

## TP1 : INTRODUCTION À MATLAB

RÉSUMÉ. Matlab est un logiciel de calcul numérique, utilisé dans de nombreux domaines d'application. Il est basé sur le calcul matriciel. Matlab est d'ailleurs un raccourci pour "Matrix Laboratory". Le but de ce document est d'aider les débutants en Matlab, en introduisant les commandes les plus courantes.

### 1. L'ACCÈS

Pour lancer l'exécution de Matlab, tapez dans une fenêtre xterm : `matlab6.5`. L'interface graphique de Matlab apparaît alors. A droite, la partie où l'on entre les commandes. Le caractère `»` signifie que Matlab attend une instruction. A gauche, en haut, les variables d'environnement, en bas, les dernières commandes tapées. Regardez le contenu des menus déroulants.

**Exercice 1.** Exécutez l'introduction de Matlab, chargez l'aide. Bref, faites un petit tour du propriétaire selon le tableau ci-dessous...

<code>intro</code>	lance une introduction à Matlab
<code>help</code>	produit une liste de toutes les commandes par thèmes
<code>demo</code>	démonstration donnant une représentation des fonctionnalités de bases de Matlab
<code>info</code>	information sur la boîte à outils disponibles

### 2. L'AIDE DANS MATLAB

Mieux vaut apprendre à se repérer tout seul que de demander en permanence à son voisin comment faire. Ne serait-ce qu'au cas où il faudrait utiliser dans l'examen une fonction dont on ne se souvient que vaguement quelle est sa syntaxe...

<code>helpwin</code>	ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations
<code>help</code>	donne la liste de toutes les commandes par thèmes
<code>help nom</code>	décrit la fonction <code>nom.m</code>
<code>lookfor nom</code>	recherche une instruction à partir du mot clé <code>nom</code>

**Exercice 2.** Trouvez la fonction qui donne les valeurs propres d'une matrice. Tapez `help` de cette fonction.

## 3. COMMANDES GÉNÉRALES

3.1. **Gestion des fichiers.** Vous pouvez utiliser la petite fenêtre en haut à droite, ou à défaut :

<code>pwd</code>	affiche le nom du répertoire courant pour Matlab
<code>cd rep</code>	change le répertoire courant pour Matlab qui devient <i>rep</i>
<code>dir</code>	fournit le catalogue d'un répertoire
<code>delete</code>	efface des fichiers ou des objets graphiques

3.2. **Calculs élémentaires.** Dans la partie commandes de l'interface,

» `5+8`

Résultat : » 13

Pour conserver le résultat, il faut l'assigner dans un objet :

» `a=5+8`

» `a`

Pour ne pas faire afficher le résultat, mettez ; à la fin de la commande :

» `a=5+8;`

3.3. **Constantes prédéfinies.**

<code>pi</code>	3.1415...
<code>eps</code>	2.2204e-016
<code>Inf</code>	nombre infini
<code>NaN</code>	n'est pas un nombre; exprime parfois une indétermination

3.4. **Historique.** Matlab conserve l'historique des commandes. Il est donc possible de récupérer des instructions déjà saisies (et ensuite de les modifier dans le but de les réutiliser) :

<code>↑, ↓, →, ←</code>	permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes
-------------------------	--

3.5. **Variables d'environnement.** Matlab garde en mémoire les variables qui ont été créées. On les voit en haut, à gauche, lorsque Matlab dispose d'une interface graphique. Sinon, on peut les afficher et les effacer par la ligne de commande :

<code>who</code>	donne la liste des variables présentes dans l'espace de travail
<code>whos</code>	donne la liste des variables présentes dans l'espace de travail ainsi que leurs propriétés
<code>what</code>	donne la liste des fichiers <code>.m</code> et <code>.mat</code> présents dans le répertoire courant
<code>clear var<sub>1</sub>...var<sub>n</sub></code>	efface les variables <code>var<sub>1</sub>,...var<sub>n</sub></code> de l'espace de travail
<code>clear</code>	efface toutes les variables créées dans l'espace de travail

**Exercice 3.**

- (1) Tapez la commande `a=1:7;`. Tapez les commandes `a`, `who` et `whos`.
- (2) Utilisez `↑` pour modifier `a`: `a=1:2;`.
- (3) Tapez la commande `b=a+2;`. Réexécutez les commandes `who` et `whos` en utilisant `↑`. Tapez `clear` et `b`.

## 4. LES TYPES DE DONNÉES

Matlab traite un seul<sup>1</sup> type d'objet: *les matrices*! Les scalaires sont des matrices  $1 \times 1$ , les vecteurs lignes des matrices  $1 \times n$ , les vecteurs colonnes des matrices  $n \times 1$ .

