Didacticiel MATLAB

Introduction

Matlab, pour Matrix Laboratory, est un logiciel de calcul scientifique numérique et matriciel qui permet de simplifier les problèmes de programmation [1]. En effet, Matlab est conçu de manière à rendre le plus facile possible la manipulation de matrices (un vecteur ligne de taille N étant une matrice de taille $1 \times N$) : d'une part le langage de base est proche des notations utilisées en algèbre linéaire et d'autre part bon nombre de méthodes sont déjà présentes dans les bibliothèques pré-intégrées. Néanmoins il est nécessaire de connaître quelques extensions de syntaxe que nous allons introduire dans les feuilles qui suivent.

1 Quelques généralités.

Pour utiliser Matlab, ouvrir un terminal et `matlab&`.

Pour entrer une instruction, vous devez taper dans la fenêtre principale le nom de votre commande à la suite du prompt représenté par le signe `>>`.

Pour les noms de vos variables ou de vos scripts, éviter d'utiliser des dénominations déjà affectées par le logiciel, comme `sin`, `max` ou encore `pi` et `i`. A cet effet, vous pouvez utiliser l'aide de Matlab.

L'aide en ligne est appelée par `help`. La commande `lookfor` permet de rechercher des commandes à l'aide d'un mot clé. L'aide existe aussi sous forme HTML.

```
>> help help
>> helpbrowser
>> lookfor maximum
```

Une dernière remarque : Matlab différencie les majuscules des minuscules.

2 Matrices et vecteurs.

2.1 Création

Rappelez vous d'une chose : dans Matlab, un vecteur est une matrice dont une des dimensions vaut 1.

Pour créer un vecteur ligne, il suffit d'écrire la suite des composantes entre crochets en les séparant par un espace ou une virgule :

```
>> w = [1 pi 5 i]
w =
     1.0000    3.1416    5.0000    0 + 1.0000i
```

1. `Matlab` consiste en un langage interprété qui s'exécute dans une fenêtre dite *d'exécution*. À part sa richesse fonctionnelle, il est facile à utiliser puisqu'il n'exige pas de compilation, et les variables utilisées sont implicitement déclarées.

Pour créer une matrice, les éléments sont entrés ligne par ligne en séparant celles-ci par des points-virgules.

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

Pour créer un vecteur colonne, deux solutions se présentent : créer une matrice avec une seule colonne ou créer un vecteur ligne et le transposer.

```
>> w1 = [j ; 2 ; 5 ; 9]
w1 =
     0 + 1.0000i
     2.
     5.
     9.
>> w2 = [j 2 5 9]
w2 =
     0 + 1.0000i     2.0000     5.0000     9.0000
>> w1a = w2.'
w1a =
          0 + 1.0000i
     2.0000
     5.0000
     9.0000
>> w1b = w2'
w1b =
          0 - 1.0000i
     2.0000
     5.0000
     9.0000
```

Remarque : ne pas mettre le point avant l'apostrophe revient à calculer le transposé conjugué. Bien évidemment ceci est aussi valable pour une matrice.

Afin d'éviter que Matlab n'affiche le résultat (en particulier lorsque le temps d'affichage de la sortie est long : essayer `rand(200)`), il suffit de placer un point-virgule en fin de ligne.

A noter que Matlab ne garde en mémoire, dans une variable appelée `ans`, que le dernier résultat obtenu. C'est pour cette raison qu'il faut donner un nom à toutes les variables que l'on désire employer plusieurs fois.

Les commandes `who` ou `whos` permettent de lister les variables que vous avez créées. Vous retrouverez aussi ces informations dans la fenêtre Workspace à gauche.

Pour afficher le contenu complet d'un vecteur, il suffit de taper son nom. La commande `clear` permet d'effacer une ou toutes les variables (attention, Matlab ne demande pas de confirmation, et une fois effacées, les variables sont irrécupérables...). *Tester la suite d'instructions suivantes :*

```
>> w1b
>> clear w1b
```

```
>> w1b
>> who
>> clear
>> who
```

Les progressions arithmétiques s'obtiennent, comme vecteurs lignes, avec la syntaxe début : raison : fin. Lorsque la raison est absente elle prend, par défaut, la valeur 1. *Tester la suite d'instructions suivantes :*

```
>> v = 1 : 2 : 12
>> w = 6 : -1 : 1
>> x = 1 : 6
```

Il est aussi possible d'utiliser `linspace` (voir `help linspace`).

2.2 Accès aux éléments

Pour avoir accès, en lecture ou en écriture, à l'élément d'une matrice A situé en ligne `k` et colonne `l`, il faut taper `A(k,l)`. Pour un vecteur, cela peut se simplifier en ne désignant que le numéro de la composante : par exemple la troisième composante de v

```
>> v(3)
ans =
     5
```

Remarque : attention sous Matlab, les indices des tableaux, vecteurs ou matrices commencent à 1 et non à 0 !

```
>> v(0)
??? Subscript indices must either be real positive integers or logicals.
```

De même que pour un vecteur, il est possible de décrire les éléments d'une matrice A par un seul indice : `A(k)` est le k^{ieme} élément en parcourant la matrice colonne par colonne, de la première à la dernière.

```
>> A = [ 1 2 3 ; 4 5 6 ; 7 8 9] ;
>> A(3)
ans =
     7
```

Pour accéder à un sous-ensemble de composantes, à une ligne ou une colonne complète, voir les exercices 1 et 2 du TP1.

