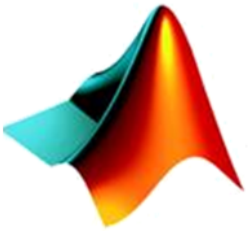
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abdelhamid Ibn Badis de Mostaganem
Faculté des Sciences Exactes et de l'Informatique

Cours d'Outils de Programmation pour les Mathématiques 2



MATLAB®

Ali Merina Houria

Année universitaire 2020/2021

Contents

Préface

Nous avons choisi MATLAB pour illustrer ce cours car, parmi les systèmes actuels, il nous semble être le seul à être à la fois d'accès facile, très diffusé (aussi bien par le nombre d'utilisateurs que par la diversité des machines le supportant). MATLAB est un langage de haut niveau et un environnement interactif qui permet de résoudre des problèmes d'ingénierie et scientifiques à grande échelle. Il permet au programmeur de définir et de manipuler des fonctions, de visualiser des expressions mathématiques, de résoudre des systèmes d'équations, d'intégrer des expressions algébriques, et bien plus encore.

Introduction

La programmation est devenue un outil indispensable dans notre vie moderne. Elle nous permet de créer des applications et des sites web, de gérer des bases de données, d'automatiser des tâches et bien plus encore, ce qui peut aider à réduire le temps et les coûts nécessaires pour effectuer certaines tâches. La programmation peut nous aider aussi à résoudre des problèmes complexes et à trouver des solutions innovantes.

Matlab est un langage de programmation très populaire dans les domaines des sciences, de la technologie, de l'ingénierie et des mathématiques (STEM). Il est utilisé pour résoudre des problèmes complexes et pour analyser des données. Il peut être utilisé pour créer des modèles mathématiques, des simulations numériques et des applications graphiques. Les étudiants en STEM peuvent tirer parti de Matlab pour résoudre leurs problèmes scientifiques et technologiques. Matlab offre une variété d'outils puissants qui peuvent aider les étudiants à comprendre leurs sujets plus rapidement et plus efficacement. Les étudiants peuvent également utiliser Matlab pour créer des visualisations complexes qui peuvent être utilisées pour expliquer leurs concepts à un public plus large.

MATLAB est un logiciel de calcul numérique commercialisé par la société **MathWorks**. Il a été initialement développé à la fin des années 70 par Cleve Moler, professeur de mathématique à l'université du Nouveau-Mexique puis à Stanford, pour permettre aux étudiants de travailler à partir d'un outil de programmation de haut niveau et sans apprendre le Fortran ou le C. C'est est un logiciel de calcul matriciel à syntaxe simple (relativement à des langages évolués comme C, C++). Avec ses fonctions spécialisées. L'avantage indéniable de MATLAB réside en une syntaxe assez simple et intuitive. L'abrévation MATLAB signifie en anglais "**Matrix laboratory**", tous les objets en jeu sont des matrices, y compris les scalaires (qui sont considérés comme matrices 1×1). En MATLAB, il existe deux modes de fonctionnement :

1. **mode interactif**: MATLAB exécute les instructions au fur et à mesure qu'elles

sont données par l'usager.

2. **mode exécutif:** MATLAB exécute ligne par ligne un "fichier.m" (programme en langage MATLAB).

Matlab trouve ses applications dans de nombreuses disciplines. Il constitue un outil numérique puissant pour la modélisation des systèmes physiques, la simulation des modèles mathématiques, la conception et la validation d'applications (tests en simulation et expérimentation) . Le logiciel de base peut être complété par de multiples boîtes à outils qu'on appelle toolboxes. Celles-ci sont des bibliothèques de fonctions dédiées à des domaines particuliers. On peut citer par exemple : l'Automatique, le traitement du signal, l'analyse statistique, l'optimisation.

Le document est structuré principalement en trois chapitres :

1 L'objectif du chapitre 1 est de donner une introduction à Matlab avec des indications pour l'installation et une description de l'interface graphique.

2 Le chapitre 2 est dédié aux fonctions, les fichiers scripts, comment les utiliser et comment les construire, ainsi , qu'aux premiers éléments de programmation : les opérateurs logiques, les structures conditionnelles et les boucles.

3 Dans le chapitre 3, on traite de l'utilisation des scripts afin d'organiser le travail et aussi des moyens de générer des graphiques en deux et trois dimensions.

Chaque chapitre contient une section d'une série d'exercices, elle sert à aider l'étudiant à pratiquer les notions vues en cours.

La partie d'annexes est constitué de textes de travaux pratiques élaborés pour étudier pratiquement les instructions Matlab déjà présentés en séances de cours.

0.1 Motivation

0.1.1 Recherche des zéros d'une fonction

Problème: On cherche x_0 tel que $f(x_0) \approx 0$.

Un exemple pratique montrant une des fonctionnalité de MATLAB est la recherche d'un zéro d'une foncton. Cependant, il existe une commande appelée **fzero** qui a deux paramètres d'entrées, la fonction f et le point initiale x_0 qui permet de trouver le zéro de f autour de $\mathbf{x}_0$.

Il faut fournir alors, d'une part la fonction f elle-même, et d'autre part une estimation de x_0.Comme toute langage de calcul numérique approché la commande **fzero** est basée sur un algorithme de résolution, l'efficacité de cet algorithme est comme toujours dépendante de la valeur estimée choisie.

Par exemple on cherche le zéro de la fonction suivante :

$$f(x) = \cos(x) - x^2$$

Les méthodes graphiques peuvent généralement trouver une estimation de x_0, comme indiqué ci-dessous (**??**), les graphes des fonctions $x \mapsto x^2$ et $x \mapsto \cos(x)$, se coupent en deux points sur $[-\pi, \pi]$, au voisinage de $1, -1$ donc x_0 peut être choisi égale à l'une de ces deux valeurs.

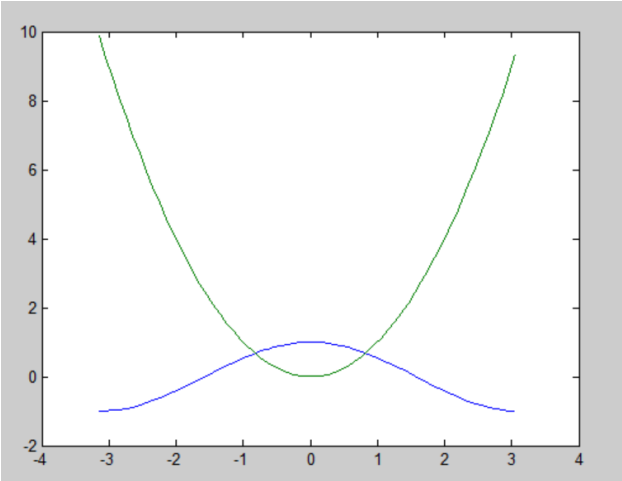


Figure 1 : Estimation de x0 graphiquement

On peut trouver numériquement la solution et résoudre ce problème en définissant une fonction anonyme à l'aide de l'opérateur "@" si on prend $x_0 = 1$, on trouve le premier zéro, si on prend $x_0 = -1$, on trouve le deuxième :

```
≫ f = @(x) (cox(x)-x^2) ;
≫fzero(f,1)
ans =
      0.8241
≫fzero(f,-1)
ans =
      - 0.8241
```

Donc, les zéros de la fonction $f(x) = \cos(x) - x^2$ sont approximativement -0.8241 et 0.8241.

Généralités et prise en main

Ce chapitre montre comment utiliser MATLAB pour faire des opérations arithmétiques. On montre ensuite comment définir des variables: scalaires, vecteurs, matrices (l'opérateur d'affectation) et utiliser ces variables dans les calculs arithmétiques.

0.2 Démarrage de MATLAB

On suppose que le logiciel est installé sur l'ordinateur et que l'utilisateur peut démarrer le programme. Une fois le programme démarré, la fenêtre du bureau MATLAB s'ouvre avec la disposition par défaut (0.2). La mise en page a une barre d'outils en haut et quatre fenêtres en dessous.

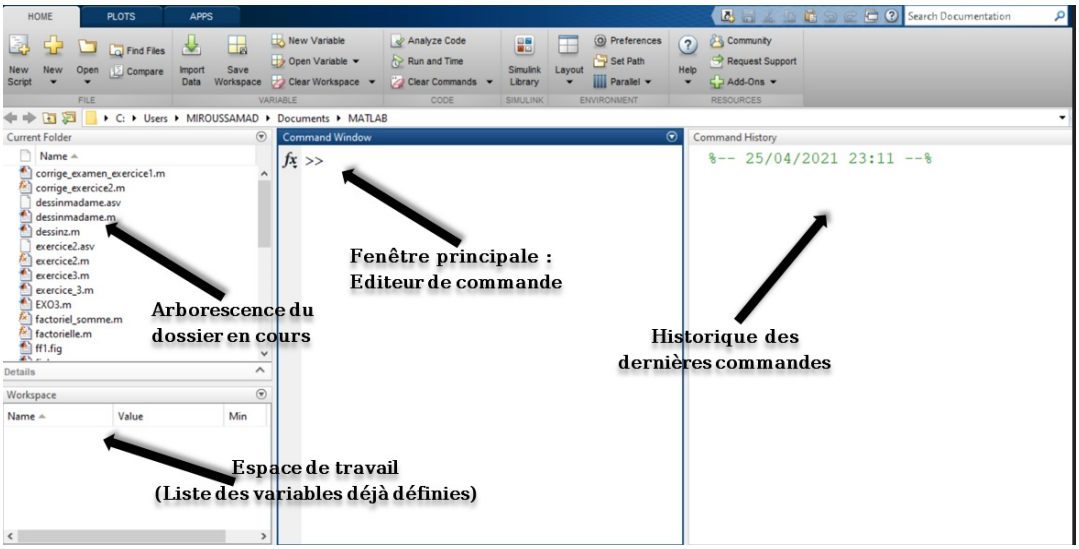


Figure 2: Interface Graphique de Matlab

Le logiciel propose un véritable environnement de travail composé de multiples fenêtres. Nous pouvons distinguer quatre blocs :

1. **Current directory** (répertoire courant) : permet de naviguer et de visualiser

le contenu du répertoire courant de l'utilisateur. Les programmes de l'utilisateur doivent être situés dans ce répertoire pour être visible et donc exécutable.

2. **Command window** (console d'exécution) : à l'invite de commande « >> », l'utilisateur peut entrer les instructions à exécuter. Il s'agit de la fenêtre principale de l'interface. Le symbole >> situé à gauche indique que l'interpréteur est prêt à recevoir une commande, chaque ligne etant exécutée immédiatement après la touche Entrée.

   Pour mettre fin à la session de travail et quitter l'environnement MATLAB, il suffit de taper la commande :

   >> quit.

   On peut vider cette fenêtre avec la commande :

   Edit −> clear Command Window.

3. **Command history :** historique des commandes que l'utilisateur a exécutées. Il est possible de faire glisser ces commandes vers la fenêtre de commande. On peut vider cette fenêtre avec la commande :

   Edit −> clear Command History.

4. **Workspace** (espace de travail) : permet de visualiser les variables définies, leur type, la taille occupée en mémoire. On peut vider l'espace de travail avec la commande :

   Edit −> clear Workspace.

0.3 Le Help

Une façon efficace de découvrir Matlab est d'utiliser **le help.** Pour obtenir de l'aide sur un sujet, une instruction ou une fonction, on tape help suivi par le sujet, l'instruction ou la fonction désirée.

1. **helpwin** ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations.

2. **help** : donne la liste de toutes les commandes par thèmes.

3. **help nom** : décrit la fonction nom.m (si on connait exactement la commande).

4. **lookfor nom** : recherche une instruction à partir du mot clé **nom** (si on ne connait pas la commande) exemple **lookfor** plot donne une liste de toutes les commandes qui ont un rapport avec plot. **lookfor sin** : on trouvera que la commande sin correspond, sans surprise, à la fonction **sinus**.

0.4 Opérations arithmétiques sur les scalaires

Dans cette partie, on aborde uniquement les opérations arithmétiques sur des scalaires, qui sont des nombres (tableau 1×1). Comme cela sera expliqué plus loin dans le chapitre, les nombres peuvent être utilisés dans les calculs arithmétiques directement (comme avec une calculatrice) ou ils peuvent être affectés à des variables, pour ensuite être utilisées dans les calculs.