**4.1. Construction explicite.** On peut former des vecteurs et des matrices en entrant leurs coefficients.

- **scalaires**

- » `s=30`

- **vecteurs numériques**

- » `x=[1;2;3]` (les `;` séparent les éléments d'un vecteur colonne)

- » `x=[1,2,3]` (les `,` ou les blancs séparent les éléments d'un vecteur ligne)

- » `x'`

- » `y = [x,x,x]`

- » `z = [x x x]`

- **matrices**

- » `M=[11 12 13 14;21 22 23 24; 31 32 33 34; 41 42 43 44]`

(où les `;` séparent les lignes d'une matrice)

Construction à partir de plusieurs vecteurs de même longueur :

- » `y=[11;22;33];`

- » `mat1=[x' y]`

- **vecteurs de chaîne de caractères**

La chaîne de caractères est un vecteur ligne. Pour le créer, on entre les caractères en commençant et en terminant par `'` (*quote*).

- » `ch='matlab'`

- **les nombres complexes**

Dans Matlab, un nombre complexe est de la forme:  $z = a + ib$ .

- » `c=2+i`

- **les polynômes**

Matlab représente un polynôme sous forme d'un vecteur ligne contenant les coefficients classés dans l'ordre des puissances décroissances. Par exemple le polynôme  $P$  d'expression  $P(x) = x^2 - 6x + 9$  est représenté par

---

1. ou presque... Voir l'appendice!

» P=[1 -6 9]

**Exercice 4.** Entrez les différents vecteurs et matrices et donnez la longueur et la taille de chacun (Utilisez `help` pour trouver les fonctions qui donnent longueur et taille).

**4.2. Création rapide.** Certaines commandes permettent de créer plus rapidement des vecteurs précis :

```

» l1=1:10 (Un vecteur contenant les entiers de 1 à 10)
» l2=1:1:10
» l3=10:-1:1
» l4=1:0.3:pi
» l1(2)=l3(3)
» l4(3:5)=[1,2,3]
» l4(3:5)=[]
» l5=linspace(1,5,5)
» help linspace
» who
» whos
» clear l1 l2 l3 l5
» who
» clc (efface le contenu de la fenêtre de commande)
» clear

```

**NB :** Une ligne de commande commençant par le caractère `%` n'est pas exécutée par Matlab. Cela permet d'insérer des lignes de commentaires. Et il faut commenter ses programmes, ... surtout ceux de l'examen !

**Exercice 5.** Construire :

- (1) une suite partant de  $-8$  et allant à  $-5$  par pas de  $0.25$ .
- (2) une suite décroissante d'entiers de  $15$  à  $3$ .
- (3) une suite de longueur  $100$  de  $-\pi$  à  $\pi$ .

**4.3. Opérations vectorielles.** Les tableaux suivants résument certaines commandes couramment utilisées.

#### VECTEURS

<code>n:m</code>	nombres de $n$ à $m$ par pas de $1$
<code>n:p:m</code>	nombres de $n$ à $m$ par pas de $p$
<code>linspace(n,m,p)</code>	$p$ nombres de $n$ à $m$
<code>length(x)</code>	longueur de $x$
<code>x(i)</code>	$i$ -ème coordonnée de $x$
<code>x(i1:i2)</code>	coordonnées $i1$ à $i2$ de $x$
<code>x(i1:i2)=[]</code>	supprimer les coordonnées $i1$ à $i2$ de $x$
<code>[x,y]</code>	concaténer les vecteurs $x$ et $y$
<code>x*y'</code>	produit scalaire des vecteurs lignes $x$ et $y$
<code>x'*y</code>	produit scalaire des vecteurs colonnes $x$ et $y$

## MATRICES

<code>size(A)</code>	nombre de lignes et de colonnes de A
<code>A(i,j)</code>	coefficient d'ordre $i, j$ de A
<code>A(i1:i2,:)</code>	lignes $i1$ à $i2$ de A
<code>A(i1:i2,:) = []</code>	supprimer les lignes $i1$ à $i2$ de A
<code>A(:,j1:j2)</code>	colonnes $j1$ à $j2$ de A
<code>A(:,j1:j2) = []</code>	supprimer les colonnes $j1$ à $j2$ de A
<code>A(:)</code>	concaténer les vecteurs colonnes de A
<code>diag(A)</code>	coefficients diagonaux de A

## MATRICES PARTICULIÈRES

<code>zeros(m,n)</code>	matrice nulle de taille $m, n$
<code>ones(m,n)</code>	matrice de taille $m, n$ dont tous les coefficients valent 1
<code>eye(n)</code>	matrice identité de taille $n$
<code>diag(x)</code>	matrice diagonale dont la diagonale est le vecteur $x$
<code>magic(n)</code>	carré magique de taille $n$
<code>rand(m,n)</code>	matrice de taille $m, n$ à coefficients i.i.d. de loi uniforme sur $[0,1]$
<code>randn(m,n)</code>	matrice de taille $m, n$ à coefficients i.i.d. de loi normale $\mathcal{N}(0,1)$

**Exercice 6.** [Extraction de composantes.] Entrez la matrice

» `A=[1 2 3; 2 3 1; 3 1 2]`

Quels sont les résultats des commandes suivantes?

