TP 1 – Introduction à Matlab

Février 2009

1 Introduction

Matlab pour « MATtrix LABoratory », est un logiciel qui a été conçu pour fournir un environnement de calcul numérique de haut niveau. Il est particulièrement performant pour le calcul matriciel car sa structure de données interne est basée sur les matrices. Il dispose également de grandes capacités graphiques pour, par exemple, la visualisation d'objets mathématiques complexes. Son fonctionnement repose sur un langage de programmation interprété qui permet un développement très rapide. Pour des applications nécessitant un temps de calcul plus élevé, un langage compilé comme le `C++` ou le `fortran`, est mieux adapté.

Ce premier énoncé peut servir d'aide mémoire pour les commandes Matlab les plus courantes, les exercices se trouvant aux deux dernières pages.

---

2 Premiers pas

2.1 Lancement de Matlab

L'interface Matlab se compose d'une fenêtre principale divisée en trois sous-fenêtres.

(*i*) En haut à gauche, il y a une fenêtre contenant deux onglets : `Launch Pad` et `Workspace`. L'onglet `Launch Pad` est visible par défaut. Le `Launch Pad` ne nous concerne pas ; il s'agit d'une interface pour obtenir de l'information et des demos sur des composantes de Matlab. Nous ne l'utiliserons pas. Le `Workspace` permet de gérer les variables utilisées. Nous y reviendrons au paragraphe 2.3.

(*ii*) En bas à gauche, il y a une fenêtre contenant deux onglets : `Command History` et `Current Directory`. L'onglet `Command History` est visible par défaut ; il indique les dernières commandes effectuées. Le `Current Directory` gère l'emplacement des fichiers. Celui-ci sera utile pour le travail avec les `m-files` . Nous y reviendrons au paragraphe 5.1.

(*iii*) Sur la droite, il y a une grand fenêtre : `Command Window`. La `Command Window` est la fenêtre d'interaction avec Matlab.

2.2 Commandes et calculs de base

Matlab fonctionne de manière similaire à un shell `Linux` ou `DOS`. L'utilisateur rentre des commandes et Matlab les exécute.

```
>> 2+2
ans =
    4
```

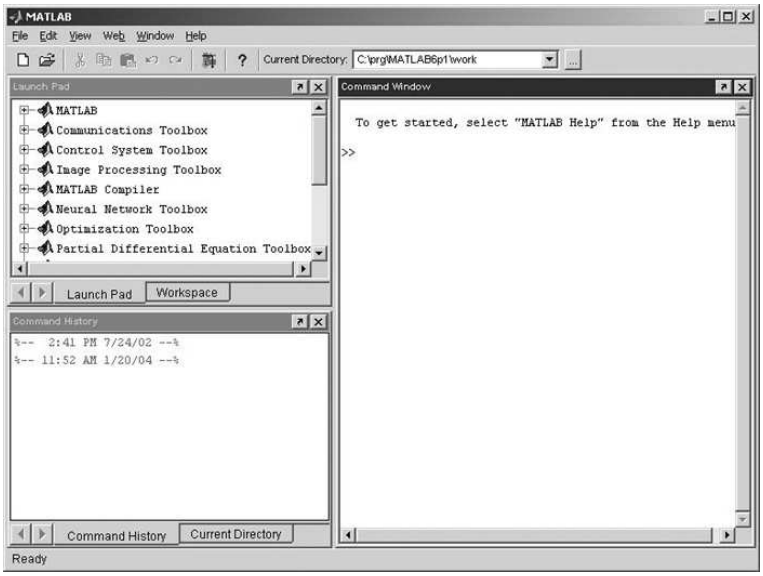

FIG. 1 – Lancement de MATLAB.

Le symbole [>>] indique à l'utilisateur où il faut rentrer la commande. On ne peut pas « revenir en arrière », c'est-à-dire, il ne faut pas essayer de placer le curseur sur une ligne au-dessus du dernier [>>]. Pour taper une autre commande on le fait à la suite.

```
>> 2+2
ans =
    4

>> 3*5
ans =
    15
```

Si on rentre des commandes erronées, MATLAB nous l'indique par un message d'erreur.

```
>> 5*
??? 5*
    |
Error: Expected a variable, function, or constant, found "end of line".

>> aaa
??? Undefined function or variable 'aaa'.
```

Les touches [↑] et [↓] permettent de naviguer parmi les dernières commandes effectuées, ce qui peut être utile si l'on commet une erreur et qu'on veut éviter de taper à nouveau toute la commande.

MATLAB possède de nombreuses fonctions prédéfinies utiles en mathématiques que nous allons étudier au cours de ces travaux pratiques.

```
>> pi
ans =
```

```
    3.1416

>> sin(pi/6)
ans =
    0.5000

>> log(1.5)
ans =
    0.4055

>> i^2
ans =
    -1
```

Il peut parfois être utile de stocker une valeur dans une variable pour l'utiliser plus tard. L'affectation d'une variable en MATLAB se fait au moyen du signe [=]. Le nom d'une variable doit commencer par une lettre (majuscule ou minuscule, sans accent) puis peut contenir des lettres (même remarque), des chiffres et des caractères soulignés [_]. Le nom peut contenir au maximum 31 caractères. La valeur d'une variable peut être un nombre, une chaîne de caractères ou un tableau (voir la section 3). Contrairement au `C++` ou au `fortran`, MATLAB n'est pas « typé ». Autrement dit, une variable contenant un entier peut contenir plus tard une chaîne de caractères ou un tableau. Précisons que MATLAB est « case-sensitive », c'est-à-dire qu'il fait la distinction entre majuscules et minuscules.

```
>> A=23
A =
    23

>> a=2.432
a =
    2.4320

>> A='salut !'
A =
    salut !
```

