



Initiation et memento MATLAB

MATLAB, abréviation de MATrix LABoratory, est un langage très performant pour le calcul numérique.

Les utilisations classiques sont :

- Calculs mathématiques
- Développement d'algorithmes
- Modélisation et simulation
- Analyse de données et visualisation
- Développement d'applications telles que les interfaces utilisateur

Les commandes matlab sont organisées en boîtes à outils aussi appelées toolboxes.

Table des matières

1	Traitements, expressions et variables	3
2	Sauvegarde d'une session	3
3	Commandes d'aide	3
4	Matrices	4
4.1	Construction	4
4.1.1	Saisie d'une liste d'éléments	4
4.1.2	Construction à l'aide de fonctions	5
4.2	Modification de matrices	6
4.2.1	Modification manuelle	6
4.2.2	Modification du format - Réorganisation	6
4.2.3	Fonctions élémentaires	6
4.3	Opérations entre matrices	7
4.4	Opérations élément par élément	8
4.5	Fonctions spécifiques aux vecteurs et aux matrices	9
4.5.1	Pour les vecteurs	9
4.5.2	Pour les matrices	9
5	Boucles, tests et relations	10
5.1	Boucles	10
5.1.1	<i>For</i>	10
5.1.2	<i>While</i>	11
5.2	Test : instruction if	11
5.3	Relations	11
5.4	Pause et break	12