» `A([2 3],[1 3])`

» `A([2 3],1:2)`

» `A([2 3],:)`

» `A([2 3],end)`

» `A(:)`

**Exercice 7.** Créez des matrices particulières. Exemple de création d'une matrice par blocs :

» `C=[A, zeros(3,2); zeros(2,3), eye(2)]`

Répliquez le vecteur colonne `[1; 3; 6]` pour en faire une matrice  $3 \times 19$ , de deux manières : en utilisant `ones` et en effectuant une multiplication matricielle, puis en trouvant la commande *ad hoc* de répllication.

**Exercice 8.** Ecrire la matrice carrée  $M$  d'ordre 12 contenant les entiers de 1 à 144 rangés par ligne. Extraire de cette matrice les matrices suivantes :

- la sous-matrice formée par les coefficients  $a_{ij}$  pour  $i = 1, \dots, 6$  et  $j = 7, \dots, 12$ ;
- celles des coefficients  $a_{ij}$  pour  $(i, j) \in \{1, 2, 5, 6, 9, 10\}^2$ ;
- celle des coefficients  $a_{ij}$  pour  $i + j$  pair.

## 5. LES OPÉRATIONS MATRICIELLES ET LES FONCTIONS

## 5.1. Les opérations matricielles.

<code>A'</code>	transposée de A
<code>rank(A)</code>	rang de A
<code>inv(A)</code>	inverse de A
<code>expm(A)</code>	exponentielle de A
<code>det(A)</code>	déterminant de A
<code>trace(A)</code>	trace de A
<code>poly(A)</code>	polynôme caractéristique de A
<code>eig(A)</code>	valeurs propres de A
<code>[U,D]=eig(A)</code>	vecteurs propres et valeurs propres de A
<code>+</code> <code>-</code>	addition, soustraction
<code>*</code> <code>^</code>	multiplication, puissance (matricielles)
<code>.*</code> <code>.^</code>	multiplication, puissance terme à terme
<code>A\b</code>	solution de $Ax = b$
<code>b/A</code>	solution de $xA = b$
<code>./</code>	division terme à terme

**Exercice 9.** Essayez des fonctions sur la matrice A. Par exemple, quels sont ses valeurs et vecteurs propres? Puis, construisez une matrice C de même taille que A. Essayez  $A+C$ ,  $A*C$ ,  $A.*C$ . Ensuite, définissez la matrice B comme étant la matrice A à laquelle on a ajouté le vecteur colonne  $[1; 2; 3]$ . Déterminez son noyau. Y a-t-il une fonction prédéfinie dans Matlab qui détermine le noyau d'une matrice? Le cas échéant, y a-t-il des différences entre les méthodes employées pour le calcul du noyau?

**Exercice 10.** Résolution d'un système sous-dimensionné :

$$\begin{cases} 2x_1 + x_2 - 3x_3 = 1 \\ x_1 - 2x_2 + x_3 = 2 \end{cases}$$

- (1) Ecrivez le système sous la forme matricielle  $Ax = b$  (où vous définissez A et b) et calculez le rang de la matrice A.
- (2) Définissez la matrice B comme étant la matrice A à laquelle on a ajouté le vecteur colonne b.
- (3) Calculez le rang de la matrice B. Conclusion?
- (4) Définissez le vecteur  $c = [1; 1; 1]$  et déterminez l'image du vecteur c par la matrice A.
- (5) Résolvez l'équation  $Ax = b$ .

**NB:**  $A\b$  est équivalent à  $inv(A)*b$  si A est inversible.

## 5.2. Les fonctions.

## FONCTIONS ÉLÉMENTAIRES

sqrt	exp	log
sin	cos	tan
asin	acos	atan
round	floor	ceil
abs	angle	conj

Certaines fonctions de Matlab s'appliquent à l'ensemble d'un vecteur. Lorsqu'on les applique à des matrices, elles opèrent colonne par colonne.