On peut évidement faire des calculs avec des variables. Le résultat d'un calcul est, par défaut, stocké dans une variable nommée `ans`. Celle-ci peut être changée pour n'importe quelle autre variable. Par défaut, MATLAB affiche le résultat de la dernière opération. Cet affichage peut être supprimé en terminant votre commande par la touche [;]. Plusieurs commandes peuvent être rentrées sur une même ligne en les séparant soit par [,] soit par [;].

```
>> x=2;y=5;
>> z=x^2+y^2
z =
    29
```

Pour une liste complète des opérations mathématiques que l'on peut faire dans MATLAB voir le paragraphe 3.3.

2.3 Gestion des variables

Dès que nous commençons à avoir un certain nombre de variables, on peut rapidement se perdre. Si l'on tape le nom d'une variable, MATLAB renvoie la valeur de celle-ci. Mais comment savoir quelle variable a été utilisée ? Pour se retrouver, MATLAB propose plusieurs solutions. La commande `who` permet de lister simplement les variables utilisées, alors que `whos` donne des informations détaillées sur toutes les variables.

```
>> who
Your variables are:

A    a    ans  x    y    z

>> whos
    Name      Size         Bytes  Class

    A         1x7             14  char array
    a         1x1              8  double array
    ans       1x1              8  double array
    x         1x1              8  double array
    y         1x1              8  double array
    z         1x1              8  double array

Grand total is 12 elements using 54 bytes
```

L'onglet `Workspace` donne une alternative graphique à la commande `whos`. En double-cliquant sur une variable on peut voir sa valeur et même la modifier.

Pour effacer complètement une variable, il suffit de rentrer la commande `clear` suivie du nom de la variable. Pour tout effacer, `clear all`.

2.4 Historique des commandes

MATLAB garde en mémoire les dernières commandes effectuées. Elles sont visibles dans l'onglet `Command History`. On peut également y accéder directement dans la `Command Window` au moyen des touches [↑] et [↓]. Ceci est particulièrement utile pour répéter la dernière commande.

2.5 Aide

MATLAB possède un grand nombre de fonctions et commandes. On ne pourra pas toutes les traiter en détail. Afin d'obtenir de l'information (nombre de paramètres d'une fonction, valeur de retour, etc), il suffit de rentrer `help nom_de_la_commande`.

La commande `lookfor` est très utile. Elle permet de chercher les fonctions par mots-clefs. Plus précisément, `lookfor XYZ` renvoie toutes les fonctions qui contiennent `XYZ` dans la première ligne de leur descriptif. Nous y reviendrons au paragraphe sur `m-files`.

Si vous êtes perdu, la commande `help help` pourra vous aider...

En programmant, il arrive que des erreurs s'insinuent dans le code. Ces erreurs peuvent être de plusieurs types. Tout d'abord des erreurs de syntaxe qui sont signalées par le

programme et sont donc "faciles" à localiser. D'autres erreurs sont plus difficiles à corriger. Il s'agit des vraies erreurs de programmation. Le programme fait quelque chose, mais pas ce que vous voulez. Pour se rendre compte de la présence de telles erreurs, on teste son programme sur des exemples faciles dont la réponse est connue.

Dans le cas où vous êtes confrontés à une telle erreur, les outils de debuggage de MatLab peuvent vous aider.

Par exemple, vous pouvez placer dans votre code (M-file) des "break point". Ceux-ci interrompent l'exécution à l'endroit choisi et vous pouvez accéder aux valeurs des variables internes à cette instant via la fenêtre "Workspace". Vous pouvez exécuter la suite du programme jusqu'au break point suivant ou ligne après ligne. Ceci peut être très utile pour comprendre son erreur de raisonnement.

2.6 Sauvegarde

MATLAB ne permet pas de sauvegarder l'historique des commandes exécutées. Il existe cependant deux solutions pour sauvegarder son travail.

(*i*) Le `Workspace`. On peut sauver l'état de la session en cours dans un fichier `.mat`. Pour cela, dans la fenêtre principale, `File` → `Save Workspace As`, et vous choisissez l'emplacement et le nom de votre fichier. MATLAB sauvegarde ainsi le nom et la valeur de chacune des variables. La prochaine fois que vous utilisez MATLAB, au moyen du menu `Files` → `Open` vous retrouvez le `Workspace` dans l'état dans lequel vous l'avez laissé. Vous ne verrez cependant pas l'historique des commandes.

(*ii*) Les `m-files`. Un peu plus loin, à la section 5, on introduira la notion de `m-files`. Il s'agit d'un fichier dans lequel on regroupe des commandes. C'est très utile pour aborder des problèmes plus complexes et éviter de retaper les mêmes commandes plusieurs fois.

(*iii*) Attention n'oubliez pas de sauvegarder vos fichiers sur votre espace mémoire étudiant ou sur un support personnel (clef USB par exemple). Ne sauvegarder pas votre travail sur le disque local de l'ordinateur, sinon il sera perdu au prochain redémarrage de l'appareil.

---

3 Vecteurs et matrices

La structure de données de MATLAB est le *tableau*[1] ; même un nombre est considéré comme une matrice 1×1. Toutes les fonctions et opérations relatives aux tableaux sont très optimisées et sont à utiliser aussi souvent que possible.

