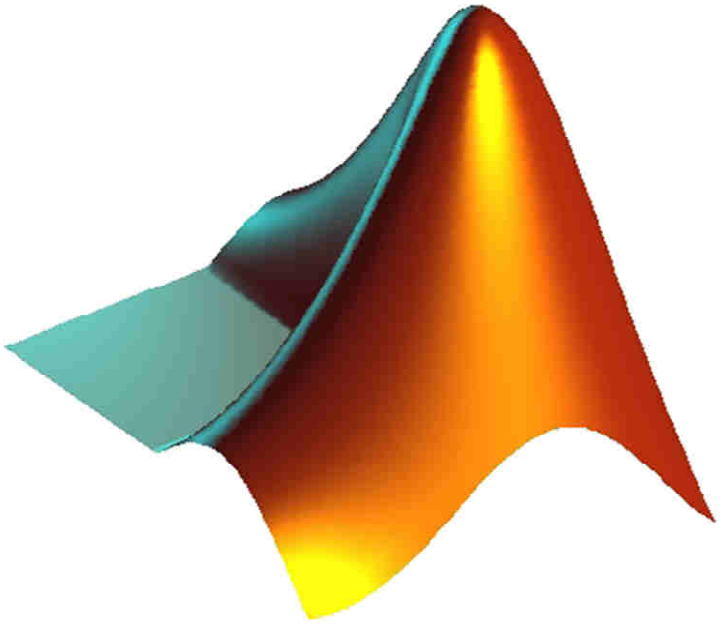
INITIATION AU

LOGICIEL MATLAB



TELECOM nancy
Ingénieurs du numérique • Inspiring your digital future

1ère année

2020/2021

Marc Tomczak

UNIVERSITÉ DE LORRAINE

1. INTRODUCTION

MATLAB (*MA Trix LA Boratory*) est un logiciel interactif de calcul scientifique et technique. Il est issu des projets *Linpack* et *Eispack*, développés dans les années 70 et visant à constituer des librairies de programmes dédiés au calcul matriciel. Bien qu'écrit à l'origine en Fortran, par *Cleve Moler*, la version actuelle[1] (due à *The Math Works Inc.*) est en langage C. Elle est complétée par un outil de simulation (*SIMULINK*).

MATLAB est disponible sur plusieurs plates-formes : Sun, Bull, HP, IBM, compatibles PC, Macintosh, ainsi que plusieurs machines parallèles. Un de ses atouts est sa portabilité : la même portion de code peut être utilisée sur différentes plates-formes sans la moindre modification.

L'environnement MATLAB comprend un noyau de base (*built-in functions*) et des bibliothèques de fonctions spécifiques (boîtes à outils ou *toolboxes*), propres à un domaine, comme par exemple la *Signal Processing Toolbox*, dédiée au traitement du signal. Actuellement, une cinquantaine de boîtes à outils sont disponibles (commande des systèmes, identification, traitement d'images, réseaux de neurones, logique floue, calcul symbolique, mathématiques financières, statistiques, communications, etc.). L'édition du code source de toutes les fonctions est possible (sauf pour les *built-in functions*), l'utilisateur est donc libre de les modifier ou de s'en inspirer pour écrire ses propres fonctions.

Du point de vue des langages informatiques, MATLAB est un interpréteur, les instructions saisies dans la fenêtre de commande sont exécutées au fur et à mesure des appuis sur la touche *entrée* ou *return*. Cependant, elles peuvent aussi être regroupées dans un fichier programme, ou *script*, reconnaissable à l'extension **.m**. L'exécution de ce dernier peut alors être lancée en tapant son nom dans la fenêtre de commande, sous réserve que le chemin d'accès (le *path*) soit indiqué.

Le "langage" MATLAB contient un minimum de structures de programmation (structure itérative, structure conditionnelle, sous-routine) mais reste assez rudimentaire. L'avantage est qu'il est très simple, assez intuitif et très rapide à programmer, offrant une grande tolérance (syntaxe simple, pas d'obligation de définition de types, etc.), ce qui permet un gain appréciable en temps de mise au point. L'ingénieur peut par ce moyen être plus efficace dans l'analyse d'un problème, en concentrant ses efforts sur celui-ci et non pas sur l'outil servant à le résoudre.

Enfin, l'interface graphique de MATLAB est sans conteste l'un des points forts du logiciel et facilite le tracé de courbes et l'obtention de graphiques 2D ou 3D de grande qualité.

Il faut tout de même noter qu'il existe des produits concurrents :

- MatrixX/SystemBuild, développé à partir de la même base par Integrated Systems Inc.
- Scilab/Scicos développé par l'INRIA en France, d'utilisation libre, qui peut être obtenu à l'adresse http://www.scilab.org/
- Octave, développé par John W. Eaton et d'autres, d'utilisation libre, qui peut être téléchargé à l'adresse http://www.gnu.org/software/octave/

[1] Ce fascicule a été rédigé à partir de la version 7.14 de Matlab, on en est aujourd'hui à la version 9.6 et de nombreuses fonctionnalités ont été ajoutées entre-temps. Mais pour vous, pauvres novices, le contenu du présent poly est plus que suffisant !

Ce dernier est particulièrement compatible avec MATLAB, quant à Scilab, sa philosophie est très proche également.

2. DÉMARRAGE[2]

Au démarrage, en configuration par défaut, plusieurs fenêtres sont ouvertes, dont :

- la **fenêtre de commande** (*command window*), qui comme son nom l'indique est la fenêtre où sont entrées les instructions ou commandes Matlab. Celle-ci est affectée de son invite (*prompt*) ">>", signe qui signifie que le logiciel est en attente d'une commande. Le contenu de cette fenêtre peut être effacé à l'aide de la commande `clc`.
- l'**historique** des commandes exécutées auparavant (*command history*). Il permet d'accéder à d'anciennes commandes. Il est également possible de naviguer parmi ces commandes à l'aide des touches ↑ et ↓ du clavier. Entrer une lettre puis taper sur la touche ↑ permet de rappeler les dernières commandes entrées commençant par cette lettre.
- l'**espace de travail** (*workspace*) qui contient l'ensemble des variables créées lors de la session (si elles n'ont pas été effacées entre-temps) et leur dimension. La gestion de l'espace de travail peut se faire soit par l'intermédiaire du menu ou du clic droit, soit par les commandes `who` et `whos` (affichage des variables existantes, de leur dimension, de la mémoire disponible), `clear` (réinitialisation de l'espace de travail : `clear all`, ou suppression d'une ou plusieurs variables : `clear` <*nom_variable*>), `load` (chargement d'un fichier de données), `save` (sauvegarde des variables de l'espace de travail dans des fichiers de données). Les noms des fichiers de données Matlab sont en général dotés de l'extension **.mat**.

Le **répertoire courant** (current folder), qui constitue le répertoire de travail où vont s'enregistrer les fichiers créés par l'utilisateur, apparaît en haut de la fenêtre principale. Ce répertoire peut bien sûr être changé, mais pour qu'il soit reconnu, par exemple lors de l'appel d'un programme qui s'y trouve, le chemin d'accès doit être spécifié dans le *path*.

D'autres fenêtres peuvent apparaître en cours d'utilisation, il s'agit :

- de l'**éditeur** de Matlab, qui permet de taper des programmes ou des fonctions. Bien sûr, tout autre éditeur peut faire l'affaire, mais l'éditeur de Matlab est équipé d'un débogueur, ce qui peut être utile.
- des **fenêtres graphiques**, en cas d'utilisation de commandes graphiques.
- de la fenêtre d'**aide**. Celle-ci peut être invoquée à partir du menu, ou directement à partir de la fenêtre de commande (`helpwin`) : `>>helpwin` <*fonction ou commande*>.

