Institut supérieur d'informatique Année Universitaire 2016-2017

MODULE : Analyse d'information multimédia

TP 1 : Initiation à Matlab (1)

1. Calculs élémentaires

Commençons par les opérateurs les plus courants : +, -, *, /, ^ (puissance).

Tapez une expression mathématique quelconque et appuyez sur «Entrée». Par exemple :

*>> (3*2)/(5+3)*
ans =
0.7500
Le résultat est mis automatiquement dans une variable appelée ans (answer). Celle-ci peut être utilisée pour le calcul suivant, par exemple :

*>> ans*2*
ans =
1.5000

2. Affichage des nombres

Matlab offre au moins quatre types d'affichage (court ou long, scientifique ou non) que l'on sélectionne avec les commandes
– format short (qui est le format par défaut)
– format short e
– format long
– format long e
Pour le nombre x = 1/700, ces formats donnent respectivement les affichages
>> 0.0014
>>1.4286e-03
>>0.00142857142857
>>1.428571428571429e-03

3. Variables et fonctions prédéfinies

a. Variables

Gros avantage sur les langages classiques : on ne déclare pas les variables. Leur type (entier, réel, complexe) s'affectera automatiquement en fonction du calcul effectué.

Pour affecter une variable, on dit simplement à quoi elle est égale. Exemple :

>> a=1.2
a =
1.2000

On peut maintenant inclure cette variable dans de nouvelles expressions mathématiques, pour en définir une nouvelle :

*>> b = 5*a^2+a*
b =
8.4000
et ensuite utiliser ces deux variables :

>> c = a^2 + b^3/2
c =
297.7920
On a maintenant trois variables a, b et c. Ces variables ne sont pas affichées en permanence à l'écran. Mais pour voir le contenu d'une variable, rien de plus simple, on tape son nom :
>> b
b =
8.4000

Retenons

– MATLAB fait la différence entre les minuscules et les majuscules.

– Les noms de variables peuvent avoir une longueur quelconque.

– Les noms de variables doivent commencer par une lettre.

b. Effacement et liste des variables

La commande ***clear*** permet d'effacer une partie ou toutes les variables définies jusqu'à présent. Il est conseillé de placer cette commande au début de vos fichiers de commandes, en particulier lorsque vous manipulez des tableaux.

Syntaxe : ***clear*** var1 var2 var3 . . .

Si aucune variable n'est spécifiée, toutes les variables seront effacées.

La commande ***who*** affiche les noms de toutes les variables en cours.

c. Variables prédéfinies

Il existe un certain nombre de variables pré-existantes. Nous avons déjà vu ***ans*** qui contient le dernier résultat de calcul, ainsi que i et j qui représentent p−1.

Il existe aussi pi, qui représente π, et quelques autres.

ATTENTION : ces variables ne sont pas protégées, donc si vous les affectez, elles ne gardent pas leur valeur. C'est souvent le problème pour i et j que l'on utilise souvent spontanément comme indices de boucles, de telle sorte qu'on ne peut plus ensuite définir de complexe ! !

d. Fonctions prédéfinies

Toutes les fonctions courantes et moins courantes existent. On retiendra que pour appliquer une fonction à une valeur, il faut mettre cette dernière entre parenthèses. Exemple :

>> sin (pi/12)
ans =
0.16589613269342

Voici une liste non exhautive :

– fonctions trigonométriques et inverses : sin, cos, tan, asin, acos, atan

– fonctions hyperboliques (on rajoute «h») : sinh, cosh, tanh, asinh, acosh, atanh

– racine, logarithmes et exponentielles : sqrt, log, log10, exp….

3. Matrices et tableaux

MATLAB ne fait pas de différence entre les deux. Le concept de tableau est important car il est à la base du graphique : typiquement pour une courbe de n points, on définira un tableau de n abscisses

et un tableau de n ordonnées. Mais on peut aussi définir des tableaux rectangulaires à deux indices pour définir des matrices au sens mathématique du terme, puis effectuer des opérations sur ces matrices.