6 Fichiers .m

6.1 Scripts

6.2 Fonctions

6.3 Textes, entrées, messages d'erreurs

6.4 Vectorisation et pré-allocation

7 Gestion des fichiers

8 Graphiques

9 Bonnes pratiques de code

12

13

13

14

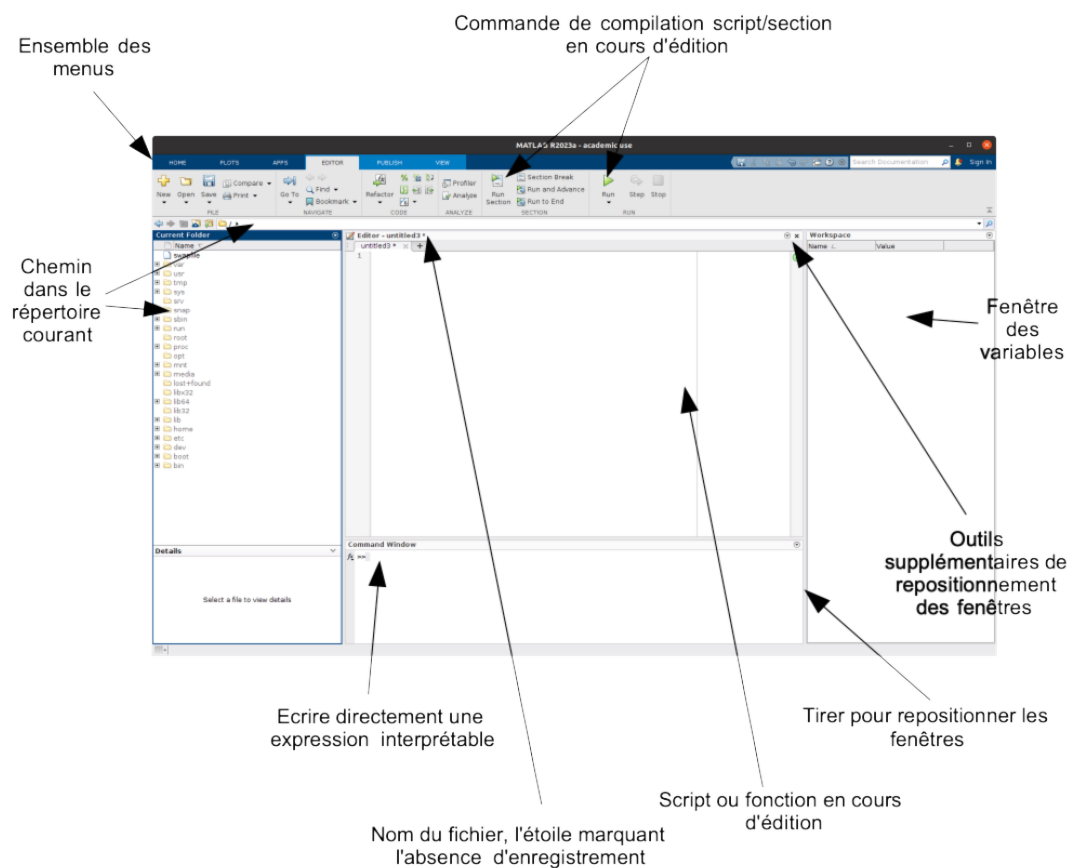
14

14

15

16

Au cours du BE, vous pouvez saisir vos instructions MATLAB soit directement dans la fenêtre de commande (ce qui est utile pour un test rapide) soit en les enregistrant dans un script (fichier portant l'extension .m, ce qui permet de les sauver et de les conserver). Dans un premier temps, vous pouvez utiliser les deux possibilités. Certaines règles à suivre pour la création de scripts et de fonctions sous MATLAB sont données dans la suite de ce document. Pour commencer, il est important de commenter vos scripts et fonctions : le caractère % permet d'insérer un commentaire en fin de ligne. En doublant ce caractère en début de ligne, il est possible de créer des sections dans le programme, c'est-à-dire des suites d'instructions qui pourront être exécutées séparément du reste du programme. Il faudra éviter de donner à vos fonctions, scripts et variables les mêmes noms que ceux de fonctions MATLAB existantes, cela ayant pour effet d'écraser (temporairement) ces fonctions existantes.



1 Traitements, expressions et variables

MATLAB est un langage qui utilise des expressions qui sont ensuite interprétées et évaluées. Les traitements ont généralement la forme suivante :

variable = (*expression*) ou simplement
expression

Les expressions sont généralement composées d'opérateurs, de fonctions et de noms de variables. L'évaluation d'une expression produit une matrice qui est ensuite affichée et assignée à la variable. Si le nom de la variable est oubliée, alors le résultat est automatiquement assigné à la variable **ans**. **Si le dernier caractère d'un traitement est un point virgule, alors l'affichage est supprimé.** Ceci est particulièrement utile lors de la création d'un script, l'affichage étant une opération extrêmement couteuse en temps de calcul. Il faudra faire attention à mettre ce symbole ";" à la fin de chaque ligne, et l'enlever uniquement lorsque l'on veut afficher un résultat ou lors du débogage.

Pour avoir une vue globale de l'espace de travail, l'on peut soit regarder la fenêtre "workspace", soit utiliser les fonctions suivantes :

who	liste les variables
whos	liste les variables, leur taille et leur type
what	liste les fichiers d'extension .m et .mat

Les variables peuvent être supprimées à l'aide de la commande **clear** suivi du nom de la variable. La commande **clear** permet de supprimer toutes les variables présentes dans l'espace de travail. Il est recommandé d'utiliser la commande **clear** en début d'un script. De même, **close all** placé en début de programme permet de fermer toutes les figures ouvertes précédemment. Enfin, la commande **clc** permet d'effacer le contenu de la fenêtre de commande (message d'erreurs et affichages de résultats précédents).

2 Sauvegarde d'une session

Un fichier MatLab s'enregistre comme tout autre type de document informatique. En revanche, lorsque l'on quitte MATLAB, toutes les variables sont perdues. On peut les sauver avec la commande **save nom_sauvegarde**, qui stocke les variables dans un fichier d'extension .mat (c'est-à-dire **nom_sauvegarde.mat**). Lorsque l'on relance MATLAB, on peut les recharger avec **load nom_sauvegarde**.

Exercice 0: Créer un fichier **BE0.m**, dans lequel vous effectuerez ce BE en autonomie

3 Commandes d'aide

Deux instructions seront très utiles :

- **lookfor** suivie d'un mot clé permet de trouver une commande relative à ce mot clé. Taper **lookfor average** pour obtenir le nom de la commande qui permet de calculer la moyenne empirique d'un ensemble de valeurs.
- **help** suivi d'un nom de commande donne l'aide sur la commande et en particulier les formats des entrées et des sorties. Afficher l'aide de la fonction qui permet de calculer la moyenne empirique.
- **help toolbox** donne toutes les commandes de la toolbox par catégories.

Exercice 1: Taper **help stats** dans la fenêtre de commande pour avoir une vision des potentialités de MATLAB en probabilités et statistiques.

4 Matrices

MATLAB travaille essentiellement sur des matrices, et toutes les variables sont représentées par des matrices. Un vecteur ou un scalaire étant une matrice avec une ligne et/ou une colonne. L'utilisation de vecteurs et de matrices permet une réduction de la complexité des calculs et permet l'économie de boucles par rapport à la plupart des langages de programmation

4.1 Construction

La saisie d'une matrice peut s'effectuer de différentes façons :

- soit en la rentrant comme une liste d'éléments ;
- soit en la générant à l'aide de fonctions ;
- soit en la chargeant depuis un fichier.

4.1.1 Saisie d'une liste d'éléments

$$A = [1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9]$$

crée une matrice 3×3 et l'affecte à la variable A. MATLAB affiche alors

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

Rappel : d'une manière générale, pour ne pas afficher la valeur d'une variable, ou le résultat d'une fonction ou d'un calcul, on écrit un “;” en fin de ligne. Ainsi,

$$A = [1 \ 2 \ 3; \ 4 \ 5 \ 6; \ 7 \ 8 \ 9];$$

construit la matrice 3×3 et l'affecte à la variable A, mais ne l'affiche pas.

Lors de la saisie d'une matrice, les éléments d'une ligne sont séparés par des espaces ou des virgules et les lignes sont séparées par des point virgules.