3.1 Création

Un tableau est délimité par des crochets. On sépare les colonnes par des espaces et les lignes par des points-virgules.

[1] Les termes *tableau* et *matrice* sont synonymes.

```
>> A=[1 1 1 ; 2 2 2]
A =
    1     1     1
    2     2     2

>> B=[1 ; 2 ; 3]
B =
    1
    2
    3

>> C=[1.1 2.2 3.3]
C =
    1.1000    2.2000    3.3000
```

Les tableaux qui n'ont qu'une seule ligne sont appelés des *vecteurs lignes* ou des *listes* ; ceux qui n'ont qu'une seule colonne sont appelés des *vecteurs colonnes* ou simplement des *vecteurs*. Si le nombre d'éléments dans chaque ligne (ou colonne) n'est pas le même, MATLAB signale une erreur.

```
>> A=[1 1 1; 1 2]
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

MATLAB propose des commandes pour créer certaines matrices particulières très simplement. Pour plus d'information, lire le `help` de chaque fonction.

Commande	**Description**
`ones(n,m)`	Matrice de taille $n \times m$ ne contenant que des 1.
`zeros(n,m)`	Matrice de taille $n \times m$ ne contenant que des 0.
`eye(n,m)`	Matrice de taille $n \times m$ contenant des 1 sur la première diagonale et des 0 ailleurs.
`diag(v)`	Matrice diagonale où les éléments de la diagonale sont les composantes du vecteur v.
`rand(n,m)`	Matrice de taille $n \times m$ contenant des nombres aléatoires entre 0 et 1.

TAB. 1 – Commandes pour créer des matrices.

MATLAB dispose également de moyens très simples pour créer des listes. La commande `[a:h:b]`[2] crée une liste dont les éléments sont

$$a,\ a+h,\ a+2h,\ \ldots,\ a+nh,$$

où $n \in \mathbb{N}$, $|a+nh| \leq |b|$ et $|a+(n+1)h| > |b|$. Le cas particulier `[a:b]` est un raccourci pour `[a:1:b]`. Si les conditions initiales sont erronées, MATLAB renvoie un message d'erreur.

```
>> x=[1:2:10]
x =
    1     3     5     7     9

>> y=[-5:0]
y =
    -5    -4    -3    -2    -1     0

>> z=[10:2:-10]
z =
    Empty matrix: 1-by-0
```

Un autre cas particulier de `[a:h:b]` est la fonction `linspace(a,b,n)`. Celle-ci crée une liste de n éléments uniformément répartis entre a et b. Autrement dit, `linspace(a,b,n)` est la même chose que `[a:`$\frac{b-a}{n-1}$`:b]`.

Il est parfois utile de travailler avec des échelles logarithmiques ; pour cela, la commande `logspace(x1,x2,n)` crée une liste de n points répartis logarithmiquement uniformément entre 10^{x_1} et 10^{x_2}.

Une dernière méthode pour créer des tableaux est la concaténation. Si `A` et `B` sont deux tableaux, alors `[A B]`, ou `[A,B]` est le tableau obtenu en collant `B` à la droite de `A`, et `[A ;B]` est le tableau obtenu en collant `B` au-dessous de `A`. Comme d'habitude, il faut faire attention aux tailles de `A` et de `B`.

```
>> A=[1,3,5], B=[2,4,1], C=[1,1;1,2]
A =
    1     3     5
B =
    2     4     1
C =
    1     1
    1     2

>> [A,B],[A;B]
ans =
    1     3     5     2     4     1
ans =
    1     3     5
    2     4     1
```

[2] Il n'y a pas de différence entre les commandes `[a:h:b]`, `(a:h:b)` et `a:h:b`.

```
>> [A,C]
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must have the
 same number of rows.
```

3.2 Accès et modifications

On présente dans ce paragraphe diverses méthodes pour accéder et modifier les éléments d'une matrice. Dans la table qui suit, `A` désigne un tableau de taille quelconque, `k` et `l` sont des nombres entiers, `v` est une liste et `M` une matrice.

Commande	**Description**
`A(k,l)`	Renvoie l'élément se trouvant à la $k^{\text{ème}}$ ligne et la $l^{\text{ème}}$ colonne.
`A(k)`	Renvoie le $k^{\text{ème}}$ élément d'une matrice. En MATLAB, les éléments d'une matrice de taille $n \times m$ sont indexés de 1 à nm de haut en bas et de gauche à droite.
`A(v)`	Renvoie une liste contenant les éléments dont l'indice appartient à `v`. Si `v` est un vecteur colonne, le résultat est le même mais sous forme de vecteur colonne.
`A(M)`	Renvoie une matrice contenant les éléments dont l'indice appartient à `M`.
`A(k,:)`	Renvoie la $k^{\text{ème}}$ ligne de la matrice.
`A(:,l)`	Renvoie la $l^{\text{ème}}$ colonne de la matrice.

TAB. 2 – Commandes pour accéder aux éléments d'une matrice.

```
>> A=[1 2 3 4; 12 13 14 15]
A =
     1     2     3     4
    12    13    14    15

>> A(2,3)
ans =
    14

>> A(2,:)
ans =
    12    13    14    15
```

```
>> A([1 3 5 7])
ans =
    1     2     3     4

>> A(:,4)
ans =
     4
    15

>> A([1 1 1])
ans =
    1     1     1

>> A([1 3; 4 8])

ans =
     1     2
    13    15
```

Pour modifier les éléments d'une matrice, on utilise les mêmes commandes que ci-dessus. On ajoute à la commande le signe [=] et la nouvelle valeur.

