Université de Genève
Laboratoire de programmation

Semestre de printemps 2020

Travaux pratiques 1 : Introduction à Matlab

1 Introduction

Matlab pour « MATtrix LABoratory », est un logiciel 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++`, le `fortran` ou autre, est mieux adapté.

Dans la première partie de cet énoncé, on va présenter les commandes Matlab les plus courantes. Les exercices se trouvent dans la dernière section.

1.1 Guide pour vos premiers pas dans Matlab

Ces travaux pratiques ne sont malheureusement pas accompagnés d'un cours, ce qui implique de votre part un certain degré d'auto-apprentissage. Pour débuter, il est fortement conseillé de procéder de la manière suivante :

1. Lisez la Sections 2 en testant les commandes présentées pour vous familiariser avec les bases du programme et du language.
2. Jetez un coup d'oeil à la Section 3, qui explique comment créer et manipuler les vecteurs et les matrices.
3. Lisez la Section 5, présentant les m-files. La solutions de chaque exercice doit être rédigée sous la forme d'un m-file.
4. Etudiez attentivement l'exemple d'exercice résolu de la Section 7. Il est exigé que vous commentiez le code de vos solutions en suivant cet exemple.
5. Après cette préparation, vous pouvez commencer à résoudre les exercices, en vous référant au guide ci-dessous et à l'aide de Matlab quand nécessaire. Les exercices sont classés approximativement par degré de difficulté, il est donc conseillé de tenter de les résoudre dans l'ordre dans lequel ils sont présentés.

2 Bases

2.1 Lancement de Matlab

L'interface Matlab se compose d'une fenêtre principale divisée en plusieurs sous-fenêtres, dont les plus importantes sont :

1. La fenêtre `Workspace` permet de gérer les variables utilisées. Nous y reviendrons au paragraphe 2.4.

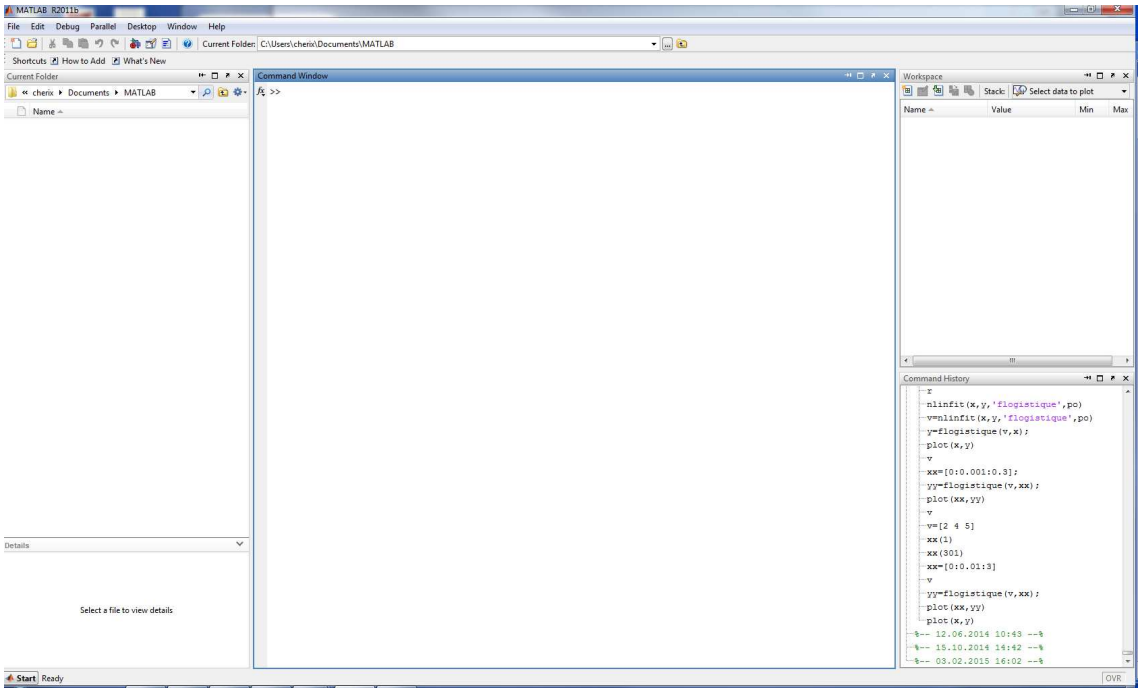

FIGURE 1 – L'espace de travail de MATLAB.

2. Le `Current Folder` gère l'emplacement des fichiers. Celui-ci sera utile pour le travail avec les `m-files` . Nous y reviendrons au paragraphe 5.1.
3. La fenêtre `Command History` n'est plus visible par défaut dans la version 2018b ; elle indique les dernières commandes effectuées. Pour l'afficher, cliquez sur l'icône représentant des fenêtres en haut à côté de la barre de recherche.
4. La plus grande fenêtre, `Command Window`, est la fenêtre d'interaction avec MATLAB.

2.2 Commandes et calculs de base

MATLAB est un interpréteur. Cela veut dire que l'utilisateur rentre des commandes et MATLAB les exécute ligne par ligne. [1]

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

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 [>>]. MATLAB fournit le résultat de la commande sous la ligne contenant `ans =`.

Toute suite de caractères suivant le symbole [%] est ignorée par MATLAB, ce qui est pratique pour commenter le code :

1. Chaque fois que des exemples de commandes comme le suivant apparaissent dans ce guide, il vous est vivement conseillé d'entrer la commande correspondante sur votre ordinateur, afin de vérifier par vous-même le comportement de MATLAB. Dans le cas présent, entrez `2+2` dans la `Command Window` tapez Enter et voyez ce que MATLAB retourne.

```
>> 2+2     % Un exemple d'addition
ans =
     4
```

Il est extrêmement important de commenter votre code. Voir les sections 5 et 7 pour plus d'informations.

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. Pour taper une autre commande, on le fait à la suite.

```
>> 2+2    % Une première commande
ans =
    4

>> 3*5    % Une deuxième commande
ans =
    15
```

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

```
    >> 3*     % Une commande erronée
 3*
   |
Error: Expression or statement is incomplete or incorrect.
```

Notez la flèche (ici une barre verticale) indiquant l'endroit où le problème est apparu. Un autre exemple est :

```
>> aa       % Une autre commande erronée
Undefined function or variable 'aa'.
```

MATLAB ne comprend pas ce que l'on entend par `aa` et retourne une erreur.

