace(1.5, 3.5, 400);
figure;
plot(x, f(x), 'b-', 'LineWidth', 2); hold on;
plot(racine, 0, 'ro', 'MarkerSize', 12, 'MarkerFaceColor', 'r
   ');
yline(0, 'k--'); hold off;
title('M thode de Bissection : f(x) = x^3 - 2x - 5');
xlabel('x'); ylabel('f(x)');
legend('f(x)', sprintf('Racine ~= %.6f', racine));
grid on;
```

Script 2.7 — Élimination de Gauss pour les Systèmes Linéaires

L'élimination de Gauss transforme systématiquement la matrice augmentée $[A \mid b]$ en forme triangulaire supérieure par des opérations sur les lignes, puis résout par substitution inverse.

```
%
    ================================================================

% Script  : elimination_gaussienne.m
% Objectif: R soudre Ax = b par   limination   de Gauss avec
    pivot partiel
%
    ================================================================

clc; clear;

A = [ 2   1  -1;
      -3  -1   2;
      -2   1   2];
b = [8; -11; -3];
n = length(b);

fprintf('Syst me Ax = b :\n'); disp([A b]);

% ---   limination   vers l'avant avec pivot partiel
    -------------
Aug = [A, b];
for col = 1:n-1
    [~, ligne_pivot] = max(abs(Aug(col:n, col)));
    ligne_pivot = ligne_pivot + col - 1;
    Aug([col, ligne_pivot], :) = Aug([ligne_pivot, col], :);
        %   changer   les lignes
    for ligne = col+1:n
        facteur = Aug(ligne, col) / Aug(col, col);
        Aug(ligne, :) = Aug(ligne, :) - facteur * Aug(col, :)
            ;
    end
end
fprintf('\nForme triangulaire sup rieure :\n'); disp(Aug);

% --- Substitution inverse
    ------------------------------------
x = zeros(n, 1);
for k = n:-1:1
    x(k) = (Aug(k, end) - Aug(k, k+1:n) * x(k+1:n)) / Aug(k,
        k);
end
fprintf('\nSolution x :\n'); disp(x);

% --- V rification
    ----------------------------------------------
```

```
residu = norm(A*x - b);
fprintf('R sidu ||Ax - b|| = %.2e\n', residu);
x_matlab = A \ b;
fprintf('Diff rence max avec la solution MATLAB : %.2e\n',
   max(abs(x - x_matlab)));
```

Script 2.8 — Simulation d'Équation Différentielle (Méthode d'Euler)

La méthode d'Euler fait avancer $\frac{dy}{dt} = f(t, y)$ pas à pas : $y(t+h) \approx y(t) + h \cdot f(t, y(t))$.

```
%
   ================================================================

% Script  : euler_ode.m
% Objectif: R soudre dy/dt = -2y + cos(t), y(0) = 1
% Comparer: M thode d'Euler vs. ode45 MATLAB vs. solution
   exacte
%
   ================================================================

clc; clear; close all;

f  = @(t, y) -2*y + cos(t);
t0 = 0; tf = 5; y0 = 1;
h  = 0.1;    % pas pour la m thode d'Euler

% --- M thode d'Euler
   ----------------------------------------
t_euler = t0:h:tf;
N = length(t_euler);
y_euler = zeros(1, N);
y_euler(1) = y0;
for k = 1:N-1
    y_euler(k+1) = y_euler(k) + h * f(t_euler(k), y_euler(k))
       ;
end

% --- MATLAB ode45 (Runge-Kutta 4/5)
   --------------------------
[t_ode, y_ode] = ode45(f, [t0 tf], y0);

% --- Solution exacte
   ----------------------------------------
exacte = @(t) (2*cos(t) + sin(t))/5 + (y0 - 2/5)*exp(-2*t);
t_exacte = linspace(t0, tf, 500);