Les éléments d'une matrice sont référencés coordonnée par coordonnée. Ainsi, $A(2,3)$ est l'élément de la deuxième ligne et troisième colonne de la matrice A. Si l'on définit un vecteur B, alors $B(4)$ est le quatrième élément de ce vecteur, indifféremment qu'il soit ligne ou colonne. Les coordonnées des éléments doivent être des entiers **strictement positifs**, MatLab commençant à compter les indices à 1.

La commande “:” aussi appelée `column` permet aussi d'accéder aux sous-matrices d'une matrice. Par exemple, $A(1:4,3)$ est le vecteur colonne composé des 4 premiers éléments de la 3^{ème} colonne de la matrice A.

$A(:,3)$ est la troisième colonne de la matrice A et $A(1:4,:)$ est composée des quatre premières lignes de la matrice A.

Exercice 2: Construire manuellement la matrice

$$B = \begin{matrix} 1 & 0 & 9 \\ -4 & 8 & 5 \end{matrix}$$

Donner la commande qui permet d'extraire le coefficient -4 , puis celle pour extraire le coefficient 9.

Extraire de cette matrice la matrice

$$C = \begin{matrix} 1 & 0 \\ -4 & 8 \end{matrix}$$

MATLAB traite aussi les nombres complexes :

$$\begin{aligned} A &= [1 \ 2; \ 3 \ 4] + i * [5 \ 6; \ 7 \ 8] \\ A &= [1 + 5 * i \ 2 + 6 * i; \ 3 + 7 * i \ 4 + 8 * i] \end{aligned}$$

i ou j peuvent être utilisés indifféremment pour la notation imaginaire. Dans le cas où ils seraient utilisés par ailleurs comme variables, on peut les recréer à l'aide de :

$$ii = \text{sqrt}(-1)$$

Le nombre π est donné par `pi`.

4.1.2 Construction à l'aide de fonctions

Il existe sous MATLAB un certain nombre de fonctions permettant de construire des matrices ou des vecteurs :

<code>zeros</code>	matrice de zéros
<code>ones</code>	matrice de uns
<code>eye</code>	matrice identité
<code>linspace</code>	vecteur en progression arithmétique
<code>logspace</code>	vecteur en progression logarithmique
<code>diag</code>	création ou extraction de la diagonale d'une matrice

Par exemple, `zeros(m,n)` crée une matrice $m \times n$ remplie de zéros, et `zeros(n)` crée une matrice remplie de zéros et de taille $n \times n$.

Si x est un vecteur, `diag(x)` est la matrice diagonale avec x sur la diagonale, les autres éléments étant nuls. Si A est une matrice carrée, `diag(A)` est un vecteur composé de la diagonale de A .

Les matrices peuvent être construites par blocs.

Exercice 3: Soit A la matrice 3×3 précédente. Construire la matrice 5×5 suivante :

$$D = [A, \text{zeros}(3,2); \text{zeros}(2,3), \text{eye}(2)].$$

La commande `linspace` permet de générer des vecteurs dont les éléments suivent une progression arithmétique. Les variables d'entrée sont la valeur de départ, la valeur de fin et le nombre d'éléments. Lorsqu'il est plus pertinent de préciser le pas que le nombre d'éléments, on pourra utiliser la commande `r1:pas:rN` qui génère un vecteur de la forme `[r1 r1+pas r1+2pas ... rN]`. Lorsque le pas vaut 1, il est possible de l'omettre. Ainsi `1:5` est le vecteur ligne `[1 2 3 4 5]`. Les incréments peuvent ne pas être des entiers positifs : `0.2:0.2:1.2` est le vecteur `[0.2 0.4 0.6 0.8 1.0 1.2]` et `5:-1:1` est `[5 4 3 2 1]`.

Exercice 4: Construire ces vecteurs sous matlab en utilisant la commande : puis la commande `linspace` (après avoir consulté l'aide associée).

Remarque Les générateurs de nombres aléatoires permettent de générer des matrices dont les éléments sont des réalisations indépendantes de variables aléatoires de distribution donnée. La commande `rand(n)`, par exemple, crée une matrice de $n \times n$ éléments uniformément distribués entre 0 et 1.

`rand(m,n)` crée une matrice de taille $m \times n$ suivant la même distribution. La commande `randi(.)` permet de générer des entiers aléatoires de distribution uniforme sur $1, \dots, N$.

Exercice 5: Simuler le résultat de 10 lancers de dé indépendants. Vérifier la taille de votre résultat, et ce qui se passe lorsque vous répétez l'expérience.

Ces générateurs de nombres aléatoires seront plus largement étudiés dans le BE suivant.

4.2 Modification de matrices

4.2.1 Modification manuelle

Après initialisation de la matrice, on peut affecter une valeur scalaire à un élément d'une matrice comme suit :

$$A(2,4) = 8$$

qui va affecter la valeur 8 à l'élément de la deuxième ligne et de la quatrième colonne de **A**.