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

```
>> pi    % Le rapport de la circonférence d'un cercle à son diamètre.
ans =
    3.1416

>> sin(pi/6)  % Un exemple d'évaluation d'une fonction
ans =
    0.5000

>> log(1.5)  % Un second exemple
ans =
    0.4055

>> i^2     % i est l'unité imaginaire
ans =
    -1
```

2.3 Variables

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 [=].

```
>> A=23   % On assigne la valeur 23 à la variable A.
A =
    23
```

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. En entrant le nom d'une variable, MATLAB retourne sa valeur.

```
>> A    % On demande la valeur enregistrée dans la variable A.
A =
    23
```

La valeur d'une variable peut être un nombre, comme dans l'exemple ci-dessus, une chaîne de caractères (inclue entre des guillement simples ['])

```
>> A='salut !'  % On enregistre une chaîne de caractères dans la variable A.
A =
    salut !
```

ou une matrice (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 une matrice, comme dans les exemples ci-dessus.

Précisons que MATLAB est « case-sensitive », c'est-à-dire qu'il fait la distinction entre majuscules et minuscules.

```
>> A=42     % On assigne la valeur 42 à la variable A.
A =
    42

>> a=2.432  % On assigne la valeur 2.432 à la variable a.
a =
    2.4320

>> A        % Le second assignement n'a pas changé la valeur de A.
A =
    42
```

Les variables `a` et `A` sont donc bien distinctes.

On peut évidement faire des calculs avec des variables. Le résultat d'un calcul est, par défaut, stocké dans une variable nommée `ans`, ce qui explique pourquoi MATLAB retourne `ans = 4` lorsque l'on entre `2+2`. On peut toutefois stocker le résultat dans n'importe quelle autre variable :

```
>> A = 2 + 2  % On enregistre le résultat d'une addition dans la variable A.
A =
    4
```

Par défaut, MATLAB affiche le résultat de la dernière opération. Cet affichage peut être supprimé en terminant votre commande par le symbole [;]. Plusieurs commandes peuvent être rentrées sur une même ligne en les séparant soit par [,] soit par [;].

```
>> x=2;y=5;    % On effectue plusieurs commandes sur la même ligne.
               % Matlab n'affiche pas le résultat à cause des
               % caractères ; à la fin de chaque commande.
>> z=x^2+y^2   % On effectue un calcul avec les variables enregistrées
               % à la première ligne.
z =
    29
```

Pour une liste plus détaillée des opérations mathématiques que l'on peut faire dans MATLAB voir le Paragraphe 3.3.

2.4 Gestion des variables

La fenêtre `Workspace`, sur la droite ou la gauche de l'espace de travail de MATLAB, fournit une liste de toutes les variables déjà utilisées avec leur type et leur valeur. Pour les vecteurs et le matrices, on peut voir la valeur de leurs éléments de matrice en double-cliquant sur la variable. On peut également modifier la/les valeurs d'une variable de cette façon. Un clic droit sur une variable ouvre un menu permettant de dupliquer ou de supprimer une variable.

Dans la `Command Window`, on peut également visionner les variables utilisées et obtenir des informations détaillées au moyen des commandes `who` et `whos` respectivement. On peut effacer une variable en utilisant la commande `clear` suivie du nom de la variable. Pour tout effacer, `clear all`.

2.5 Historique des commandes

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

2.6 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` dans la `Command Window`. En plus d'explications, vous obtenez un lien vers la page de l'aide de MATLAB correspondant à cette fonction, où vous trouverez des informations plus détaillées et des exemples.

L'aide de MATLAB peut être atteinte directement en cliquant sur le point d'interrogation en haut à droite de la fenêtre de MATLAB.

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 ne trouvez pas la fonction dont vous avez besoin, ou si vous aimeriez des exemples supplémentaires concernant l'utilisation d'une fonction, une recherche sur Google ou un autre moteur de recherche s'avère souvent fructueuse.

2.7 Sauvegarde

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

1. 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.
2. 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.

> **Attention ! ! !**
>
> Sauvegardez vos fichiers sur votre espace mémoire étudiant ou sur un support personnel (clef USB par exemple).
> Ne sauvegardez *pas* votre travail sur le disque local de l'ordinateur : il sera perdu au prochain redémarrage.

3 Vecteurs et matrices

La structure de données de base de MATLAB est la *matrice*[2] ; même un nombre est considéré comme une matrice 1×1. Toutes les fonctions et opérations relatives aux matrices sont très optimisées et sont à utiliser aussi souvent que possible.

3.1 Création

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

```
>> A=[1 1 1 ; 2 2 2]   % Une matrice de taille 3 x 2.
A =
    1     1     1
    2     2     2

>> B=[1 ; 2 ; 3]       % Un vecteur colonne de taille 3.
B =
    1
```

2. Les matrices sont parfois également appelées *tableaux*. En anglais, les termes respectifs sont *matrix* et *array*.

```
    2
    3

>> C=[1.1 2.2 3.3]     % Un vecteur ligne de taille 3.
C =
    1.1000    2.2000    3.3000
```

Les matrices qui n'ont qu'une seule ligne sont appelées des *vecteurs lignes* ou des *listes* ; celles qui n'ont qu'une seule colonne sont appelées 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]    % Cette matrice a 3 éléments dans la première ligne
                     % et deux dans la seconde -> erreur.
Dimensions of matrices being concatenated are not consistent.
```

MATLAB propose des commandes pour créer certaines matrices particulières simplement. Pour plus d'informations, lire l'aide de chaque fonction (`help ...`).

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, et dont les éléments hors diagonale sont nuls.
`rand(n,m)`	Matrice de taille $n \times m$ contenant des nombres aléatoires uniformément répartis entre 0 et 1.

TABLE 1 – Commandes pour créer des matrices.

MATLAB dispose également de moyens très simples pour créer des listes. La commande `[a:h:b]` [3] crée une liste dont les éléments sont

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

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

```
>> x=[1:2:10]   % Un vecteur dont les éléments vont de 1 à 9, par
                % incréments de 2.
```

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

```
x =
    1     3     5     7     9

>> y=[-5:0]     % Un vecteur dont les éléments vont de -5 à 0, par
                % incréments de 1.
y =
    -5    -4    -3    -2    -1     0