```
>> A
A =
     1     2     3     4
    12    13    14    15

>> A(2,2)=999
A =
     1     2     3     4
    12   999    14    15

>> A([2 3 5]) = [-1 -1 -1]
A =
     1    -1    -1     4
    -1   999    14    15

>> A(:,4)=[101 103]'
A =
     1    -1    -1   101
    -1   999    14   103
```

Remarquons cependant que dans ce cas on est autorisé à dépasser la taille de la matrice initiale. MATLAB crée automatiquement une nouvelle matrice en ajoutant aux anciennes valeurs les nouvelles. Si rien n'est spécifié, il remplit avec des 0.

```
>> A=[1 2; 3 4]
A =
```

```
     1     2
     3     4

>> A(2,5) = 34
A =
     1     2     0     0     0
     3     4     0     0    34
```

3.3 Opérations avec les matrices

Opérations de bases. Comme vu dans le cours d'algèbre, on peut faire certaines opérations avec des matrices. MATLAB est un peu plus souple et permet de faire certaines opérations qui n'ont pas une signification mathématique. Dans ce qui suit, `A` et `B` sont des tableaux et `c` est un scalaire.

Commande	**Description**
`A+B`	Addition terme à terme ; `A` et `B` doivent avoir le même format.
`A+c = c+A`	Addition de `c` aux éléments de `A`.
`A-B`	Soustraction terme à terme ; `A` et `B` doivent avoir le même format.
`A-c`	Soustraction de `c` aux éléments de `A`.
`c-A`	Tableau dont les éléments sont $c - a_{ij}$.
`A*B`	Produit matriciel standard ; nb. col. `A` doit être le même que nb. lign. `B`.
`A*c = c*A`	Multiplication de `c` aux éléments de `A`.
`A.*B`	Multiplication terme à terme ; `A` et `B` doivent avoir le même format.
`A^n` ($n \in \mathbb{Z}_+$)	`A * A * ... * A` (n fois) ; `A` doit être carrée.
`A^n` ($n \in \mathbb{Z}_-$)	`A^-1 * A^-1 * ... * A^-1` ($\lvert n \rvert$ fois) ; `A` doit être inversible.
`A.^B`	Tableau dont les éléments sont $a_{ij}^{b_{ij}}$; `A` et `B` doivent avoir le même format.

`A'`	Transposition et conjugaison.
`A.'`	Transposition ; `A.' = A'` dans le cas où `A` est réelle.
`B/A`	Le résultat est un tableau X tel que $XA = B$. Si A est inversible, alors $X = BA^{-1}$; nb. col. `A` doit être le même que nb. col. `B`.
`A\B`	Le résultat est un tableau X tel que $AX = B$. Si A est inversible, alors $X = A^{-1}B$; nb. lign. `A` doit être le même que nb. lign. `B`.
`A./B`	Division terme à terme des éléments de `A` par ceux de `B` ; `A` et `B` doivent avoir le même format.
`A.\B`	Division terme à terme des éléments de `B` par ceux de `A` ; `A` et `B` doivent avoir le même format.
`A/c`	Division des éléments de `A` par `c`.

TAB. 3 – Opérations avec des matrices.

Précisons que MATLAB ne renvoie pas un message d'erreur lors d'une division par 0, mais donne le résultat `Inf`. Attention néanmoins à ne pas travailler avec `Inf` comme avec un nombre.

```
>> A=[1 2 3; 0 0 1; 1 0 0]
A =
    1     2     3
    0     0     1
    1     0     0

>> B=[-1 -2 -3; 0 0 -1; -1 0 0]
B =
    -1    -2    -3
     0     0    -1
    -1     0     0

>> A+B
ans =
     0     0     0
     0     0     0
     0     0     0

>> A*B
ans =
    -4    -2    -5
    -1     0     0
    -1    -2    -3
```

```
>> A^(-2)
ans =
         0    1.0000         0
   -0.7500    1.7500    1.2500
    0.5000   -1.5000   -0.5000

>> A.*B
ans =
    -1    -4    -9
     0     0    -1
    -1     0     0

>> A/B
ans =
    -1     0     0
     0    -1     0
     0     0    -1
```

Important. Pour la résolution de systèmes d'équations, utilisez toujours les commandes `B/A` et `A\B`. N'inversez JAMAIS une matrice !

Fonctions sur les matrices. étant donnée une matrice A, il y a un certain nombre de choses que l'on peut calculer en rapport avec A. Au cours d'algèbre on a vu deux fonctions importantes définies sur l'ensemble des matrices carrés : le déterminant et la trace ; dans le cas d'un vecteur, vous avez également vu la norme. Nous présentons ici quelques fonctions définies dans MATLAB prenant comme paramètre des tableaux. Pour plus d'information, lire le `help` de chaque fonction.

Commande	**Description**
`det(A)`	Renvoie le déterminant de `A` ; celle-ci doit être carrée.
`trace(A)`	Renvoie la trace de `A`.
`rank(A)`	Renvoie le rang de `A` (dimension de l'image de l'application associée à `A`).
`null(A)`	Renvoie une base du noyau de `A` ; l'argument supplémentaire `'r'` donne une « meilleure » base (voir `help null`).
`diag(A)`	Renvoie la première diagonale de `A`.
`norm(v)`	Renvoie la norme euclidienne de `v` ; `v` est un vecteur. Il est aussi possible de calculer d'autres normes ;

	par exemple, `norm(x,'inf')`, renvoie la norme infinie.
`mean(A)`	Renvoie une liste contenant la moyenne des éléments de chaque colonne.
`sum(A)`	Renvoie une liste contenant la somme des éléments de chaque colonne.
`prod(A)`	Renvoie une liste contenant le produit des éléments de chaque colonne.
`max(A)`	Renvoie une liste contenant la valeur maximale de chaque colonne.
`min(A)`	Renvoie une liste contenant la valeur minimale de chaque colonne.
`length(A)`	Renvoie le maximum entre le nombre de lignes et de colonnes ; si `A` est un vecteur, `length(A)` est le nombre d'éléments dans le vecteur.

TAB. 4 – Fonctions sur des matrices.

Finalement, on précise que toutes les fonctions mathématiques classiques (cos, sin, log, exp, etc) s'appliquent également aux tableaux. Le résultat est un tableau où l'on a appliqué terme à terme la fonction en question.