Il est possible de modifier en une opération (sans boucle sur les éléments) des lignes ou des colonnes particulières d'une matrice. Ainsi, taper `A(:, [2 4 5]) = B(:, 1:3)` remplacera les colonnes 2, 4 et 5 de **A** par les trois premières colonnes de **B**. Cette opération n'est possible que s'il y a adéquation des dimensions.

4.2.2 Modification du format - Réorganisation

La commande `reshape(A,m,n)` reconstruit la matrice **A** avec **m** lignes et **n** colonnes, en considérant les éléments colonne après colonne. Ainsi, si

$$E = \begin{matrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{matrix},$$

`F=reshape(E,2,3)` donne

$$F = \begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix}$$

Exercice 6: Utiliser la commande : pour générer un vecteur en progression arithmétique puis en appliquant la commande `reshape` retrouver les deux matrices **A** et **B**.
Consulter également l'aide associée aux commandes `fliplr`, `flipud`, `flip` et construire la matrice

$$G = \begin{matrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{matrix}$$

4.2.3 Fonctions élémentaires

La plupart des fonctions de base s'appliquent, sous MATLAB directement sur les matrices. MATLAB permet en effet de réaliser en une seule instruction des opérations qui nécessiteraient des boucles en Pascal ou en C. Les fonctions élémentaires disponibles sous MATLAB, s'appliquent aussi bien à des matrices qu'à des scalaires. Les plus courantes sont :

`sin, cos, tan, asin, acos, atan`
`sinh, cosh, tanh, asinh, acosh, atanh`
`exp, log, log10`
`rem` (pour reminder), `abs, angle, real, imag, conj, sqrt, sign`
`round, floor, ceil, fix, nchoosek`

4.3 Opérations entre matrices

Les opérations suivantes sont disponibles dans MATLAB :

+	addition
−	soustraction
*	multiplication
^	puissance
'	conjugué transposé si la matrice est complexe ou transposé si elle est réelle
.'	transposé
\	division à gauche
/	division à droite
inv	inversion d'une matrice carrée
kron	produit de Kronecker

Ces commandes s'appliquent aussi aux scalaires, aux vecteurs et aux matrices dès lors que les tailles sont compatibles. Si les tailles des matrices sont incompatibles, alors un message d'erreur apparaît, sauf dans le cas d'une opération entre un scalaire et une matrice où l'opérateur va s'appliquer au scalaire et aux éléments de la matrice.

Exercice 7: Construire en modifiant le format d'une matrice puis en utilisant les opérations élémentaires la matrice

$$H = \begin{bmatrix} 10 & 9 & 8 & 7 & 6 & 5 \end{bmatrix}$$

Laquelle des deux méthodes est, selon vous, moins gourmande en opérations ?

Exercice 8: Utiliser les commandes **kron** et **ones** pour générer le vecteur [123123123]. On pourra également effectuer un produit entre vecteurs et utiliser la fonction **reshape** pour obtenir le même résultat.

Remarque concernant la division matricielle : soit A une matrice inversible, et b un vecteur respectivement colonne ou ligne, alors :

$\mathbf{x} = A \backslash \mathbf{b}$ est la solution de $A * x = b$, et

$\mathbf{x} = A / \mathbf{b}$ est la solution de $x * A = b$.

Les divisions à droite et à gauche sont liées par $\mathbf{b} / A = (A' \backslash \mathbf{b}')'$.

- pour résoudre le système $A * x = b$, on peut aussi écrire $\mathbf{x} = \text{inv}(A) * \mathbf{b}$, mais ceci n'est pas équivalent (dans son fonctionnement) à $\mathbf{x} = A \backslash \mathbf{b}$. En effet, $\mathbf{x} = \text{inv}(A) * \mathbf{b}$ fait le calcul explicite de l'inverse de A , alors que $\mathbf{x} = A \backslash \mathbf{b}$ résout le système sans inverser la matrice, ce qui est plus rapide.
- on peut aussi résoudre le système $A * x = b$ à l'aide de $\mathbf{x} = A \backslash \mathbf{b}$ dans le cas où la matrice est de taille $m \times n$ avec $m \geq n$.

4.4 Opérations élément par élément

Les opérations précédentes sont les opérations matricielles classiques. Les opérations $*$, \wedge , \backslash , $/$ peuvent aussi s'appliquer élément par élément sur une matrice ou un vecteur en les faisant précéder d'un point. Par exemple :

`[1 2 3 4].*[1 2 3 4]` ou `[1 2 3 4].^2`
donnent `[1 4 9 16]`.