>> z=[10:2:-10] % Un vecteur dont les éléments vont de 10 à -10, par
                % incréments de 2. -> vecteur vide
z =
    Empty matrix: 1-by-0
```

Une autre formulation 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 équivalente à `[a:c:b]` où $c = \frac{b-a}{n-1}$.

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 matrice peut aussi être créée par concaténation. Si `A` et `B` sont deux matrices, alors `[A B]` ou `[A,B]` est la matrice obtenue en collant `B` à la droite de `A`. `[A;B]` est la matrice obtenue en collant `B` au-dessous de `A`. Les tailles de `A` et de `B` doivent être compatibles.

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

>> [A,B],[A;B]  % Concaténations horizontale et verticale de A et B
ans =
    1     3     5     2     4     1
ans =
    1     3     5
    2     4     1

>> [A,C]    % Impossible de concaténer horizontalement des matrices
            % n'ayant le même nombre de lignes.
Error using horzcat
Dimensions of matrices being concatenated are not consistent.
```

3.2 Accès et modifications

On présente dans cette section diverses méthodes pour accéder et modifier les éléments d'une matrice. Dans la table qui suit, `A` désigne une matrice de taille $n \times m$, $1 \leq$ `k` $\leq n$ et $1 \leq$

`l` $\leq m$ sont des nombres entiers et `M` une matrice d'entiers positifs inférieurs ou égaux à nm.

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(M)`	Renvoie une matrice de même taille que `M` dont chaque élément `A(k)` a pour valeur `A(M(k))`. (Le $k^{\text{ème}}$ élément de `M` est donc un indice, déterminant un élément de la matrice `A` à placer en position k dans `A(M)`.)
`A(k,:)`	Renvoie la $k^{\text{ème}}$ ligne de la matrice.
`A(:,l)`	Renvoie la $l^{\text{ème}}$ colonne de la matrice.
`A(k:l,p:q)`	Renvoie sous-matrice de `A` composée de l'intersection des lignes k à l (inclue) et des colonnes p à q (inclue).

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

```
>> A=[1 2 3 4; 12 13 14 15]   % Une matrice 2 x 4.
A =
     1     2     3     4
    12    13    14    15

>> A(2,3)        % L'élément de matrice de A en position (2,3)
ans =
    14

>> A(2,:)       % La seconde ligne de A
ans =
    12    13    14    15

>> A([1 3 5 7]) % Un vecteur ligne contenant les éléments no.1, 3, 5, 7.
ans =
    1     2     3     4

>> A(:,4)    % La quatrième colonne de A.
ans =
     4
    15
```

```
>> A([1 1 1])  % Un vecteur ligne contenant l'élément no.1 trois fois.
ans =
    1     1     1

>> A([1 3; 4 8])  % Une matrice contenant les éléments no.1,3,4,8.

ans =
     1     2
    13    15

>> A(1:2, 1:3)  % Une sous-matrice donnée par l'intersection des lignes
                % 1 et 2 et des colonnes 1, 2 et 3. Equivalent à A(:,1:3).

ans =
     1     2     3
    12    13    14
```

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   % Notre matrice
A =
     1     2     3     4
    12    13    14    15

>> A(2,2)=999   % On assigne la valeur 999 à l'élément de matrice en
                % position (2,2).
A =
     1     2     3     4
    12   999    14    15

>> A([2 3 5]) = [-1 -1 -1]  % On assigne la valeur -1 aux éléments
                            % no.2,3,5.
A =
     1    -1    -1     4
    -1   999    14    15

>> A(:,4)=[101 103] % Change la valeur des éléments de matrice de la
                    % colonne 4.
A =
     1    -1    -1   101
    -1   999    14   103
```

Remarquons cependant que dans ce cas, on est autorisés 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]  % Une matrice 2 x 2
```

```
A =
    1     2
    3     4

>> A(2,5) = 34   % On assigne la valeur 34 à l'élément de matrice en
                 % position (2,5). A est convertie en une matrice de
                 % taille 2 x 5.

A =
    1     2     0     0     0
    3     4     0     0    34
```

3.3 Opérations avec les matrices

Opérations de bases. Dans ce qui suit, `A` et `B` sont des matrices et `c` est un nombre. On note les éléments de matrices de `A` et de `B` par (a_{ij}) et (b_{ij}).

Commande	**Description**
`A+B`	Matrice dont l'élément (i,j) est $a_{ij}+b_{ij}$. `A` et `B` doivent avoir les mêmes dimensions.
`A+c = c+A`	Matrice dont l'élément (i,j) est $a_{ij}+c$.
`A-B`	Matrice dont l'élément (i,j) est $a_{ij}-b_{ij}$. `A` et `B` doivent avoir les mêmes dimensions.
`A-c`	Matrice dont l'élément (i,j) est $a_{ij}-c$.
`c-A`	Matrice dont l'élément (i,j) est $c-a_{ij}$.
`A*B`	Matrice dont l'élément (i,j) est $\sum_j a_{ik}b_{kj}$. C'est le produit matriciel standard. Le nombre de colonnes de `A` doit être le même que le nombre de lignes de `B`.
`c*A` (ou `A*c`)	Matrice dont l'élément (i,j) est ca_{ij}. C'est la multiplication d'une matrice par un scalaire.
`A.*B`	Matrice dont l'élément (i,j) est $a_{ij}b_{ij}$. C'est la multiplication élément par élément, peu utilisée en algèbre linéaire, mais souvent pratique. `A` et `B` doivent avoir les mêmes dimensions.
`A^n` ($n \in \mathbb{Z}_+$)	`A * A * ... * A` (n fois) ; `A` doit être carrée.

`A^(-1)`	Matrice inverse, telle que `A * A^(-1) = A^(-1) * A` est la matrice identité. `A` doit être inversible.
`A^(-n)` ($n \in \mathbb{Z}_+$)	`A^(-1) * A^(-1) * ... * A^(-1)` (n fois) ; `A` doit être inversible.
`A.^B`	Matrice dont l'élément (i, j) est $a_{ij}^{b_{ij}}$. `A` et `B` doivent avoir les mêmes dimensions.
`A'`	Transposition et conjugaison complexe.
`A.'`	Transposition. `A.' = A'` dans le cas où `A` est réelle.
`B/A`	Matrice X telle que $XA = B$. Si A est inversible, alors $X = BA^{-1}$. Le nombre de colonnes de `A` doit être le même que le nombre de colonnes de `B`.
`A\B`	Matrice X telle que $AX = B$. Si A est inversible, alors $X = A^{-1}B$. Le nombre de lignes de `A` doit être le même que le nombre de lignes de `B`.
`A./B`	Matrice dont l'élément (i, j) est a_{ij}/b_{ij}. `A` et `B` doivent avoir les mêmes dimensions.
`A.\B`	Matrice dont l'élément (i, j) est b_{ij}/a_{ij}. `A` et `B` doivent avoir les mêmes dimensions.
`A/c`	Matrice dont l'élément (i, j) est a_{ij}/c.