```
>> I=[0:0.2:1]
I =
    0    0.2000    0.4000    0.6000    0.8000    1.0000

>> exp(I)
ans =
    1.0000    1.2214    1.4918    1.8221    2.2255    2.7183
```

4 Graphisme

4.1 Courbes dans le plan

étant donné deux vecteurs de même taille, `x` et `y`, la fonction `plot(x,y)` trace le graphe de `y` en fonction de `x`. En fait MATLAB relie les points de coordonnées `(x(k),y(k))` pour $1 \leq$ `k` $\leq$ `length(x)`. En prenant un grand nombre de points dans le vecteur `x` et en définissant ensuite `y = f(x)` pour une certaine fonction `f`, la fonction `plot(x,y)` nous donnera le graphe de la fonction `f`.

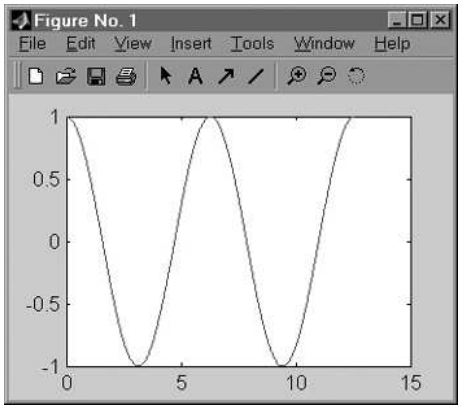

FIG. 2 – Graphe de $x \mapsto \cos(x)$.

```
>> x=[0:0.01:4*pi];
>> y=cos(x);
>> plot(x,y)
```

Plusieurs courbes. On s'aperçoit que les commandes

```
>> z=sin(x);
>> plot(x,y)
>> plot(x,z)
```

ne tracent pas deux graphes, mais un seul. En fait, le deuxième `plot(x,z)` vient remplacer le premier `plot(x,y)`. Pour remédier à cela, MATLAB propose plusieurs méthodes suivant si l'on désire que les courbes apparaissent dans une ou plusieurs fenêtres. Pour voir les graphiques sur deux fenêtres, il suffit de dire à MATLAB de construire une nouvelle fenêtre avec la commande `figure`.

```
>> plot(x,y)
>> figure
>> plot(x,z)
```

Pour avoir les deux courbes dans la même fenêtre, il existe deux méthodes équivalentes : soit avec les commandes `hold on` et `hold off`,

```
>> hold on,plot(x,y),plot(x,z),hold off
```

soit en donnant plus de paramètres à la commande `plot`.

```
>> plot(x,y,x,z)
```

Options d'affichage. On présente rapidement quelques réglages que l'on peut faire sur l'affichage des graphiques. Ce n'est pas une liste exhaustive ; la commande `help plot` donne plus de détails.

On peut commencer par régler les axes. Deux options intéressantes sont `axis equal` et `axis off`. La première met la même échelle sur les deux axes et la deuxième supprime les axes. On peut également combiner les deux.

```
>> plot(x,y), axis equal
>> plot(x,y), axis equal off
```

Les couleurs et le style du tracé peuvent également être modifiés. Pour cela, il suffit d'ajouter à `plot` une chaîne de caractères spécifiant le style ; voir `help plot` pour toutes les possibilités.

```
>> x=[0.5:0.1:5];y=log(x);
>> plot(x,y,'co'), axis equal
>>
>> x=[0:0.05:1];y=exp(x);z=log(y);
>> plot(x,y,'mo',y,z,'g>'),axis equal
```

4.2 Affichage de matrices

étant donnée A une matrice $n \times m$, MATLAB propose une méthode simple pour « visualiser » le contenu de A à l'écran. Plus précisément, MATLAB dessine un rectangle partagé en $n \times m$ petits rectangles où la couleur du rectangle (i, j) dépend de la valeur de l'élément a_{ij} de la matrice.