Exercice 9: A l'aide des commandes : et **reshape**, créer la matrice $K(i, j)$ telle que $K(i, j) = 2(j - 1) + i$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, c'est-à-dire :

$$K = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 \end{bmatrix}.$$

Créer alors la matrice $L(i, j)$ telle que $L(i, j) = 16 - (2(j - 1) + i)$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, c'est-à-dire :

$$L = \begin{bmatrix} 15 & 13 & 11 & 9 & 7 & 5 & 3 & 1 \\ 14 & 12 & 10 & 8 & 6 & 4 & 2 & 0 \end{bmatrix}.$$

par soustraction matricielle. Construire également L de la même façon que K en utilisant les commandes : et **reshape**.

À partir des matrices K et L , créer alors la matrice $M(i, j)$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, telle que :

$$M(i, j) = \frac{(16 - (2(j - 1) + i))^{1/2}}{\sin(2(j - 1) + i)}$$

Exercice 10: Créer le vecteur $N = [\cos(0) \ \cos(\frac{\pi}{3}) \ \cos(\frac{2\pi}{3}) \ \cos(\pi)]$. On veut créer la matrice

$$P = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}.$$

On remarque qu'il s'agit d'une matrice de Toeplitz. Utiliser les commandes **lookfor** et **help** pour utiliser cette propriété pour créer la matrice P . Résoudre alors le système :

$$\begin{bmatrix} 9 & 4 & 1 & 0 \\ 4 & 9 & 4 & 1 \\ 1 & 4 & 9 & 4 \\ 0 & 1 & 4 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{\cos(0)} \\ \frac{1}{\cos(\frac{\pi}{3})} \\ \frac{1}{\cos(\frac{2\pi}{3})} \\ \frac{1}{\cos(\pi)} \end{bmatrix}$$

4.5 Fonctions spécifiques aux vecteurs et aux matrices

La commande `size` donne les dimensions d'une matrice ou d'un vecteur. La commande `length` donne la plus grande des dimensions d'une matrice ou la longueur d'un vecteur. La commande `numel` donne le nombre d'éléments d'une matrice ou d'un vecteur.

4.5.1 Pour les vecteurs

Certaines fonctions de MATLAB s'appliquent sur des vecteurs et renvoient un scalaire. Dans le cas où on les applique sur des matrices, elles renvoient un vecteur qui correspond généralement au résultat obtenu pour chaque colonne de la matrice. Ce sont :

`max`, `min`,
`sort`,
`sum`, `prod`, `median`, `mean`, `std`
`any`, `all`

Par exemple, l'élément maximal d'une matrice A est donné par `max(max(A))`, et non pas par `max(A)` qui renvoie un vecteur.

Exercice 11: Calculer les moyennes des 2 lignes de la matrice M construite précédemment, définie par $M(i, j)$, pour $1 \leq i \leq 2$, $1 \leq j \leq 8$, telle que :

$$M(i, j) = \frac{(16 - (2(j - 1) + i))^{1/2}}{\sin(2(j - 1) + i)}$$

Calculer les 8 moyennes des valeurs dans ses colonnes. Comment obtenir la moyenne sur tous les éléments de la matrice ? *On pourra utiliser la fonction `help`*

4.5.2 Pour les matrices

Les opérations matricielles les plus courantes sont :

<code>eig</code>	valeurs propres
<code>svd</code>	décomposition en valeurs singulières
<code>poly</code>	polynôme caractéristique
<code>det</code>	déterminant
<code>inv</code>	inverse
<code>expm</code>	matrice exponentielle
<code>sqrtn</code>	matrice racine carrée
<code>size</code>	taille d'une matrice

Exercice 12: À l'aide des commandes `ones` et `eye`, créer la matrice 10×10

$$A = \begin{bmatrix} 6 & 4 & \dots & 4 \\ 4 & 6 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 4 \\ 4 & \dots & 4 & 6 \end{bmatrix}.$$

Remplacer la 2^e ligne par la ligne `[2 3 2 2 ... 2]`.

Vérifier alors que les valeurs propres de A^{-1} sont les inverses des valeurs propres de A .

Exercice 13: On souhaite créer un vecteur $x = [x_1, x_2, \dots, x_{10}]$ tel que

$$x_i = \frac{1}{10} \sum_{j=1}^{10} \exp \left(\frac{1}{\log [10(j-1) + i] + 1} \right), \quad 1 \leq i \leq 10.$$

On commence par créer la matrice A de taille 10×10 telle que $A(i, j) = 10(j-1) + i$, c'est-à-dire

$$A = \begin{bmatrix} 1 & 11 & \dots & 91 \\ 2 & 12 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 99 \\ 10 & \dots & 90 & 100 \end{bmatrix}.$$

On pourra utiliser pour cela la commande **reshape**. On crée alors à partir de A la matrice B telle que $B(i, j) = \exp \left(\frac{1}{\log(10(j-1)+i)+1} \right)$, en utilisant les opérations élément-par-élément. On obtient alors le vecteur x en calculant la moyenne de B (commande **mean**).