TABLE 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`, un symbole dénotant une quantité infinie. Attention néanmoins à ne pas travailler avec `Inf` comme avec un nombre.

```
>> A=[1 2 3; 0 0 1; 1 0 0]    % Une matrice 3 x 3
A =
    1     2     3
    0     0     1
    1     0     0

>> B=[-1 -2 -3; 0 0 -1; -1 0 0]   % Une autre matrice 3 x 3
B =
    -1    -2    -3
     0     0    -1
    -1     0     0

>> A+B       % La somme des matrices A et B.
```

```
ans =
     0     0     0
     0     0     0
     0     0     0

>> A*B       % Le produit matriciel de A et B.
ans =
    -4    -2    -5
    -1     0     0
    -1    -2    -3

>> A^(-2)    % Le carré de l'inverse de A.
ans =
         0    1.0000         0
   -0.7500    1.7500    1.2500
    0.5000   -1.5000   -0.5000

>> A.*B      % Le produit élément par élément de A et B.
ans =
    -1    -4    -9
     0     0    -1
    -1     0     0

>> A/B       % Une matrice ans satisfaisant ans*B = A.
ans =
    -1     0     0
     0    -1     0
     0     0    -1
```

Important. Pour la résolution de systèmes d'équations `AX = B` ou `XA = B`, utilisez toujours les commandes `A\B` ou `B/A`. Il n'est *pas* nécessaire d'inverser la matrice `A` pour résoudre le système. L'inversion de A est coûteuse en temps de calcul et moins précise que la résolution directe du système.

Fonctions sur les matrices. Nous présentons ici quelques fonctions définies dans MATLAB prenant comme paramètres des matrices. Pour plus d'information, tapez `help` suivi du nom de la fonction. Dans le tableau qui suit, `A` est une matrice et `v` est un vecteur.

Commande	**Description**
`det(A)`	Déterminant de `A` ; la matrice `A` doit être carrée.
`trace(A)`	Trace de `A`. Si `A` n'est pas carrée, la trace est définie comme la somme des éléments dans la diagonale principale, commençant au coin supérieur gauche.

`rank(A)`	Rang de `A` (la dimension de l'image de l'application associée à `A`).
`null(A)`	Base du noyau de `A`. L'argument supplémentaire `'r'` donne une « meilleure » base (voir `help null`).
`diag(A)`	Diagonale principale de `A`.
`norm(v)`	Norme euclidienne de `v`. Il est aussi possible de calculer d'autres normes, voir `help norm`.
`mean(A)`	Liste (vecteur ligne) contenant la moyenne des éléments de chaque colonne.
`sum(A)`	Liste contenant la somme des éléments de chaque colonne.
`prod(A)`	Liste contenant le produit des éléments de chaque colonne.
`max(A)`	Liste contenant la valeur maximale de chaque colonne.
`min(A)`	Liste contenant la valeur minimale de chaque colonne.
`length(v)`	Nombre d'éléments du vecteur `v`. Appliqué à une matrice `A`, `length(A)` retourne le maximum entre le nombre de lignes et de colonnes.
`eig(A)`	Liste des valeurs propres de A.
`[M,D]=eig(A)`	Enregistre sous la forme de variables une matrice diagonale `D` contenant les valeurs propres de `A` et une matrice `M` contenant comme colonnes les vecteurs propres de `A`.

TABLE 4 – Fonctions sur des matrices.

Toutes les fonctions mathématiques classiques (`cos`, `sin`, `log`, `exp`, etc...) peuvent également être appliquées à une matrice `A`. Le résultat est la matrice obtenue de `A` en appliquant la fonction séparément à chaqun des éléments de matrice de `A`. Ceci est pratique pour calculer rapidement une fonction sur un ensemble de valeurs.

```
>> I=[0:0.2:1]  % Un vecteur dont les éléments vont de 0 à 1 par pas de
                % 0.2.
I =
    0    0.2000    0.4000    0.6000    0.8000    1.0000

>> exp(I)       % Un vecteur dont les éléments sont les exponetielles des
                % éléments de I.
```

```
ans =
    1.0000    1.2214    1.4918    1.8221    2.2255    2.7183
```

4 Graphisme

4.1 Courbes dans le plan

Courbe Étant donnés deux vecteurs de même taille, `x` et `y`, la fonction `plot(x,y)` dessine les points de coordonnée `(x(k),y(k))` pour $1 \leq$ `k` $\leq$ `length(x)`.. Par défaut, MATLAB relie les points par des segments de droite.

```
>> x = [-1 -.5 -1 0 1 .5 1 0 -1];  % Un vecteur de coordonnées horizontales.
>> y = [-1 0 1 .5 1 0 -1 -.5 -1];  % Un vecteur de coordonnées verticales.
>> plot(x,y)                       % On dessine les neuf points du plan de
                                   % coordonnées (x,y), reliés par des
                                   % segments de droite.
```

La figure dessinée apparaît dans une nouvelle fenêtre.

Graphe d'une fonction En prenant un grand nombre de points régulièrements espacés dans le vecteur `x` et en définissant ensuite `y = f(x)` pour une fonction `f`, la fonction `plot(x,y)` dessine le graphe de la fonction `f`.

```
>> x=[0:0.01:4*pi];   % Un vecteur dont les éléments vont de 0 à 4*pi,
                      % par pas de 0.01.
>> y=cos(x);          % Un vecteur dont les éléments sont les cosinus des
                      % éléments de x.
>> plot(x,y)          % On dessine les points de coordonnées (x,y), reliés
                      % par des segments de droite.
```

Plusieurs courbes La suite de commandes

```
>> z=sin(x);           % Un vecteur dont les éléments sont les sinus des
                       % éléments de x.
>> plot(x,y)           % Dessine les points de coordonnées (x,y).
>> plot(x,z)           % Dessine les points de coordonnées (x,z).
```

ne trace pas deux graphes, mais un seul. En fait, le deuxième `plot(x,z)` vient effacer et 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)   % Comme ci-dessus
>> figure      % Demande à Matlab de dessiner sur une nouvelle fenêtre.
>> plot(x,z)   % Dessine les points de coordonnées (x,z) dans la nouvelle
               % fenêtre.
```

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  % Après hold on, Matlab dessine
                                         % dans la même fenêtre, sans
                                         % effacer les dessins précédents.
                                         % La commande hold off annule la
                                         % commande hold on pour les
                                         % prochains dessins.
```

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

```
>> plot(x,y,x,z)  % Dessine les points de coordonnées (x,y), puis les
                  % points de coordonnées (x,z), dans la même fenêtre.
```

Options d'affichage On présente rapidement quelques options de la commande `plot`. Ce n'est pas une liste exhaustive, voir `help plot` pour plus de détails.