```
>> a=ones(11,11)
>> a([1:2:121])=0
>> imagesc(a), axis equal
>>
>> x=rand(10,10);y=rand(100,100);z=rand(1000,1000);
>> imagesc(x),axis off
>> figure
>> imagesc(y),axis off
>> figure
>> imagesc(z),axis off
```

Pour la visualisation de surfaces en 3 dimension données par des équations du type $z = f(x, y)$, MATLAB met à disposition deux fonctions : `mesh` et `surf`. La seule différence entre les deux vient du rendu graphique : `mesh` affiche la surface en fil-de-fer et `surf` en surface remplie. Essayez l'exemple ci-dessous :

```
>> [x,y]=meshgrid(-3:0.1:3,0:0.2:5);
>> z = x.^2 - y.^2;
>> surf(x,y,z)
```

Pour mieux comprendre ce que fait la fonction `meshgrid` voir `help meshgrid`.

---

5 Travail avec les `m-files`

Nous avons vu jusqu'à présent un certain nombre de commandes et outils MATLAB. Il est cependant désagréable de devoir taper les commandes plusieurs fois. Pour l'élaboration de programmes complexes cette méthode s'avère inutilisable. Nous allons voir comment regrouper des commandes dans un fichier appelé un `m-file`[3]. Il y en a deux types, les

[3]Le nom `m-file` vient de l'extension de ces fichiers (`.m`).

scripts et les *fonctions*.

5.1 Scripts

La première étape est de créer un répertoire (on en choisir un existant) où vous allez ranger les fichiers. En utilisant l'onglet `Current Directory`, placez-vous dans votre répertoire personnel (probablement sur le disque `H:\`). Créez ensuite un répertoire pour vos travaux MATLAB, par exemple `tps_matlab` (dans l'onglet `Current Directory`, bouton droit de la souris et `New` → `Folder`). Dans ce répertoire, vous pouvez, par exemple, créer un sous-répertoire pour le travail pratique en question : `tp1`. Créez ensuite un `m-file` (menu principal, `File` → `New` → `M-file`). Apparaît alors une nouvelle fenêtre ressemblant à un éditeur de texte, c'est le `M-file Editor`. Dans cette fenêtre, on rentre toutes les commandes qu'on désire que MATLAB exécute. Remarquez que, contrairement à la `Command Window`, les commandes ne s'exécutent pas directement. La touche [`%`] permet d'ajouter des commentaires.

```
%Mon premier m-file
A=ones(4)
v=[1 2 3 4]'
w=A*v
```

Quand vous avez fini de rentrer les commandes souhaitées, il faut enregistrer le fichier dans le répertoire `tp1` avec comme nom, par exemple, `test1.m`. Pour exécuter les commandes on rentre le nom du fichier (sans le `.m`) dans la `Command Window`.

```
>> test1
A =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
v =
     1
     2
     3
     4
w =
    10
    10
    10
    10
```

Pour la suite des travaux pratiques, on utilisera essentiellement le système des `m-files` pour rentrer des commandes.

Remarquons que pour exécuter un `m-file`, il faut « être dans le bon répertoire ». Autrement dit, vous devez voir vos fichiers dans l'onglet `Current Directory`. La prochaine fois que vous relancez MATLAB, n'oubliez pas de choisir de répertoire où sont rangés vos `m-files` avant d'essayer de les exécuter.

5.2 Fonctions

Une *fonction* est un `m-file` particulier : on lui passe des arguments et il retourne une valeur. Il faut simplement commencer le `m-file` par le mot `function`, lui préciser les valeurs de retour et les arguments comme dans l'exemple ci-dessous. Cet exemple prend comme paramètre un entier n et affiche les valeurs d'une matrice de taille $n \times n$. Il renvoie ensuite la valeur moyenne des éléments du tableau. Si on fait tendre n vers l'infini, vers quoi devrait tendre cette moyenne ?

```
function m = moyenne(n)
A=rand(n);
imagesc(A);
axis equal off;
m=mean(mean(A));
```

La forme générale de la déclaration d'une fonction est

`function` *nom_variable_retour* = `nom_fonction` (*nom_paramètres*)

Quelques remarques :

(*i*) le `m-file` et la fonction doivent avoir le même nom ;

(*ii*) il peut y avoir plusieurs paramètres (il suffit de les séparer par des virgules) ou aucun ;

(*iii*) il peut y avoir plusieurs variables de retour (les mettre entre crochets et les séparer par des virgules) ou aucune.

(*iv*) afin de vous souvenir de ce que vous avez programmé, il est recommandé de mettre un commentaire au début de votre `m-file`. Le `m-file` précédent ressemblera donc à

```
function m = moyenne(n)
%MOYENNE - calcul de la valeur moyenne d'une matrice
%  cette fonction calcule...

A=rand(n);
imagesc(A);
axis equal off;
m=mean(mean(A));
```

La première ligne est importante car elle est reconnue par la commande `lookfor`. C'est-à-dire, si vous entrez la commande `lookfor moyenne`, MATLAB renvoie `MOYENNE - calcul de la valeur moyenne d'une matrice`. De plus en tapant `help moyenne`, vous aurez tout le descriptif.

---

6 Programmation

En plus des commandes vues jusqu'à maintenant, MATLAB permet d'inclure dans des `m-files` des instructions de programmation classiques.

6.1 Conditions – `if ... else ... end`

La syntaxe est la suivante :

```
if (test)
    commandes
else
    autres commandes
end
```

On peut également imbriquer des `if ... else` les uns dans les autres à l'aide de l'instruction `elseif`.

```
if (test 1)
    commandes
elseif (test 2)
    commandes
elseif (test 3)
    ...
else
    commandes
end
```

Le `test` est une expression booléenne. En MATLAB la comparaison d'égalité se fait à l'aide de [==] et l'inégalité à l'aide de [∼=]. Le « faux » est représenté par 0 et le « vrai » par 1.

```
>> a=1;b=2;
>> a==b,a~=b
ans =
    0
ans =
    1
```

6.2 Répétitions – `for ... end`

La syntaxe est la suivante :

```
for k = liste
    commandes
end
```

On peut naturellement imbriquer des boucles `for ... end` les unes dans les autres. Attention à ne pas utiliser les variables [i] et [j] comme itérateurs car ces variables représentent le nombre complexe $\sqrt{-1}$.