Attention : Matlab fait la distinction entre lettres minuscules et majuscules, or, toutes les instructions Matlab sont en **minuscules**[3], bien que, dans l'aide, elles apparaissent écrites en majuscules afin de les mettre en évidence.

---

[2] Un didacticiel interactif Matlab (en anglais) est disponible à l'adresse suivante :
http://www.mathworks.fr/academia/student_center/tutorials/register.html?s_cid=ACD0809frFT&s_v1=1-AA4DCC_1-B5QTP

[3] Avec les versions les plus récentes, ceci n'est plus vrai pour l'intégralité des commandes, mais dans le doute…

L'aide peut également être invoquée et obtenue dans la fenêtre de commande (`help`), selon le schéma : `>>help` <*fonction ou commande*>.

L'instruction `lookfor` fournit la liste des fonctions et commandes contenant le **mot-clé** spécifié dans la première ligne de leur texte d'aide : `>>lookfor` <*mot-clé*>.

3. QUELQUES GÉNÉRALITÉS

L'instruction `which` <*nom_fonction*> retourne le **chemin d'accès** de la fonction spécifiée. La sélection de ce dernier et un clic droit permettent alors d'éditer le code correspondant.

L'instruction `type` <*nom_fichier*> permet de **visualiser le contenu** de ce fichier.

L'instruction `what` renvoie **la liste des fichiers** spécifiques à Matlab, notamment ceux possédant l'extension **.m**, **dans le répertoire courant**.

La commande `pack` permet de **défragmenter** la mémoire.

Pour **sauvegarder** la session en cours dans un fichier texte, utiliser la commande `diary`.

Enfin, les commandes DOS `cd`, `pwd`, `dir` sont utilisables dans Matlab.

La séquence de touches **Ctrl-C** permet de mettre fin à toute opération ou exécution en cours.

Les noms des fichiers de programmes simples (**scripts**) et des **fonctions** doivent être suivis de l'extension **.m**. **Ces noms ne doivent comporter ni caractères spéciaux** (accents, slash, etc.) **ni espaces**.

Lors de l'exécution, les variables créées dans un script entrent dans l'espace de travail (ce sont des **variables globales**). Au contraire, les variables créées dans une fonction sont **locales**, les fonctions permettent donc d'effectuer des passages de paramètres. Pour qu'un fichier programme soit reconnu comme une fonction, son entête doit être au format :

`function` [*liste des arguments de sortie*] =nom_de_fonction(*liste des arguments d'entrée*)

Exemple : `function [x1,x2]=soluce_trinome(a,b,c)`

Ces listes utilisent la virgule comme séparateur : `arg1,arg2,…`

Attention : bien que ce ne soit pas obligatoire, il est fortement recommandé de toujours donner le même nom à une fonction et au fichier correspondant (nom_de_fonction = nom_de_fichier).

Si un script peut être invoqué en tapant simplement son nom (sans l'extension) dans la fenêtre de commande, **l'appel à une fonction** se fait selon le schéma :

[*liste des arguments de sortie*] =nom_de_fonction(*liste des arguments d'entrée*)

Exemple : `[sol1,sol2]=soluce_trinome(2,3,1)`

`sol1` et `sol2` sont les variables du *workspace* dans lesquelles sont enregistrées les valeurs de `x1` et `x2`, variables locales.

Tout ce qui suit le caractère "`%`" (pourcent) sur une ligne n'est pas interprété par Matlab et est donc considéré comme un **commentaire**. Les premières lignes de commentaires en tête de fichier sont celles qui apparaissent lors d'une demande d'aide sur le fichier en question par la commande `help`.

Lorsqu'un programme comporte une erreur, l'exécution de celui-ci s'arrête au niveau de l'erreur en question. Matlab renvoie alors un **message d'erreur** (en rouge), ce dernier est souvent très explicite (bien qu'en anglais) et **ne doit donc pas être négligé lors du débogage !** D'autres messages, en noir, peuvent intervenir, ce sont des **avertissements** (par exemple, pour signaler l'utilisation d'une commande obsolète ou une division par 0).

Les **séparateurs d'instructions** sont les caractères "`,`" (virgule) et "`;`" (point-virgule). Que ce soit dans un fichier programme ou dans la fenêtre de commande, le point-virgule permet d'**éviter l'affichage à l'écran** du résultat de la commande qui le précède. Ces deux séparateurs s'utilisent également lors de la définition de matrices.

Matlab possède des **variables prédéfinies** : `pi`, qui contient une approximation de π, `Inf` pour $+\infty$, `NaN` (Not-a-Number) pour le résultat d'une opération non définie comme $0/0$, `i` et `j` pour $\sqrt{-1}$ (pour travailler sur des complexes), ou encore `ans` qui contient le résultat de la dernière instruction exécutée.
Attention : ces variables peuvent être redéfinies (parfois sans y prêter attention) par l'utilisateur. Ce peut être le cas par exemple avec `i` et `j` qui sont souvent utilisés comme indices de boucle !

Sous Matlab, les calculs sont effectués avec une **arithmétique à précision finie**. Ceci le différencie des logiciels de calcul symbolique tels que Maple, mais la comparaison n'a pas lieu d'être. Calcul numérique et calcul symbolique sont des outils complémentaires du calcul scientifique. Par ailleurs, Matlab possède une boîte à outils optionnelle de calcul symbolique.
La variable prédéfinie `eps` correspond à la **précision relative** des nombres. Elle est égale à 2^{-52} et est donc de l'ordre de 10^{-16} ($2{,}2204.10^{-16}$), l'arithmétique sur ordinateur n'étant pas exacte (l'étendue des nombres représentables s'étend de $2^{-1022} = 2{,}2251.10^{-308}$ à $(2 - \texttt{eps}).\, 2^{1023} = 1{,}7977.10^{308}$).

Bien que tous les calculs soient effectués en double précision, il existe différents **formats d'affichage** des nombres, parmi lesquels `short` (format par défaut : virgule fixe avec 5 chiffres), `long` (virgule fixe avec 15 chiffres), `short e` (virgule flottante et 5 chiffres), `long e` (virgule flottante et 15 chiffres), `rat` (approximation par un ratio d'entiers), etc.

Exemple : `>> format long, pi → ans = 3.14159265358979`

Avec Matlab on travaille essentiellement avec un seul type d'objet : **des matrices**. Sous Matlab, une matrice est un tableau rectangulaire de nombres représentés en virgule flottante avec une double précision, c'est à dire 8 octets pour un nombre réel et 16 octets pour un nombre complexe. Une matrice 1×1 est interprétée comme un scalaire, celles ayant une seule ligne ou une seule colonne comme des vecteurs.

Les commandes `format loose` et `format compact` permettent **de régler l'interligne à l'affichage**.

4. PRINCIPALES COMMANDES MATLAB

Opérateurs de base : ceux-ci agissent sur les scalaires et les matrices. Dans le cas matriciel, les dimensions des matrices doivent s'y prêter !

+	Addition
−	Soustraction
*****	Multiplication
/	Division
****	Division à gauche
^	Élévation à une puissance

La division à gauche `A\B` donne le résultat de l'opération $A^{-1}B$ (ce qui est utile pour résoudre un système linéaire, voir aussi l'annexe en fin de document). Avec cet opérateur `backslash`, la réponse est donc telle que `A*ans = B`. Lorsque `A` est une matrice carrée de rang plein et `B` un vecteur colonne avec un nombre de lignes égal à la dimension de `A`, il s'agit d'un problème classique de résolution d'un système d'équations linéaire, dont l'unique solution est donnée par $A^{-1}B$.
Plus généralement, lorsqu'il n'y a pas qu'une solution, l'opérateur `backslash` calcule une solution au sens des moindres carrés et retourne une solution basique comportant au plus `r` valeurs non-nulles, où `r` est le rang de `A`. La solution à norme minimale peut être obtenue à l'aide de la commande `pinv`.
Remarque : dans le cas de deux matrices, l'opération `A/B` équivaut à `(B'\A')'`.