`axis equal` met les deux axes à la même échelle. `axis off` supprime les axes. Ces deux commandes peuvent être combinées comme ci-dessous.

```
>> plot(x,y), axis equal      % Dessine les points de coordonnées (x,y)
                              % sur une figure avec des axes à la même
                              % échelle.
>> plot(x,y), axis equal off  % La même chose, en ne dessinant pas les
                              % axes.
```

Les couleurs et le style du tracé peuvent également être modifiés en passant à `plot` comme argument une chaîne de caractères.

```
>> x=[0.5:0.1:5];y=log(x);             % Un vecteur de coordonnées et
                                       % leur logarithme.
>> plot(x,y,'co'), axis equal          % Dessine les points de coordonée
                                       % (x,y) en cyan ('c'), sous la
                                       % forme de cercles ('o').
>> x=[0:0.05:1];y=exp(x);z=log(y);     % Comme ci-dessus.
>> plot(x,y,'mo',y,z,'g>'),axis equal  % Dessins les points (x,y) sous la
                                       % forme de cercles en magenta ('m')
                                       % et les points (x,z) en vert ('g').
```

Voir `help plot` pour toutes les possibilités.

4.2 Affichage de surfaces

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);  % Crée une grille (voir ci-dessous)
>> z = x.^2 - y.^2;                   % Calcule une fonction z = x^2 - y^2
>> surf(x,y,z)                        % Dessine la surface correspondante.
```

La fonction `meshgrid(-3:0.1:3,0:0.2:5)` crée deux matrices `x` et `y` de taille 26 x 61, que l'on peut voir comme les coordonnées (x, y) de points d'une grille de 26 x 61 points. Les coordonnées x vont de -3 à 3 par pas de 0.1 et les coordonnées y vont de 0 à 5 par pas de 0.2. Pour plus d'explications, voir `help meshgrid`.

On calcule alors la fonction z sur chaque point de la grille à partir de ses coordonnées, puis on dessine la surface au moyen de `surf`.

4.3 Affichage de matrices

Étant donnée une matrice A de taille $n \times m$, MATLAB propose une méthode, `imagesc`, pour visualiser le contenu de A. 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)           % Crée une matrice 11 x 11 remplie de 1.
>> a([1:2:121])=0          % Change les éléments de numéro pair à 0.
>> imagesc(a), axis equal  % Visualise a.


>> x=rand(10,10);        % Matrice aléatoire de taille 10 x 10
>> y=rand(100,100);      % Matrice aléatoire de taille 100 x 100
>> z=rand(1000,1000);    % Matrice aléatoire de taille 1000 x 1000
>> imagesc(x),axis off   % Visualise x
>> figure                % Nouvelle fenêtre
>> imagesc(y),axis off   % Visualise y
>> figure                % Nouvelle fenêtre
>> imagesc(z),axis off   % Visualise z
```

5 Travail avec les m-files

Pour l'élaboration de programmes complexes, il n'est pas pratique de taper les commandes une à une dans la `Command Window`. Nous allons voir comment regrouper des commandes dans un fichier appelé un *m-file*[4]. Il y a deux types de m-files, les *scripts* et les *fonctions*. On utilisera dans ces TPs seulement les scripts, combinés avec la possibilité offerte par la version 2016 de MATLAB d'inclure les fonctions dans les fichiers scripts.

5.1 Scripts

La première étape est de créer un répertoire (ou en choisir un existant) où les fichiers seront stockés. En utilisant l'onglet `Current Folder`, 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 Folder`, 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 script (onglet `Editor`, `New` → `Script`). Apparaît alors une nouvelle fenêtre ressemblant à un éditeur de texte, c'est l'`Editor`. Le script est édité dans cette fenêtre.

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

Un script est un ensemble de commande que Matlab exécutera séquentiellement, lorsque le script sera lancé. Comme d'habitude, le symbole [%] permet d'ajouter des commentaires.

```
% Mon premier m-file
A=ones(4)      % Une matrice 4 x 4 remplie de 1.
v=[1 2 3 4]'   % Un vecteur colonne
w=A*v          % Transformation du vecteur par la matrice.
```

Remarquez que, contrairement à la `Command Window`, les commandes ne s'exécutent pas directement. Quand vous avez fini de rentrer les commandes souhaitées, enregistrez le fichier dans le répertoire `tp1` avec comme nom, par exemple, `test1.m`. Pour exécuter les commandes entrées dans le m-file, 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, la console peut-être utilisée pour tester des commandes, mais les solutions des exercices devront être rédigée dans des `m-files`. Lors des tests, vous devrez rendre un `m-files` par exercice.

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 Folder`. 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

Les fonctions sont traditionellement implémentées dans leur propre m-file, dont le nom doit coincider avec le nom de la fonction. Pour des raison pratiques et pour faciliter la correction, il vous est cependant demandé d'inclure vos fonctions dans le script de l'exercice correspondant (voir ci-dessous) lors des tests. Ceci est possible dans les versions 2016 et supérieures de MATLAB ("local functions").

Une *fonction* reçoit des arguments et retourne une valeur. Dans un script, on peut définir des fonctions après la dernière commande du script, en utilisant le mot-clef `function`. Il faut également préciser les valeurs de retour et les arguments comme dans l'exemple ci-dessous.

La définition de la fonction doit se terminer par le mot-clef `end`. Le script peut utiliser toutes les fonctions définie dans le même m-file, mais celles-ci ne sont pas accessibles hors du m-file.