3.1. Définition d'un tableau

On utilise les crochets [et] pour définir le début et la fin de la matrice. Ainsi pour définir une variable M contenant la matrice :

$$\begin{bmatrix} 1 & 2 & 3 \\ 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$$

On écrira:

```
>> M = [1 2 3
11 12 13
21 22 23]
M =
 1   2   3
11  12  13
21  22  23
```

On peut également utiliser le symbole, qui sert de séparateur de colonne et ; de séparateur de ligne. Ainsi pour définir la matrice précédente on aurait pu taper :

```
>> M = [1,2,3;11,12,13;21,22,23]
```

ou bien, en remplaçant la virgule par des blancs :

```
>> M = [1 2 3;11 12 13;21 22 23]
```

On peut aussi définir des vecteurs, ligne ou colonne, à partir de cette syntaxe. Par exemple pour définir un vecteur ligne, on peut écrire:

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

alors que :

```
>> V = [11
12
13]
V =
11
12
13
```

définit un vecteur colonne. On aurait pu aussi définir ce dernier par :

```
>> V= [11;12;13]
```

3.2. Accès à un élément d'un tableau

Il suffit d'entrer le nom du tableau suivi entre parenthèses d'un ou des indices dont on veut lire ou écrire la valeur. Exemple si je veux la valeur de M32 :

```
>> M(3,2)
ans =
```

22
Pour modifier seulement un élément d'un tableau, on utilise le même principe. Par exemple, je veux que M3 2 soit égal à 32 au lieu de 22 :

>> M(3,2) = 32
M =
1 2 3
11 12 13
21 32 23

Vous remarquerez que MATLAB réaffiche du coup toute la matrice, en prenant en compte la modification. Voir ce qui se passe si on affecte la composante d'une matrice qui n'existe pas encore. Exemple :

>> P(2,3) = 3
P =
0 0 0
0 0 3

MATLAB construit automatiquement un tableau suffisamment grand pour arriver jusqu'aux indices spécifiés, et met des zéros partout sauf au terme considéré.

On remarque que contrairement aux langages classiques, il est inutile de dimensionner les tableaux à l'avance : ils se construisent au fur et à mesure !

3 .3. Extraction de sous-tableaux

Il est souvent utile d'extraire des blocs d'un tableau existant. Pour cela on utilise le caractère **:** Il faut spécifier pour chaque indice la valeur de début et la valeur de fin. La syntaxe générale est donc la suivante (pour un tableau à deux indices) :

Tableau (début :fin, début :fin)

Pour extraire le bloc $\begin{bmatrix} 2 & 3 \\ 12 & 13 \end{bmatrix}$ on tapera :

>> M(1:2,2:3)
ans =
2 3
12 13

Si on utilise le caractère : seul, ça veut dire prendre tous les indices possibles. Exemple :

>> M(1:2,:)
ans =
1 2 3
11 12 13

C'est bien pratique pour extraire des lignes ou des colonnes d'une matrice. Par exemple pour obtenir la deuxième ligne de M :

>> M(2,:)
ans =
11 12 13

3.4. Construction de tableaux par blocs :

Par exemple, à partir des matrices et vecteurs précédemment définis, on peut définir la matrice :

$N = \begin{bmatrix} M & V \\ U & o \end{bmatrix}$

qui est une matrice 4x4. Pour faire ça sous MATLAB, on fait comme si les blocs étaient des scalaires, et on écrit tout simplement :

```
>> N= [M V
U 0]


N =
 1   2   3  11
11  12  13  12
21  32  23  13
 1   2   3   0
```

ou bien en utilisant le caractère ;

```
>> N= [M V; U 0]
```

Cette syntaxe est très utilisée pour allonger des vecteurs ou des matrices, par exemple si je veux ajouter une colonne à M, constituée par V :

```
>> M = [M V]
M =
 1   2   3  11
11  12  13  12
21  32  23  13
```

Si je veux lui ajouter une ligne, constituée par U :

```
>> M = [M;U]
M =
1   2   3
11  12  13
21  32  23
1   2   3
```

3.5. Opérations sur les tableaux

3.5.1. Addition et soustraction :

Les deux opérateurs sont les mêmes que pour les scalaires. A partir du moment où les deux tableaux concernés ont la même taille, Le tableau résultant est obtenu en ajoutant ou en soustrayant les termes de chaque tableau.