5 Boucles, tests et relations

Dans MATLAB, ce type de fonctions opèrent quasiment de la même façon que pour les langages de type Pascal ou C.

5.1 Boucles

L'exécution répétitive d'une suite d'instructions peut-être réalisée à l'aide de **for** ou **while**. Remplacer une boucle par un traitement matriciel, lorsque cela est possible, conduit à un gain de temps d'exécution très important. Il est donc toujours préférable de chercher une formulation matricielle.

5.1.1 For

La syntaxe de la boucle **for** est la suivante :

```
for variable
    instructions
end
```

Par exemple, soit n donné :

```
x = [ ]; %crée une matrice vide
for i = 1 : n
    x = [x, i.^2];
end
```

Exercice 14: Bien souvent sur MatLab, une boucle peut être remplacée par une opération matricielle. Montrer que la boucle de l'exemple peut ainsi être avantageusement remplacée par une seule instruction.

Ce genre de modification est presque toujours avantageux en termes de coûts en opération et en mémoire d'un programme, et doit donc être recherché lors de n'importe quel traitement numérique de données.

5.1.2 While

La syntaxe de la boucle **while** est la suivante :

```
while expression
    traitement;
end
```

Le traitement va être répété tant que l'expression est vraie. Les groupes d'instructions qui vont être exécutées ou non selon la condition, peuvent être délimitées par les instruction **case** et **otherwise**. Consulter l'aide de la commande **while** pour avoir un exemple d'utilisation.

5.2 Test : instruction if

Cette instruction permet d'exécuter une suite d'instructions conditionnelles. Sa syntaxe est la suivante :

```
if condition
    instructions;
end
```

Le traitement va être exécuté uniquement si la condition est vraie. Des branchements multiples sont aussi possibles :

```
if condition1
    instructions1;
elseif condition2
    instructions;
else
    instructions;
end
```

5.3 Relations

Les principaux opérateurs relationnels sont les suivants :

<	plus petit
>	plus grand
<=	plus petit ou égal
>=	plus grand ou égal
==	égal (= correspond à une affectation)
~ =	différent

Ils peuvent être combinés avec des opérateurs logiques suivants :

&	et
	ou
~	non

Lorsque une relation est appliquée à des scalaires, le résultat est alors égal à 1 ou 0 selon qu'elle est vraie ou fausse. Sur des matrices ou des vecteurs de même taille, l'opération est effectuée élément par élément. Le résultat est une matrice (ou un vecteur) dont les éléments prendront les valeurs 1 ou 0 (suivant que l'expression logique est vraie ou fausse). Cette interprétation souple des Booléens est parfois utile, par exemple pour des calculs de fréquence

Une relation entre matrices est interprétée par **while** et **if** comme vraie si elle est vérifiée par tous les éléments de la matrice. Par conséquent, si l'on veut exécuter un traitement quand des matrices A et B sont égales, on peut faire :

```
if A == B
    traitement;
end
```

tandis que si on veut l'exécuter lorsqu'elles ne sont pas égales, on peut faire :

```
if any(any(A~=B))
    traitement;
end
```

ou tout simplement

```
if A == B else
    traitement;
end
```

Il faut noter que :

```
if A~=B
    traitement;
end
```

ne va pas donner le résultat escompté, car le traitement va être exécuté seulement si pour chaque indice de position, les éléments de A et de B en cette position sont différents.

Les fonctions **any** et **all** permettent de réduire les relations entre matrices à des vecteurs ou des scalaires. Consulter l'aide à leur sujet.

La fonction **find** est souvent très pratique : elle renvoie l'indice des éléments obéissant à une condition. Par exemple, soit y un vecteur, alors **find**($y > 2$) va renvoyer le rang de tous les éléments de y qui sont strictement supérieurs à 2.

Exercice 15: Générer le vecteur $y = [123...10]$ puis calculer le vecteur "logique" $b = (y < 5)$.

Exercice 16: Comparer la valeur booléenne de $A \sim B$ et de $\sim(A == B)$.

Remarque : Comme dans tous les langages, un test d'égalité entre matrices de nombres flottants n'a pas de sens. Dans ce cas, il vaut mieux faire un test comparant la différence à un seuil.

5.4 Pause et break

Lors de l'exécution d'un programme, on peut, à l'aide de l'instruction **pause**, suspendre son exécution et la relancer si l'utilisateur appuie sur une touche.

L'instruction **break** arrête l'exécution d'une boucle **while** ou **for**. La combinaison des touches "Ctrl" et "c" sur la fenêtre de commande MATLAB permet de stopper l'exécution d'un programme en cours.

6 Fichiers .m

MATLAB peut exécuter un ensemble d'instructions contenues dans un programme. Ces programmes ont une extension **.m**. Il y a deux types de programmes **.m** :

- les scripts
- les fonctions

Les corps des programmes de MATLAB peuvent être observés avec la commande `type`, avec la commande suivante :

```
type sum.m
```

6.1 Scripts

Un fichier script est composé d'un ensemble d'instructions MATLAB. Si le fichier a pour nom `prog.m`, alors la commande `prog` va l'exécuter. Les variables d'un programme script sont globales, et son exécution va changer leurs valeurs dans l'espace de travail.