**Exercice 11.** Construire un vecteur quelconque et essayer les fonctions ci-dessus.

Le tableau suivant décrit le résultat de quelques unes de ces fonctions lorsqu'elles sont appliquées à un vecteur  $x$  :

## FONCTIONS VECTORIELLES

max(x)	maximum
min(x)	minimum
sort(x)	tri par ordre croissant
[y, I] = sort(x)	retourne en plus les indices des éléments de x
find(x)	retourne les indices non nuls de x
[y, I] = find(x)	retourne des lignes (dans le vecteur I) et des colonnes (dans le vecteur J) des éléments non nuls du x
sum(x)	somme des éléments de x
cumsum(x)	vecteur contenant la somme cumulée des éléments de x
prod(x)	produit des éléments de x
cumprod(x)	vecteur contenant le produit cumulé des éléments de x
diff(x)	vecteur des différences entre deux éléments consécutifs de x
mean(x)	moyenne des éléments de x
median(x)	médiane
std(x)	écart type

**Exemple 1.** Regardez l'effet des instructions suivantes.

```

» x=rand(1,5)
» mean(x)
» std(x)
» median(x)
» sort(x)

```

```

» A=rand(3)
» sort(A)
» [B, I]=sort(A)
» sort(A')
» max(A)
» max(A')
» max(max(A))
» sum(A)
» cumsum(A)
» prod(A)
» diff(A)

» D=A([1,2],1:3)
» sum(D,1)
» sum(D,2)

```

**Exercice 12.** Soit  $X$  une matrice  $2 \times n$  contenant les coordonnées de  $n$  points du plan. Comment faire pour obtenir une matrice où les points sont ordonnés par ordre croissant des abscisses?

**Exercice 13.** Calculer  $10!$ .

**Exercice 14.**

- (1) Soit le vecteur de dimension 8 de composantes : 3.2, 4.8, 3.3, 3.2, 3.1, 4.2, 3.2, 3.3. Entrez le vecteur  $\mathbf{y} = (y_i)_{i=1,\dots,8}$  correspondant.
- (2) Construisez à l'aide des fonctions précédentes la suite des moyennes mobiles,

$$\bar{y}_n = \frac{1}{n} \sum_{i=1}^n y_i .$$

Extrayez  $\bar{y}_8$ . Donnez une fonction qui calcule directement  $\bar{y}_8$  à partir de  $\mathbf{y}$ .

**Exercice 15.** Tirez 20 nombres aléatoirement dans l'intervalle  $[0,1]$ . Quelle est la valeur minimale du vecteur et la position du coefficient qui la réalise? Vérifiez.

## 6. OPÉRATEURS RELATIONNELS ET LOGIQUES

Ils permettent de relier logiquement deux *matrices*.

Opérateurs relationnels	<, <=, >=, == (égalité), ~= (différent)
Opérateurs logiques	& (et),   (ou), ~ ou not (non)

**NB:** Attention de ne pas confondre = qui sert à affecter une valeur à une variable et == qui sert à tester l'égalité.



Les opérateurs relationnels peuvent être utilisés avec des scalaires ou des matrices. Le résultat d'évaluation d'une expression relationnelle est 1 (vrai) ou 0 (faux). Appliqués à une matrices, ils rendent une matrice de même dimension, formée de 1 et de 0.

**Exemple 2.**

```
u=4
u==4
u<=12
```

Reprenez la matrice  $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$  en tapant  $A = [ \uparrow ]$ .  
 $A_r = (A <= 2)$

$[B, I] = \text{find}(A == 1)$  (Retourne les coordonnées égales à 1, ainsi que leur position.)

**Exercice 16.** Reprenons le vecteur  $y$ . Pour rappel, tapez  $y$ .

- (1) Faites répondre Matlab à la question suivante : existe-t-il une coordonnée du vecteur  $y$  inférieure à 3.3?
- (2) Construisez un vecteur logique  $z$  tel que la  $i$ -ème coordonnée de ce vecteur sera 1 si la  $i$ -ème coordonnée du vecteur  $y$  est à l'extérieur de l'intervalle  $[\bar{y}_8 - \sigma_8, \bar{y}_8 + \sigma_8]$  où  $\sigma_8$  est la racine carrée de la variance d'échantillonnage.

**Exercice 17.**

- (1) Tirez 100 nombres aléatoirement (et uniformément) dans l'intervalle  $[0,1]$  et groupez-les dans un vecteur  $\mathbf{x} = (x_i)_{i=1, \dots, 100}$ .
- (2) Prenez  $y_i = 2 * x_i$  pour tout  $i = 1 \dots, 100$ .
- (3) Prenez la partie entière de ces nombres (à l'aide de la fonction `floor`):  $z_i = [y_i]$ . Ceci définit un vecteur  $\mathbf{z}$ . (Au passage, notez qu'il existe plusieurs fonctions parties entières, avec des comportements différents, `ceil` par exemple...)
- (4) Donnez la fréquence de 1 sur l'échantillon  $\mathbf{z}$ . Pouvaient-on s'attendre à ce résultat?

Réinitialisez l'espace de travail en tapant `clear` (effacement des variables d'environnement) puis `clc`.

## 7. REPRÉSENTATION GRAPHIQUE DES RÉSULTATS.

**7.1. Représentations de points dans le plan.** Il existe plusieurs possibilités pour représenter un ensemble de points  $(x(i), y(i))$ . Les plus utilisées sont énumérées ci-dessous.