Les symboles des opérations arithmétiques sont:

Opération	Symbole	Exemple
Addition	$+$	$5+3$
Soustraction	$-$	$5-3$
Mutiplication	$*$	$5*3$
Division	$/$	$5/3$
Puissance	^	$5\hat{}3\ (5^3 = 125)$

Example 0.4.1

Si l'on regarde ce que donne l'exemple d'addition :

$$\gg 2+3$$
$$ans = \quad 5$$

On peut remarquer que lorsqu'aucune variable d'affectation n'est pas spécifiée pour stocker le résultat de l'opération, MATLAB stocke le résultat dans une variable appelée **ans** (diminutif pour answer). Cette variable sera écrasée à chaque nouvelle opération. Pour éviter cela, on peut spécifier le nom d'une variable pour stocker le résultat. On pourra alors réutiliser le résultat de l'opération plus tard. Par exemple :

$$\gg x = 3+2$$
$$x = \quad 5$$

x prend alors la valeur 5. Cette variable peut alors être réutilisée dans un calcul suivant :

$\gg x/2$

$ans =$

2.5000

0.4.1 Priorité des opérateurs

MATLAB exécute les calculs selon l'ordre de priorité affiché ci-dessous. Cet ordre est le même que celui utilisé dans la plupart des calculatrices.

1. ()
2. ^
3. * /
4. +

Example 0.4.2

$\gg 3 + 2 * 4\hat{}2$

16 1) Evaluation de $4\hat{}2$ car $\hat{}$ a la priorité par rapport aux autres opérateurs $*$ et $+$

32 2) Evaluation de $2 * 16$ car $*$ a la priorité par rapport à l'opérateur $+$

$ans = 35$ 3) Pour finir, évalutation de $3 + 32$

Example 0.4.3

$\gg ((3 + 2) * 4)\hat{}2$

5 1) Evaluation de $3 + 2$ car () à la priorité sur $*$

20 2) Evaluation de $5 * 4$ car () à la priorité sur $\hat{}$

$ans = 400$ 3) Pour finir, évalutation de $20\hat{}2$

0.4.2 Utilisation et affectation d'une variable

Une variable est une zone mémoire qui stocke une valeur et qui peut être référencée par un nom. Elle est utilisée pour stocker des données et des informations qui peuvent être modifiées au cours de l'exécution d'un programme..

On distingue plusieurs types de variable selon les données qu'elles servent à stocker (nombre, tableau, matrice, structure). Contrairement à d'autres langages de programma-

tion, sous MATLAB le type des variables n'a pas besoin d'être spécifié, MATLAB infère le type d'une variable en fonction de la donnée que l'on y stocke.

Les variables sont créées lors de la première affectation (opération qui permet d'attribuer une valeur à une variable).

$A = 0$ on affecte la valeur 0 à la variable A, si A n'existait pas auparavant, elle est créée.

$B = 11$ on affecte la valeur 11 à la variable B.

On peut alors utiliser une variable dans un calcul ou pour affecter son contenu à une autre variable.

$A = B$ on affecte la valeur contenue dans B à la variable A, A vaut à présent 11, B reste inchangée.

$A = A + 1$,on affecte la valeur contenue dans A incrémentée de 1 à la variable A, A vaut à présent 12.

Sous MATLAB, l'affectation se fait à l'aide de l'opérateur $=$

$$\boxed{\begin{array}{l} \gg A = 0 \\ \gg B = 11 \\ \gg A = B \\ \gg A = A + 1 \end{array}}$$

Remark 0.4.1 1) Le symbole ";" est utilisé par Matlab dans deux situations différentes:

-Pour ne pas afficher le résultat d'une commande(par exemple :A=0; si on regarde dans le workspace, on va voir que la variable a été crée, mais matlab ne va pas l'afficher).

-Pour séparer les lignes d'un tableau (matrice, vecteur).

2) Il faut éviter d'utiliser comme **nom de variable** des noms déjà employés comme nom de fonctions (par ex : min, max, exp ...). MATLAB générera également une erreur si un des mots-clés réservés suivant est utilisé : for, end, if, while, function, return, elseif, case, otherwise, switch, continue, else, try, catch, global, persistent, break. Ces mots-clés font partie de la syntaxe du langage MATLAB, nous verrons dans la suite des exemples d'utilisation de certains de ces mots-clés.

3) On a un choix très ouvert pour le nom, mais ce nom doit commencer par une lettre. Par exemple, si vous commencez par un nombre 12A égale 3, ça ne pourra pas marcher. Et en effet, vous avez une erreur.12A= 3 Error: Unexpected MATLAB expression

4) Il n'est pas nécessaire de déclarer une variable. Sa taille ou son type sont définis automatiquement.

5) Le nombre de charactères doit être inferieur à 63 (namelengthmax).

6) Aussi, il faut savoir que si vous avez une variable en minuscules et une variable qui s'appelle presque pareil, mais en majuscules, MatLab va considérer ces 2 variables comme différentes.

7) Le résultat d'une opération est affiché avec 5 chiffres significatifs. La précision par défaut correspond au format short. Pour plus de précision, on choisira le format long.

Aspects élémentaires

Il est également possible de lister les variables du workspace via l'instruction « whos, who »,

who affichage des variables dans l'espace de travail,

whos affichage détaillé des variables dans l'espace de travail.

On peut aussi utiliser la commande **class**: elle retourne le nom de la classe à laquelle appartient la variable donnée comme argument.

La commande « **clear** » permet de supprimer une variable du workspace (« **clear all** » les supprime toutes, clear var1 var2 efface que les variables var1 et var2). Toutes les commandes tapées dans la Command window peuvent être retrouvées et éditées grâce aux touches de direction. Appuyez sur la touche ⇑ pour remonter dans les commandes précédentes, ⇓ pour redescendre.

Des commandes permettent de vérifier le type d'une variable.

isnumeric, **ischar** ou **islogical** permet de vérifier si une variable est respectivement numérique, de type caractères ou logique. Ces fonctions renvoient 0 si c'est faux, et 1 si c'est vraie.

Pour les variables de types caractères: **char,** la déclaration se fait à l'aide du symbole apostrophe, on peut aussi lier des mots à l'aide des crochets.

Example 0.4.4

On veut créer la chaine de caractères Good Moring, on fait :

```
≫s1='Good';
≫s2='Morning';
≫s=[s1 s2]
s = Good Morning
```

Constantes prédéfinies

Il existe des symboles auxquels sont associés des valeurs prédéfinies. En voici quelques uns :

pi le nombre π, i ou j le nombre complexe, *eps* :2.2204e-16, NAN not a number

Attention, ces valeurs peuvent être écrasées si le symbole est redéfini

```
≫pi=1
pi=
1
```

```
≫clear pi
≫pi
ans =
        3.1416
```

Matlab peut aussi manipuler explicitement des nombres complexes:

```
≫a = 2 + i*3;
≫ b = 1 - i;
≫ a + b
ans=
     3 + 2i
```

0.4.3 Quelques fonctions mathématiques

Fonctions trigonométriques :

Symbole	Fonction
sin(x)	la fonction sinus
cos(x)	la fonction cosinus
tan(x)	la fonction tangente
asin(x)	la fonction arcsinus
acos(x)	la fonction arccosinus
atan(x)	la fonction arctangente
sinh(x)	la fonction sinus hyperbolique
cosh(x)	la fonction cosinus hyperbolique
tanh(x)	la fonction tangente hyperbolique

Fonctions puissances, exponentielle, racine carrée et logarithmes :

power ou (^)	sqrt	exp	log	log10

Fonctions réels vers entiers (arrondi, troncature. . .):

round(x) : entier le plus proche de x

floor(x) : arrondi par défaut de x.

ceil(x) : arrondi par excés de x.

Fonctions autres :

sign(var): (signe) retourne 1 si var>0, -1 si var<0 et 0 si var=0.

abs(var): module de var.

real(var): partie réelle de var.

imag(var): partie imaginaire de var.

gcd(var1,var2): plus grand diviseur commun des entiers var1 et var2.

lcm(var1,var2): plus petit multiple commun des entiers var1 et var2.

QUELQUES GÉNÉRALITÉS

1. 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.

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

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

4. 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.

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

6. 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.

7. Le symbole "%" sert à faire un commentaire sous Matlab, toute instruction écrite aprés "%" ne sera pas évaluer.

8. Pour une liste des fonctions mathématiques élémentaires, taper

help elfun

Pour une liste de fonctions mathématiques plus avancées ou sur les matrices, taper :**help specfun, help elmat**

Quelques fonctions, comme **sqrt** et **sin** sont des fonctions (built-in). Ceci signifie qu'elles font parties intégrantes du noyau Matlab.

0.5 Définir les vecteurs

La méthode la plus simple pour définir un vecteur est de donner sa description explicite à l'aide de la commande [ ], par exemple :

≫ v=[1 2 3] vecteur ligne
≫u=[1;2;3] vecteur colonne

On peut concaténer deux vecteurs :

```
≫vec1 = [1 3 5];
≫vec2 = [9 10 11];
≫vec = [vec1 vec2]
vec=
     1 3 5 9 10 11
```

Et on peut également prendre la transposée(ou par la commande **transpose**) pour passer d'une ligne à une colonne ou réciproquement :

```
≫vec1 = [1 3 5];
≫vec = vec1'
vec=
     1
     3
     5
```

Remark 0.5.1 Il n'est pas nécessaire de définir la taille d'un vecteur (c'est automatique), par contre la commande **length()** permet de retourner cette quantité.

```
≫length(vec)
ans =
     3
```

Lorsque l'on veut afficher le graphe d'une fonction **f** telle que y = f(x) sous Matlab, il faut définir deux vecteurs : un pour le vecteur x, l'autre pour le vecteur y = f(x). Matlab ne fait pas du calcul formel, il faut donc discretiser l'axe des x. On utilise pour ce faire la fonction **linspace(a,b,n).** Cette commande génère un vecteur ligne de **n** éléments espacés linéairement entre a et b. Par défaut n = 100.

Une autre méthode pour générer des vecteurs espacés linéairement consiste à utiliser l'opérateur (:), on fait alors : [a:s:b]. On crée alors un vecteur entre a et b avec un espacement s :

```
≫vec=[1:2:10]
vec =
1 3 5 7 9
```

0.5.1 Manipuler un vecteur

Il est important également de se familiariser à la manipulation de vecteurs, c'est-'a-dire être capable d'extraire des sous-vecteurs à l'aide des indices. Le **kième** élément d'un vecteur vec peut être affiché grâce à la commande **vec(k)**. **k** doit être un entier sinon Matlab retournera une erreur :

```
≫ vec = linspace(1,10,10)
vec =
      1 2 3 4 5 6 7 8 9 10
≫vec(4)
ans =
       4
≫ vec(4.2)
Subscript indices must be real positive integers or logical.
```

On peut également utiliser l'opérateur (:) pour extraire un sous-vecteur :

```
≫ vec = linspace(1,89,9)
vec =
        1  12 23 34 45 56 67  78 89
≫ vec(3:6)
ans =
     23 34 45 56
≫ vec(4:2:8)
ans =
      34 56 78
≫ subvec=[1 3 5]; vec(subvec)
ans =
      1 23 45
```

le mot-clé **end** permet d'accéder au dernier élement.

```
≫vec(end) %dernier élément
ans=
       89
```

0.5.2 Opérations sur les vecteurs

Les opérations algébriques usuelles +, -, *, / doivent être prises avec précautions pour les vecteurs. La somme et la différence sont des opérations termes à termes, et nécessitent donc des vecteurs de même dimension. Le produit * est le produit matriciel. Nous y reviendrons dans la section sur les matrices. Pour utiliser la mutliplication ou la division termes à termes on doit remplacer * par .* et / par ./

```
≫ vec = [1 3 5 6]; vec2 = [10 20 30 40];
≫vec+vec2
ans =
     11 23 35 46
≫vec-vec2
ans =
     -9  -17 -25 -34
≫vec.*vec2
ans =
      10 60 150 240
≫vec2.*vec
ans =
      10.0000 6.6667  6.0000 6.6667
```

De la même manière que pour les scalaires, on peut appliquer toutes les fonctions définies précédemment pour les vecteurs. Par exemple :

```
≫vec = linspace(1,10,10);
≫out=sqrt(vec)
out =
Columns 1 through 5
1.0000  1.4142 1.7321 2.0000  2.2361
Columns 6 through 10
2.4495 2.6458 2.8284 3.0000 3.1623
≫out2=vec.^2
out2 =
        Columns 1 through 5
          1 4 9 16 25
        Columns 6 through 10
         36 49 64  81 100
≫ out3=cos(vec)
out3 =
        Columns 1 through 5
       0.5403 -0.4161 -0.9900 -0.6536 0.2837
        Columns 6 through 10
      0.9602  0.7539  -0.1455 -0.9111 -0.8391
```

Il existe aussi des commandes qui sont propres aux vecteurs.

- **sum(x)** : somme des éléments du vecteur x.
- **prod(x) :** produit des éléments du vecteur x.
- **max(x) :** plus grand élément du vecteur x.
- **min(x)** : plus petit élément du vecteur x.
- **mean(x) :** moyenne des éléments du vecteur x.
- **sort(x) :** ordonne les éléments du vecteur x par ordre croissant.
- **fliplr(x)** : renverse l'ordre des éléments du vecteur x.
- **[J, I] = sort(x)** : retourne en plus les indices des élements d'un vecteur x.
- **find(x)** : retourne les indices non nuls d'un vecteur x.