Le script ci-dessous utilise une fonction, `moyenne`, qui prend un entier comme argument et retourne ensuite la valeur moyenne des éléments de matrice aléatoire de taille `n x n`.

```
n = 4;       % On choisit une valeur pour n.
moyenne(n)   % On utilise la fonction moyenne() définie ci-dessous.

function m = moyenne(n)
    % Calcule la moyenne des éléments d'une matrice aléatoire.
    % Cette fonction prend comme paramètre un entier n et affiche les
    % valeurs d'une matrice aléatoire A de taille n x n. Elle retourne
    % la moyenne des éléments de matrice de A.

    A = rand(n);       % Une matrice aléatoire de taille n x n

    imagesc(A);        % Affiche les éléments de matrice sous forme de
                       % couleurs

    axis equal off;    % Met les axes à la même échelle et les supprime
                       % de la figure.

    m = mean(mean(A)); % Calcule les moyennes des colonnes, puis la
                       % moyenne du vecteur ainsi obtenu. Ceci est
                       % équivalent à calculer la moyenne de tous les
                       % éléments de matrice.
end                    % Termine la définition de la fonction
```

La forme générale de la déclaration d'une fonction est

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

Quelques remarques :

1. Il peut y avoir plusieurs paramètres, auquel cas il suffit de les séparer par des virgules, ou aucun.
2. Il peut y avoir plusieurs variables de retour, auquel cas il faut les mettre entre crochets et les séparer par des virgules, ou aucune.
3. Pour les fonctions définies dans un script, comme ci-dessus, la définition des fonctions doit apparaître à la fin du script. (Pas de code du script après la définition d'une fonction.)
4. Afin de vous souvenir de ce que vous avez programmé, et de permettre à d'autre personnes d'utiliser vos fonction, il est indispensable de mettre un commentaire expliquant ce que la fonction effectue. Traditionellement, ce commentaire est placé juste sous la définition des variables de retour et des paramètres, c'est-à-dire sous la première ligne (voir l'exemple ci-dessus).
5. Dans le cas où les fonctions sont définies dans leur propre m-file (ce qu'on évitera sauf indication contraire dans ces TPs), la première ligne de commentaires est importante

car les mots de cette ligne sont utilisés par la commande de recherche de fonction de MATLAB , `lookfor`.

6. Pour des fonctions plus compliquées, il est recommandé d'énumérer les paramètres d'entrée et les variables retournées en expliquant leur nature.

5.3 Debuggage

En programmant, il arrive (toujours...) que des erreurs s'insinuent dans le code. Ces erreurs peuvent être de plusieurs types. Les erreurs de syntaxe 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, il faut tester 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 m-file des *break point*. Ceux-ci interrompent l'exécution à l'endroit choisi et vous pouvez accéder aux valeurs des variables internes à cet instant via la fenêtre `Workspace`. Vous pouvez exécuter la suite du programme jusqu'au break point suivant ou ligne par ligne. Les break points peuvent être ajoutés au moyen du menu au dessus de la `Command windows` ou de l'`Editor`.

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 Expressions booléennes

Une variable booléenne est une variable prenant deux valeurs : vrai ou faux. Les variables booléennes de MATLAB prennent les valeurs 1 (vrai) ou 0 (faux).

En MATLAB le test d'égalité se fait à l'aide de [==] et celui d'inégalité à l'aide de [∼=]. Ces tests retournent une valeur booléenne suivant la véracité de l'expression. On peut combiner des variables et valeurs booléennes au moyen des opérateurs logiques "et" [&] et "ou" [|].

Voici un exemple :

```
>> a=1;b=2;           % On définit deux variables a = 1, b = 2
>> a==b               % "a égal b" est faux.
ans =
     0
>> a~=b               % "a inégal à b" est vrai.
ans =
     1
>> a == 1 & b == 1    % "a égal 1 et b égal 1" est faux.
ans =
     0
>> a == 1 | b == 1    % "a égal 1 ou b égal 1" est vrai.
ans =
     1
```

6.2 Conditions – `if ... else ... end`

La commande `if` permet d'executer un bloc de commandes seulement si une condition est vraie. La condition est une variable booléenne (cf. ci-dessus). La syntaxe est la suivante :

```
if (test)          % Si la condition test est vraie,...
    [commandes]    % ...on exécute les commandes dans ce bloc.
end
```

Notez le mot-clé `end` qui termine le bloc de commandes. Par souci de lisibilité du code, il est recommandé d'indenter le texte du bloc contenu dans la commande `if`.

Pour exécuter un autre bloc de commande au cas où la condition est fausse, on utilise `else` :

```
if (test)          % Si la condition test est vraie,...
    [commandes]    % ...on exécute les commandes dans ce bloc.
else               % Si test est fausse,...
    [commandes]    % ...on exécute les commandes dans ce bloc.
end
```

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

```
if (test1)        % Si la condition test1 est vraie,...
    [commandes]   % ...on exécute les commandes dans ce bloc.
elseif (test2)    % Si test1 est fausse et test2 est vraie,...
    [commandes]   % ...on exécute les commandes dans ce bloc.
elseif (test3)    % Si test1 et test2 sont fausses et test3 est vraie,...
    [commandes]   % ...on exécute les commandes dans ce bloc.
    ...           % On peut ajouter autant de elseif que nécessaire.
else              % Si toutes les conditions sont fausses,...
    [commandes]   % ...on exécute les commandes dans ce bloc.
end
```

Voici maintenant un exemple de bloc `if`, une fonction qui teste si une variable entière `n` est paire ou impaire.

```
function estpaire(n)
% Cette fonction affiche un message disant si l'argument n est pair ou impair.
% n est un entier

if mod(n,2) == 0                % Teste si le reste de la division
                                % Euclidienne par 2 est égal à zero.
                                % Si c'est le cas,...

  disp('L''argument est pair')  % ...l'argument est pair.

elseif mod(n,2) == 1            % Si ça n'est pas le cas, on teste si le
                                % reste vaut 1. Si c'est le cas,...
```

```
  disp('L''argument est impair') % ...l'argument est impair.

else                             % Si ça n'est pas le cas,...

  disp('L''argument n'est pas un entier')  % Le nombre fourni n'était pas un
                                           % entier.

end
```

6.3 Boucles – `for ... end`

Une boucle `for` permet de répéter un groupe de commandes pour chaque élément d'une liste. La syntaxe est la suivante :

```
for k = liste          % Une boucle for
    [commandes]
end
```

La variable `k` est appelée l'*itérateur*. MATLAB assigne successivement à `k` toutes les valeurs dans le vecteur `liste` et exécute à chaque fois les commandes du bloc `[commandes]`. Le bloc de commandes à répéter se termine obligatoirement par le mot clé `end`. Pour des raisons de lisibilité du programme, il est bon d'indenter le bloc de commandes dans la boucle `for`.