<code>plot(x,y,'s')</code>	tracé d'une courbe ou d'un nuage de points
<code>bar(x,y,'s')</code>	tracé sous forme d'un histogramme
<code>stem(x,y,'s')</code>	diagramme en bâtons
<code>stairs(x,y)</code>	tracé en escalier des valeurs discrètes
<code>fplot</code>	représente des fonctions
<code>hist</code>	trace des histogrammes

's' est un paramètre facultatif constitué d'une chaîne de caractères qui spécifie le type de tracé (couleur, différents tracés en pointillés, symboles pour le tracé de points). Par défaut, le tracé est continu. Tapez `help plot` pour avoir la liste des valeurs possibles pour 's'.

## 7.2. Gestion de la fenêtre graphique.

<code>hold on</code>	les prochains tracés se superposeront aux tracés déjà effectués
<code>hold off</code>	le contenu de la fenêtre graphique active sera effacé lors du prochain tracé
<code>clf</code>	efface le contenu de la fenêtre graphique active
<code>figure(n)</code>	affiche ou rend active la fenêtre graphique numéro <code>n</code>
<code>close</code>	ferme la fenêtre graphique active
<code>close all</code>	ferme toutes les fenêtres graphiques
<code>subplot(n,m,p)</code>	partage la fenêtre graphique active en <code>m × n</code> espaces graphiques et sélectionne le <code>p</code> -ième.

## 7.3. Axes et légendes.

<code>axis([xmin xmax ymin ymax])</code>	pour définir les échelles des axes
<code>grid</code>	quadrillage du graphique
<code>grid off</code>	
<code>title('titre')</code>	titre pour le graphique
<code>xlabel('titre')</code>	légende pour l'axe des abscisses
<code>ylabel('titre')</code>	légende pour l'axe des ordonnées
<code>legend('titre1','titre2',...)</code>	légende pour chaque courbe du graphique
<code>text(x,y,'texte')</code>	texte explicatif à la position $(x,y)$
<code>gtext('texte')</code>	texte positionné à l'aide de la souris

**NB:** Si `a` est une variable contenant le nombre  $\sqrt{2}$ , la commande `text(x,y,['le resultat est ' num2str(a,3)])` affichera le texte "le resultat est 1.41" à partir de la position `x,y` de la fenêtre graphique.

### Exemple 3.

```
x= -pi:0.1:3*pi; y=x.*sin(x);
plot(x,y)
clf
plot(x,y) axis([-pi,3*pi,-6,9])
xlabel('x') ylabel('y')
title('graphe de la fonction x sin(x) sur l''intervalle [' num2str(x(1)) ' , '
num2str(x(end)) ']'')
plot(x,y,x,2*y)
plot(x,[y;2*y])
plot(x,y,'r--',x,2*y,'g+')

fplot('[x*sin(x),2*x*sin(x)]',[-pi,3*pi])
```

```
fplot('x*sin(x)',[-pi,3*pi],'b-') hold on
fplot('2*x*sin(x)',[-pi,3*pi],'yo') hold off

t=0:0.1:2*pi;
plot(sin(t),sin(2*t))
plot(sin(t),sin(2*t),'c-')
```

**Exercice 18.** Soit  $f$  et  $g$  les fonctions définies sur l'intervalle  $[0,10]$  par :

$$f(x) = \exp\left(-\frac{x}{25}\right), \quad g(x) = \cos\left(\frac{x}{10}\right).$$

Tracez ces deux fonctions (à l'aide de `fplot`) d'abord dans une même fenêtre graphique mais sur des graphes différents, puis dans une même fenêtre graphique et sur le même graphe.

**7.4. La sauvegarde d'une figure.** Une figure peut être sauvegardée sous plusieurs formats :

- sous un format propre à matlab avec l'extension `.fig` (pour les versions récentes de Matlab). Pour cela, cliquer sur la commande `Save as` du menu `File` de la fenêtre graphique et entrer un nom de fichier avec l'extension `.fig` dans l'encadré qui apparaît. Un tel fichier peut être visualisé en utilisant la commande `Open` du menu `File`.
- sous un format PostScript en utilisant la commande `Export` du menu `File` d'une fenêtre graphique, ou, plus manuellement, en tapant `print -dps nomfichier`. Dans ce cas, un fichier `nomfichier.ps` est créé dans le répertoire courant. On peut visualiser les fichiers PostScript en utilisant le logiciel GhostView (`gv`), l'impression s'en faisant par la commande linux `lpr`.

**Exercice 19.**

- (1) Tirer 100 couples de points  $(x,y)$  aléatoirement dans le carré  $[0,1] \times [0,1]$ .
- (2) Représenter le nuage de points obtenus dans une fenêtre graphique.
- (3) Calculer le centre de gravité  $G$  du nuage de points.
- (4) Ajouter en rouge au nuage de points le centre de gravité.
- (5) Sauvegarder la figure sous le nom `nuage.fig`. Fermer la fenêtre graphique. Ouvrir une nouvelle fenêtre graphique et faire réapparaître le fichier `nuage.fig`.