% --- Comparaison graphique
```

```
    ----------------------------------
figure;
plot(t_exacte, exacte(t_exacte), 'k-',  'LineWidth', 2.5, '
    DisplayName', 'Exacte');
hold on;
plot(t_euler, y_euler, 'r--', 'LineWidth', 1.5, ...
     'DisplayName', sprintf('Euler (h=%.2f)', h));
plot(t_ode, y_ode, 'b:', 'LineWidth', 2, 'DisplayName', '
    ode45');
hold off;
title('Solution ODE : dy/dt = -2y + cos(t)');
xlabel('t'); ylabel('y(t)');
legend('Location', 'northeast'); grid on;

erreur_euler = max(abs(y_euler - exacte(t_euler)));
fprintf('Erreur max Euler (h=%.2f) : %.4e\n', h, erreur_euler
    );
```

Script 2.9 — Approximation par Série de Fourier

Une série de Fourier représente une fonction périodique comme une somme infinie de sinus et de cosinus. Ce script illustre les approximations par séries de Fourier partielles d'un signal carré (phénomène de Gibbs).

```
%
    ============================================================

% Script  : serie_fourier.m
% Objectif: Approximer un signal carr  avec N termes de
    Fourier
%
    ============================================================

clc; clear; close all;

x  = linspace(-pi, pi, 1000);
sq = sign(sin(x));     % signal carr  (exact)

figure;
plot(x, sq, 'k-', 'LineWidth', 1, 'DisplayName', 'Signal
    carr ');
hold on;

couleurs = {'r', 'b', 'g', 'm'};
N_vals = [1, 3, 9, 25];     % nombre de termes de Fourier

for idx = 1:length(N_vals)
    N = N_vals(idx);
    approx = zeros(size(x));
    for k = 1:2:2*N-1          % harmoniques impaires
```

```
        uniquement
        approx = approx + (4/pi) * (1/k) * sin(k*x);
    end
    plot(x, approx, [couleurs{idx} '--'], 'LineWidth', 1.5,
        ...
        'DisplayName', sprintf('N=%d termes', N));
end
hold off;
title('S rie de Fourier : Approximation du Signal Carr ');
xlabel('x'); ylabel('f(x)');
legend('Location', 'southeast');
ylim([-1.5 1.5]); grid on;
```

Script 2.10 — Pipeline Complet d'Analyse de Données

Ce script final illustre un workflow complet d'analyse de données : génération de données synthétiques, ajustement d'un modèle, analyse statistique et production d'une figure multi-panneaux professionnelle.

```
%
    ================================================================

% Script  : pipeline_analyse_donnees.m
% Objectif: Analyse compl te de donn es de temp rature
    simul es
%
    ================================================================

clc; clear; close all;
rng(42);     % fixer la graine al atoire pour la
    reproductibilit

% --- G n rer des donn es de temp rature annuelle
    synth tiques ( C ) ---
annees = (2000:2023)';
n      = length(annees);
tendance = 0.05 * (annees - 2000);    % tendance au
    r chauffement
bruit    = 0.5 * randn(n, 1);
temps    = 15 + tendance + bruit;

% --- Ajustement lin aire de tendance
    ---------------------------------
p     = polyfit(annees, temps, 1);
t_fit = polyval(p, annees);
taux_tendance = p(1);     %  C  par an

% --- Statistiques
    ------------------------------------------------------
stats.moyenne = mean(temps);
```

```
stats.ecart    = std(temps);
stats.min      = min(temps);
stats.max      = max(temps);
stats.etendue = stats.max - stats.min;

fprintf('=== Analyse des temp ratures (2000-2023) ===\n');
fprintf('Moyenne     : %.2f  C \n',  stats.moyenne);
fprintf(' cart -type : %.2f  C \n',  stats.ecart);
fprintf(' tendue     : %.2f  C  (%.2f    %.2f)\n', ...
        stats.etendue, stats.min, stats.max);
fprintf('Tendance    : %+.4f  C /an\n', taux_tendance);
fprintf('R chauffe. : %+.2f  C  sur 24 ans\n', taux_tendance
   *24);

% --- Figure multi-panneaux
   ----------------------------------------
figure('Position', [50 50 1100 850]);