Exercice 17: Pour cet exercice, l'on demande d'écrire un script sauvé sous le nom `lancers.m`.

On considère le problème suivant. On lance p dés à n faces numérotées de 1 à n . On cherche à vérifier par simulation la probabilité d'avoir :

1. exactement un 1 (probabilité théorique : $\frac{p}{n} \left(1 - \frac{1}{n}\right)^{p-1}$);
2. au moins un 1 (probabilité théorique : $1 - \left(1 - \frac{1}{n}\right)^p$);
3. au plus un 1 (probabilité théorique : $\left(1 - \frac{1}{n}\right)^{p-1} \left(1 + \frac{p-1}{n}\right)$);
4. exactement deux 1 (probabilité théorique : $\frac{p(p-1)}{2} \left(\frac{1}{n}\right)^2 \left(1 - \frac{1}{n}\right)^{p-2}$);

(a) À l'aide de la fonction `unidrnd`, générer une matrice `lancers` simulant 10000 lancers indépendants de $p = 5$ dés à $n = 10$ faces.

(b) À l'aide des fonctions `find`, `length`, `sum`, et/ou `mean`, estimer les différentes probabilités avec les fréquences empiriques, et donner le pourcentage d'erreur par rapport aux probabilités théoriques.

6.2 Fonctions

Les variables d'une fonction sont locales, mais peuvent aussi être déclarées comme globales.

Soit la fonction :

```
function y = moyenne(x)
    %MOYENNE valeur moyenne.
    y = sum(x)/length(x);
end
```

Cette fonction va calculer la moyenne des éléments d'un vecteur. Dans la ligne de commande, on peut alors écrire :

```
vect = [1 2 3 4 5 6];
moy = moyenne(vect)
```

qui va renvoyer la valeur moyenne des éléments du vecteur `vect`.

La première ligne du script d'une fonction doit obligatoirement être de la forme :

```
function [resultat1,resultat2,...] = nom_fonction(argument1,argument2,...)
```

Cette fonction doit être sauvegardée dans un fichier qui a pour nom `nom_fonction.m` (ce fichier doit être accessible depuis MATLAB : pour cela, se placer grâce à la commande `cd` dans le répertoire où la fonction a été sauvegardée).

6.3 Textes, entrées, messages d'erreurs

Les textes doivent être encadrés par des quotes :

```
s = 'texte'
```

Ils peuvent être affichés :

```
disp('affichage')
```

Les messages d'erreurs peuvent être affichés à l'aide de la commande **error** qui arrête aussitôt l'exécution du programme :

```
error('desole!')
```

On peut aussi les saisir lors de l'exécution :

```
iter = input('nombre d iterations?')
```

6.4 Vectorisation et pré-allocation

Dans le but d'accélérer l'exécution de programmes, il est important de vectoriser les algorithmes dans les fichiers **.m**. En effet, là où d'autres langages utilisent des boucles, MATLAB peut utiliser des opérations vectorielles ou matricielles. Considérons l'exemple suivant :

```
i = 0;
for t = 0 : 0.01 : 10,
    i = i + 1;
    y(i) = sin(t)
end;
```

Si on vectorise ce programme, il suffit d'écrire :

```
t = 0 : 0.01 : 10;
y = sin(t);
```

Dans le cas où il n'est pas possible de vectoriser, les boucles **for** seront plus rapides si les vecteurs ou les matrices dans lesquels les résultats seront stockés sont pré-alloués. Par exemple :

```
y = zeros(1, 100);
for i = 1 : 100
    y(i) = det(A.^i);
end
```

En effet, si on ne pré-alloue pas un vecteur, MATLAB va devoir changer la taille du vecteur *y* à chaque boucle d'itération, ce qui ralentit considérablement l'exécution du programme.

7 Gestion des fichiers

Lors de la gestion de programme, il est parfois nécessaire d'utiliser des outils de gestion. L'ouverture de l'éditeur de programmes se fait à l'aide de :

```
edit ou
edit nom_du_prog
```

Les commandes DOS classiques sont valables dans MATLAB :

pwd	indique le chemin du répertoire courant
cd	change de répertoire
dir	liste tous les éléments d'un répertoire
what	liste les fichiers .m , .mat et .mex d'un répertoire

D'autres commandes DOS peuvent être exécutées si elles sont précédées de "!" :

```
!del nom_fichier
```

La commande `path` permet d'obtenir ou d'initialiser un chemin. Pour plus d'information utiliser le `help`.

8 Graphiques

MATLAB permet de réaliser des graphiques 2-D et 3-D.

Pour avoir un aperçu des capacités de MATLAB en matière de graphismes, utiliser la commande `demo`.