3.5.2. Multiplication, division et puissance terme à terme :

Ces opérateurs sont notés .*, ./ et .^ (attention à ne pas oublier le point). Ils sont prévus pour effectuer des opérations termes à terme sur deux tableaux de même taille. Ces symboles sont fondamentaux lorsque l'on veut tracer des courbes.

3.5.3. Multiplication, division et puissance au sens matriciel :

Puisque l'on peut manipuler des matrices, il paraît intéressant de disposer d'une multiplication matricielle. Celle-ci se note simplement * et ne doit pas être confondue avec la multiplication terme à terme. Il va de soi que si l'on écrit A*B le nombre de colonnes de A doit être égal au nombre de lignes de B pour que la multiplication fonctionne.

La division a un sens vis-à-vis des inverses de matrices. Ainsi A/B représente A multipliée (au sens des matrices) à la matrice inverse de B.

Il existe aussi une division à gauche qui se note \. Ainsi A\B signifie l'inverse de A multiplié par B.

La puissance $n^{ième}$ d'une matrice représente cette matrice multipliée n fois au sens des matrices par elle-même.

Pour bien montrer la différence entre les opérateurs .* et *, un petit exemple faisant intervenir la matrice identité multipliée à la matrice $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$. Voici la multiplication au sens des matrices :

```
>> [1 0; 0 1] * [1 2; 3 4]
ans =
1  2
3  4
```

et maintenant la multiplication terme à terme :

```
>> [1 0; 0 1] .* [1 2; 3 4]
ans =
1 0
0 4
```

3.5.4. Transposition

L'opérateur transposition est le caractère ' et est souvent utilisé pour transformer des vecteurs lignes en vecteurs colonnes et inversement.

3.5.5. Synthèse

Le tableau suivant résume les différents opérateurs applicables aux matrices.

Syntaxe MATLAB	Ecriture mathématique	Terme général
A	A	Aij
B	B	Bij
A+B	A+B	Aij + Bij
A-B	A-B	Aij – Bij
A.*B		AijBij
A./B		Aij/Bij
A.^B		$A_{ij}{}^{B_{ij}}$
A.^s		$A_{ij}{}^{s}$
A*B	AB	$\sum_k A_{ik} B_{kj}$
A/B	AB^{-1}	
A\B	A^{-1}B	
A^n	A^n	
A'	A^T	A_{ji}

3.6. Longueurs des tableaux

La fonction ***size*** appliquée à une matrice renvoie un tableau de deux entiers : le premier est le nombre de lignes, le second est le nombre de colonnes. La commande fonctionne aussi sur les vecteurs et renvoie 1 pour le nombre de lignes (resp. colonnes) d'un vecteur ligne (resp colonne).

Pour les vecteurs, la commande ***length*** est plus pratique et renvoie le nombre de composantes du vecteur, qu'il soit ligne ou colonne.

3.7. Génération rapide de tableaux

3.7.1. Matrices classiques

On peut définir des matrices de taille donnée ne contenant que des 0 avec la fonction zeros, ou ne contenant que des 1 avec la fonction ***ones***. Il faut spécifier le nombre de lignes et le nombre de colonnes.

ATTENTION, pour engendrer des vecteurs lignes (resp. colonnes), il faut spécifier explicitement «1» pour le nombre de lignes (resp. colonnes). Voici deux exemples :

```
>> ones (2,3)
ans =
1 1 1
1 1 1
>> zeros(1,3)
ans =
0 0 0
```

L'identité est obtenue avec eye. On spécifie seulement la dimension de la matrice (qui est carrée. . .)

```
>> eye(3)
ans =
1 0 0
0 1 0
0 0 1
```

3.7.2. Listes de valeurs

Cette notion est capitale pour la construction de courbes. Il s'agit de générer dans un vecteur une liste de valeurs équidistantes entre deux valeurs extrêmes.