L'opération d'élévation à une puissance n'a de sens que si l'un des opérandes au moins est un scalaire. Si l'un des opérandes est une matrice, celle-ci doit être carrée.

Opérateurs "élément par élément" : ils s'utilisent dans le cas matriciel. Là encore, les dimensions des matrices (ou l'orientation dans le cas de vecteurs) doivent correspondre.

.*	Multiplication élément par élément
./	Division élément par élément
.^	Élévation à une puissance élément par élément

Autres opérateurs matriciels :

inv(.)	Inversion d'une matrice
' (A')	Transposition-conjugaison
.' (A.')	Transposition simple
fliplr(.)	Modification de l'ordre des colonnes dans le sens gauche-droite
flipud(.)	Modification de l'ordre des lignes dans le sens haut-bas

Voir aussi : `rot90`, `flipdim`, `permute`, `reshape`, …

Dans le cas où la matrice `A` ne contient que des réels, l'opérateur `'` n'effectue qu'une transposition.

Opérateurs relationnels :

<	Strictement inférieur
<=	Inférieur ou égal
>	Strictement supérieur
>=	Supérieur ou égal
==	Égal
~=	Différent

Matlab renvoie 1 pour un événement vrai et 0 pour un événement faux.

Opérateurs logiques :

&	ET logique (AND)
|	OU logique (OR)
~	NON logique (NOT)
xor	OU exclusif (XOR)

Instructions mathématiques de base :

Celles-ci s'appliquent aux scalaires comme aux matrices. Dans le cas matriciel, elles s'appliquent à chaque élément de la matrice.

sqrt(.)	Racine carrée	**abs(.)**	Valeur absolue ou module
exp(.)	Exponentielle	**angle(.)**	Phase
log(.)	Logarithme naturel	**conj(.)**	Conjugaison complexe
log10(.)	Logarithme décimal	**imag(.)**	Partie imaginaire
log2(.)	Logarithme de base 2	**isreal(.)**	Vrai si réel
nextpow2(.)	Puissance de 2 supér.	**real(.)**	Partie réelle
cos(.)	Cosinus	**fix(.)**	Arrondi vers zéro
acos(.)	Arc cosinus	**floor(.)**	Arrondi vers -∞
sin(.)	Sinus	**ceil(.)**	Arrondi vers +∞
asin(.)	Arc sinus	**round(.)**	Arrondi au plus proche entier
cot,tan(.)	(Co)Tangente	**sign(.)**	Signe
atan(.)	Arc tangente	**factorial(.)**	Factorielle
sinc(.)	Sinus cardinal	**rem(.)**	Reste division euclidienne
cosh,sinh, tanh(.)	Fonctions hyperboliques	**mod(.)**	Reste modulo

Instructions pour les matrices :

det(.)	Déterminant
diag(.)	Création d'une matrice diagonale ou sélection de la diagonale
zeros(.)	Création d'une matrice de zéros
ones(.)	Création d'une matrice de uns
eye(.)	Matrice identité
size(.)	Dimensions d'une matrice
length(.)	Dimension maximale d'une matrice (longueur d'un vecteur)
expm(.)	Exponentielle matricielle
rank(.)	Rang de la matrice

Voir aussi : `cond`, `norm`, `trace`, `eig`, `poly`, …

Instructions polynomiales :

conv(.)	Produit de polynômes et produit de convolution
deconv(.)	Division polynomiale et déconvolution
polyval(.)	Calcul des valeurs d'un polynôme
roots(.)	Racines d'un polynôme
poly(.)	Définition d'un polynôme à partir des racines

Voir aussi : `polyder`, `polyfit`, `residue`, …

Instructions statistiques :

min(.)	Valeur minimale	**xcov(.)**	Fonction d'auto ou d'intercovariance
max(.)	Valeur maximale	**xcorr(.)**	Fonction d'auto ou d'intercorrélation
mean(.)	Valeur moyenne	**psd(.)**	Estimateur de la densité spectrale de puissance
median(.)	Valeur médiane	**hist(.)**	Histogramme
std(.)	Écart-type	**rand**	Matrice aléatoire uniforme
cov(.)	Covariance	**randn**	Matrice aléatoire gaussienne
var(.)	Variance	**sort(.)**	Tri des éléments

La commande `cov` utilisée sur un vecteur renvoie la variance et est équivalente dans ce cas à la commande `var`.

Caractères spéciaux :

:	Boucle implicite : génération de vecteurs. Indice : "toutes les lignes" ou "toutes les colonnes"	**[.]**	Définition de matrices ou arguments de sortie de fonctions
…	Continuation à la ligne	**(.)**	Expressions math. Ou arguments d'entrée de fonctions
,	Sépare 2 éléments sur une ligne, sépare 2 instructions en autorisant l'affichage du résultat de la première	**;**	Sépare 2 lignes, sépare 2 instructions en interdisant l'affichage
.	Point décimal	**..**	Répertoire parent
'	Délimiteur de chaîne de caractère (à doubler en cas d'apostrophe)	**=**	Affectation

Instructions graphiques :

plot(.)	Graphe simple	**grid**	Grille de graduation
subplot(.)	Plusieurs graphes sur une figure	**hold**	Maintien du graphe
stem(.)	Représentation par impulsions	**axis(.)**	Gestion des axes
bar(.)	Représentation type histogramme	**xlabel(.)**	Légende abscisses
stairs(.)	Représentation en escalier	**ylabel(.)**	Légende ordonnées
figure(.)	Nouvelle figure	**title(.)**	Titre de la figure
clf	Effacer la figure courante	**legend(.)**	Légende de la figure
close(.)	Fermer la figure	**semilogx(.)**	Échelle log sur les abscisses
mesh(.)	Graphe 3D	**semilogy(.)**	Échelle log sur les ordonnées
polar(.)	Graphe en coordonnées polaires	**loglog(.)**	Échelle log sur les 2 axes

Voir aussi : `axes`, `ginput`, `surf`, `zlabel`, …

Instructions de contrôle du temps :

clock	Date et heure sous forme de vecteur
cputime	Temps CPU
date	Date du jour (chaîne de caractères)
now	Date du jour et heure (nombre)
etime(.)	Intervalle de temps écoulé
tic, toc	Chronomètre

Autres instructions (en vrac) :

fft(.)	Transformée de Fourier discrète	**end**	Index de fin par défaut
ifft(.)	TFD inverse	**input(.)**	Entrée de l'utilisateur
fftshift(.)	Échange des 2 moitiés d'un vecteur	**fprintf(.)**	Affichage dans un fichier ou à l'écran
filter(.)	Filtrage	**disp(.)**	Affichage à l'écran (texte et matrice)
boxcar(.)	Fenêtre rectangulaire	**num2str(.)**	Conversion d'un nombre en chaîne de caractères
rectwin(.)	Idem : fenêtre rectangul.	**int2str(.)**	Conversion d'un entier en chaîne de caractères
linspace(.)	Vecteur à espacement linéaire	**strcmp(.)**	Comparaison de chaînes de caractères
logspace(.)	Vecteur à espacement logarithmique	**eval(.)**	Exécution d'une chaîne de caractères comme une commande
sum(.)	Somme des éléments	**any(.)**	Vrai si au moins un élément non-nul
prod(.)	Produit des éléments	**all(.)**	Vrai si tous les éléments sont non-nuls
cumsum(.) **cumprod(.)**	Somme cumulée des éléments, produit cumulé	**find(.)**	Indices d'éléments non-nuls
nargin(.)	Nombre d'arguments d'entrée d'une fonction	**exist(.)**	Vérifie si une fonction ou une variable existe
nargout(.)	Nombre d'arguments de sortie d'une fonction	**isempty(.)**	Vrai pour une matrice vide