Soit deux vecteurs `x` et `y` de même taille. La commande `plot(x,y)` va tracer `y` en fonction de `x`. Soit à tracer la fonction sinus de 0 à 4 :

```
x = 0 : 0.01 : 4;  
y = sin(x); plot(x,y);
```

Si l'on souhaite tracer un deuxième graphique sur une figure différente, taper :

```
figure(2)
```

tandis que si l'on souhaite tracer un deuxième graphique sur la même figure, taper :

```
hold on
```

L'effet de cette commande s'annule avec `hold off`.

L'écriture de texte sur les axes d'un graphe ou sur le graphe s'effectue à l'aide des commandes :

<code>title</code>	titre du graphe
<code>xlabel</code>	titre de l'axe des abscisses
<code>ylabel</code>	titre de l'axe des ordonnées
<code>gtext</code>	placement d'un texte avec la souris
<code>text</code>	positionnement d'un texte spécifié par des coordonnées

La commande `grid` place une grille sur un graphe.

Les commandes précédentes doivent être saisies après la commande `plot`.

Par défaut les courbes sont tracées par des lignes continues, mais il y a d'autres choix :

- -, :, - ., ., +, *, o et x

De même, on peut préciser les couleurs avec :

y, m, c, r, g, b, w et k

Par exemple, `plot(x,y,'r*')` va tracer en rouge et avec des * la courbe de `y` en fonction de `x`.

La commande `subplot` permet de partitionner un graphe en plusieurs graphes :

```
subplot(m,n,p)
```

divise la fenêtre de la figure en une matrice $m \times n$ de petits sous-graphes et sélectionne le p -ième sous-graphe pour la figure courante.

Les commandes `semilogx` et `semilogy` à la place de `plot` permettent d'avoir respectivement l'axe des abscisses ou des ordonnées en échelle logarithmique.

Exercice 18: Tracer la superposition de plusieurs sinus de phases respectives 0, $\Pi/6$ et $\Pi/3$ sur la même figure, en utilisant une légende pour les distinguer, en indiquant que la variable sur l'axe des abscisses est le temps, en faisant apparaître un titre. Faire deux figures sur le même modèle que la précédente, alignée sur une même fenêtre, pour deux valeurs de la fréquence. On utilisera pour cela la commande `subplot`.





































































Line Plots	Scatter and Bubble Charts	Data Distribution Plots	Discrete Data Plots	Geographic Plots	Polar Plots	Contour Plots	Vector Fields	Surface and Mesh Plots	Volume Visualization	Animation	Images
											
											
											
											
											
											
											
											

FIGURE 1 – Quelques types de graphique MathLab, extrait de [mathworks](https://www.mathworks.com)

9 Bonnes pratiques de code

Voici une liste de l'ensemble des bonnes pratiques à respecter lors de l'écriture d'une portion de code sur MatLab.

- *Je commence un nouveau projet ayant vocation à durer plus de 30 seconde ?*
J'enregistre mon fichier avant d'écrire la moindre ligne de code (section 2).
- *Je commence un nouveau projet ayant vocation à durer plus d'une heure ?*
Je commente mon code à l'aide du symbole %, le découpe au besoin en section à l'aide du symbole %% suivi d'un espace, je respecte l'indentation et place tous les symboles ; nécessaires. Lors du développement, je supprime les données inutiles avec `clear`, `clc` et `close all` (section 1).
- *Je cherche le nom d'une fonction ou son fonctionnement ?*
J'utilise les commandes d'aide `lookfor` ou `help` respectivement (section 3).
- *Je définis une nouvelle variable ?*
Je choisis un nom, pertinent sans être trop long, et qui n'écrase pas de fonction MatLab. Si possible, je la crée du bon type, et avec la bonne taille s'il s'agit d'une matrice.
- *Je cherche à faire une opération connue de manière répétée ?*
J'utilise une écriture matricielle (section 4).
- *Je cherche à faire une opération inconnue de manière répétée impossible à faire de la manière précédente (typiquement opération de récurrence) ?*
J'utilise une boucle, `for` si on connaît le nombre d'opérations, `while` si on ne le connaît pas (section 5).
- *J'ai besoin de répéter de nombreuses fois un même code ?*
Je l'enregistre dans une fonction, que je place dans le même répertoire (section 6).
- *Je ne trouve pas le chemin vers une fonction que j'ai écrite précédemment ?*
Je peux utiliser les outils de gestion de fichier (section 7).

- *J'ai besoin d'une représentation graphique ?*

Je réfléchis à en choisir une pertinente pour mon propos (section 8). Je n'oublie pas de lui donner un titre, une légende et de nommer les axes.

- *J'ai fini d'écrire mon code ?*

Je le teste avec des exemples simples, repère un maximum d'erreurs et les corrige. En particulier, je vérifie que mes variables sont de bonne taille et de bon type (section 1).