2.3 Opérations

Pour les opérations simples telles que addition, soustraction et multiplication matricielle, les notations sont classiques pour l'algèbre linéaire. Cependant il ne faut pas oublier d'être attentif aux dimensions des vecteurs ou matrices que l'on manipule, sinon gare aux messages d'erreur ! ! **Il y a en particulier une contradiction entre la définition habituelle des vecteurs en algèbre linéaire, qui sont des vecteurs colonnes, et la définition par défaut des vecteurs par Matlab, qui sont des vecteurs lignes**.

En cas de doute, les commandes `size` et `length` vous permettront d'obtenir la taille et la dimension d'une matrice ou d'un vecteur. *Tester la suite d'instructions suivantes :*

```
>> size(v)
>> size(v')
>> length(v)
>> size(A)
>> length(A)
```

Il faut bien noter que les opérateurs classiques `+`, `-`, `*` et `^`(puissance) réalisent des opérations matricielles. Mais il peut arriver que l'on veuille faire ces opérations composantes par composantes. Matlab permet d'effectuer cela très rapidement (beaucoup plus rapidement que si l'on utilise une boucle !), simplement en accolant devant l'opérateur un point (par exemple `.*` ou `./`).

```
>> v = [1 2 3] ; b = [2 4 6] ;
>> v.*b
ans =
     2     8    18
>> b./v
ans =
     2     2     2
```

A ne pas confondre avec :

```
>> v*b.'
ans =
    28.
```

Faire l'exercice 3 du TP1.

Par ailleurs la plupart des fonctions mathématiques de Matlab (`log`, `exp`, `cos`,...) agissent, naturellement, termes à termes, sur les matrices. Par contre, d'autres (`max`, `sum`, `mean`, `prod`,...) agissent sur les colonnes ou sur les lignes. Pour cette raison, il vous est conseillé de bien lire l'aide avant toute utilisation d'une commande et de faire un essai sur un cas simple. *Tester sur A.*

3 Boucles et tests

Les instructions de type boucle ou test sont celles usuellement utilisées en programmation avec la syntaxe suivante : un mot clé de début (`for`, `while` ou encore `if`) et finissent avec le mot clé `end`. Pour avoir la syntaxe complète, consulter l'aide.

Utiliser le moins possible ces structures de boucle ; en particulier exploiter au maximum les potentialités de Matlab avec les opérations composantes par composantes qui sont beaucoup plus efficaces que les boucles. *Tester l'illustration ci-dessous en utilisant la montre automatique :*

```
>> clc
>> y = [0 :0.01 :100*%pi] ; M = length(y)
>> tic(), z=y.*[1 :M] ; TimeDirect=toc()
>> tic(), for i=1 :M, y(i) = y(i)*i ; end, TimeBoucle=toc()
>> TimeBoucle/TimeDirect
>> max(abs(y-z))
```

Le temps d'éxécution d'une boucle est beaucoup plus long !!! Et pourtant on calcule bien la même chose, car la dernière commande montre que toutes les composantes de $y - z$ valent 0.

Remarque : constater dans la suite d'instructions précédentes l'emploi de `clc` et de la virgule ou du point-virgule entre deux instructions tapées sur la même ligne.

Tester ce dernier exemple pour ***if*** *:*

```
>> clc, p=0.4 ; y=rand()
>> if y< p x=1 ; elseif y>= p x=0 ; end, x
>> y<p
>> u=rand(10,10)
>> u<p
```

On remarquera que si le test logique est vrai, alors 1 est retourné, sinon on obtient 0. Par ailleurs, on observera le fonctionnement d'un test logique sur une matrice.

4 Programmation.

Les notions de script et de fonction existent sous Matlab. Ce sont des fichiers externes qui contiennent une séquence d'instructions Matlab et qui sont utiles dès que les calculs à effectuer requièrent plus de quelques lignes de commande. Un script est juste une suite d'instructions qui s'exécutent automatiquement, alors qu'une fonction enrichit le logiciel Matlab de nouvelles potentialités et peut posséder des arguments d'entrée et de sortie. Mais la différence fondamentale entre un script et une fonction réside en ce que les variables utilisées sont conservées et accessibles dans l'espace de travail (workspace) pour le premier (scipt) tandis que pour le second (fonction), les variables internes sont détruites dès que la fonction a retourné ses valeurs. Ainsi, lorsqu'on rédige un programme conséquent, on a tout intérêt, au début, à utiliser des scripts à la place des fonctions de façon à faciliter la détection des erreurs.