Voir aussi les commandes : `str2num`, `isequal`, `isinf`, `isnan`, `issparse`, `isstr`, …

En plus de la fenêtre rectangulaire déjà citée, il existe également plusieurs autres types de fenêtres utilisées en traitement du signal : `bartlett`, `barthannwin`, `blackman`, `blackmanharris`, `bohmanwin`, `chebwin`, `gausswin`, `hamming`, `hann`, `kaiser`, `nuttalwin`, `parzenwin`, `triang`, `tukeywin`

Instructions de contrôle :

break	Termine l'exécution d'une boucle (obsolète, remplacé par `return`)
continue	Passe à l'itération suivante en sautant les instructions suivantes dans une boucle
else	Sinon, utilisé avec `if`
elseif	Sinon si, utilisé avec `if`
end	Termine `for`, `if` et `while`
error(.)	Termine un programme et retourne le message d'erreur spécifié
for	Répétition
if	Instruction conditionnelle
otherwise	Sinon, utilisé avec `switch`
pause	Arrêt momentané d'un programme (attente d'appui sur une touche)
return	Retour à la fonction appelante ou au clavier
switch	Branchement conditionnel
while	Tant que

a) **Branchement conditionnel : if … then … else**

L'instruction `if` permet d'exécuter un bloc d'instructions en fonction de la valeur logique d'une expression. **Syntaxe :**

```
if <expression>
      <instructions>
end
```

Le groupe d'instructions est exécuté si seulement si l'expression est vraie.

Il est possible d'utiliser des branchements multiples, exemple :

```
if length(x)~=1
      disp('nonscalaire')
elseif isnan(x)
      disp('NaN')
elseif x>0
      disp('positif')
elseif x<0
      disp('negatif')
else
      disp('nul')
end
```

Si `A` et `B` sont deux matrices, il est possible d'utiliser l'expression `A==B` comme test logique. Le bloc d'instructions venant à la suite du `if A==B` est exécuté si et seulement si les deux matrices sont égales. Attention cependant à ce test d'égalité, car il renvoie un message d'erreur si les deux matrices n'ont pas les mêmes dimensions. Il est préférable d'utiliser la fonction `isequal` pour tester l'égalité entre plusieurs matrices.

b) **Branchement conditionnel multiple : switch**

Syntaxe :

```
switch <expression>
case <valeur1>
```

```
        <instructions1>
    case <valeur2>
        <instructions2>
    ...
    otherwise <instructions>
    end
```

L'expression doit être un scalaire ou une chaîne de caractères. Dans le cas scalaire, elle est comparée successivement avec `valeur1`, `valeur2`, etc. Dès que le premier test `<expression>==valeur` retourne la valeur vraie, le bloc d'instructions qui suit est exécuté. Si aucun test n'est validé, le bloc qui suit `otherwise` est exécuté.

Exemple :

```
switch mod(x,3)
case 0
    disp('multiple de 3')
case 1
    disp('x=3 [1]')
case 2
    disp('x=3 [2]')
otherwise
    disp('autre')
end
```

c) Boucle finie : for

La boucle `for` permet de répéter un bloc d'instructions un certain nombre de fois.

Syntaxe :

```
for <variable>=<expression>
    <instructions>
end
```

Exemple : édition d'une table de multiplication.

```
n=10;
t=1:n;      % le double point est ici utilisé pour effectuer une
            % boucle implicite, créant un vecteur constitué des %
            entiers de 1 à n.
T=[];
for i=1:n
    T=[T;t*i];
end
T
```

Bien sûr, plusieurs boucles peuvent être imbriquées.

Remarque : la variable peut être un vecteur et l'expression une matrice. Exemple :

```
>> for u=eye(4),u',end
ans =
     1     0     0     0
ans =
     0     1     0     0
ans =
     0     0     1     0
ans =
     0     0     0     1
```

d) Boucle infinie : while

Une boucle `while` permet d'exécuter un bloc d'instructions tant qu'une expression logique est vraie. **Syntaxe :**

```
while <expression>
      <instructions>
end
```

Exemple : calcul de la précision machine `eps`.

```
e=1;
while 1+e>1
      e=e/2;
end
epsilon=2*e
```

Remarques :

Bien que ce point soit en cours de correction avec les nouvelles versions de Matlab, dans la plupart des cas **l'exécution des boucles `for` et `while` reste lente comparativement à des instructions matricielles**. Or, il est souvent possible de remplacer une telle boucle par une ou plusieurs instructions matricielles équivalentes.

Exemple : création d'un vecteur contenant 1001 valeurs de la fonction sin(t).

Boucle "classique" :

```
n = 0;
for t = 0:.01:10      % de 0 à 10 par pas de 0,01
    n = n+1;
    y(n) = sin(t);
end
```

Programmation Matlab :

```
t = 0:.01:10;
y = sin(t);
```

Une autre façon d'accélérer un programme Matlab consiste à **pré-allouer de l'espace mémoire**, pour une variable dont la taille est censée être augmentée au cours d'une boucle, par exemple en créant une variable de la taille finale souhaitée, mais ne contenant que des valeurs nulles (instruction `zeros`).

Enfin, du point de vue de la rapidité d'exécution, il est toujours préférable de **programmer sous forme de fonctions** plutôt que de scripts. En effet, après le premier appel, une fonction est plus rapide car Matlab l'aura compilé en pseudo-code, alors qu'un script est toujours exécuté ligne par ligne.

5. EXERCICES D'INITIATION :

Exercice 1 : *Matlab utilisé comme une calculatrice*

On travaille directement dans la fenêtre de commande. Définir les scalaires `a` et `b` :

```
>>a=5;
>>b=2
>>a+b
```

Vérifier l'effet du point-virgule en fin de ligne, vérifier le contenu de la variable `ans`. Vérifier le contenu de l'espace de travail.

Tester les autres principaux opérateurs `(-, *, /, \, ^)` sur les scalaires `a` et `b`.

Effacer le contenu de l'espace de travail.

Exercice 2 : *opérations de base sur des vecteurs*

On travaille toujours dans la fenêtre de commande. Définir les vecteurs `A`, `B`, `C` et `D` :

```
>>A=[1 2 3] ,B=[2 3 4] ,C=[1,2,3], D=[2 3 4]'
```

Vérifier l'égalité entre `A` et `C`. Examiner le contenu de l'espace de travail. Quelle relation y a-t-il entre `B` et `D` ? Effectuer chacune des opérations suivantes et commenter les différents résultats :

```
A+B; A-B; A-2; A*B; A.*B; A'*B; A*B'; A*2; A.*2; A\B; A.\B; 2\A; 2.\A;
A\2; A.\2; A/B; A./B; A/2; A./2; 2/A; 2./A; B^A; B.^A; A^2; A.^2; 2^A;
2.^A; (A+i*B).'; (A+j*B)'; Z=A+2i; A(2)*2; A(3)+B(1); A(4); A(0); A(-
1); A(1.5);
```

Réexaminer le contenu de l'espace de travail. Effacer son contenu **en conservant les vecteurs** `A` et `B` pour l'exercice suivant.

NB : à propos des opérateurs de division matricielle à droite (`/`) et à gauche (`\`), consulter l'aide (`help mrdivide` et `help mldivide`) ainsi que l'annexe en fin de document.