**7.5. Les histogrammes.** La représentation d'un histogramme peut se faire à l'aide des fonctions `hist` ou `histo`. Nous décrivons la fonction `hist` :

- `hist(y,C)` dessine l'historgramme pour les valeurs du vecteur  $y$  réparties en  $C$  classes si  $C$  est un scalaire, ou sur les classes centrées sur les valeurs du vecteur  $C$  si  $C$  est un vecteur. Par défaut,  $C$  vaut 10.
- `[N,C]=hist(y,C)` ne dessine pas d'historgramme mais retourne les vecteurs  $N$  et  $C$  où  $C$  est le vecteur des milieux des classes et  $N$  est le nombre de valeurs de  $y$  appartenant à chaque classe. L'historgramme s'obtient par `bar(C,N)`.

**Exercice 20.** Reprendre l'exercice précédent et faire afficher dans une même fenêtre graphique deux histogrammes, un pour les abscisses et l'autre pour les ordonnées des points tirés.

## 8. UTILISATION DE FICHIERS.

**8.1. Les fichiers de sauvegarde.** La commande `diary nomfichier` crée un fichier (journal de bord) intitulé `nomfichier` qui garde la trace de toutes les commandes que vous avez tapées dans la fenêtre de commandes ainsi que les réponses de l'ordinateur. `diary off` permet d'arrêter l'écriture du journal de bord, tandis que `diary on` permet de la reprendre.

<code>save nomfichier var<sub>1</sub> ... var<sub>n</sub></code>	sauve la valeur des variables <code>var<sub>1</sub> ... var<sub>n</sub></code> dans un fichier binaire <code>nomfichier.mat</code>
<code>save nomfichier</code>	sauve l'ensemble des variables existant dans l'espace de travail de Matlab dans un fichier binaire <code>nomfichier.mat</code>
<code>save -ASCII nomfichier var<sub>1</sub> ... var<sub>n</sub></code>	sauve la valeur des variables <code>var<sub>1</sub> ... var<sub>n</sub></code> dans un fichier texte <code>nomfichier</code>
<code>load nom-f</code>	permet de récupérer toutes les données sauvegardées dans le fichier <code>nomfichier.mat</code>

**Exercice 21.** Sauvegardez `x`, `y` et le centre de gravité `G` dans un fichier binaire `sauv.mat`. Ouvrez ce fichier sous Linux (avec Emacs par exemple). Dans Matlab, avec la commande `clear`, effacez `x`, `y` et `G`. Puis, chargez le fichier `sauve.mat` afin de les récupérer. Vérifiez la récupération.

**8.2. Les programmes *script* (ou fichiers d'instructions).** Ce sont des fichiers texte avec une extension `.m`. Ils contiennent des suites d'instructions Matlab qui sont exécutées les unes après les autres.

**Exemple 4.** Sauvez dans le répertoire courant les lignes suivantes sous le nom `losange.m`:

```
x=[0 -1 0 1; -1 0 1 0 ]
y=[-1 0 1 0; 0 1 0 -1]
plot(x,y)
```

La commande `losange` affichera `x`, puis `y`, puis tracera un losange. Si ce fichier est placé dans un répertoire accessible<sup>2</sup>, la commande `losange` devient une commande Matlab comme toutes les autres.

**8.3. Les fichiers de fonctions.** Comme les fichiers d'instruction, ce sont des fichiers texte avec une extension `.m`. Leur syntaxe est particulière. Ils contiennent la définition d'une fonction et portent le nom de cette fonction.

**Exemple 5.**

```
function d=densnorm(x)
% densnorm: densite de la loi N(0,1)
% densnorm(x) retourne (1/sqrt(2*pi))*exp(-x^2/2)
d=(1/sqrt(2*pi))*exp((-x.^2)/2);
```

---

<sup>2</sup> voir ci-dessous le paragraphe "Gestion du path"

Si ce fichier est placé dans un répertoire accessible, la fonction `densnorm` devient une fonction Matlab comme toutes les autres. Le texte placé en commentaire est le contenu de l'aide pour la nouvelle fonction.

**Exercice 22.** Utilisez la fonction ci-dessus pour tracer (avec `plot`) la densité de la loi normale centrée réduite entre -5 et 5 en faisant calculer 100 points. Aurait-on pu utiliser `fplot`? Calculez  $\mathbb{P}(-5 \leq X \leq 1.96)$  où  $X$  est une variable aléatoire de loi  $N(0,1)$ .