% Panneau 1 : Donn es brutes + droite de tendance
subplot(2, 2, 1);
plot(annees, temps, 'bo-', 'LineWidth', 1.5, 'MarkerFaceColor
   ', 'b');
hold on;
plot(annees, t_fit, 'r-', 'LineWidth', 2.5);
hold off;
title('Temp rature Annuelle Moyenne');
xlabel('Ann e'); ylabel('Temp rature ( C )');
legend('Donn es', sprintf('Tendance : %+.3f  C/an',
   taux_tendance));
grid on;

% Panneau 2 : Anomalies ( cart  par rapport    la moyenne)
subplot(2, 2, 2);
anomalie = temps - stats.moyenne;
bar(annees, anomalie, 'FaceColor', [0.2 0.5 0.8]);
title('Anomalie de Temp rature ( cart  par rapport    la
   moyenne)');
xlabel('Ann e'); ylabel('Anomalie ( C )');
yline(0, 'k-'); grid on;

% Panneau 3 : Histogramme
subplot(2, 2, 3);
histogram(temps, 8, 'FaceColor', [0.2 0.6 0.4], 'EdgeColor',
   'white');
title('Distribution des Temp ratures');
xlabel('Temp rature ( C )'); ylabel('Fr quence');
xline(stats.moyenne, 'r--', 'LineWidth', 2, 'Label', 'Moyenne
   ');
grid on;

% Panneau 4 : R chauffement cumul
```

```
subplot(2, 2, 4);
cumulatif = cumsum(temps - temps(1));
area(annees, cumulatif, 'FaceColor', [0.9 0.3 0.2], '
   FaceAlpha', 0.4);
hold on;
plot(annees, cumulatif, 'r-', 'LineWidth', 2);
hold off;
title('Changement de Temp rature Cumul ');
xlabel('Ann e'); ylabel('Changement cumul  ( C )');
grid on;

sgtitle('Analyse Compl te des Donn es de Temp rature', '
   FontSize', 14, 'FontWeight', 'bold');
```

EXERCICE

Exercice final — Mini-Projet

En utilisant le pipeline d'analyse de données ci-dessus comme modèle, étendez le script pour :

1. Ajouter un second ajustement polynomial (degré 2) et le tracer aux côtés de l'ajustement linéaire.
2. Calculer et afficher la valeur R^2 pour les deux ajustements.
3. Ajouter un 5e sous-graphique montrant la différence entre les ajustements linéaire et quadratique.
4. Sauvegarder la figure finale sous `'rapport_temperature.png'` en utilisant `saveas()`.

Rappel : $R^2 = 1 - \dfrac{\sum(y - \hat{y})^2}{\sum(y - \bar{y})^2}$

SOLUTION

```
% Ajustement quadratique
p2     = polyfit(annees, temps, 2);
t_fit2 = polyval(p2, annees);

% R  pour l'ajustement lin aire
SS_res1 = sum((temps - t_fit).^2);
SS_tot  = sum((temps - mean(temps)).^2);
R2_lin  = 1 - SS_res1 / SS_tot;

% R  pour l'ajustement quadratique
SS_res2 = sum((temps - t_fit2).^2);
R2_quad = 1 - SS_res2 / SS_tot;

fprintf('R  lin aire    = %.4f\n', R2_lin);
fprintf('R  quadratique = %.4f\n', R2_quad);

% Ajouter l'ajustement quadratique au sous-graphique 1
subplot(2, 2, 1);
```

```
hold on;
plot(annees, t_fit2, 'g--', 'LineWidth', 2, ...
     'DisplayName', sprintf('Quadratique R =%.3f',
        R2_quad));
hold off;
legend;