Exercice 3 : *opérations de base sur des matrices, concaténations, extractions*

Si `A11` est une matrice de format `(m1,n1)` (noté aussi `m1xn1`), `A12` une matrice `m1xn2`, `A21` une matrice `m2xn1`, et `A22` une matrice `m2xn2`, alors on peut définir la matrice `A` :

$$A = \begin{bmatrix} A11 & A12 \\ A21 & A22 \end{bmatrix}$$

par *concaténation* des matrices précédentes. La syntaxe est la suivante :

```
A=[A11 A12; A21 A22]
```

On peut procéder de même avec plus de blocs encore. D'autre part, la fonction `zeros(m,n)` renvoie une matrice de `0` avec m lignes et n colonnes. De plus, comme on l'a déjà vu dans le cas particulier des vecteurs, les opérateurs `'` (transposition/conjugaison) et `.'` (transposition) permettent de transposer une matrice (ainsi, dans le cas général où A est complexe, A^H s'écrit donc `A'` et A^T s'écrit `A.'`).

Une instruction très utilisée dans Matlab (boucle implicite) est de la forme :

```
ind = deb : inc : fin
```

Elle permet de créer un vecteur ligne dont la première composante est `deb`, les composantes suivantes étant obtenues par incrémentation de `inc` jusqu'à ne pas dépasser `fin`. Si `inc` est strictement positif (resp. strictement négatif) et si `deb > fin` (resp. `deb < fin`), aucune composante n'est créée et l'on obtient un objet appelé matrice vide. Par défaut, l'incrément est `1`.

Pour l'extraction de vecteurs ou de matrices, on peut en fait utiliser les expressions `v(ind)`, comme on l'a vu dans l'exercice précédent, et `A(indi,indj)` si les vecteurs `ind`, `indi` et `indj` ne contiennent que des indices entiers bien situés, c'est-à-dire si `ind` ne contient aucun entier (str.) inférieur à `1` et (str.) supérieur au nombre de composantes de `v` et de même pour les 2 autres relativement au nombre de lignes et de colonnes de `A`.
D'autres possibilités vont être explorées dans cet exercice.

a) Toujours dans la fenêtre de commande, vérifier que les 4 lignes d'instructions suivantes définissent la même matrice :

```
>>M1=[1 2 3;2 3 4;3 4 5]
>>M2=[1 2 3                % entrée
2 3 4                      % entrée
3 4 5]
>>M3=[1:3;2:4;3:5]
>>M4=[A;B;3:1:5]
```

Expliquer l'utilisation du point-virgule ainsi que celle du double point.

Obtenir le résultat suivant d'au moins deux manières différentes :

```
M =
     1     2     3     1
     2     3     4     2
     3     4     5     6
```

b) Réinitialiser l'espace de travail et créer la matrice `A` suivante : $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$

Commenter le résultat des commandes suivantes :

```
>> A([1,3],2), A([1,3],[2,1,4])
```

Trouver la commande qui retourne le résultat suivant :

```
B =   5    6   7
      9    10  11
      13   14  15
```

Tester la commande `A(:)` puis expliquer le résultat de la commande :

```
>> A([1 2 3; 4 5 6])
```

Prévoir la réponse à la commande `>> A(10:-1:5)`

Comment créer un vecteur contenant les éléments de `A` dans l'ordre suivant :

`16 12 8 4 15 11` ... ?

Quel est le résultat de l'instruction `[A(1,:),5]` ? Expliquer.

Comment créer un vecteur contenant les éléments de `A` dans l'ordre suivant :

`1 2 3 4` ... ?

Effectuer la somme des éléments de la diagonale principale de `A` et vérifier le résultat avec la commande `trace`.

Quel est le résultat de `diag(diag(A))` ?

Créer une matrice identité 4x4 de deux manières différentes (commande `eye` et `diag`).

À l'aide de la commande `reshape`, transformer la matrice `A` en une matrice de dimension 2x8.
À l'aide des commandes `zeros` et `ones`, compléter la matrice `B` de façon à obtenir le résultat :

```
C =
     5     6     7     0
     9    10    11     0
    13    14    15     0
     1     1     1     1
```

À l'aide de la commande `rand`, créer une matrice aléatoire `D` de distribution uniforme et de dimension 4x4.

Utiliser la commande `isequal` pour vérifier l'équivalence entre les commandes `D\A` et `inv(D)*A`. Expliquer le résultat en passant en `format long`.

Obtenir le déterminant et le rang de la matrice `D`.

Proposer une méthode pour extraire les valeurs positives de la matrice `E=D-.5`.

Exercice 4 : *premier script, première fonction*

Soit : $\mathtt{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, $\mathtt{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Écrire un script (que l'on pourra appeler `script1.m`), qui :
- permette de donner une valeur à l'angle θ (utiliser la commande `input`),
- définisse la matrice `A` et le vecteur `x`,
- calcule `y=Ax`.

Expérimenter votre script avec plusieurs valeurs pour θ et pour le vecteur `x`.

À partir de la fenêtre de commande, comment :
- extraire la deuxième composante de `x` ?
- extraire l'élément $a_{1,2}$?
- extraire la première ligne de `A` ?
- extraire la deuxième colonne de `A` ?

Modifier le script pour en faire une fonction dont l'argument d'entrée est la valeur de θ, et tester la fonction.

Exercice 5 : *encore une matrice*
Après avoir consulté le `help` et expérimenté la commande `diag(a,k)`, où `a` est un vecteur et `k` un entier relatif, écrire un script qui génère la matrice d'ordre `n` suivante :

$$\mathtt{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}$$

Exercice 6 : *premières boucles, instruction* `if`*, commandes* `find`*,* `eval`*,* `num2str`

a) Recopier dans un fichier appelé `eratosthene.m` le script suivant. Faire exécuter ce script et expliquer l'algorithme.

```
n = 49;
T = ones(1,n);
for k = 2:sqrt(n)
     if T(k)==1
           T(2*k:k:n)=0;
     end
end
find(T)
```

b) À l'aide des commandes `eval` et `num2str`, et d'une boucle `for`, écrire un script qui génère automatiquement des matrices appelées `I2` à `I10` et correspondant aux matrices identités de dimensions 2x2, 3x3, …, 10x10.

Exercice 7 : *polynômes, premier graphe*

Dans MATLAB, un polynôme p(x) est représenté par la liste de ses coefficients dans l'ordre des puissances décroissantes.

Exemple : $p(x) = x^3 - 8x^2 - 32x - 13$ est représenté par : `p=[1  -8  -32  -13]`

En utilisant la commande `polyval`, évaluer le polynôme p(x) en `x = 2` et en `x =` π.
Déterminer les racines du polynôme p(x) (commande `roots`).

Soit le polynôme $q(x)=4x^2+3$, déterminer les coefficients du polynôme résultant du produit p(x).q(x), puis les coefficients et le reste de la division de p(x) par q(x) (commandes `conv` et `deconv`).

NB : attention au nombre d'arguments de sortie pour la fonction `deconv` (consulter le `help`).

Obtenir le graphe de p(x) sur l'intervalle [-5, 12] avec un pas de 0,01 (commande `plot`).