- **[J, I] = find(x)** retourne des lignes (dans le vecteur I) et des colonnes (dans le vecteur J) des élements non nuls du x.
- **cumsum(x)** : vecteur contenant la somme cumulée des éléments de x.
- **prod(x)** : produit des éléments de x.
- **cumprod(x)** : vecteur contenant le produit cumulé des éléments de x.
- **diff(x)** : vecteur des différences entre deux éléments consécutifs de x.
- **mean(x)** : moyenne des éléments de x.
- **median(x)** : médiane.
- **std(x)** : écart type.

sqrt	exp	log	sin	cos
tan	asin	acos	atan	round
floor	ceil	abs	angle	conj

Ces fonctions peuvent être appliquées aux vecteurs même pour les matrices

Certaines fonctions de Matlab s'appliquent à l'ensemble d'un vecteur. Lorsqu'on les applique à des matrices, elles opèrent colonne par colonne.

Application : Construire un vecteur quelconque et essayer les fonctions ci-dessus.

0.6 Définir une matrice

Une matrice va se définir de façon similaire à un vecteur avec la commande [ ]. On définit la matrice A :

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

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

Cela est équivalent à

```
≫A = [1 2
4 3]
A =
    1 2
    4 3
```

Une matrice est composée de **m** lignes et **n** colonnes. Si on souhaite connaitre la valeur de **m** ou **n**, on utilise la commande **size(A)**

```
≫A = [1 2 5 ; 4 3 6]
 A =
    1 2 5
      4 3 6
≫ [m n] = size(A)
m =
    2
n =
    3
≫ size(A,1)
ans =
      2
```

On peut construire trés simplement une matrice **"par blocs"**. Si les quatres matrices (aux dimensions compatibles), on définit la matrice

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

par l'instruction M = [A B ; C D].

Il existe des commandes matlab pour des matrices prédéfinies :

eye(n) : la matrice identité (carrée de taille n).

ones(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 1.

zeros(m,n) : la matrice à m lignes et n colonnes dont tous les éléments valent 0.

rand(m,n) : une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1.

magic(n) : une matrice magique de dimension n.

```
≫eye(3)
ans =
     1 0 0
     0 1 0
     0 0 1
≫ones(2,4)
ans =
     1 1 1 1
     1 1 1 1
≫ zeros(2)
ans =
      0 0
      0 0
≫rand(2,3)
ans =
     0.8147 0.1270  0.6324
     0.9058 0.9134 0.0975
```

Matrices creuses (help sparfun)

Les matrices creuses (sparse matrix) sont des matrices caractérisées par un nombre très important d'éléments nuls. D'un point de vue informatique, un traitement particulier de ces matrices permet un gain en place mémoire et en coût calculatoire. Fonctions définition sparse création d'une matrice creuse full conversion d'une matrice creuse **issparse** retourne 1 si la matrice est creuse **speye** matrice identité creuse.

0.6.1 Manipuler une matrice

Pour extraire un élément de la matrice on indique la ligne et la colonne de celui-ci :

```
≫A = [1 2 5 ; 4 3 6]
A =
     1 2 5
     4 3 6
≫A(2,1)
ans =
      4
```

Lorsque l'on souhaite extraire une colonne ou une ligne entière on utilise le symbole (:) comme on va le voir dans l'exemple suivant :

```
≫A(2,:)
ans =
       4 3 6
≫A(:,1)
ans =
        1
        4
```

Toutes les combinaisons sont alors possibles. On peut extraire 2 colonnes par exemple en faisant :

```
≫A(:,[1 2])
ans =
      1 2
      4 3
```

Comme pour les vecteur il est possible d'obtenir la transposée d'une matrice avec la commande '.

```
≫A'
ans =
       1 4
       2 3
       5 6
```

Il existe des commandes matlab permettant de manipuler globalement des matrices. La commande **diag** permet d'extraire la diagonale d'une matrice : si A est une matrice,

diag(A) est le vecteur composé des éléments diagonaux de A. La même commande permet aussi de créer une matrice de diagonale fixée : si v est un vecteur de taille n, **A=diag(v)** est la matrice diagonale dont la diagonale est v.

```
≫A=eye(3)
≫diag(A)'
A =
     1 0 0
     0 1 0
     0 0 1
ans =
      1 1 1
≫v=[1:3];
≫diag(v)
ans =
        1 0 0
        0 2 0
        0 0 3
```

0.6.2 Opérations matricielles : addition et soustraction

On passe maintenant au opérations matricielles qui sont à la base du calcul sous Matlab.. L'objectif ici est de revoir les opérations matrcielles telles que : l'addition et la soustraction, le produit et l'inversion des matrices.

On commence par l'addition et la soustraction. Ces opérations sont possibles uniquement sur des matrices de taille identique. Ce sont des opérations termes à termes, similaires aux opérations scalaires. Par exemple :

$$\begin{pmatrix} 1 & 5 \\ -3 & 2 \end{pmatrix} + \begin{pmatrix} 4 & 2 \\ 0 & -4 \end{pmatrix} = \begin{pmatrix} 5 & 7 \\ -3 & -2 \end{pmatrix}$$

```
≫A = [1 5 ; -3 2]; B = [4 2 ; 0 -4];
≫M = A+B
M =
     5 7
    -3 -2
```

0.6.3 Produit Matriciel

Par contre la multiplication mérite une attention particulière. Il existe deux types de multiplication : la multiplication dite matricielle et la multiplication termes à termes. Commençons par cette dernière : la multiplication termes à termes est l'analogue de l'addition et de la soustraction vues précédemment. Sous Matlab elle se note de façon spécifique pour la distinguer de la vraie multiplication matricielle : **A.*B**

```
≫ A = [1 5 ; -3 2]; B = [4 2 ; 0 -4];
≫M = A.*B
M =
     4 10
     0 -8
```

De même si l'on souhaite obtenir le carré d'une matrice (au sens du produit termes à termes de cette matrice par elle même) on écrit : **A.^2**

```
≫A = [1 5 ; -3 2 ; 1 3]; M=A.^2
M =
      1 25
      9 4
      1 9
```

Sous Matlab, on calcule le produit matriciel en utilisant simplement le signe A*B :

```
≫A = [4 2 ; 0 1]; B = [1 2 3; 5 4 6]; M=A*B
M =
    14 16 24
     5  4  6
```

Comme on a dit précédemment, il faut faire attention à la taille des matrices pour que le produit soit défini. Par exemple si l'on essaie de faire calculer le produit BA on obtient une erreur :

```
≫A = [4 2 ; 0 1]; B = [1 2 3; 5 4 6]; B*A
Error using *Inner matrix dimensions must agree.
```

0.6.4 Inverse et division

Maintenant que l'on dispose du produit matriciel, on peut définir l'inversion. En effet, on note A^{-1}, l'inverse de A (quand il existe) et on définit A^{-1} par :

```
≫A = [4 2 ; 0 1]; M = inv(A)
M =
      0.2500 -0.5000
      0   1.0000
```

La division se définit à partir de l'inverse :

$A/B = AB^{-1}$.

Elle nécessite donc que B soit inversible et que les dimensions de A et B soient compatibles. Elle s'implémente sous Matlab de façon simple :

```
≫A = [4 2 ; 0 1]; B = [1 2 ; 5 4]; M = A/B
M =
    -1.0000 1.0000
    0.8333 -0.1667
```

0.6.5 Résoudre un système linéaire

Nous allons maintenant utiliser le formalisme matriciel pour résoudre un système d'équations. Le système que nous cherchons à résoudre est le suivant :

$$\begin{cases} 3x + 5y + z = 1 \\ 7x - 2y + 4z = -3 \\ -6x + 3y + 2z = 3. \end{cases}$$

On définit alors la matrice M à partir des coefficients de ce système

$$M = \begin{pmatrix} 3 & 5 & 1 \\ 7 & -2 & 4 \\ -6 & 3 & 2 \end{pmatrix}.$$

Si on note $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, on peut voir que les trois lignes du produit MX correspondent aux trois lignes du système. On peut alors définir la matrice $b = \begin{pmatrix} 1 \\ -3 \\ 3 \end{pmatrix}$,

tel que

$$MX = b.$$

Pour obtenir les solutions du système linéaire, il ne reste plus qu'à inverser la matrice M. On a donc :

$$X = M^{-1}b.$$

Il existe deux méthodes pour implémenter ce calcul sous Matlab. La première méthode reprend ce que nous avons appris précédemment. On doit définir la matrice M et le vecteur b, calculer l'inverse de M et faire le produit avec b :

```
≫M = [3 5 1 ; 7 -2 4 ; -6 3 2]; b = [1 -3 3]';
≫X=inv(M)*b
X =
   -0.3100
    0.3886
    0.0131
```

La seconde méthode est plus élégante. Elle introduit la division à gauche. Lorsque l'on note sous Matlab $M\backslash b$ cela signifie $inv(M) * b$. C'est la division à gauche. Rappelons nous que la division à droite (la division usuelle) était définit à l'aide de l'inverse : b/M signifie $b * inv(M)$.

Remark 0.6.1 la fonction prédéfinie **"solve"** permet de calculer les racines d'une ou un système d'équations. Si nous voulons l'appliquer sur notre exemple, il suffit d'écrire:

```
≫syms x y z
≫[x,y,z]=solve('3*x+5*y+z==1,7*x-2*y+4*z==-3,-6*x+3*y+2*z==3',[x,y,z])
x=
   -71/229
y=
    89/229
z=
   -3/229
```

Il existe des commandes Matlab prédefinies pour manipuler les matrices

- A' : transposée d'une matrice A.

- rank(A) : rang d'une matrice A.
- inv(A) : inverse d'une matrice A.
- expm(A) : exponentielle d'une matrice A.
- det(A) : déterminant d'une matrice A.
- trace(A) : trace d'une matrice A.
- poly(A) : polynôme caractéristique d'une matrice A.
- eig(A) : valeurs propres d'une matrice A.
- [U,D]=eig(A) : vecteurs propres et valeurs propres d'une matrice A.

0.7 Exercices d'application

Exercice 1: (Exercice de type Calculatrice) Faire les calculs suivants (en ligne de commande) :

- $x = 1 + 3/2$
- $y = x^2 + 1$
- $w = (1 + y)^x$
- $v1 = 4/3\pi R^3$ (où R= 7) % π dans Matlab s'écrit « pi »
- $a = cos(\pi) + sin(\pi)$ %cos(x) et sin(x) sont des fonctions prédéfinies
- $b = ln(v1)$ % logarithme népérien de x s'écrit log(x)
- $c = e^{y+x}$ % exponentielle de y+x s'écrit exp(y+x)
- $d = \sqrt{a^3 + b^2}$ %racine carrée de a^3+b^3 s'écrit sqrt(a^3+b^3)
- $s = |y|$% valeur absolue de y s'écrit abs(y)

Exercice 2: Donner les commandes Matlab permettant de calculer les expressions suivantes :

1. $\sqrt{\frac{4+\sqrt{7}}{5}}$
2. $\frac{b^2+\sqrt{b^2-4ac}}{2a}, a = 3, b = -3, c = 4$
3. $e^{-x^3}x^2, x = 6$
4. $e^{-x}\sin x, x = 0.5$

Exercice 3: Evaluer les expressions suivantes :

1. $\dfrac{5+11}{12\times 6}$
2. $2\dfrac{\frac{1}{3}+\frac{1}{5}+\frac{1}{6}}{\frac{2}{3}+\frac{4}{5}+\frac{5}{6}}$
3. $\dfrac{1}{12-6^2}+\dfrac{2}{3}$
4. $\dfrac{1+\left(\frac{2}{3}\right)^{\frac{1}{2}}}{12-6^2}$

Exercice 4: Celcius et Fahrenheit. Opérations matricielles

La relation entre les échelles de températures Celsius et Fahrenheit est linéaire.

On peut la représenter par une relation de la forme

$$T_C = aT_F + b,$$

où TC est la température en degrés Celsius et TF la température en degés Fahrenheit.

Deux points sont bien connus sur ces deux échelles : le point de fusion de l'eau, de à $T_F = 32$ et $T_C = 0$ et le point d'ébullition à $T_F = 212$ et $T_C = 100$.

On peut donc identifier les coefficients a et b grâce à ces données.

a. Donner le système d'équation qui permet d'obtenir a et b.

b. Donner la forme matricielle de ce système.

c. Résolver ce système en utilisant l'inversion matricielle puis en utilisant la division à gauche.