```
%nombres de Fibonacci et nombre d'or
a=1;
fib=1;
for k = [1:10]
```

```
    tmp=a;
    a=fib;
    fib=fib+tmp
    or=fib/a
end
```

6.3 Répétitions – `while ... end`

La syntaxe est la suivante :

```
while test
    commandes
end
```

L'exemple ci-dessus affiche des matrices aléatoires en boucle. La commande `drawnow` force MATLAB à dessiner à chaque boucle et non une seule fois à la fin.

```
k=0;
figure;
while k < 50
    k = k+1;
    A=rand(10,10);
    imagesc(A);
    axis off equal;
    drawnow;
end
```

6.4 La récursivité

Dans la définition d'une fonction f, il arrive que l'on désire rappeler la même fonction f pour des valeurs différentes. L'exemple typique étant le cas de la définition de la fonction factorielle sur les entiers.

$$\text{factoriel}(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * \text{factoriel}(n-1) & \text{si } n > 0 \end{cases}$$

Ceci est possible en MatLab, comme dans bien des langages de programmation, mais il faut être attentif au fait que cet outil amène souvent des boucles sans fin, si la condition d'arrêt n'est pas bien pensée. Voilà l'exemple de la factorielle programmée en MatLab.

```
function nn =factoriel(n)
%calcule le n factoriel en fonction de n.
reponse=1;
if n==0
   nn=1;
else
   reponse= n*factoriel(n-1);
end; %if
nn=reponse;
```

7 Exercices

1 Opérations sur les vecteurs. Calculer, quand c'est possible, les opérations suivantes. Utiliser uniquement les opérations de base (pas de `for`, ni de `while`). Quand vous le pouvez, interpréter géométriquement ou algébriquement le résultat.

$$q = \begin{pmatrix}1\\2\end{pmatrix}, \quad r = \begin{pmatrix}3\\-6\end{pmatrix}, \quad s = \begin{pmatrix}5\\6\\4\end{pmatrix}, \quad t = \begin{pmatrix}1\\1\\1\end{pmatrix}, \quad u = \begin{pmatrix}3\\4\\2\end{pmatrix}, \quad v = \begin{pmatrix}-1\\\vdots\\-9\end{pmatrix}, \quad w = \begin{pmatrix}1\\\vdots\\9\end{pmatrix};$$

Effectuez

(*i*) $q + r$; $q + 3v$; $s - 2t - u$;

(*ii*) $\langle q|r\rangle$; $\langle s|s\rangle - 2\langle s|t\rangle - \langle s|u\rangle$; $\langle s|v\rangle$, $\langle v|w\rangle$; où $\langle q|r\rangle$ représente le produit scalaire entre q et r.

(*iii*) $\|v\|$, $\|s\|^2$.

2 Différence entre `A*B` et `A.*B`. Trouver deux matrices inversibles (3×3) `A` et `B` telles que `A.*B` $= 0$. Est-ce possible pour le produit matriciel standard ?

3 Soit A une matrice carrée. En une ligne, construire une matrice diagonale B ayant la même diagonale que A. Autrement dit, quelles sont les commandes Matlab permettant de passer de $\begin{pmatrix} a_1 & * & * \\ * & \ddots & * \\ * & * & a_n \end{pmatrix}$ à $\begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix}$.

4 Considérons les deux matrices $A = \begin{pmatrix} 2 & -1 \\ 1 & 3 \\ 3 & 0 \end{pmatrix}$ et $B = \begin{pmatrix} 2 & 5 & 1 \\ 0 & 0 & -3 \end{pmatrix}$ et le vecteur $v = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$. Désignons par C le produit matriciel $C = AB$. Calculer les vecteurs suivants :

(*i*) $x = Bv$;

(*ii*) $y = Ax$;

(*iii*) $z = Cv$.

Que constate-t-on ? Pourquoi ?

5 Résolution matricielle de systèmes linéaires. Résoudre matriciellement les systèmes suivants. Si Matlab affiche un message d'erreur, dire pourquoi.

(*i*)

$$\begin{array}{rcrcrcl} 6x & + & y & - & 5z & = & 10 \\ 2x & + & 2y & + & 3z & = & 11 \\ 4x & - & 9y & + & 7z & = & 12 \end{array}$$

(*ii*)

$$\begin{array}{rcrcrcl} 6x & + & y & - & 5z & = & 10 \\ 2x & + & 2y & + & 3z & = & 2 \\ 8x & + & 3y & - & 2z & = & 12 \end{array}$$

(*iii*)

$$\begin{array}{rcrcrcrcl} x & + & 2y & + & 3z & + & 4t & = & 1 \\ 2x & + & 3y & + & 4z & + & t & = & -2 \\ -2x & + & 4y & - & 5z & + & 2t & = & 0 \\ 8x & + & y & - & z & + & 3t & = & 1 \end{array}$$

6 FIGURES GÉOMÉTRIQUES. Au moyen de la fonction `plot`, afficher les figures géométriques suivantes :

(*i*) un carré ;

(*ii*) un pentagone régulier ;

(*iii*) un heptagone régulier ;

(*iv*) un « cercle ».

7 GRAPHE DE FONCTIONS. Tracer le graphe des fonctions suivantes (faire en sorte qu'il y ait la même échelle sur les deux axes) :

(*i*) $x \mapsto x^2 \sin(x)$ $x \in [0, 2\pi]$;

(*ii*) $x \mapsto \sqrt{x^2}$ $x \in [-2, 2]$;

(*iii*) $x \mapsto \sqrt[n]{nx}$ $x \in [0, 1]$ et $n = 1, \ldots, 5$ (tous sur la même figure).