% Sauvegarder la figure
saveas(gcf, 'rapport_temperature.png');
fprintf('Figure sauvegard e.\n');
```

RÉSUMÉ DE SECTION

— La Fenêtre de Commandes est idéale pour les calculs rapides et ciblés ; les scripts dans l'Éditeur gèrent les workflows complexes en plusieurs étapes.
— Incluez toujours `clc; clear; close all;` au début des scripts pour la reproductibilité.
— Utilisez `fprintf` pour une sortie formatée et lisible ; organisez le code en sections Entrée / Calcul / Sortie.
— Les méthodes numériques (bissection, Euler, élimination de Gauss) sont facilement implémentées et comparées.
— Les fonctions intégrées de MATLAB (`ode45`, `polyfit`, `roots`, `integral`) complètent les implémentations personnalisées.
— La visualisation avec `subplot` crée des figures multi-panneaux professionnelles ; `saveas` les exporte.

Remarques / Extensions

Les ouvrages suivants ont été utilisés comme sources de référence pour l'élaboration de ce cours. Ils constituent également des ressources complémentaires recommandées pour approfondir les notions abordées.

Bibliographie

[1] *Data Analysis Software : GNU Octave, Mathematica,* MATLAB, *Maple, Scilab, Social Network Analysis Software, LabVIEW, Eicaslab.* Books LLC, 2010.

[2] J.-T. Lapreste. *Outils mathématiques pour l'étudiant, l'ingénieur et le chercheur avec* MATLAB. Ellipses, 2008.

[3] J.-P. Grenier. *Débuter en Algorithmique avec* MATLAB *et Scilab.* Ellipses, 2007.

[4] S. Attaway. MATLAB *: A Practical Introduction to Programming and Problem Solving.* Butterworth-Heinemann, 2013.

[5] J. Koko. *Calcul scientifique avec* MATLAB *: outils* MATLAB *spécifiques, équations aux dérivées partielles.* Ellipses, 2009.

[6] J. Sizemore et J. P. Mueller. MATLAB *for Dummies.* John Wiley & Sons, 2014.

[7] A. Biran et M. Breiner. MATLAB *pour l'ingénieur.* Pearson Education, 2004.

[8] M. Mokhtari et A. Mesbah. *Apprendre et maîtriser* MATLAB. Springer-Verlag, 1997.

[9] J. Bastien et J.-N. Martin. *Introduction à l'analyse numérique : applications sous* MATLAB — *cours et exercices corrigés.* Dunod, 2003.

Lectures recommandées

Les ressources suivantes, non citées directement dans le cours, constituent des compléments utiles pour approfondir les thèmes abordés.

Algorithmique et programmation

— T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms* (3e éd.). MIT Press, 2009.

— R. Sedgewick et K. Wayne. *Algorithms* (4e éd.). Addison-Wesley, 2011.

Algèbre linéaire et matrices

— G. Strang. *Introduction to Linear Algebra* (5e éd.). Wellesley-Cambridge Press, 2016.

— L. N. Trefethen et D. Bau. *Numerical Linear Algebra.* SIAM, 1997.

Analyse numérique et optimisation

— J.-P. Demailly. *Analyse numérique et équations différentielles*. EDP Sciences, 2006.

— W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *Numerical Recipes : The Art of Scientific Computing* (3e éd.). Cambridge University Press, 2007.

— J. Nocedal et S. J. Wright. *Numerical Optimization* (2e éd.). Springer, 2006.

Statistiques et traitement de données

— S. Haykin. *Neural Networks and Learning Machines* (3e éd.). Prentice Hall, 2009.

— C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Ressources en ligne

— Documentation officielle MATLAB (MathWorks) :
https://www.mathworks.com/help/matlab/

— GNU Octave — logiciel libre compatible MATLAB :
https://www.gnu.org/software/octave/

— Scilab — alternative open-source :
https://www.scilab.org/

— MATLAB Central (forum et échange de fichiers) :
https://www.mathworks.com/matlabcentral/

— Developpez.com — tutoriels francophones :
https://matlab.developpez.com/

NOTE

[Note à l'étudiant] Cette liste bibliographique ne constitue qu'un point de départ. L'étudiant est vivement encouragé à consulter les ressources en ligne et à explorer la documentation officielle MATLAB pour compléter sa formation et mener à bien ses projets.