Exercice 5: Manipulation de matrices

a. Construire la matrice $M = \begin{pmatrix} 12 & \pi \\ 3i+4 & 1 \end{pmatrix}$.

b. Construire la matrice T tridiagonale à l'aide de la commande diag( ) utilisée 3 fois :

$$T = \begin{pmatrix} 1 & 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

c. Extraire de T les deux premières colonnes.

d. Extraire de T les éléments des colonnes et des lignes 2 à 4.

e. Créer une matrice T2 où la ligne 1 est échangée avec la ligne 3 puis la colonne 2 est remplacée par les valeurs de la colonne 4.

f. Construire les matrice A, B

$$A = \begin{pmatrix} 6 & 0 & 1 & 0 & 1 \\ 1 & 4 & 1 & 0 & 1 \\ 1 & 0 & 4 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 8 & 0 & 2 & 0 & 2 \\ 2 & 4 & 2 & 0 & 2 \\ 2 & 0 & -1 & 0 & 2 \\ 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

en sachant que v = [6;4;-3;-2;1];

A = diag(5:-1:1)

A(1:3,[1,3,5]) = A(1:3,[1,3,5]) + ones(3,3)

A = diag(v);

A(1:3,[1,3,5]) = A(1:3,[1,3,5]) + 2*ones(3,3);

Introduction à la programmation sous Matlab

Dans ce chapitre, on va présenter les mécanismes d'écriture et d'éxécution des programmes MALAB.

0.8 Les fichiers .m sur Matlab

La plupart du travail sous Matlab va consister à créer ou modifier des fichiers .m . Lorsque l'on réalise une tâche sous Matlab, il est trés souvent possible de le faire en utilisant uniquement la Command Window. Cependant lorsque cette tache devient plus complexe (plusieurs dizaines de ligne de code) ou que l'on souhaite pouvoir la transmettre à quelqu'un d'autre simplement, on utilise la fenêtre Editor. On crée un fichier .m qui peut être au choix un script ou une fonction (function en anglais).

0.8.1 Ecrire des scripts M-files

Un fichier de script est un fichier contenant une suite d'instruction MATLAB. Les fichiers script ont une extension de nom de fichier .m. Les M-files peuvent être des scripts qui exécutent simplement une suite d'instructions ou peuvent être des fonctions.

Example 0.8.1

Créer le script "test script" (soit vous tapez >>edit test_script.m, soit vous faites 'File'->'New'->'M-file' puis 'Save As' en spécifiant "test script.m" comme nom) avec la suite

d'instructions suivante :

```
clear all %efface toutes les variables du workspace
a = 1;
b = 2;
c = 3, d = 4;
e = a*b/(c+d)+5;
scal = 11;
k=scal+2*e
```

Sauvegardez puis exécutez le script (menu Debug->Save&Run ou Fleche verte ou F5). Observez la sortie sur la ligne de commande et conclure quant à l'utilisation des ; et ,

Il est également possible d'exécuter le script depuis la ligne de commande, taper >>test_script.

0.8.2 Ecrire des fichiers functions

Matlab permet de définir des fonctions. Une fonction est un ensemble d'instructions regroupées de manière à ne pas devoir les répéter régulièrement. Une fonction peut prendre des arguments et renvoyer des valeurs. (Par exemple, si l'on fourni à une fonction une date de naissance, cette fonction peut renvoyer l'âge de la personne)

La définition d'une fonction se fait de la manière suivante :

```
function arguments de sortie = nom_De_Ma_Fonction (arguments d'entrée)
```

Attention, pour que matlab reconnaisse une fonction et sache l'utiliser à partir de l'espace de travail, il est obligatoire que le nom du M-file soit identique au nom donné à la fonction.

A titre d'exemple, nous allons créer une fonction de calcul de la moyenne. Dans le répertoire courant, créer le fichier myMean.m (>>edit myMean.m) contenant le code

suivant :

```
function [mu,minv,maxv] = myMean( tab )
% Ligne de définition de fonction, Définie le nom de la fonction ainsi que le nombre
% et l'ordre des arguments  d'entrée et de sortie
%*************************************************************************
%                                   Texte d'aide
% Description du programme qui sera affiché lorsqu'on demandera
%
% MYMEAN(TAB) calcul de moyenne sur un tableau 1D
% Pour un tableau 'tab' à 1 dimension [mu,minv,maxv] = myMean( tab )
% retourne la moyenne 'mu', le minimum 'minv' le maximum 'maxv' des
% éléments de 'tab'
%*************************************************************************
%*************************************************************************
%                                   Corps de la fonction
%*************************************************************************
mu = sum(tab)/ length(tab);
minv = min(tab);
maxv = max(tab);
```

Pour notre exemple de calcul de la moyenne, le fichier devra obligatoirement s'appeler myMean.m.

Pour exécuter cette fonction. - Créer un vecteur (par ex, >>note = [2,18,5,15]) et appeler la fonction myMean() des manières suivantes :

```
≫myMean(note)
≫a = myMean(note)
≫[a,b] = myMean(note)
≫[a,b,c] = myMean(note)
```

- Essayer d'accéder à l'aide (>>help myMean).

Créer un fichier script appelé programme principal:

```
%*****************************************************%
%                    Programme Principal                %
%*****************************************************%
clear all
clc
note = [2,18,5,15]  % déclaration de mon vecteur
[a,b,c] = myMean(note) % appel à ma fonction myMean()
```

Il est indisponsable de sauvegarder ce fichier script dans le même dossier que le fichier de la fonction myMean() .

0.9 Les Opérateurs de comparaisons

Les opérateurs de comparaison sont :

Symbole	Signification
==	égal à
>	strictement superieur
<	strictement inferieur
>=	superieur ou égal
<=	inferieur ou égal
~=	différent

Le résultat d'une évaluation d'un test logique à l'aide des opérateurs de comparaison peut être "vrai" ou "faux" c-à-d 1 si vrai , 0 si faux

Example 0.9.1

```
≫x=3,y=4,x==3,x==y,x~=3,x~=y,x>3,x<3
x=
    3
y=
    4
ans=
     1
ans=
     0
ans=
     0
ans=
     1
ans=
     0
ans=
     0
```

Question

Que se passe-t'il si l'un des opérandes est un tableau de valeurs numériques ? Et si les deux opérandes sont des tableaux ?

Comparaison d'un tableau et d'un scalaire

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

Les termes de A supérieurs à 2 donnent 1 (true), les autres 0 (faux).

Comparaison de deux tableaux

Prenons maintenant deux tableaux quelconques de même dimension, et comparons-les :

```
≫A=[1 2 ; 3 -2];
≫B=[-3 2.5 ; 1 -1];
≫T = A > B
T =
   1 0
   1 0
```

Exercice : Que se passe-t'il si A='MATLAB',B='MaTHEMATICA'

A='MATLAB', B='OCTAVE'

A='MATLAB',B='octave'

Tester le résultat des ces commandes :A==B, A~=B ?

0.10 Opérateurs logiques

Les opérateurs logiques sont des opérateurs qui s'appliquent exclusivement sur des valeurs de type logique. Ils permettent la combinaison de conditions logiques.

	Syntaxe Matlab	
Opérateur logique	Symbole	Fonction
A et B	A&B	and(A,B)
A ou B	A|B	or(A,B)
Négation de A	~A	not(A)

A et B peuvent être des valeurs scalaires logiques ou des tableaux de valeurs logiques de même dimensions. Pour les tableaux, ces opérations s'appliquent terme à terme.

Example 0.10.1

```
≫a = 1; b = 0; c = 1; d = 0;
≫a&c
ans =
     1
≫a&b
ans =
     0
≫b&d
ans =
     0
≫b&a
ans =
     0
```

Un ET logique entre 2 propositions renvoie VRAI si et seulement si les deux propositions sont vraies.

```
≫a|c
ans =
     1
≫a|b
ans=
     1
≫b|d
ans=
     0
≫b|a
ans=
     1
```

Un OU logique entre 2 propositions renvoie FAUX si et seulement si les deux propositions

sont fausses.

```
≫~a
ans =
     0
≫~b
ans=
     1
≫~b|a
ans =
     1
≫~(b|a)
ans =
     0
```

0.11 Branchement conditionnel IF

Forme 1

if <expression booléenne>

<suite d'instructions exécutée si l'expression est VRAI>

end

Forme 2

if <expression booléenne>

<suite d'instructions 1 exécutée si l'expression est VRAI>

else

<suite d'instructions 2 exécutée si l'expression est FAUSSE>

end

Forme 3

if <condition1>

<suite d'instructions 1 exécutée si condition1 est VRAI>

elseif <condition 2>

<suite d'instructions 2 exécutée si condition1 est FAUSSE et que condition 2 est VRAI>

. . .

else

<suite d'instructions n exécutée si aucune des conditions n'est VRAI >

end

Example 0.11.1

x = rand(); %renvoie un nombre aléatoire compris entre 0 et 1 selon une loi uniforme

if x > 0.5

disp('pile') %la commande disp permet d'afficher un entier ou une chaine de caractère

else

disp('face')

end

Les opérateurs de comparaison et les opérateurs logiques sont utilisés essentiellement dans les instructions de contrôle if et while.

0.12 Boucle for

La boucle for répète une suite d'instruction un nombre fini de fois. Sa syntaxe est la suivante :

for compteur=valeur initiale:pas:valeur finale

une suite d'instructions

end

Example 0.12.1

```
N = 5 ;
for k = 1:N
x(k)= 1/2*k;
end
```

Cet exemple construit élément par élément un vecteur x de dimension 5.

0.13 Boucle while

Il arrive que nous souhaitions répéter une suite d'instructions jusqu'à qu'une condition soit satisfaite. Si l'on ne connait pas le nombre d'itérations nécessaire à l'avance, une boucle while est préférable par rapport à une boucle for.

while <condition>

<suite d'instructions>

end

La <suite d'instructions> va être répétée tant que <condition> est vrai, ou dit autrement jusqu'à ce que <condition> soit fausse.

Le terme expression est une expression logique. Si cette dernière est vraie, le bloc instructions est exécuté. Puis, expression est de nouveau testée. L'exécution du bloc est répétée tant que le test est vrai.

Example 0.13.1

```
compteur = 0;
while compteur < 10
disp('toujours dans la boucle') ;
compteur = compteur + 1;
end
```

Cet exemple affiche 10 fois la chaîne de caractère toujours dans la boucle.

Example 0.13.2

Faire un programme qui calcule la somme suivante :

$s = 1 + 2 + 3 + ...$, on arrête le calcul quand s>5

```
clear all
clc
s=1;i=1;%initialisation de deux variables i, s
while s<=5 %le test d'arrêt
i=i+1%incrémentation de i
s=s+i % l'expression de s
end
```

L'éxécution nous donne:

```
i=
   2
s= 3
i=
   3
s=
   6
```

0.13.1 Instruction break

L'instruction break : provoque l'arrêt de la première boucle for, while englobante.

L'instruction break interrompt l'exécution du bloc d'instructions en cours d'exécution, et sort totalement de la boucle for ou while, en ignorant les itérations suivantes.

L'instruction for ci-dessous est stoppée au premier i tel que t(i) est nul :

```
t = -2:10;
for i = 1:length(t)
if t(i) == 0
break;
end
end
```

0.13.2 La commande SWITCH

L'instruction SWITCH est une instruction de choix comme le IF mais avec la particularité de pouvoir effectuer plusieurs IF.

La commande SWITCH doit être utilisée dans le cas ou, par exemple, en fonction de la valeur d'une variable, elle permet de gérer plus élégamment les tests portant sur un seul critère pouvant prendre plus de deux valeurs entières, voici sa syntaxe :

Switch Variable

case Cst_1

script 1 (séquence d'instructions)

case Cst_2

script 2 (séquence d'instructions)

...

case Cst_n

script n (séquence d'instructions)

otherwise

(séquence d'instructions)

end

L'argument variable est de type numérique ou chaîne de caractères ;

Cst_1, ..., Cst_n sont des constantes numérique ou des constantes chaîne de caractères;

Script i est une séquence d'instructions à exécuter si variable == Cst_i.

Example 0.13.3

```
switch x
case 0,
resultat = a + b;
case 1,
resultat = a * b;
case 2,
resultat = a/b;
case 3,
resultat = a^b;
otherwise
resultat = 0;
end
```

En fonction de la valeur de x une opération particulière est effectuée.

0.13.3 Instruction continue

L'instruction continue : dans une instruction for, while, l'instruction continue provoque l'arrêt de l'itération courante, et le passage au début de l'itération suivante.

L'instruction continue interrompe l'exécution du bloc d'instructions en cours d'exécution, et passe à l'itération suivante de la boucle for ou while.

Supposons que l'on parcoure un vecteur t pour réaliser un certain traitement sur tous les éléments, sauf ceux qui sont positifs :

```
for i = 1:length(t)
if (t(i) > 0 )
continue; % on passe au i suivant dans le for
end
... % traitement de l'élément courant
end
```

0.14 Exercices d'application

Exercice **1 :**

1- Ecrire une fonction qui calcule la moyenne pondérée $\overline{x}$, d'une série statistique.