Exercice 8 : *`linspace`, opérations élément par élément, graphes multiples*

La fonction `linspace(a,b,n)` permet d'obtenir un vecteur ligne avec `n` composantes régulièrement réparties entre `a` et `b`.
Toutes les fonctions mathématiques classiques (sin, cos, exp,…) sont connues de Matlab et s'appliquent aussi à des vecteurs ou matrices, au sens élément par élément. Par exemple, `w=sin(v)` donnera un vecteur de même format que `v` et tel que $w_i=\sin(v_i)$. Enfin, si deux matrices `A` et `B` sont de même format alors `C=A.*B` et `D=A./B` donnent des matrices de même format que `A` et `B` avec $c_{i,j}=a_{i,j}b_{i,j}$ et $d_{i,j}=a_{i,j}/b_{i,j}$. Il est toléré que l'une des deux matrices soit un scalaire. Ceci fonctionne également avec l'opérateur de puissance `^`, comme on l'a vu dans le cas des vecteurs.

a) Définir un vecteur `x` avec 100 composantes permettant de mailler régulièrement l'intervalle $[0, 2\pi]$. On souhaite représenter la fonction $f_1(x) = \sin x$ sur cet intervalle : définir le vecteur `y` tel que $y_i=\sin(x_i)$ puis tracer la courbe.

b) Même question avec successivement :

$$f_2(x) = (x-1)(x+1), x \in [-2, 2]$$

$$f_3(x) = \frac{1}{1+x^2}, x \in [-3, 3]$$

$$f_4(x) = \cos\left(\frac{1}{0{,}1+x^2}\right), x \in [-3, 3]$$

Il est possible d'adapter le nombre de points du maillage au besoin. Représenter f_3 et f_4 sur la même figure.

Pour créer plusieurs figures au fur et à mesure des besoins, on pourra utiliser la commande `figure(numéro de figure)`.

Exercice 9 : *résolution d'une équation du second degré*

Écrire une fonction pour résoudre une équation du second degré, sur le modèle :

```
function [x1,x2]=resoud_eq_2d(a,b,c)
% calcul des racines de ax^2+bx+c = 0
% a, b et c peuvent être réels ou complexes et a doit être non nul
```

(dans le cas contraire, envoyer un message d'erreur à l'aide de la commande `error`).

La commande `roots` peut être utilisée mais uniquement pour vérifier le résultat.

Exercice 10 : *création d'un signal*

On se propose d'écrire un script qui permette de simuler le signal de mesure fourni par un débitmètre. On suppose que le débit mesuré est constant par morceaux mais un bruit de moyenne nulle vient se superposer à la mesure. Le signal résultant est supposé échantillonné à une fréquence de 100 Hz, pendant 12 secondes. On suppose qu'il est de moyenne nulle pendant les 4 premières secondes, puis de moyenne 10 lors des 2 secondes suivantes, de moyenne nulle lors des 3 secondes suivantes, puis de moyenne 15 pendant les 2 secondes suivantes, et enfin à nouveau de moyenne nulle pendant la dernière seconde. Le bruit de mesure est supposé gaussien, d'écart-type égal à 2.

a) Créer tout d'abord un vecteur `t` contenant la base de temps (de `0` à `12` s.).
b) Créer ensuite un vecteur `s` contenant le signal non-bruité.
c) Engendrer un vecteur `b` contenant les valeurs du bruit et effectuer les modifications nécessaires pour que ce bruit soit de moyenne nulle et de variance égale à 4. Vérifier les caractéristiques du bruit à l'aide des commandes `mean`, `std`, `var` et `hist`.
d) Créer enfin le vecteur `sb` par superposition de `s` et `b`.
e) Représenter le résultat en ayant soin de graduer l'axe des temps et de munir le graphe d'un titre, et d'un nom pour chacun des 2 axes.
f) Représenter sur un même graphe le signal bruité et l'original non-bruité, toujours en fonction du temps.

Conserver le contenu de l'espace de travail pour l'exercice suivant.

Exercice 11 : *filtre moyenneur*

a) Soit l'estimateur de la moyenne pour un signal x de N échantillons :

$$\hat{m} = \frac{1}{N}\sum_{k=1}^{N} x(k)$$

On souhaite développer une version séquentielle de cet estimateur, c'est-à-dire qui fournisse une nouvelle estimation au fur et à mesure de l'acquisition des nouveaux échantillons du signal $x(n)$. Établir l'expression de $\hat{m}(n)$ puis de $\hat{m}(n+1)$ et en déduire la relation de récurrence entre les deux. Écrire la fonction Matlab

correspondante et la tester sur le signal de l'exercice précédent (argument d'entrée : le signal, argument de sortie : le vecteur des moyennes successives).

b) Proposer une modification de l'algorithme précédent qui permette d'obtenir un estimateur de moyenne sur fenêtre glissante de durée donnée L (donc sur les L derniers échantillons). Le paramètre L est bien sûr un argument d'entrée de la fonction. Tester la nouvelle fonction sur le signal de l'exercice précédent. Lors de l'initialisation de l'algorithme, on pourra utiliser la fonction précédente.

c) Étudier la commande `median` de Matlab. À quoi correspond-elle ? Sur le même modèle que l'algorithme de moyenne sur une fenêtre glissante, écrire un programme qui calcule la valeur médiane sur fenêtre glissante et le tester sur le signal bruité.

d) Pour essayer d'améliorer la prise en compte des dynamiques rapides, on se propose de coupler l'algorithme de moyenne glissante avec un détecteur de saut. Celui-ci est très rudimentaire mais peut s'appliquer dans le cas d'un bruit de distribution quelconque. Il repose sur l'inégalité de Chebychev : $P(|X-m| \geq \lambda) \leq \frac{\sigma^2}{\lambda^2}$ pour tout $\lambda > 0$. Ceci s'écrit aussi : $P(|X-m| \geq \lambda\sigma) \leq \frac{1}{\lambda^2}$, σ représentant bien sûr l'écart-type du bruit. Écrire la fonction, la tester sur le signal en échelon et comparer avec les résultats précédents.

6. EXERCICES D'AUTO-ÉVALUATION :

Exercice 1 :

a) Créer le vecteur `V=[0 1 2 3 ... 49 50]` et déterminer sa longueur. Créer ensuite le vecteur `W` contenant les cinq premières valeurs de `V`, puis le vecteur `X` contenant les cinq premiers et les cinq derniers éléments de `V`. Enfin, créer le vecteur `Z=[0 2 4 ... 48 50]` à partir de `V`.

b) Créer à présent la matrice $M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \end{bmatrix}$. Extraire de cette dernière les matrices $N = \begin{bmatrix} 1 & 2 \\ 11 & 12 \\ 21 & 22 \end{bmatrix}$, $P = \begin{bmatrix} 8 & 9 & 10 \\ 18 & 19 & 20 \\ 28 & 29 & 30 \end{bmatrix}$ et $Q = \begin{bmatrix} 3 & 7 \\ 23 & 27 \end{bmatrix}$, ainsi que la matrice `R` obtenue en ne retenant qu'une colonne sur deux de `M`.

c) Créer les matrices `S=[2 4 6 8 ... 98 100]` et `T=[-1 -3 -5 ... -97 -99]` puis construire le vecteur `U=[-1 2 -3 4 -5 6 ... -99 100]`.

d) Construire une matrice aléatoire `H` à 4 lignes et sept colonnes (instruction `rand`). Déterminer le nombre d'éléments de `H` strictement supérieurs à 0,5 ainsi que leur position. Construire ensuite la matrice `K` obtenue à partir de `H` en remplaçant tous ses éléments inférieurs ou égaux à 0,4 par 0 et tous ses éléments strictement supérieurs à 0,4 par 1. Enfin, de même, créer la matrice `L` obtenue à partir de `H` en remplaçant tous ses

éléments inférieurs ou égaux à 0,4 par -5 et tous ses éléments strictement supérieurs à 0,4 par 12.

Exercice 2 :

Créer une matrice 10x4 constituée des puissances de 0 à 9 des nombres 2, 3, 4 et 5.

Réponse :

```
A =
           1           1           1           1
           2           3           4           5
           4           9          16          25
           8          27          64         125
          16          81         256         625
          32         243        1024        3125
          64         729        4096       15625
         128        2187       16384       78125
         256        6561       65536      390625
         512       19683      262144     1953125
```

Exercice 3 :

Créer une matrice de la forme :

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 94.4043 & 107.8118 & 97.3439 & 0 \\ 0 & 104.4365 & 105.6896 & 88.1222 & 0 \\ 0 & 90.5010 & 91.7829 & 77.9768 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

où les éléments de la matrice 3x3 centrale sont choisis suivant une loi gaussienne de moyenne 100 et d'écart-type 10.