#### 8.4. Dialogue avec l'utilisateur.

<code>disp(var)</code>	affiche le contenu de <i>var</i>
<code>rep=input('texte')</code>	affiche la chaîne de caractère <code>texte</code> et donne la main à l'utilisateur pour qu'il entre la valeur de la variable <code>rep</code>
<code>pause(n)</code>	arrête l'exécution d'un programme pendant <code>n</code> secondes ; par défaut (sans <code>n</code> ), reprend l'exécution du programme lorsque l'utilisateur appuie sur une touche du clavier
<code>keyboard</code>	arrête l'exécution d'un programme et donne la main à l'utilisateur jusqu'à ce qu'il tape presse la touche Entrée

**Exemple 6.** Essayez les commandes suivantes :

```
n=input('entrez le nombre de simulations')
disp(['Vous avez tapé: ' num2str(n)])
```

**Exercice 23.** Ecrire une fonction `simule.m` qui prend `n` en argument et :

- (1) construit un vecteur de  $n = 100$  nombres aléatoires  $(y_i)_{i=1,\dots,n}$  issus d'une loi normale de moyenne 2 et de variance 1,
- (2) renvoie la moyenne et la variance de l'échantillon observé,
- (3) sépare la fenêtre graphique en deux espaces graphiques et superpose respectivement sur chacun d'eux :
  - l'histogramme de  $(y_i)_{i=1,\dots,n}$  et la densité de la loi normale de moyenne 2 et de variance 1, donne un titre à la figure et affiche la moyenne et la variance,
  - la fonction de répartition empirique (utiliser la fonction `sort`) et la fonction de répartition théorique (utiliser la fonction `pnorm`), et donne un titre à la figure.

Faites tourner le programme pour d'autres valeurs de  $n$ , plus grandes ou plus petites. Indication : la première ligne du fichier de fonction est `function [m,v] = simule(n)`.

## 9. LES COMMANDES STRUCTURÉES

9.1. **L'instruction for.** La syntaxe en est la suivante :

```
for  variable = vecteur
    instructions
end
```

Les colonnes du '*vecteur*' sont affectées, l'une après l'autre à la variable '*variable*' et pour chacune de ces valeurs, les '*instructions*' sont exécutées. Remarquez qu'il faut privilégier les opérations vectorielles à l'utilisation de boucles.

**Exemple 7.** Ces quelques lignes calculent  $n!$  pour  $n = 1$  à 100. `fact = zeros(1,100);`  
`n=100;`  
`fact(1)=1;`  
`for k=2:n`  
`fact(k)=fact(k-1)*k;`  
`end`

Mais on peut aussi taper simplement : `n=100; fact=cumprod(1:n);`

L'exécution du premier programme prend significativement plus de temps que l'exécution du deuxième programme. Pour le voir, et comparer l'efficacité des algorithmes, on peut utiliser `tic` (à placer en début de programme) et `toc` (à placer en fin de programme) qui permettent de compter le temps CPU écoulé.

**Exercice 24.** Exécutez les deux programmes ci-dessus en utilisant les commandes `tic`, `toc` afin de comparer leur temps d'exécution respectif.

9.2. **Les intructions conditionnelles.** La manière la plus brute de procéder est d'utiliser un bloc `if ... then ... else ... end`. La syntaxe en est la suivante :

```
if conditions
    instructions
end
```

Les '*instructions*' ne sont exécutées que si les '*conditions*' sont vérifiées, plus précisément si '*conditions*' a une valeur différente de 0. Une variante plus élaborée est :

```
if conditions
    instructions          (exécutées si les 'conditions' sont vérifiées)
else
    instructions        (exécutées si les 'conditions' ne sont pas vérifiées)
end
```

Ou encore (avec des emboîtements) :

```

if conditions1
    instructions      (exécutées si les 'conditions1' sont vérifiées)
elseif conditions2
    instructions      (exécutées si les 'conditions1' ne sont pas vérifiées
                      mais si les 'conditions2' le sont)
else
    instructions      (exécutées si ni les 'conditions1' ni les 'conditions2' ne sont vérifiées)
end

```

**Exemple 8.** Le programme suivant simule le lancer d'une pièce.

```

p=0.5;
u=rand;
if u<p
    disp('pile')
else
    disp('face')
end

```

**NB:** Rappelons qu'une proposition logique (par exemple,  $(x>0 \ \& \ x<10)$ ) a pour valeur 1 si elle est vraie et 0 sinon. Dans les calculs, on peut donc éviter d'utiliser l'instruction `if` en introduisant des indicatrices d'ensembles.

**Exercice 25.** En tenant compte de cette remarque, simuler la réalisation d'une variable aléatoire de loi de Bernoulli de paramètre  $p$ . Puis, après cet échauffement, écrire un programme *script* `EstimeParam.m` qui :

- (1) simule un 1000-échantillon de variables aléatoires de Bernoulli de paramètre 0.2,
- (2) affiche le nombre de succès contenus dans ce 1000-échantillon,
- (3) et trace la courbe des moyennes mobiles en fonction du rang.