Rappel : $\overline{x} = \frac{1}{N} \sum_{i=1}^{i=N} n_i x_i$.

Exercice **2 :**

-Ecrire un programme qui calcule le maximum d'un vecteur.

-Ecrire une fonction qui calcule le factoriel d'un nombre n.

Exercice **3** :

-Ecrire une fonction qui calcule le factoriel de la somme de deux éléments consécutifs d'un vecteur.

-Ecrire une fonction qui calcule le factoriel de la somme terme à terme de deux vecteurs.

Représentation graphique en 2D et 3D

Matlab possède un grand nombre de commandes pour l'élaboration de graphiques plus sophistiqués. : graphiques standards en 2D ou 3D avec axes linéaires, semi-log ou logarithmiques, histogrammes, en escalier, représentation en coordonnées polaires, de courbes de niveau (« contour ,surface »). Ce chapitre présente les commandes les plus importantes à connaitre pour la représentation des graphiques les plus fréquents.

0.15 La commande « plot » et la commande « figure »

La commande plot est utilisée pour créer un graphique en 2D avec axes linéaires. Il s'agit de la façon la plus simple de créer une représentation graphique dans Matlab :

plot (x,y) %2-D graphique avec axes linéaires

ici **x** représente l'abscisse et **y** représente l'ordonnée (x et y sont des vecteurs).

Quand la commande "plot" est exécutée, une fenêtre appelée "figure" apparait automatiquement.

Il est possible de créer une figure pour chaque graphique avec la commande "figure"

la syntaxe pour dessiner deux fonctions dans deux figures différentes est la suivante :

```
y(x),z(x)
figure (1),plot(x,y(x))
figure(2),plot(x,z(x))
```

Example 0.15.1

```
x=0:.1:pi;y=sin(x);y2=cos(x);
figure(1),plot(x,y)
figure(2),plot(x,y2)
```

Tous les paramètres graphiques (épaisseur des lignes, échelles, limite d'axes) peuvent être modifiés en ajoutant des arguments (optionnels):

plot (x,y, 'LineSpec', 'PropertyName', 'PropertyValue')

LineSpec: « line specifiers » qui définissent le style et la couleur des lignes ainsi que le type de marqueurs. Ils peuvent se combiner entre eux et l'ordre n'importe pas.

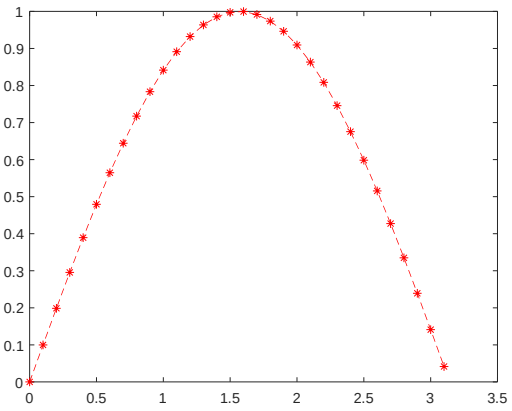
STYLE	LineSpec
Solide	-
Traitillé	- -
Pointillé	:
Point-trait	-.

Couleur	LineSpec
Rouge	r
Vert	g
Bleu	b
Cyan(bleu-vert)	c
Magenta(rouge violacé)	m
Jaune	y
Noir	k
Blanc	w

Marqueur	LineSpec
plus	+
cercle	o
astérisque	*
point	.
croix	x
triangle	^,<,>,v
carré	s
diamant	d
étoile	p,h

Example 0.15.2

```
x=0:.1:pi;y=sin(x);
plot(x,y,' - - r *')
```



Graphe de la fonction sin(x)

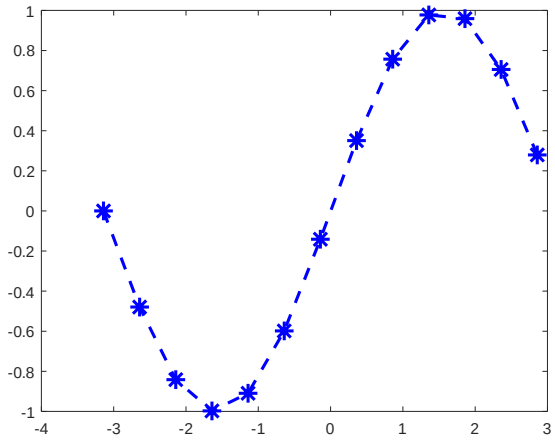
PropertyName et PropertyValue permettent de spécifier l'épaisseur des lignes, la taille

des marqueurs ainsi que leur apparence (p.ex. couleur du contour).

PropertyName	PropertyValue	Description
LineWidth	réel (défaut=0.5)	épaisseur de la ligne
Markersize	réel(défaut=10)	taille du marqueur
MarkerEdgeColor	couleur(r,g,b,...)	couleur de la bordure du marqueur
MarkerFaceColor	couleur(r,g,b,...)	couleur du marqueur

Example 0.15.3

```
x = [-pi:0.5:pi]
y = sin(x)
plot(x,y,'–b*','LineWidth',2,'MarkerSize',12)
```



Graphe de la fonction sin(x)

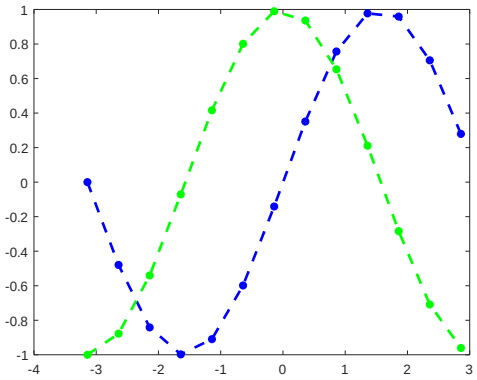
0.16 Graphiques multiples

La commande plot permet aussi de représenter plusieurs graphes dans un même graphique. Les paramètres graphiques se comportent de la même façon.

plot (x1,y1, x2,y2,x3,y3, . . . , xn,yn) 2-D graphique avec axes linéaires

Example 0.16.1

```
x1 = [-pi:0.5:pi]
y1 = sin(x1);
y2 = cos(x1);
plot(x1,y1,'–b*',x1,y2,'–g*','LineWidth',2)
```

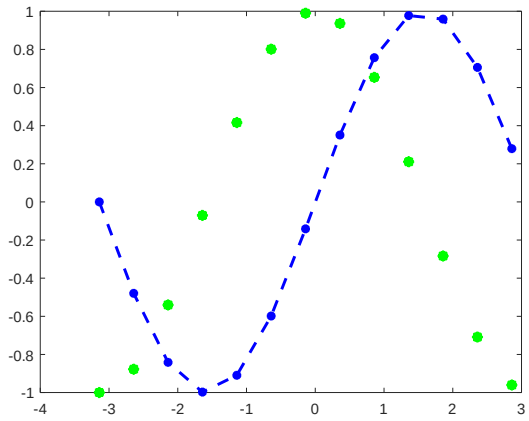


Graphe des fonctions sin(x) et cos(x)

La commande **hold on** permet de garder une fenêtre (figure) ouverte et ainsi ajouter d'autres graphiques dans cette fenêtre. La commande **hold off** arrête ce procédé.

Example 0.16.2

```
x1 = [-pi:0.5:pi]
y1 = sin(x1); y2 = cos(x1);
plot(x1,y1,'–b*','LineWidth',2)
hold on
plot(x1,y2,'g*','LineWidth',5)
hold off
```



Graphe des fonctions sin(x) et cos(x)

(commandes hold on, hold off)

0.17 Formater un graphique

C.-à-d. spécifier le « look » et ajouter des informations dans le graphique.

"xlabel" et "ylabel": ces deux commandes nous permettent de donner un nom à l'axe de x et l'axe y.

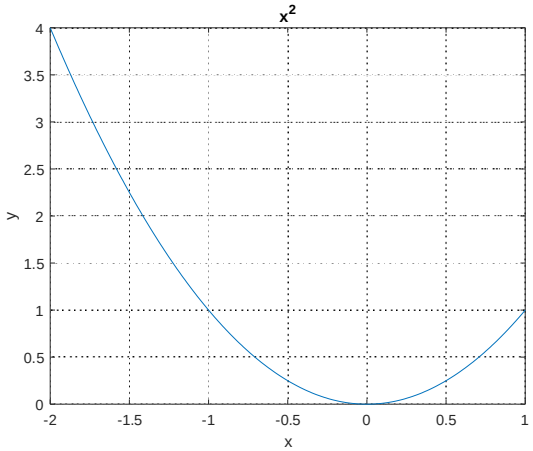
La commande "title": cette commande nous permet de donner un titre à la courbe(ou graphique)

plot(x,y)

title('titre')

Example 0.17.1

```
x=-2:0.01:1;
y=x.^2;
plot(x,y)
xlabel('x')
ylabel('y')
title('x^2'),grid
```

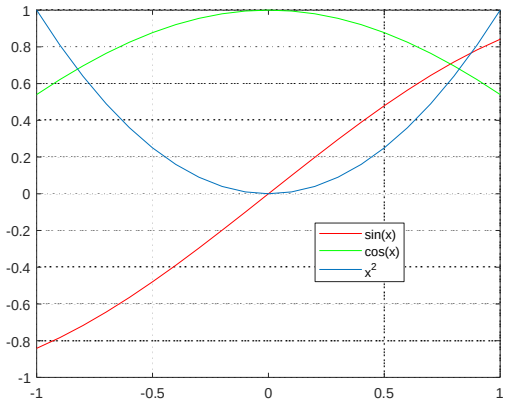


Graphe de la fonction x^2

La commande "legend": cette commande est utilisée quand on a plusieurs graphiques, elle permet de donner un nom à chaque graphique.

Exemple:

```
x1 = [-1:0.1:1]
y1 = sin(x1); y2 = cos(x1);y3=x1.^2;
plot(x1,y1,'r',x1,y2,'g',x1,y3),grid
legend('sin(x)','cos(x)','x^2')
```



Graphes des fonctions sin(x),cos(x),x^2

0.18 Types de représentations graphiques

Se référer à Matlab help pour les différents arguments optionnels de chaque type

Type	Description	commande
semilogy	axe des y en log de base 10 et axe des x linéaire	semilogy(x,y)
semilogx	axe des x en log et axes des y linéaire	semilogx(x,f(x))
loglog	les deux axes sont en log de base 10	loglog(x,y)
bar,barh	graphique à bars verticales ou horizontales	bar(x,y),barh(x,y)
hist	histogramme	hist(y,nbins); nbins=nbre des barreaux
stairs	graphique en escalier	stairs(x,y)
stem	graphique de données discrètes(digramme en batons)	stem(x,y)
polar	graphique à coord. polaires	polar(theta, radius)

0.19 Représentation de plusieurs repères dans une même figure

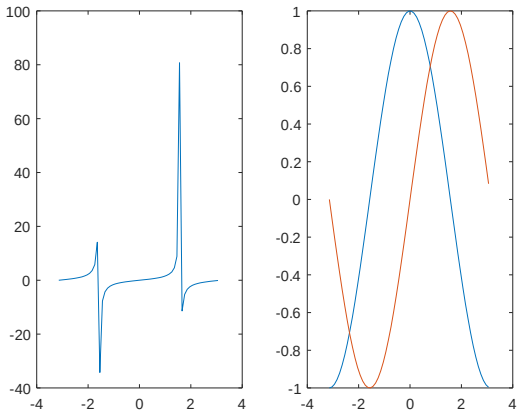
Plusieurs repères peuvent être mis l'un à côté de l'autre dans une même figure avec la commande "subplot" cette commande divise la figure en mxn sous fenêtres contenant chacune un graphique. La forme générale de cette commande est la suivante :

subplot(m,n,p)ou subplot(mnp), avec:

m:le nombre de ligne,**n**:nombre de colonne,**p**: la position

Example 0.19.1

```
x=[-pi:0.1:pi];
subplot(1,2,1)
plot(x,tan(x))
subplot(1,2,2)
plot(x,cos(x))
hold on
plot(x,sin(x))
hold off
```



Graphes des fonctions 1)tan(x), 2) sin(x),cos(x)

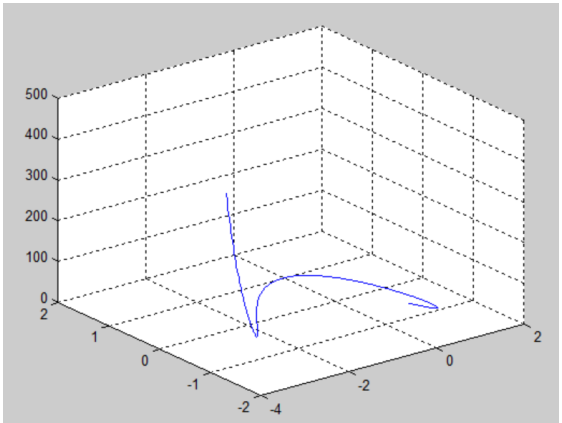
0.20 Représentation graphique en 3D

0.21 Courbes en 3D

Une courbe en 3D par une liste de triplets (x, y, z). L'instruction **plot3** attend trois arguments : les x, les y et les z.