8 SURFACES. Afficher la surface donnée par l'équation $z = \sin(x)\cos(y)$ pour $x \in [0, 1]$ et $y \in [2, 5]$.

9 GRAM-SCHMIDT. Soit $\mathcal{B} = \{b_1, \ldots, b_n\}$ une base de $\mathbb{R}^n$. écrire une fonction qui prend comme paramètre $\mathcal{B}$ et retourne la base $\mathcal{B}'$ obtenue par le procédé d'orthonormalisation de Gram-Schmidt.

10 LE DÉTERMINANT.

(*i*) Programmer une fonction récursive qui calcule le déterminant par la méthode des mineurs. Attention ne testez pas cette fonction sur des matrices trop grandes.

(*ii*) Comparez le temps de calcul de cette fonction sur des matrices de tailles raisonnables (pas plus grandes que 10x10) avec celui utilisé par la fonction "det" implantée dans Matlab.

(*iii*) La décomposition LU est une méthode permettant d'écrire une matrice A comme un produit d'une matrice triangulaire supérieure U et d'une matrice triangulaire inférieure L (i.e. $A = LU$). Cet algorithme est assez similaire à une décomposition de Gauss avec pivot. Il s'agira pour vous de comprendre le code de la fonction decompLUincomplete.m et de le compléter pour obtenir une telle décomposition. A l'aide de cette décomposition, écrivez une autre fonction calculant le déterminant d'une matrice A plus rapidement que votre première fonction

11 Écrire une fonction qui prend en paramètre un entier n et qui construit la matrice ci-dessous. Ne pas utiliser de boucles `for`, ni `while`.

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{n} & 0 & 4 & \cdots & 0 & 0 & 0 \\ 0 & \frac{2}{n} & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & (n-1)^2 & 0 \\ 0 & 0 & 0 & \cdots & \frac{n-1}{n} & 0 & n^2 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix}$$

Pour $n = 1$, cela donne $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ et pour $n = 2$, c'est la matrice $\left(\begin{smallmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & 4 \\ 0 & 1 & 0 \end{smallmatrix}\right)$.

12 Changement de base. Soit $\mathcal{E} = \{e_1, \ldots, e_n\}$ la base canonique de $\mathbb{R}^n$. Soit M la matrice associé à un certain endomorphisme ϕ de $\mathbb{R}^n$ écrite dans la base $\mathcal{E}$, c'est-à-dire $M := M_{\mathcal{E}}^{\mathcal{E}}\phi$.

Soit $\mathcal{F} = \{f_1, \ldots, f_n\}$ une autre base de $\mathbb{R}^n$. écrire une fonction qui prend comme paramètre M et $\mathcal{F}$ et retourne $\tilde{M}$, la matrice associée à ϕ dans la base $\mathcal{F}$, c'est-à-dire $\tilde{M} := M_{\mathcal{F}}^{\mathcal{F}}\phi$.

13 Applications linéaires entre espaces de polynômes. Désignons par $\mathfrak{P}_n$ l'espace vectoriel des fonctions polynomiales de $\mathbb{R}$ dans $\mathbb{R}$ de degré $\leq n$.

(*i*) Pour $p, q \in \mathfrak{P}_n$ deux applications polynomiales, nous désignons par $r(p, q)$ le reste de la division euclidienne de p par q. Considérons l'application linéaire

$$\begin{aligned} R : \mathfrak{P}_n &\longrightarrow \mathfrak{P}_n \\ p &\longmapsto r\big(p, x^2\big). \end{aligned}$$

écrire une fonction qui prend comme paramètre un entier n et renvoie la matrice M_R de l'application R dans la base $\{1, x, x^2, \ldots, x^n\}$. À l'aide de cette matrice, calculer le reste de division par x^2 pour les polynômes suivants :

(a) $7x^8 + 411x^7 - 231x^5 + 31x^4 + 451x^3 - 231x - 42$;

(b) $x^7 + \frac{5}{21}x^5 + 0.432x^4 - 22x^3 + 51x^2 - \frac{1}{39}x + 4.431$.

Calculer pour $n = 6, 7, 8$ le noyau de R. Que constate-t-on ?

(*ii*) Considérons l'application linéaire *dérivée* :

$$\begin{aligned} d : \mathfrak{P}_n &\longrightarrow \mathfrak{P}_n \\ \sum_{k=0}^{n} a_k x^k &\longmapsto \sum_{k=1}^{n} k a_k x^{k-1}. \end{aligned}$$

Refaire le point (*i*) avec l'application d à la place de R et déterminer les valeurs propres et les vecteurs propres associés de l'application d.

Question théorique : Existe-t-il des fonctions propres pour l'opérateur dérivée autre que 0 et si oui lesquelles ?

(*iii*) Considérons encore les deux compositions $d \circ R$ et $R \circ d$. Que font ces applications ? Sont-elles les mêmes ? Quel est le lien avec $M_d M_R$ ou $M_R M_d$?

(*iv*) Écrire une procédure prenant comme paramètre deux polynômes p et d et retournant comme résultat les deux polynômes q et r ; q étant le quotient de la division polynomiale de p par d et r en étant le reste.