Exécuter le programme. Que remarque-t-on ? Quel est le théorème qui explique le phénomène ?

**Exercice 26.** Nous allons censurer des variables. Soit  $u$  une réalisation d'un 15-échantillon  $X_1, \dots, X_{15}$  de loi uniforme sur  $[-5, 1]$ . Mettez les observations plus petites que  $-2$  à  $-10$ . Ceci forme  $v$ , réalisation d'un 15-échantillon  $Y_1, \dots, Y_{15}$ . Quelle est la loi commune aux  $Y_i$  ?

**9.3. L'instruction while.** Ce format de boucle permet de s'arrêter conditionnellement (et non plus à rang fixé, comme dans une boucle `for`). La syntaxe en est la suivante :

```

while conditions
    instructions
end

```

Les '*instructions*' sont exécutées tant que les '*conditions*' sont vérifiées.

**Exemple 9.** Soit  $f$  une fonction continue. La fonction suivante recherche par dichotomie une valeur approchée à  $\epsilon$  près d'une solution de l'équation  $f(x) = 0$  dans l'intervalle  $[a, b]$ , lorsque

$f(a)$  et  $f(b)$  sont de signes différents (ce qui forme une condition suffisante d'existence de ce point d'annulation  $x$ ).

```
function s=dicho(f,a,b,e)
% s=DICHO(f,a,b,e)
% f : chaine de caracteres contenant le
%      nom d'une fonction (definie a
%      l'aide d'un fichier .m de type
%      fonction)
% a,b : bornes de l'intervalle de recher
%      che( a<b )
% e : precision avec laquelle on veut la
%      valeur approchee de la solution
%      de f(x)=0
d=b-a;
ya=feval(f,a);
yb=feval(f,b);
if ya==0
    s=a;
elseif yb==0
    s=b;
elseif ya*yb<0
    while (d>e)
        x=(b+a)/2;
        y=feval(f,x);
        if y==0
            s=y;
            return
        elseif y*ya<0
            b=x;
            yb=y;
        else
            a=x;
            ya=y;
        end
        d=b-a;
    end
    s=x;
else error('la fonction ne change pas de signe')
end
```

Pour mémoire, on donne les instructions de rupture de séquence. Il vaut cependant mieux programmer suffisamment proprement pour s'en dispenser.

**Exercice 27.** Ecrire une fonction  $y = \text{Puiss}(n,M)$  qui étant donné un entier  $n$  et une valeur maximale  $M > n$ , calcule les puissances entières de ce nombre, les agrège dans le vecteur  $y$  et s'arrête lorsque le résultat dépasse la valeur maximale



<code>break</code>	termine l'exécution de la boucle la plus proche
<code>return</code>	Termine l'exécution d'un fichier <i>script</i> sans aller jusqu'à sa fin
<code>error('message')</code>	affiche la chaîne de caractères <i>message</i> et interrompt l'exécution du programme en cours

## RÉFÉRENCES

- [1] M. Mokhtari et A. Mesbah *Apprendre et maîtriser Matlab*. Springer.
- [2] S. Lemaire. Documents pour la préparation à l'agrégation d'Orsay.
- [3] B. Laurent. Documents pour les TP de Matlab de la Maîtrise de mathématiques ingénierie

## QUESTIONS ET PROBLÈMES SUPPLÉMENTAIRES POUR UTILISATEURS AVERTIS

Renseignez-vous, *via* l'aide intégrée à Matlab, sur les sujets suivants :

- les types de données et la conversion d'un type à l'autre,
- la gestion des chemins d'accès (i.e. le *path*).

**Exercice 28.** Essayez de mettre les mots 'chat', 'chien', 'oiseau' et '45' dans un même objet Matlab. Remarquez que Matlab refuse que ledit objet soit de type vecteur : la commande

```
» ['chat'; 'chien'; 'oiseau'; '45']
```

provoque une erreur. Pourquoi? Quel type faut-il choisir alors? Ajoutez à l'objet ainsi formé un cinquième élément : la valeur de  $\pi$ , avec 9 décimales. Récupérez ensuite la valeur numérique du quatrième élément dans un scalaire `x`. Enfin, écrivez une procédure qui mette bout à bout les premières lettres de chacun des trois mots, puis les deuxièmes, *etc.*

**Exercice 29.** Créez un répertoire de travail TP2 et ajoutez-le au *path*. Fermez votre session et rouvrez-la. Démarrez Matlab. TP2 est-il toujours dans la liste des chemins? Si non, créez un fichier `pathdef.m` (Menu `File` puis `Set path`) dans votre répertoire de démarrage. A chaque lancement de Matlab, il vous suffira de taper la commande `pathdef` pour que de multiples chemins deviennent accessibles. Ceci permet de classer correctement ses documents dans une arborescence claire.