Voici un exemple de courbe paramétrée :

```
t=linspace(-2,2,300);
plot3(exp(-3*t/10).*2.*sin(t), exp(-t/20).*cos(2*t), exp(-3*t))
grid
```



Graphe d'une courbe parametree

0.22 Surfaces

Génération des points (meshgrid)

Pour définir une surface, il faut un ensemble de triplets (x, y, z). Si on a en tout $n \times m$ points, cela signifie que l'on a $n \times m$ valeurs de x, $n \times m$ valeurs de y et $n \times m$ valeurs de z. Il apparaît donc que abscisses, ordonnées et cotes des points de la surface peuvent être stockés dans des tableaux de taille $n \times m$.

Toutes les instructions de tracé du surface, par exemple **surf, mesh** auront donc la syntaxe suivante

surf(M_x, M_y, M_z)

M_x : tableau d'abscisses, M_y : tableau d'ordonnées, M_z : tableau de cotes.

Il reste maintenant à construire ces tableaux. Prenons tout d'abord le cas de la surface définie par :

$$z = 5x^2 + 4y^2,$$

dont on veut tracer la surface représentative sur $[-1, 1] \times [-2, 2]$. Pour définir un quadrillage de ce rectangle, il faut définir une suite de valeurs $x_1, ...x_m$ pour x et une suite de valeurs $y_1, ...y_n$ pour y, par exemple :

```
 x = -1:0.2:1
x =
    Columns 1 through 7
  -1.0000 -0.8000 -0.6000 -0.4000 -0.2000 0 0.2000
    Columns 8 through 11
   0.4000 0.6000 0.8000 1.0000
y = -2:0.2:2
y =
    Columns 1 through 7
   -2.0000 -1.8000 -1.6000 -1.4000 -1.2000 -1.0000 -0.8000
    Columns 8 through 14
   -0.6000 -0.4000 -0.2000 0 0.2000 0.4000 0.6000
    Columns 15 through 21
  0.8000 1.0000 1.2000 1.4000 1.6000 1.8000 2.0000
```

En combinant toutes ces valeurs de x et y, on obtient $n \times m$ points dans le plan (x, y). Il faut maintenant construire deux tableaux, l'un contenant les $n \times m$ abscisses de ces

points l'autre les $n \times m$ ordonnées, soit :

La fonction **meshgrid** s'en charge :

>> [X,Y] = meshgrid(x,y);

Il reste maintenant à calculer les $z = 5x^2 + 4y^2$ correspondants. C'est là que les calculs terme à terme sur les matrices montrent leur efficacité : on applique directement la formule aux tableaux X et Y, sans oublier de mettre un point devant les opérateurs *, / et ^

>> Z = 5*X.^2 +4*Y .^2 ;

Tracé de la surface

Ensuite on peut utiliser toutes les fonctions pour tracer de surface, par exemple **mesh**

>> mesh(X,Y,Z)

Les instructions les plus courantes sont

– **mesh** graphique de type maille

– **meshc** qui fonctionne comme **mesh** mais ajoute les courbes de niveau dans le plan (x, y)

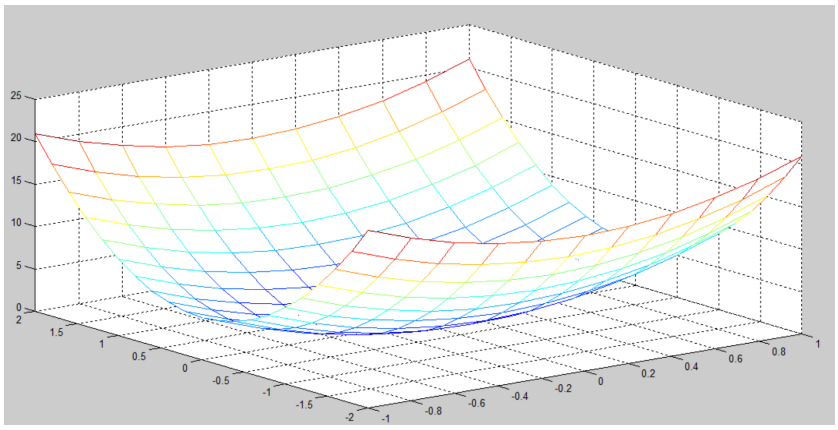
– **surf** graphique de type surface.

– **surfl** qui «peint» la surface comme si elle était éclairée.

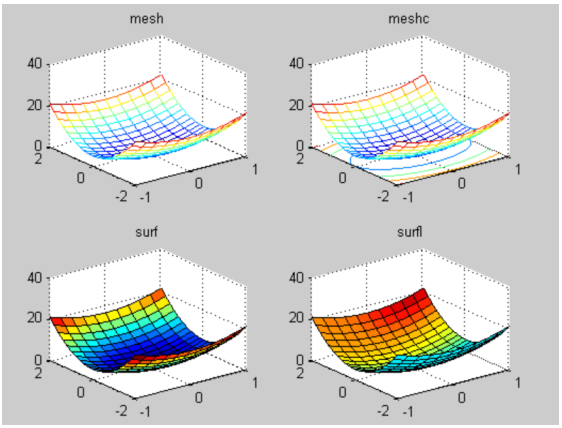
– **surfc** qui fonctionne comme surf mais ajoute les courbes de niveau dans le plan (x, y).

Exemple :

```
x = -1:0.2:1;
y = -2:0.2:2;
[X,Y] = meshgrid(x,y);
Z = 5*X.^2 +4*Y .^2 ;
figure(1)
mesh(X,Y,Z)
title('mesh')
figure(2)
subplot(2,2,1)
mesh(X,Y,Z)
title('mesh')
subplot(2,2,2)
meshc(X,Y,Z)
title('meshc')
subplot(2,2,3)
surf(X,Y,Z)
title('surf')
subplot(2,2,4)
surfl(X,Y,Z)
title('surfl')
figure(3)
contour(X,Y,Z,[-0.5 0 0.5 1 2 3 ])
ch = contour(X,Y,Z, [-0.5 0 0.5 1 2 3 ]);
clabel (ch),grid
title('contour')
```



Graphe de la surface z=5*x^2+4*y^2



Difference entre mesh, meshc, surf, surfl

Pour tracer les courbes $z = Cte$ d'une surface $z = f(x, y)$, on peut utiliser la fonction **contour**. Elle s'utilise comme les instructions précédentes, mais fournit un graphe 2D dans le plan (x, y). Plusieurs paramètres optionnels peuvent être spécifiés, notamment le nombre de courbes de contours à afficher. Avec les matrices X, Y, Z construites précédemment, on pourra par exemple essayer, pour tracer 10 lignes de niveau :

>> **contour(X,Y,Z, 10)**

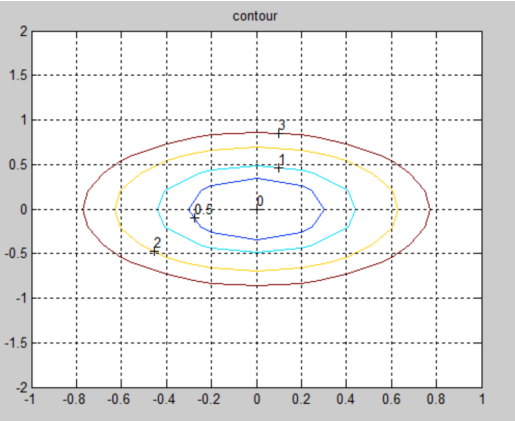
Notons que l'on peut indiquer à MATLAB quelles lignes de niveaux afficher. Par exemple pour dessiner les courbes $z = -0.5, 0, 0.5, 1, 2$ et 3, on peut écrire :

>> **contour(X,Y,Z, [-0.5 0 0.5 1 2 3])**

Dans ce cas, il est également intéressant d'ajouter les valeurs de z le long des différentes lignes de contour. Pour cela il faut affecter une variable avec la sortie de la commande **contour** et utiliser la fonction **clabel** :

>> ch = contour(X,Y,Z, [-0.5 0 0.5 1 2 3]);

>> clabel (ch)



Graphe par la fonction contour

0.23 Exercices d'application

Exercice 1 :

En utilisant la commande plot, tracer la fonction suivante :

$$f(x) = \begin{cases} \sin(2x+1) \text{ si x>0,} \\ \cos(4x) \text{ sinon.} \end{cases}$$

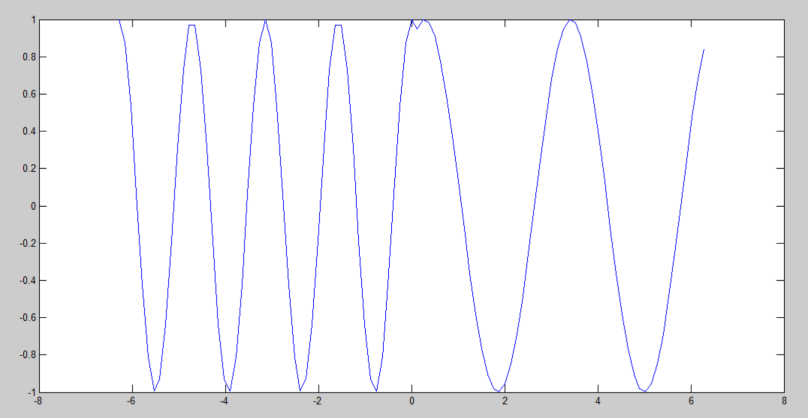
C'est une fonction y=f(x), où f est définie par morceaux puisque sa valeur dépend d'un test logique définissant, dans cet exemple, deux domaines de définition.

Voilà comment définir y en fonction de x :

```
≫x=-2*pi:pi/25:2*pi;
≫y = sin(2*x+1) .* (x>0) + cos(4*x) .* not(x>0);
```

On ajoute les deux expressions sin(2x+1) et cos(4x) en les pondérant par la condition logique définissant leurs domaines de validité puis on la dessine à l'aide de la commande plot.

≫plot(x,y)



Graphe d'une fonction definie par morceaux

Exercice 2 :

Tracer les fonctions suivantes :

-$f(x) = 2x^2 - 4$, avec x dans $[-10, 10]$.

- la fonction $f(x) = x^2$ avec la couleur rouge et le marqueur * et la fonction $g(x) = x^3$ avec la couleur vert et le marqueur +, sur le domaine $[-5, 5]$.

-la fonction $f(x) = 4x_1^2 + x_2^2$ sur le plan $[-2, 2] \times [-2, 2]$.

Annexes

0.24 TP1:Prise en main de Matlab : manipulation des vecteurs et des matrices

0.24.1 1- Manipulation des vecteurs

L'objectif de ce TP est de familiariser l'étudiant avec le langage MATLAB, sa structure, ses objets et ses fonctions.

Déroulement

Votre module comporte plusieurs séances de travaux pratiques. Le but de ces séances est multiple :

- Aborder certaines notions du cours de mathématiques sous un angle plus concret/visuel.
- Vous donner une première approche d'un logiciel professionnel de calcul scientifique.

Echauffement et premier contact

On suppose que vous avez réussi à installer Matlab. Commencez par lancer l'interface graphique de MATLAB. Vous observez une fenêtre (déja vu en cours). Pour commencer :

L'environnement MATLAB se présente sous la forme d'un espace de travail (Command Window), où un interpréteur de commandes exécute des opérations et des fonctions.

Exercice1 : (Quelques commandes Matlab, Commandes de type Calculatrice)

Commençant par tester les commandes suivantes :

– **clock :** affiche l'année, le mois, le jour, l'heure, les minutes et les secondes.

– **date :** Affiche la date.

– **ans :** quand on introduise des instructions anonymes (sans variables en sortie), le Matlab considère une variable **'ans'** par défaut pour enregistrer le résultat.

– **input :** permet de lire une valeur à partir du clavier (l'instruction habituelle lire). Exemple : **a = input (' taper un nombre :')**

– **disp :** permet d'afficher un tableau de valeurs numériques ou de caractères. L'autre façon d'afficher un tableau est de taper son nom. La commande **'disp'** se contente d'afficher le tableau sans écrire le nom de la variable, ce qui peut améliorer certaines présentations. On utilise fréquemment la commande **disp** avec un tableau qui est une chaîne de caractères pour afficher un message. Exemple :

>> disp('la valeur saisie est').