La syntaxe générale est : variable = valeur début: pas: valeur fin.

Cette syntaxe crée toujours un vecteur ligne. Par exemple pour créer un vecteur x de valeurs équidistantes de 0.1 entre 0 et 1 :

```
>> x = 0:0.1:1
x =
Columns 1 through 11
0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000
```

Il est conseillé de mettre un point-virgule à la fin de ce type d'instruction pour éviter l'affichage fastidieux du résultat.

Autre exemple pour créer 101 valeurs équi-réparties sur l'intervalle [0, 2Л] :

```
>> x = 0: 2*pi/100 : 2*pi;
```

4. Graphique 2D :

Pour tout logiciel, une courbe 2D est représentée par une série d'abscisses et une série d'ordonnées. Ensuite, le logiciel trace généralement des droites entre ces points. MATLAB n'échappe pas à la règle. La fonction s'appelle plot.

4.1. L'instruction plot :

4..1.1. Tracer une courbe simple :

L'utilisation la plus simple de l'instruction plot est la suivante :

plot (vecteur d'abscisses, vecteur d'ordonnées)

[x1 x2 . . . xn] [y1 y2 . . . yn]

Les vecteurs peuvent être indifféremment ligne ou colonne, pourvu qu'ils soient tous deux de même type. En général ils sont lignes car la génération de listes de valeurs vue à la fin du chapitre précédent fournit par défaut des vecteurs ligne.

Par exemple, si on veut tracer sin(x) sur l'intervalle [0, 2Л], on commence par définir une série (disons 100) de valeurs équidistantes sur cet intervalle :

*>> x = 0: 2*pi/100 : 2*pi;*

puis, comme la fonction sin peut s'appliquer terme à terme à un tableau :

>> plot(x, sin(x))

On remarquera que tout ce que demande plot, c'est un vecteur d'abscisses et un vecteur d'ordonnées. Les abscisses peuvent donc être une fonction de x plutˆot que x lui-même. En d'autres termes, il est donc possible de tracer des courbes paramétrées :

>> plot(cos(x), sin(x))

4.1.2. Superposer plusieurs courbes

Il suffit de spécifier autant de couples (abscisses, ordonnées) qu'il y a de courbes à tracer. Par exemple pour superposer sin et cos :

>> plot (x,cos(x),x,sin(x))

Les deux courbes étant en réalité dans des couleurs différentes. Cette méthode fonctionne même si les abscisses des deux courbes ne sont pas les mêmes.

Dans le cas plus fréquent où les abscisses sont les mêmes, il existe un autre moyen de superposer les courbes. On fournit toujours le vecteur des abscisses, commun aux deux courbes, et on fournit autant de vecteurs d'ordonnées qu'il y a de courbes. Tous ces vecteurs d'ordonnées sont regroupés dans un même tableau, chaque ligne du tableau représentant un vecteur d'ordonnées :

plot *(vecteur d'abscisses, tableau d'ordonnées)*

[x1 x2 . . . xn] $\begin{bmatrix} y_1^1 & y_2^1 & \dots & y_n^1 \\ y_1^2 & y_2^2 & \dots & y_n^2 \\ & . & & \\ & . & & \\ y_1^m & y_2^m & \dots & y_n^m \end{bmatrix}$ $\begin{matrix} Première\ courbe \\ \dots \\ m^{ième}\ courbe \end{matrix}$

Par exemple, pour superposer sin et cos, on devra fournir à plot les arguments suivants :

plot ([x1 x2 . . . xn], $\begin{bmatrix} cos\ (x1) & cos\ (x2) & \dots & cos\ (xn) \\ sin\ (x1) & sin\ (x2) & \dots & sin\ (xn) \end{bmatrix}$ *)*

Le deuxième tableau se construit très facilement avec le point-virgule.