Exercice 4 :

Créer une fonction avec un entier `n` comme paramètre d'entrée et la matrice `M` suivante comme paramètre de sortie :

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1/n & 2 & \frac{n-1}{n} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 2/n & 3 & \frac{n-2}{n} & \ddots & 0 & 0 & 0 \\ 0 & 0 & 3/n & 4 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 4/n & \ddots & 3/n & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & n-1 & 2/n & 0 \\ 0 & 0 & 0 & 0 & \ddots & \frac{n-1}{n} & n & 1/n \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & n+1 \end{bmatrix}$$

Exercice 5 :

Générer le vecteur `a=[0 1 2 3 4 5 6]`. Sans utiliser de structure bouclée ni la fonction `sum`, en déduire le vecteur `b=[0 1 4 9 16 25 36]` et calculer la somme :

$$s = \sum_{n=0}^{6} b_n e^{-a_n/10}.$$

Résultat obtenu :

```
s =
   56.4927
```

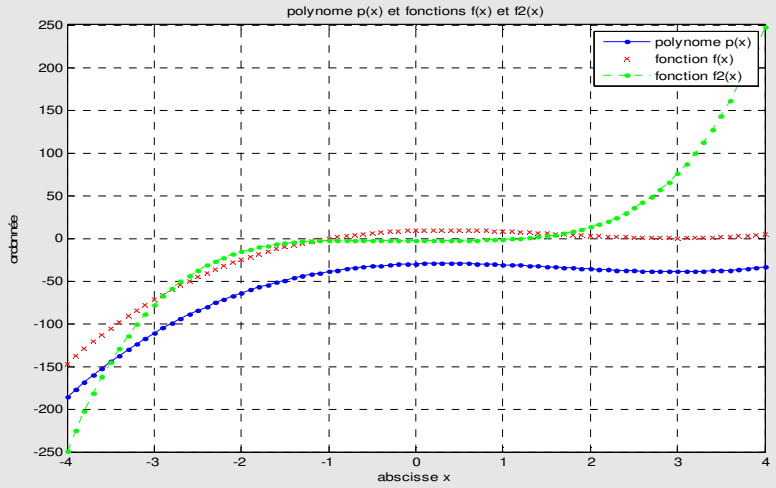
Exercice 6 :

Créer un vecteur `x` s'étendant de -4 à 4 avec un pas de 0,1.

Créer le vecteur `p=[1 -5 3 -30]` contenant les coefficients du polynôme $p(x)$.

Soient les fonctions $f(x) = (x-3)^2(x+1)$ et $f_2(x) = \frac{x^5-3}{\sqrt{x^2+1}}$, représenter sur un seul et même graphe le polynôme $p(x)$ et les deux fonctions sur l'intervalle `x`. La figure sera munie d'un titre, d'axes, d'une grille de graduation et d'une légende.

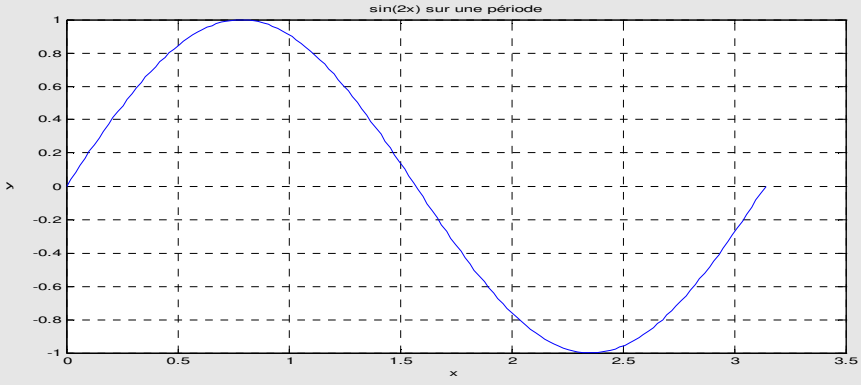
Figure obtenue :



Exercice 7 :

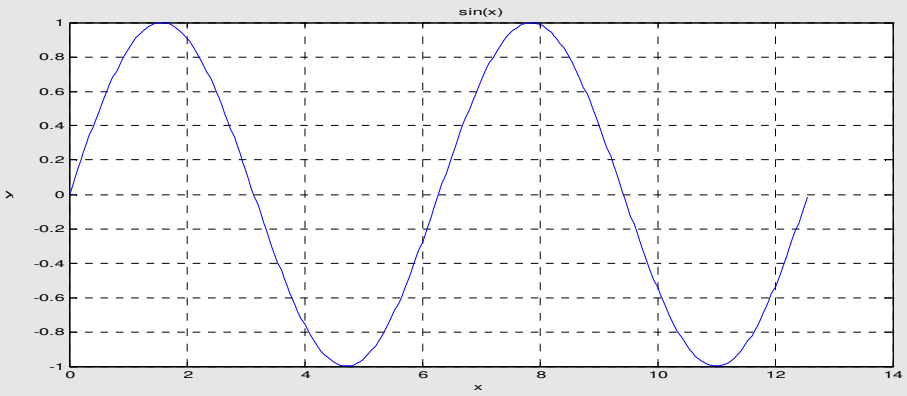
a) Tracer la fonction $y = \sin(2x)$ sur une période. L'intervalle de valeurs sera créé avec la commande `linspace` et comprendra 150 valeurs.

Figure obtenue :



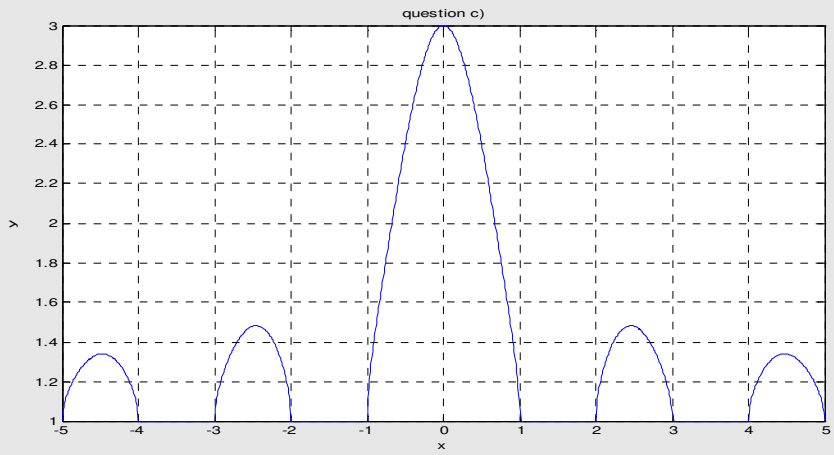
b) Tracer la fonction $y = \sin(x)$ dans l'intervalle $[0, 4\pi]$ avec un pas de 0,05.