3*5,4 : Effectue l'opération demandée.

$x = 3 * 5$: le résultat de l'opération est affecté à la variable x.

x : permet d'affficher x.

$Y = 3 * x;$: Création de Y, important: terminer la commande par ; supprime l'affichage du résultat.

Y : affiche la valeur de Y

$Y = 1 + Y\hat{}3$ Opération de mise à une certaine puissance remarquer que Y à changer, sa première ecriture est oubliée, **on utilisera souvent cette réaffectation.**

– **who** : donne la liste des variables définies dans l'espace de travail actuel (essayer whos).

– **whos** : donne la liste des variables définies dans l'espace de travail avec plus de détails.

$E = x + Y$: affiche le résultat de $x + Y$ affecté à la variable E.

who: vérifier que E a été rajoutée.

– **clear** : permet de détruire une variable de l'espace de travail (si aucune n'est spécifiée, toutes les variables seront effacées).

– **clc :** efface le contenu de la fenêtre des commandes et affiche uniquement l'invite « ».

L'aide dans Matlab

Mieux vaut apprendre à se repérer tout seul que de demander en permanence à son voisin comment faire. Ne serait-ce qu'au cas où il faudrait utiliser dans l'examen une fonction dont on ne se souvient la syntaxe que vaguement.

intro lance une introduction à MATLAB.

helpwin ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations.

help donne la liste de toutes les commandes par thèmes.

help nom décrit la fonction nom.m.

doc clc donne une description complète de la commande clc.

lookfor nom recherche une instruction à partir du mot clé nom.

Exercice 2 : Trouver la commande qui calcule le conjugé d'un nombre complexe . Taper help de cette commande.

Application

Soit z un nombre complexe tel que : $z = 8 + 4i$

1. calculer le conjugué de z .
2. afficher la partie réelle de z.
3. afficher la partie imaginaire de z.

Exercice 3 :

Donner les commandes Matlab permettant de calculer les expressions suivantes :

1. $\sqrt{\frac{3+\sqrt{5}}{17}}$
2. $\frac{b^2+\sqrt{b^2-4ac}}{2a+4c}, a = 2, b = -3, c = 1$
3. $e^{-x^2}\left(\ln x\right)^2, x = 2$
4. $(1+x)^5 e^{-x}\cos(x), x = 0.5$

Matlab traite un seul type d'objet : **les matrices ou tableaux**! Les scalaires sont des matrices 1×1, les vecteurs lignes des matrices $1 \times n$, les vecteurs colonnes des matrices $n \times 1$.

On peut construire des vecteurs et des matrices en entrant leurs coefficients.

Exemples :

1. vecteurs

   >> x=[4;5;6] % les ";" séparent les éléments d'un vecteur colonne.

   >>n=length(x)%calcule la taille du vecteur x

   >> x=[1,2,3] % les "," ou les blancs séparent les éléments d'un vecteur ligne.

   >> x'%calcule le transposé du vecteur x

   >> y = [x,x,x]

   >> z = [x x x]

2. Matrices

   >> M=[11 12 13 14;21 22 23 24; 31 32 33 34; 41 42 43 44] % où les ";" séparent les lignes d'une matrice.

   On peut construire une matrice à partir de plusieurs vecteurs de même longueur :

   >> y=[22;33;44];

   >> mat1=[x' y]

   >>[n,m]=size(mat1)

3. Les polynômes : Matlab représente un polynôme sous forme d'un vecteur ligne contenant les coefficients classés dans l'ordre des puissances décroissances.

   >> P=[1 -7 10] % Représente Le polynôme $P(x) = x^2 - 7x + 10$.

Tester l'effet des commandes suivantes(en ligne de commande)

>> l11 = 1:10 % Un vecteur contenant les entiers de 1 à 10)

>> l22 = 1:1:10

>> l33 = 10:-1:1

>> l44 = 1:0.3:pi

>> l11(2) = l3(3)

>> l44(3:5) = [1,2,3]

>> l44(3:5) = [ ]

>> l5 = linspace(1,5,5)

>> help linspace

>> who

>> whos

>> clear l11 l22 l33 l5

>> who

>> clc % efface le contenu de la fenêtre de commande.

>> clear

Un bon programme est un programme BIEN COMMENTE.

Application:

(1) Construire une suite partant de -3 et allant à -5 par pas de 0.5.

(2) Construire une suite décroissante d'entiers de 15 à 3.

(3) Construire une suite de longueur 100 de $-\pi$ à π.

Application sur les opérations vectorielles : Donnez la commande de

(1) vecteur de nombres de n à m par pas de 1

(2) vecteur de nombres de n à m par pas de p

(3) vecteur de p nombres de n à m

(4) la longueur d'un vecteur x

(5) la i-ème coordonnée d'un vecteur x

(6) les coordonnées $i1$ à $i2$ d'un vecteur x

(7) supprimer les coordonnées $i1$ à $i2$ d'un vecteur x

(8) concaténer les vecteurs x et y

(9) le produit scalaire des vecteurs lignes x et y

(10) le produit scalaire des vecteurs colonnes x et y

(11) créer une matrice de taille $[u, v]$, à partir de x

Exercice 4 :

Le nom MATLAB provient de MATRIX LABORATORY. C'est un logiciel qui bien utilisé permet d'effectuer de façon relativement efficace des opérations sur des tableaux, des vecteurs ou des matrices

- Taper $u = [4, 5, 6]$, $v = [2, 4, 6]$, $w = [2; 4; 6]$. Quelle est la différence entre v et w ?

- À quoi servent les commandes u(2) ? w(3) ?
- Que renvoie $2u$? $u+v$? $u+w$?
- Que produit selon vous la commande$M = [1,2,3;4,5,6;7,8,9]$? que donne la commande M(2,3).
- Construire les matrices $A = \begin{pmatrix} 3 & \pi \\ -1 & 2 \\ 5 & 10^{-12} \end{pmatrix}$ et $B = \begin{pmatrix} 4 & 5 & 6 \\ -1 & -2 & -3 \end{pmatrix}$
- Exécuter les commandes AB, AM, MA, Mv, Mw,vv... Lesquelles renvoient une erreur ?
- Calculer $y = Mw$, puis $M\backslash y$. Que fait la commande «\»?
- Réessayer les commandes précédentes en remplaçant «∗» par «.∗»
- Tester l'effet des opérateurs «ˆ», «.ˆ», «/» et«./» sur les matrices et les vecteurs.
- Que donnent les commandes suivantes : $0:10, 2:0.5:6, 1:0.3:5$
- Construire en une seule ligne un vecteur ligne contenant les carrés de tout les multiples de 3 compris entre 0 et 100. Construire un vecteur ligne contenant les 20 premières puissances de 2
- Si v est un vecteur, **sum(v)** renvoie la somme de toutes les composantes de v. Utiliser cette fonction pour évaluer $\sum_{k=1}^{10^6} \frac{1}{k^3}$.

0.25 TP2: Prise en main de Matlab : manipulation des vecteurs et matrices

0.25.1 2- Manipulation des matrices

Application sur les opérations matricielles : Donner la commande qui :

(1) affiche le nombre de lignes et de colonnes d'une matrice A

(2) affiche le coefficient d'ordre i, j d'une matrice A

(3) affiche les coefficients des lignes $i1$ à $i2$ d'une matrice A

(4) supprime la $k^{ème}$ ligne d'une matrice A

(5) affiche les coefficients diagonaux d'une matrice A

Application : Entrer la matrice

>> A=[1 2 5; 2 4 -1; -3 1 8]

Quels sont les résultats des commandes suivantes?

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

>> A([2 3],1:2)

>> A([2 3],:)

>> A([2 3],end)

>> A(:)

>> A(5)

>> reshape(A(:),size(A))

Exercice 1 :

Ecrire la matrice carrée M de taille 11×11 contenant les entiers de 1 à 121 rangés. Extraire de cette matrice les matrices suivantes :

– la sous-matrice formée par les coefficients m_{ij} pour $i = 1, ..., 5$ et $j = 8, ...11$;

– celles des coefficients m_{ij} pour $(i, j) \in \{1, 2, 5, 6, 9, 10\} \times \{1, 2, 5, 6, 9, 10\}$;

Exercice 2 :

1. Essayer des fonctions sur la matrice A. Par exemple, quels sont ses valeurs et vecteurs propres? Puis, construire une matrice C de même taille que A. Essayer $A+C$, $A*C$, $A.*C$.

2. Résolution du système

$$\begin{cases} 2x_1 + 4x_2 = 6 \\ x_2 + 2x_1 = -11 \end{cases}$$

Ecrire le système sous la forme matricielle $Ax = b$ (où vous définissez A et b) et trouver la solution?

Regarder l'effet des instructions suivantes.

- Vecteur

  >> x=rand(1,5)

  >> mean(x)

  >> std(x)

  >> median(x)

  >> sort(x)

- Matrice

  >> A=rand(3)

  >> sort(A)

  >> [B, I]=sort(A)

  >> sort(A')

  >> max(A)

  >> max(A')

  >> max(max(A))

  >> sum(A)

  >> cumsum(A)

  >> prod(A)

  >> diff(A)

  >> D=A([1,2],1:3)

  >> sum(D,1)

  >> sum(D,2)

Application :

- Calculer 10!.
- Tirer 30 nombres aléatoirement dans l'intervalle $[0, 1]$. Quelle est la valeur minimale du vecteur et la position du coefficient qui la réalise? Vérifier.

Opérateurs relationnels et logiques

Ils permettent de relier logiquement deux matrices.

Opérateurs relationnels $<, <=, >=, ==$ (égalité), $\tilde{}=$ (différent)

Opérateurs logiques & (et), | (ou), $\sim$ ou *not* (non)

> Attention de ne pas confondre = qui sert à affecter une valeur à une variable et == qui sert à tester l'égalité. Les opérateurs relationnels peuvent être utilisés avec des scalaires ou des matrices. Le résultat d'évaluation d'une expression relationnelle est 1 (vrai) ou 0 (faux). Appliqués à une matrice, ils rendent une matrice de même dimension, formée de 1 et de 0.

Application

>> u=5

>> u==5

>> u<=12

Reprendre la matrice

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

>> Ar=(A<=2)

>> [J,I]= find(A==1)

Exercice 3 :

- Reprenons le vecteur $y1 = [3.2, 4.8, 3.3, 3.2, 3.1, 4.2, 3.2, 3.3]$.

  Donner la commande qui répondre à la question suivante : **existe-t-il une coordonnée** du vecteur $y1$ inférieure à 3.2?

- Tirer 110 nombres aléatoirement dans l'intervalle $[0, 1]$ et grouper les dans un vecteur $x = (x_i)_{i=1,...,110}$.
- Prendre $a_i = 2 * x_i$ pour tout $i = 1..., 110$.
- Prendre la partie entière de ces nombres (à l'aide de la fonction floor) : $b_i = [a_i]$.

Exercice d'application :

1. Créer le vecteur $D = [cos(0), cos(\frac{\pi}{3}), cos(\frac{2\pi}{3}), cos(\pi)]$. puis créer de deux façons différentes la matrice,

$$L = \begin{pmatrix} 3 & 2 & 1 & 0 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 0 & 1 & 2 & 3 \end{pmatrix},$$

Résoudre alors le système :

$$\begin{pmatrix} 9 & 4 & 1 & 0 \\ 4 & 9 & 4 & 1 \\ 1 & 4 & 9 & 4 \\ 0 & 1 & 4 & 9 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \frac{1}{cos(0)} \\ \frac{1}{cos(\frac{\pi}{3})} \\ \frac{1}{cos(\frac{2\pi}{3})} \\ \frac{1}{cos(\pi)} \end{pmatrix}$$

2. A l'aide des commandes *ones* et *eye*, créer la matrice 10×10,

$$A = \begin{pmatrix} 6 & 4 & \cdots & 4 \\ 4 & 6 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 4 \\ 4 & \cdots & 4 & 6 \end{pmatrix}$$

Remplacer la $2^{ème}$ ligne par la ligne $[2\ 3\ 2\ 2\ \cdots 2]$.

3. Créer la matrice suivante:

$$H = \begin{pmatrix} 18 & 17 & 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 & 8 & 7 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

Utiliser H,pour:

(1) Créer un vecteur colonne à six éléments appelé **va** qui contient les éléments du deuxième et cinquième colonnes de H.

(2) Créer un vecteur colonne à sept éléments appelé **vb** contenant les éléments 3 à 6 de la troisième ligne de B et les éléments de la deuxième colonne de H.

(3) Créer un vecteur colonne à neuf éléments, appelé **vc**, contenant les éléments du deuxième, quatrième et sixième colonnes de H .

0.26 TP 3: Les fichiers scripts et functions

Exercice 1 :

Créer la matrice $M1$, à partir de la commande suivante :

>>M1 = magic(6)

Cette commande va créer une matrice de dimension (6×6) ayant des propriétés particulières.