>> plot(x, [cos(x);sin(x)])

4.1.3. Attributs de courbes

Vous aurez remarqué que MATLAB attribue des couleurs par défaut aux courbes. Il est possible de modifier la couleur, le style du trait et celui des points, en spécifiant après chaque couple (abscisse, ordonnée) une chaine de caractères (entre quotes) pouvant contenir les codes suivants (obtenus en tapant help plot) :

Couleurs	Styles de points	Styles de lignes
y yellow	. point	- solid
m magenta	o circle	: dotted
c cyan	x x-mark	-. dashdot
r red	+ plus	-- dashed
g green	* star	
b blue	S square	
w white	d diamond	
k black	v triangle (down)	
	^ triangle (up)	
	< triangle (left)	
	> triangle (right)	
	p pentagram	
	h hexagram	

Lorsque l'on utilise seulement un style de points, MATLAB ne trace plus de droites entre les points successifs, mais seulement les points eux-mêmes. Ceci peut être pratique par exemple pour présenter des résultats expérimentaux. Les codes peuvent être combinés entre eux. Par exemple

>> plot(x,sin(x),':',x,cos(x),'r-.')

4.2. Décoration des graphiques

4.2.1. Titre

C'est l'instruction ***title*** à laquelle il faut fournir une chaine de caractères2. Le titre apparaît en haut de la fenêtre graphique :

>> plot(x,cos(x),x,sin(x))

>> title('Fonctions sin et cos')

4.2.2. Labels

Il s'agit d'afficher quelque chose sous les abscisses et à coté de l'axe des ordonnées

>> plot(x,cos(x))

>> xlabel('Abscisse')

>> ylabel('Ordonnée')

4.2.3. Légendes

C'est l'instruction ***legend***. Il faut lui communiquer autant de chaines de caractères que de courbes tracées à l'écran. Un cadre est alors tracé au milieu du graphique, qui affiche en face du style de chaque courbe, le texte correspondant. Par exemple :

>> plot (x,cos(x),':',x,sin(x),'-.',x,sqrt(x),'--')

>> legend ('cosinus','sinus','racine')

4.2.4. Tracer un quadrillage

C'est l'instruction ***grid***, qui utilisé après une instruction plot affiche un quadrillage sur la courbe. Si on tape à nouveau ***grid***, le quadrillage disparaît.

4.3. Afficher plusieurs graphiques (subplot)

Voilà une fonctionnalité très utile pour présenter sur une même page graphique un grand nombre de résultats, par exemple :

L'idée générale est de découper la fenêtre graphique en pavés de même taille, et d'afficher un graphe dans chaque pavé. On utilise l'instruction ***subplot*** en lui spécifiant le nombre de pavés sur la hauteur, le nombre de pavés sur la largeur, et le numéro du pavé dans lequel on va tracer :

>> subplot (Nbre pavés sur hauteur, Nbre pavés sur largeur, Numéro pavé)

La virgule peut être omise. Les pavés sont numérotés dans le sens de la lecture d'un texte : de gauche à droite et de haut en bas.

>> subplot(221)

>> plot(x,sin(x))

>> subplot(222)

>> plot(x,cos(x),x,sin(x),'-.')

>> subplot(223)

>> plot(cos(x),sin(x))

>> subplot(224)

*>> plot(sin(2*x),sin(3*x))*

4.4. Maintien/effacement du graphique

Par défaut une instruction plot efface systématiquement le graphique précédent. Il est parfois utile de le conserver et de venir le surcharger avec la nouvelle courbe. Pour cela on utilise la commande ***hold***. Pour démarrer le mode surcharge, taper ***hold*** on, pour revenir en mode normal, ***hold off***. Il est conseillé de ne pas abuser de cette commande.

Pour effacer la fenêtre graphique, tapez simplement ***clf***. Cette commande annule également toutes les commandes ***subplot*** et ***hold*** passées.