Figure obtenue :



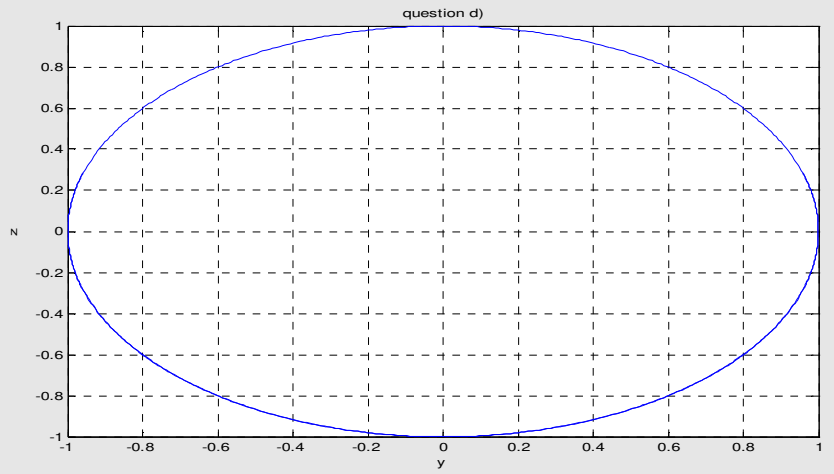
c) Tracer la fonction $y = \left|3^{\sqrt{\text{sinc}(x)}}\right|$ sur l'intervalle $[-5, 5]$.

Figure obtenue :



d) Soient $y = \sin(x)$ et $z = \cos(x)$. Tracer la courbe de z en fonction de y sur le même intervalle que précédemment.

Figure obtenue :



Exercice 8 :

a) Soient les deux polynômes suivants :

$$a(x) = x^3 + 2x^2 + 3x + 4$$

$$b(x) = x^3 + 4x^2 + 9x + 16$$

Déterminer les coefficients du polynôme $c(x) = a(x).b(x)$.

Réponse :

```
c = 1     6    20    50    75    84    64
```

b) Soit le polynôme $d(x) = a(x) + b(x)$. Calculer ses racines. À partir des racines, retrouver le polynôme.

Réponses :

```
r =
  -2.2879
  -0.3560 + 2.0601i
  -0.3560 - 2.0601i

d2 =
    2.0000    6.0000   12.0000   20.0000
```

c) Calculer le quotient et le reste de la division $c(x)/b(x)$.

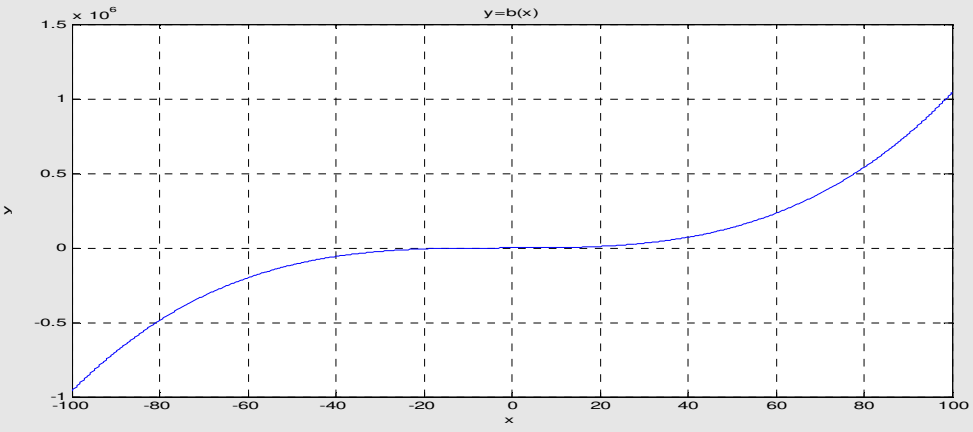
Réponses :

```
Q =
     1     2     3     4

R =
     0     0     0     0     0     0     0
```

d) Tracer la courbe $b(x)$ en fonction de x, entre -100 et 100 (10000 valeurs).

Figure obtenue :



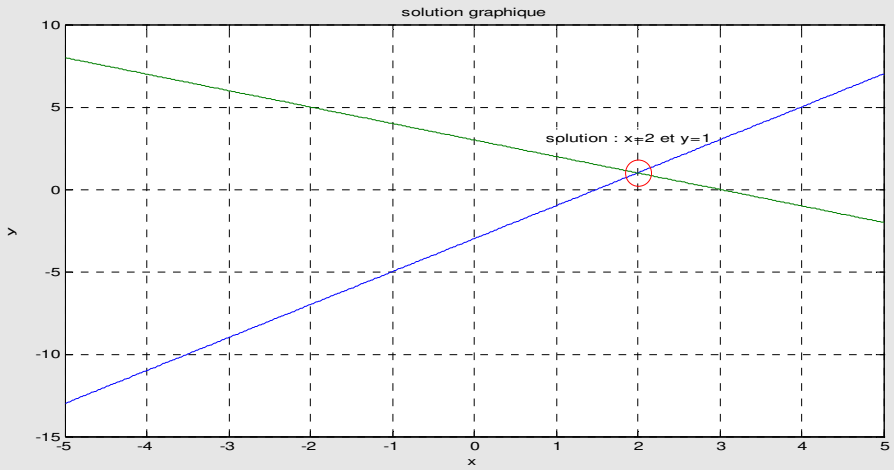
Exercice 9 :

Soit le système d'équations linéaires suivant : $\begin{cases} -2x + y = -3 \\ x + y = 3 \end{cases}$

a) À l'aide de Matlab, résoudre graphiquement ce système d'équations.

Réponse :

```
soluce =
             2
             1
```



b) Toujours sous Matlab, résoudre le système avec l'instruction `inv`.

Réponse :

```
soluce2 =
             2
             1
```

Exercice 10 :

À l'aide de la commande `backslash` (\), résoudre le système d'équations suivant :

$$\begin{cases} 2x + 3y + 4z = 3 \\ x - y - z = 0 \\ -x + 4y + z = 5 \end{cases}$$

Réponse :

```
ans =
    0.7778
    1.6667
   -0.8889
```

7. ANNEXE : précisions sur les opérateurs `/` et `\`

Soit l'opération `A\B`, il s'agit d'une division à gauche (`backslash`), la réponse est telle que `A*ans = B`.
Lorsque `A` est une matrice carrée de rang plein et `B` un vecteur colonne avec un nombre de lignes égal à la dimension de `A`, il s'agit d'un problème classique de résolution d'un système d'équations linéaire, dont l'unique solution est donnée par $A^{-1}B$.
Plus généralement, lorsqu'il n'y a pas qu'une solution, l'opérateur `backslash` calcule une solution au sens des moindres carrés et retourne une solution basique comportant au plus `r` valeurs non-nulles, où `r` est le rang de `A`. La solution à norme minimale peut être obtenue à l'aide de la commande `pinv`.

`x=A\B` résout donc le système d'équations linéaire $A * x = B$ où les matrices A et B ont le même nombre de lignes.
Si A est scalaire, `A\B` effectue la division des éléments de B par A.
Si A est une matrice carrée (n, n) et B un vecteur colonne de longueur n ou une matrice de n lignes, `x=A\B` est la solution à l'équation $A * x = B$ (si elle existe).
Si A est une matrice rectangulaire (m, n) et B un vecteur colonne de longueur m ou une matrice de m lignes, `A\B` renvoie une solution au sens des moindres carrés de l'équation $A * x = B$.

Bien sûr, `x=B/A` résout cette fois le système d'équations linéaire $x * A = B$ où les matrices A et B ont le même nombre de colonnes.
Si A est scalaire, `B/A` effectue la division des éléments de B par A.
Si A est une matrice carrée (n, n) et B un vecteur ligne de longueur n ou une matrice de n colonnes, `x= B/A` est la solution à l'équation $x * A = B$ (si elle existe).
Si A est une matrice rectangulaire (m, n) et B un vecteur ligne de longueur n ou une matrice de n colonnes, `B/A` renvoie une solution au sens des moindres carrés de l'équation $x * A = B$.

Remarque : soit X solution de $x * A = B$ (`X=B/A`), on a : $(X * A)^T = B^T$ i.e. $A^T * X^T = B^T$, donc `X'=A'\B'`, soit `X=(A'\B')'`.