Voici un exemple, un script calculant le 12ème nombre de Fibonacci. On rappelle que les nombre de Fibonacci f_n sont définis par la récurrence $f_{k+2} = f_{k+1} + f_k$, avec $f_1 = f_2 = 1$:

```
% Calcul le 12ème nombre de Fibonacci
fibk=1;                     % Le kième nombre de Fibonacci
fibkp1=1;                   % Le (k+1)ième nombre de Fibonacci
fibkp2=1;                   % Le (k+2)ième nombre de Fibonacci
for k = [1:10]              % Une boucle pour résoudre la relation de
                            % récurrence. k va de 1 à 10 par pas de 1.
    fibk = fibkp1;          % Le kième n.F. est le (k+1)ième n.F. du pas
                            % précédent, où k valait 1 de moins.
    fibkp1 = fibkp2;        % Le (k+1)ième n.F. est le (k+2)ième n.F. du
                            % pas précédent.
    fibkp2 = fibkp1+fibk;   % La relation de récurrence. Le dernier pas
                            % de la boucle est effectué avec k = 10, donc
                            % fibkp2 est le 12ième nombre de Fibonacci.
end                         % La commande end finit le bloc de commande
                            % que Matlab répète dans la boucle.
fibkp2                      % Affiche le résultat
```

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 $\sqrt{-1}$.

6.4 Boucles – `while ... end`

La commande `while` est similaire à `for`, dans le sens qu'elle permet de répéter un bloc de commande. La différence est qu'elle prend comme argument une variable booléenne. Tant

que la variable booléenne est vraie, le bloc de commandes est répété. Lorsque la variable booléenne devient fausse, le programme exécute les commandes suivant le mot-clé `end`. La syntaxe est la suivante :

```
while test        % test est une variable booléenne. Tant que test est
                  % vrai...
    [commandes]   % ...ce bloc de commandes s'exécute
end               % Le mot clé end termine le bloc de commandes.
```

L'exemple ci-dessous affiche des matrices aléatoires en boucle.

```
k=0;                  % k est une variable qui va augmenter de 1 lors de
                      % chaque répétition du bloc while.
figure;               % Ouvre une nouvelle figure
while k < 50          % Tant que k est plus petit que 50, le bloc
                      % suivant s'exécute.
    k = k+1;          % k est augmenté de 1. Nécessaire pour que le
                      % programme ne soit pas pris dans une boucle qui
                      % se répète infiniment.
    A=rand(10,10);    % Une matrice aléatoire 10 x 10
    imagesc(A);       % Dessine la matrice sous la forme de carrés colorés.
    axis off equal;   % Pour des axes à la même échelle et invisibles
    drawnow;          % Force Matlab à dessiner à chaque boucle et non
                      % une seule fois à la fin.
end                   % Termine le bloc de commandes while.
```

7 Exemple d'exercice résolu

Considérons l'exercice suivant :

Exemple d'exercice

1. Ecrire une commande qui donne une matrice 4×4 aléatoire à coefficients entiers (tirés uniformément) compris entre -5 et 5.
2. Utiliser la commande précédente 10 fois et calculer le déterminant de chacune de ces matrices. Combien y-a-t-il de déterminants égaux à 0 ?
3. Pouvez-vous expliquer cela ?

Le m-file `ExempleExerciceResolu.m` est un exemple de solution, rédigé sous la forme exigée pour vos solutions. Téléchargez-le sur moodle et ouvrez-le. Allez sous l'onglet `Publish` et cliquez sur le bouton `Publish`. Un document html apparaît sur lequel vous pouvez lire la solution.

Les règles de bases pour commenter son code sont :

1. Tout script débute par un bloc de commentaire expliquant la stratégie générale utilisée pour résoudre le problème. Utilisez le bouton `Preformatted text` sous l'onglet `Publish` pour créer un bloc de commentaires.

2. Les noms des variables doivent décrire explicitement leur fonction. Par exemple une variable encodant une somme sera nommée `somme` au lieu de `x`.
3. Les parties non-triviales du code doivent être commentées pour expliquer leur fonction.

Ajoutez une section pour chaque point numéroté de l'exercice (onglet `Publish`, `Section with Title`).

Gardez à l'esprit que les assistants qui corrigeront vos travaux n'ont pas le temps de deviner ce que votre code fait ou est sensé faire, c'est pourquoi il est indispensable de tout expliquer. Une solution sans commentaire vaut zéro point, même si le code fonctionne. C'est une bonne idée de rédiger vos exercices durant le laboratoire suivant le modèle ci-dessus est de les présenter à un assistant pour vous assurez qu'ils sont lisibles et corrigeables.

8 Série d'exercices

8.1 Exercices

Exercice 1 Opérations sur les vecteurs. On définit

$$q = \begin{pmatrix} -2 \\ 1 \end{pmatrix}, \quad r = \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \quad s = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}, \quad t = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad u = \begin{pmatrix} -2 \\ 4 \\ -1 \end{pmatrix}, \quad v = \begin{pmatrix} 24 \\ 23 \\ \vdots \\ -25 \end{pmatrix}, \quad w = \begin{pmatrix} 100 \\ 98 \\ \vdots \\ 2 \end{pmatrix};$$

Calculez, quand c'est possible, les expressions suivantes en utilisant les commandes présentées dans la Section 3. Lorsqu'une expression n'a pas de sens, expliquez pourquoi dans votre m-file sous la forme d'un commentaire.

1. $q - r$, $q + 3v$, $s + 3t - u$.
2. $\langle q|r\rangle$, $\langle s|s\rangle - 2\langle s|t\rangle - \langle s|u\rangle$, $\langle s|v\rangle$, $\langle v|w\rangle$, $\langle v|v\rangle$, où $\langle q|r\rangle$ représente le produit scalaire entre q et r.
3. $\|v\|$, $\|s\|^2$.

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

Exercice 3 Figures géométriques. Au moyen de la fonction `plot`, dessinez les figures géométriques suivantes :

1. un carré, penché d'un angle de 30° par rapport à l'horizontale ;
2. un polygone régulier à n côtés ;
3. un pentagone étoilé (pentagramme) ;
4. un cercle.