Pour créer ces fichiers, vous pouvez utiliser votre éditeur habituel ou l'éditeur intégré de Matlab (accessible à partir du menu) : cette deuxième solution est à privilégier car les erreurs de syntaxe sont signalées. La sauvegarde doit être effectuée avec l'extension " .m " pour les deux types de fichiers. Pour plus de simplicité, nous vous conseillons de placer toutes vos fonctions dans le même répertoire.

4.1 Fonctions

La définition d'une fonction commence obligatoirement par une ligne qui déclare le nom de la fonction, les variables d'entrée et le vecteur des variables de sortie, et se termine avec la commande `end` (facultatif).

```
function [args1,args2,...] = nomfonction(arge1,arge2,...)
   instructions
end
```

args1,args2,... : arguments de sortie de la fonction qui peuvent être de n'importe quel type

arge1,arge2,... : arguments d'entrée de la fonction qui peuvent être de n'importe quel type

instructions : bloc d'instructions quelconque devant affecter les arguments de sortie

L'appel à la fonction s'opère de la façon suivante, en restant vigilant sur la compatibilité de dimension entre x et ***arge1,arge2,...*** et celle entre y, et ***args1,args2,...***, si y avait été créée avant l'appel à la fonction :

```
--> y=nomfonction(x)
```

Voir l'exercice 4 du TP1 pour illustrer l'emploi d'une fonction.

4.2 Scripts

Comme nous l'avons dit plus haut, un script est un fichier contenant une suite d'instructions à exécuter ; Lancer un script permet donc d'exécuter une suite d'instructions de manière automatique.

Pour lancer un script, vous avez deux solutions :
- depuis l'éditeur de Matlab, aller dans le menu Debug et cliquer sur Run (ou utiliser le raccourci clavier F5) ;
- dans la fenêtre d'exécution, se placer dans le répertoire où le script a été enregistré et taper son nom.

Créer le script donné ci-dessous qui réalise l'algorithme du pivot de Gauss.

```
% Effacer toutes les variables
clear

% Création de la matrice à traiter
A=[1 3 -4 ;0 -1 5 ; 2 0 4] ;

% Récupération des dimensions de A
S=size(A) ;

%Initialisation de B
B=A ;

% Boucle du pivot de Gauss
for l=1 :S(2)-1
   for k=l+1 :S(1)
      B(k, :)=B(k, :)-B(l, :)*B(k,l)/B(l,l) ;
   end
end
B
```

Remarque : on notera que les commentaires sont précédés de % sur la même ligne.

Il est vivement recommandé d'avoir une présentation claire et aérée de vos fichiers, en n'écrivant qu'une seule instruction par ligne et en commentant abondamment.

Voir l'exercice 5 du TP1 pour illustrer l'emploi d'un script.

5 Les graphiques

Matlab possède un vaste ensemble de fonctionnalités graphiques 2D et 3D. Nous ne présentons ici que quelques principes de base. Que ce soit pour tracer le graphe d'une fonction dans le plan ou dans l'espace, la technique peut se décomposer en trois temps :

1. discrétiser le domaine de représentation ;
2. évaluer la fonction en chaque point de ce domaine discrétisé ;
3. exécuter l'instruction graphique avec les données précédentes.

Pour commencer quelques remarques et commandes générales :

– Par défaut, lorsque l'on effectue deux tracés successifs, Matlab efface la fenêtre courante et affiche le second graphe dans cette même fenêtre.
– `figure` crée une nouvelle fenêtre graphique ou permet de sélectionner la fenêtre dans laquelle l'on souhaite afficher le graphe.
– `hold on` permet de supperposer plusieurs graphes dans une même fenêtre graphique. Dans ce cas, attention de choisir des couleurs différentes pour distinguer les graphes.
– `close` ferme une fenêtre graphique (`close all` les ferme toutes en même temps).

Graphiques 2d

La fonction `plot` permet de tracer des graphes de fonctions en deux dimensions. Elle prend essentiellement deux arguments : un vecteur d'abscisses et un vecteur d'ordonnées qui doivent être de dimensions compatibles. Un troisième argument facultatif permet de fixer le style du graphe.

Prenons un exemple avec les fonctions $\sin(x)$, $\sin(2x)$ et $x.\sin(x)$ sur l'intervalle $[-\pi, \pi]$ avec 201 points. *Exécuter les commandes suivantes :*

```
>> x=-pi :pi/100 :pi ;
>> y=sin(x) ; plot(x,y) ;
>> z=sin(2*x) ; plot(x,z) ;
>> figure
>> y=sin(x) ; plot(x,y) ;
>> hold on
>> t=x.*sin(x) ; plot(x,t,'r') ;
>> grid on
```

La deuxième instruction ci-dessus dessine une courbe passant par les points $(x(i), y(i))$. Les vecteurs x et y doivent donc avoir la même longueur.

Il est possible de tracer des graphes en escalier ou en bâtons avec d'autres commandes et il existe de nombreuses options pour contrôler l'affichage des courbes (couleur, axe, légendes, commentaires, ...) : voir la rubrique "Functions" catégorie "Graphics" de l'aide ou taper `help graphics`.