- Créer une matrice M_1 composée uniquement de la première colonne de $M1$
- Créer une matrice M_2 composée uniquement de la deuxième ligne de $M1$
- Créer une matrice M_3 composée des 2 premières colonnes de $M1$
- Créer une matrice M_4 composée des 2 dernières lignes de $M1$
- Créer une matrice M_5 composée de l'intersection des $1^{ère}$et $2^{ème}$ lignes de $M1$ 1et des $4^{ème}$, $5^{ème}$ et $6^{ème}$ colonnes de $M1$

Les fichiers Scripts et Functions

Un script est un ensemble des instructions (commandes) Matlab qui joue le rôle du programme principal.

Pour lancer un script aller à l'onglet « File », « new » et puis « M-file ». C'est dans cette fenêtre qu'on peut écrire l'ensemble des instructions Matlab. Il est en effet beaucoup plus simple de modifier des instructions dans un script que de retaper un ensemble d'instructions MATLAB dans la fenêtre de commandes.

Une fois, vous écrivez le programme vous devez le sauvegarder et lui donner un nom Cliquer sur « file » puis « save as »

Remarques importantes :

1. Le script doit avoir une extension de la forme « .m ».
2. Dans l'appelation des scripts, Il faut éviter :
3. D'utiliser les caractères désignant un opérateur spécifique à Matlab tel que « -», « ; », « * »,... etc.
4. D'utiliser le nom d'une fonction Matlab telle que : sqrt, sin, cos,... etc.
5. D'utiliser deux mots séparées (par exemple : program st, program 1).

Pour exécuter un script, on écrit tout simplement son nom dans la fenêtre de commandes puis on clique sur entrer.

Si le script contient une (des) erreur(s), la ligne contenant l'erreur ainsi le type d'erreurs sont affichées dans la fenêtre des commandes.

Matlab contient un grand nombre de fonctions prédéfinies comme sin, cos,sqrt, sum,etc. Il est possible de créer nos propres fonctions en écrivant leurs codes"sources" dans des fichiers "M-Files" en représentant la syntaxe suivante :

function $[s_1, s_2, ..., s_n]$ =nom_fonction $(\arg_1, \arg_2, ..., \arg_n)$

% le corps de la fonction

s_1 = ...% la valeur retournée pour s_1

s_2 = ...% la valeur retournée pour s_2

...

s_n = ...% la valeur retournée pour s_n

end

Où:$s_1, s_2, ..., s_n$ sont les valeurs retournées(sorties),$\arg_1, \arg_2, ..., \arg_n$ sont les arguments (entrées).

Le rôle d'une fonction est d'éffectuer des opérations sur une ou plusieurs entrées pour obtenir un résultat qui sera appelé sortie.

Voici un exemple simple d'utilisation de fonctions (créer une fonction matlab appelée "addition" qui a pour but de calculer la somme et la différence de deux nombres x, y).

Etape 1 : Création de la fonction

Dans un fichier nommé addition.m, on écrit le texte suivant :

```
function [result1,result2] = addition(x,y)
result1 = x + y;
result2=x-y;
end
```

Etape 2 : Création d'un script qui utilise la fonction :

Dans un fichier nommé monscript.m (qui se trouve dans le même répertoire que addition.m), on écrit le texte suivant :

```
[a,b]=addition(2,3)( % a ici représente la somme, b représente la différence
```

Il ne reste plus qu'à exécuter monscript.m.

Exercice 2 :

Ecrire un fichier script pour créer une matrice J de taille (3×10) de la forme suivante.

$$J = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0.5831 & 0.4062 & 0.2354 & 0.4088 & 0.9711 & 0.8083 & 0.6523 & 0.2193 & 0.3891 & 0.5491 \end{pmatrix}$$

(Utiliser les fonctions ones, zeros et rand ainsi que la concaténation de matrices)

Exercice 3 :

- Ecrire un fichier script pour créer la matrice F suivante :

$$F = \begin{pmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 & 49 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 \\ 61 & 62 & 63 & 64 & 65 & 66 & 67 & 68 & 69 \\ 71 & 72 & 73 & 74 & 75 & 76 & 77 & 78 & 79 \\ 81 & 82 & 83 & 84 & 85 & 86 & 87 & 88 & 89 \\ 91 & 92 & 93 & 94 & 95 & 96 & 97 & 98 & 99 \end{pmatrix}$$

- Modifier le script pour calculer aussi le minimum de la deuxième ligne de F.

Exercice 4 :

1. Ecrire un script qui cherche le minimum d'un vecteur sans utiliser la fonction "min". Pour se faire, initialiser une variable "mini" avec le premier élément du vecteur, puis écrire une boucle "for" qui passera en revue chaque élément du vecteur et qui testera si celui-ci est inferieur à "mini", si il est inferieur, on mettra à jour "mini" avec l'élément courant du vecteur. Ainsi une fois tout le vecteur parcouru, "mini" devrait contenir la valeur minimale du vecteur.

2. Modifier le script pour également sauvegarder la position du minimum dans le vecteur.

Exercices 5 :

1. Écrire une fonction qui prend en entrée un vecteur et retourne le nombre de valeurs distinctes dans ce vecteur.

Exercice d'application :

1. Écrire un script qui calcule le produit scalaire de deux vecteurs.

2. On considère la fonction suivante :

$$g(x) = \begin{cases} \sin\left(x^2 + a\right) & \text{si } x < 0 \\ \frac{4+\sqrt{x}}{x^2+2x+1} & \text{si } 0 \leqslant x \leqslant \pi \\ 0 & \text{si } x \geqslant \pi \end{cases} :$$

avec a une constante donnée. Ecrire une fonction Matlab pour $g(x)$.

3. Ecrire une fonction MATLAB qui prend en entrée un vecteur et renvoie le produit des éléments du vecteur.

0.27 TP 4: Les graphiques

Graphiques 2D

Un graphique sert à représenter dans une grande majorité des cas une grandeur physique.

Il est donc primordial de gérer les unités des axes que l'on utilise. De nombreuses fonctions servent à personnaliser les axes en ajoutant vos propres labels, légendes et titres. Différents types de représentations sont aussi possibles, suivant le type de données que l'on désire représenter. Nous ne présentons ici que les graphiques de type courbes, mais il existe aussi des fonctions réalisant des histogrammes, des barres ou des camenberts pour les données de types statistiques ou proportionnelles (help specgraph pour plus de détails).

Les représentations se font en coordonnées cartésiennes mais aussi polaires ou cylindriques.

Les principales fonctions d'affichage sont :

- *plot* : relie deux points successifs par une ligne, ce qui donne une courbe.
- stem : les différents points ne sont pas reliés entre eux, mais apparaissent comme des échantillons.
- stairs les valeurs des points restent identiques jusqu'au point suivant. Similaire à un bloqueur d'ordre 0 pour un échantillonneur (capacité très grande).
- comet trace la courbe de manière continue, comme une comète qui passe.
- semilogx, semilogy, loglog idem que plot sur une courbe, semi logarithmique en x ou y ou sur une courbe logarithmique.

La première utilisation d'une de ces fonctions entraîne la création d'une fenêtre appelée figure. Les suivantes se tracent par défaut dans la même figure, en effaçant la courbe précédente. Il est donc nécessaire de créer une nouvelle figure pour afficher 2 graphiques différents. La commande figure crée une nouvelle fenêtre et la commande close ferme la fenêtre en cours d'utilisation. Toute les figures sont repérées par un numéro qui peut être utilisé comme argument de figure ou close pour rappeler une ancienne figure. La syntaxe est la même pour toutes ces fonctions. Elle permet notamment de tracer plusieurs courbes en même temps :

>> plot(x1,y1,'couleur1',x2,y2,'couleur2',...)

L'argument couleur est un caractère, correspondant à une couleur dont la liste est donnée dans l'aide de la fonction plot. Quand on ne désire tracer qu'une seule courbe, les arguments x1 et 'couleur' sont optionnels. Attention, car on perd dans ce cas là l'information sur l'axe des abscisses, et donc sur une unité (souvent le temps ou la fréquence) de la grandeur représentée.

Il est aussi possible de superposer les courbes à l'aide de la commande hold on, qui maintient l'affichage. De la même manière hold off supprime le maintient de l'affichage.

Ainsi :

*>> x = 0 :0.01 :2*pi;*

*>> y1 = sin(2*x);*

>> y2 = cos(x);

>> plot(x,y1,'r',x,y2,'b')

Équivaut à

>> plot(x,y1,'r')

>> hold on;

>> plot(x,y2)

Créer un fichier ".m"permettant de comparer les différents types de visualisations sur le signal sinusoïdal généré précédemment.

La fonction polar permet de représenter des fonctions en coordonnées polaires. Ceci présente un grand intérêt pour les diagrammes de directivité. Par exemple, ici un diagramme de directivité de type cardioïde.

*>> theta = 0 :0.01 :2*pi;*

>> r = 1 + cos(theta+2);

>> polar(theta,r)

Remarque : il peut être nécessaire de faire un hold off avant!

Gestion des axes

Les différentes fonctions suivantes permettent de :

1. Gérer les labels des axes .
2. Commenter sur les figures.

- title : title('Texte du titre') ajoute un titre à la figure.
- xlabel : xlabel('l'axe des x')
- ylabel : ylabel('l'axe des y')
- legend : legend('Nom de la courbe 1','nom de la courbe 2', ...)
- grid : grid on, grid off, quadrille ou non le graphique.
- zoom : permet de zoomer dans la figure a l'aide de la souris (clic gauche = zoom +, clic droite = zoom-, double clic droite = retour à l'affichage de départ, bouton gauche appuyé ! dessin de la zone à zoomer).
- clf : efface la figure en cours d'utilisation.

- ginput : ginput(n) récupère les coordonnées de n points cliqués à la souris dans la figure en cours.

Plusieurs repères dans une figure "subplot"

subplot(m,n,p) ou subplot(mnp) divise la fenêtre graphique courante en une matrice m x n matrice de repères et sélectionne la p-ième repère comme emplacement de dessin par défaut. Le numéro du repère est compté ligne par ligne, c'est à dire que l'emplacement (i,j) de la matrice porte le numéro (i-1)*n + j.

Exemples

subplot(221)

plot()

subplot(222)

plot3()

subplot(2,2,3)

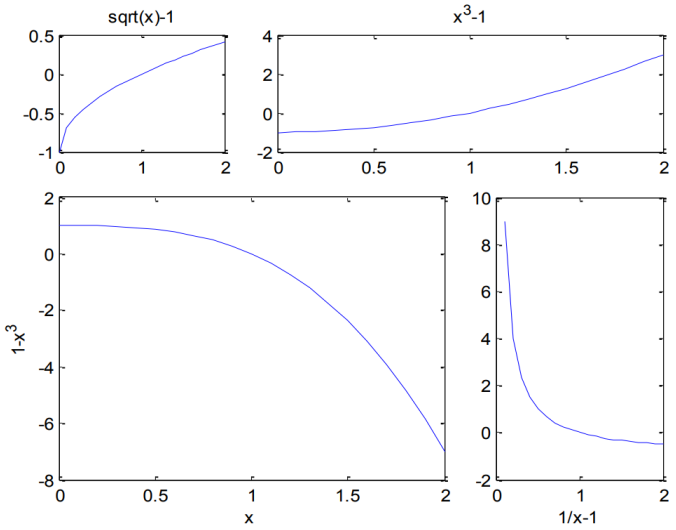
plot()

subplot(2,2,4)

surf()

Exercices :

Ecrire la syntaxe Matlab qui donne les figures suivantes :



Graphique en 3D

Il y a plusieurs types de graphiques 3D :

- *plot3(x,y,z,'style')* : Graphique type ligne en 3D.
- *mesh(x,y,z)* : mesh Graphique type maille en 3D.
- *surf(x,y,z)* : surf Graphique type surface en 3D.

Exemple :

>> t = 0:pi/25:10*pi;

>> plot3(sin(t+3),cos(2*t),t)

Exercice

Ecrire un programme qui permet de tracer la surface $z = \cos(2x)\sin(y+x)$ en utilisant $x = [-6\ 6]; y = x;$

Bibliography

[1] MATLAB for Engineers, Holly Moore, Pearson Education,2012.

[2] A Guide TO MATLAB for Beginners and Experienced Users, HUNT, Brian R., LIPSMAN, Ronald L.,et ROSENBERG, Jonathan M. Cambridge university press 1995.

[3] An Engineer's Introduction to Programming with MATLAB 2019, By Shawna Lockhart, Eric Tilleson Published June 17, 2019.

[4] Essential MATLAB for Engineers and Scientists, Brian H. Hahn and Daniel T. Valentine, Sixth Edition 2017.