Dans le cas du polygone régulier, votre script doit admettre le nombre de côtés n comme un paramètre ajustable.

Exercice 4 Graphes de fonctions.

1. Tracez le graphe de toutes les fonctions suivantes sur la même figure (Voir la Section 4.1) :

$x \mapsto x^x$;
$x \mapsto 3$;
$x \mapsto \log_3 x$
$x \mapsto \frac{1}{x}$.

2. A l'aide de ce graphique, estimez la solution des équations suivantes $x^x = 3$ et $\frac{1}{x} = \log_3 x$.
3. (Facultatif) Démontrez analytiquement que ces deux solutions coïncident.

Exercice 5 Résolution matricielle de systèmes linéaires. Résolvez matriciellement les systèmes d'équations linéaires suivants (voir Section 3.3). Si MATLAB affiche un message d'erreur ou un avertissement, expliquez pourquoi.

1.
$$\begin{array}{rcrcrcr} 6x & + & y & - & 5z & = & 6 \\ 2x & + & y & + & 3z & = & 11 \\ 4x & - & 9y & + & 7z & = & 12 \end{array}$$

2.
$$\begin{array}{rcrcrcr} 6x & + & y & - & 7z & = & 10 \\ 2x & + & 2y & + & 3z & = & 2 \\ 8x & + & 3y & - & 4z & = & 12 \end{array}$$

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

Exercice 6 Spectre d'une application linéaire. On donne une application linéaire T par la matrice suivante

$$\frac{1}{4}\begin{pmatrix} 1 & -3 & 1 & 1 \\ -3 & 1 & 1 & 1 \\ 1 & 1 & 7 & -9 \\ 1 & 1 & -9 & 7 \end{pmatrix}$$

1. Déterminez le noyau de T, ainsi que sa dimension.
2. Décrivez l'orthogonal H du noyau de T.

   Indication : Matlab a une fonction `null` qui peut être utile, voyez l'aide. Sinon, il assez facile de déterminer l'orthogonal à la main, en utilisant le point 1.
3. Ce sous-espace H est-il invariant par l'application linéaire T ? Vérifiez votre réponse avec Matlab.
4. Déterminez le spectre de cette application, ainsi que les vecteurs propres associés.
5. Construisez une base orthonormée formée de vecteurs propres, si c'est possible. Sinon, expliquez pourquoi.
6. Soit $P(x) = x^2 + 2x + 1$, on définit l'application $P(T) = T^2 + 2T + \mathrm{Id}$, où Id est la matrice identité de taille 4×4. Déterminez le spectre de $P(T)$. Quelle est la relation entre le spectre de $P(T)$ et le spectre de T ?

Exercice 7 Extraction de la diagonale. Soit A une matrice carrée. Construisez une matrice diagonale B ayant la même diagonale que A. Autrement dit, votre code doit transformer

$$\begin{pmatrix} a_1 & * & * \\ * & \ddots & * \\ * & * & a_n \end{pmatrix} \quad \text{en} \quad \begin{pmatrix} a_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & a_n \end{pmatrix}.$$

Indication : En utilisant judicieusement une fonction de Matlab, ceci est possible en une seule ligne de code.

Exercice 8 Matrice de Vandermonde. Écrire une fonction qui prend en paramètre un entier n et qui construit la matrice ci-dessous.

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 \\ 1 & 2 & 3 & \cdots & (n-2) & (n-1) & n \\ 1 & 2^2 & 3^2 & \cdots & (n-2)^2 & (n-1)^2 & n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 2^{n-3} & 3^{n-3} & \cdots & (n-2)^{n-3} & (n-1)^{n-3} & n^{n-3} \\ 1 & 2^{n-2} & 3^{n-2} & \cdots & (n-2)^{n-2} & (n-1)^{n-2} & n^{n-2} \\ 1 & 2^{n-1} & 3^{n-1} & \cdots & (n-2)^{n-1} & (n-1)^{n-1} & n^{n-1} \end{pmatrix}$$

Mettez votre fonction dans un script qui l'utilise pour afficher la matrice de Vandermonde de taille 5×5.

Exercice 9 Changement de bases. On suppose donné

1. une matrice $m \times k$ représentant une application linéaire de $\mathbb{R}^k$ dans $\mathbb{R}^m$ relativement aux bases canoniques ;
2. deux bases de $\mathbb{R}^k$ et de $\mathbb{R}^m$, données en composantes relativement aux bases canoniques, i.e. sous la forme de matrices de taille $k \times k$ et $m \times m$.

Ecrivez une fonction qui accepte ces trois matrices comme variables d'entrée, et qui retourne la matrice de l'application linéaire relativement aux nouvelles bases.

Exercice 10 Applications linéaires entre espaces de polynômes (plus difficile).
Désignons par $\mathfrak{P}_n$ l'espace vectoriel des fonctions polynomiales de $\mathbb{R}$ dans $\mathbb{R}$ de degré $\leq n$.

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

$$\begin{aligned} R : \mathfrak{P}_n &\longrightarrow \mathfrak{P}_n \\ p &\longmapsto r\big(p, x^2+1\big). \end{aligned}$$

   Écrivez une fonction qui prend comme paramètre un entier n et qui renvoie la matrice M_R de l'application R dans la base $B = \{1, x, x^2, \ldots, x^n\}$.
   Indication : Ecrivez tout d'abord la matrice de l'application R dans la base

$$\tilde{B} = \{1; x; x^2+1; x(x^2+1); x^2(x^2+1); \cdots; x^{n-2}(x^2+1)\}$$

   et utilisez les matrices de changements de bases $M_B^{\tilde{B}}(Id)$ et $M_{\tilde{B}}^{B}(Id)$, pour exprimer la matrice de R dans la base B.

2. À l'aide de M_R, calculez le reste de la division par x^2+1 des polynômes suivants :
   (a) $7x^8 + 411x^7 - 231x^5 + 31x^4 + 451x^3 - 10x - 42$;
   (b) $x^7 + \frac{5}{21}x^5 + 0.432x^4 - 22x^3 + 51x^2$.
3. Considérons l'application linéaire *dérivée* :

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

   Ecrivez la matrice M_d de l'application d dans la base B. Quels sont les valeurs propres et les vecteurs propres de l'application d ?
4. Calculez les matrices des applications $d \circ R$ et $R \circ d$ dans la base B. Que remarquez-vous ?