REPUBLIQUE ALGERIENNE DÈMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÈ DE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DU GENIE DE LA CONSTRUCTION
DEPARTEMENT DE GENIE CIVIL

COMPLÉMENT DE PROGRAMMATION

"Mastère 1 - Structures "

Dr. ALICHE AMAR

TABLE DES MATIÈRES

TABLE DES MATIÈRES 2
Introduction 5
Objectifs pédagogiques 5
CHAPITRE I : RAPPEL SUR LES TECHNIQUES DE PROGRAMMATION ET STRUCTURATION DES PROGRAMMES 6
I.1. GÉNÉRALITÉS ET NOTIONS DE BASE 6
I.2. ALGORITHMES 12
I.1.3. LES STRUCTURES DE CONTRÔLE 19
I.1.4. UTILISATION DES PROCÉDURES ET FONCTION 22
CHAPITRE II : INITIATION À MATLAB 25
II.1. DÉMARRAGE ET QUITTER 25
II.2. INTERFACE GRAPHIQUE 26
II.3. ESPACE DE TRAVAIL 28
II.4. AIDE ET DOCUMENTATION EN LIGNES 30
II.5. SYNTAXE D'UNE LIGNE D'INSTRUCTIONS 31
II.6. SYNTAXE DES PRINCIPALES COMMANDES PRÉDÉFINIT EN MATLAB 31
II.6.1. LES COMMANDES 32
II.6.3. OPÉRATEURS ARITHMÉTIQUES 32
II.6.4. OPÉRATEURS LOGIQUE ET DE COMPARAISON 32
CHAPITRE III : VARIABLES ET FONCTIONS DANS MATALAB 33
III.1. NOTION DE VARIABLES ET AFFECTATION DE VALEUR 33
III.2. LES VARIABLES DANS LES CALCULS SIMPLE 33
III.3. PRINCIPAUX TYPES DE VARIABLES 34
III.3.1. TYPE COMPLEXE 35
III.3.2. TYPE CHAINE DE CARACTÈRES 36
III.3.3. TYPE LOGIQUE 37
III.3.4. AUTRES TYPES DE VARIABLES, SPÉCIFIQUES À MATLAB 37
cell : type parfait pour gérer des tableaux de données hétérogènes 38
struct : type parfait pour créer des variables structurées 38
table : le type parfait pour des données hétérogènes tabulées 39
III.4. VARIABLES ET FONCTIONS PRÉDÉFINIES 40
III.4.1. VARIABLES PRÉDÉFINIES 40
III.4.2. FONCTIONS PRÉDÉFINIES 40
CHAPITRE IV : VECTEURS ET MATRICES 42
IV.1. DÉFINITION D'UN TABLEAU 42

IV.2. LES MATRICES ET VECTEURS 43
IV.3.1. DÉCLARATION DE VECTEURS 44
IV.3.2. DÉCLARATION DE MATRICES 46
IV.3.3. OPÉRATIONS SUR LES MATRICES 48
TP N°01 54
Exercice IV.1 54
Exercice IV.2 54
Exercice IV.3 54
Exercice IV.4 55
Exercice IV.5 55
Exercice IV.6 55
Exercice IV.7 55
Exercice IV.8 56
CHAPITRE V. REPRÉSENTATIONS GRAPHIQUES SOUS MATLAB 57
V.1. GESTION DES FENÊTRES GRAPHIQUES 57
V.2. COURBES ET REPRÉSENTATIONS 2D 58
V.2.1. TRACER LE GRAPHE D'UNE FONCTION 58
V.2.2. TRACER UN GRAPHIQUE DANS UNE NOUVELLE FENÊTRE 59
V.2.3. SUPERPOSER PLUSIEURS COURBES SUR UNE FIGURE 60
V.2.4. CRÉER UNE FIGURE AVEC PLUSIEURS GRAPHIQUES 61
V.2.5. TRACER UNIQUEMENT LES POINTS DE DONNÉES 61
V.2.6. PLOT AVEC DEUX AXES D'ORDONNÉES 62
V.2.7. ATTRIBUTS DE COURBES 62
V. 2.8. COULEUR 63
V.2.9. STYLE DE MARQUEURS 63
V.2.10. STYLE DE LIGNE 63
V.2.11. SPÉCIFIER LE STYLE DE TRAIT, LA COULEUR ET LES MARQUEURS, POUR PLUSIEURS COURBES 64
V.2.12. AUTRES FORMES DE REPRÉSENTATION 65
V.2.13. GESTION DE LA FENÊTRE GRAPHIQUE 66
V.3. REPRÉSENTATION 3D 71
V.3.1.COURBES DANS UN ESPACE 3D 71
V.3.1. TRACÉ DE SURFACES EN 3D 71
V.3.2. ADAPTATION DES TRACÉS 3D 73
CHAPITRE VI : PROGRAMMATION MATLAB 75
VI.1. LES FICHIERS SCRIPT ET FUNCTIONS 75

VI.1.1. LES SCRIPTS 75
VI.1.2. LES FONCTION SUR MATLAB 75
VI.2. LES ENTRÉES / SORTIES 77
VI.1.2.1. AFFICHAGE 77
VI.3. STRUCTURES DE CONTRÔLES (BOUCLES / TESTS) SUR MATLAB 79
VI.3.1. L'INSTRUCTION FOR 79
VI.3.2. L'INSTRUCTION While 81
VI.3.3. L'INSTRUCTION CONDITIONNELLE if 82
CHAPITRE VII : EXEMPLES D'APPLICATION 86

Introduction

Ce document est un support de cours et d'exercices sur une partie du domaine de programmation. Il couvre un champ défini de fonctionnalités que nous avons estimé utiles à nos étudiants, dans le cadre de leur formation.

C'est donc avec un certain parti-pris que nous avons choisi d'aborder la notion d'algorithmes et de leur programmation sur un logiciel comme MATALAB. Le choix de MATALAB est dicté par les caractéristiques que présente ce logiciel à savoir : une bibliothèque assez riche de fonctions prédéfinies, simplicité et fluidité dans son utilisation dans notre contexte d'utilisation qu'est la résolution de problèmes numériques, pour l'ingénieur.

Par ce document, nous souhaitons , à travers des exemples et des exercices d'applications, initie les étudiants à la notion de programmation et faciliter au passage la prise en main de MATALAB, ainsi que présenter les outils disponibles pour résoudre des problèmes numériques, lies au domaine de génie civil, qu'un technicien, un ingénieur ou un scientifique, peut rencontrer dans le cadre de son activité professionnelle. Aussi, Nous présenterons les facilités qu'offre ce type d'outils pour le traitement, l'exploitation et la présentation de données numériques, issues de calculs ou de mesures expérimentales. Dans ce contexte, les courbes 2D ou 3D seront abordées, mais aussi tous les problèmes liés à l'écriture et à la lecture de fichiers de données.

Objectifs pédagogiques

- Aborder la notion d'algorithmes et la programmation
- Faciliter la prise en main du logiciel MATALAB ;
- Présenter les fonctionnalités utiles au travail scientifique ;
- Présenter les fonctions graphiques 2D et 3D ;
- Aborder quelques problèmes de calcul numérique et de génie civil

CHAPITRE I : RAPPEL SUR LES TECHNIQUES DE PROGRAMMATION ET STRUCTURATION DES PROGRAMMES

Jusqu'ici, nous avons travaillait seulement dans la fenêtre de commandes, très pratique pour démarrer, ou pour réaliser de petits calculs, en créant des variables dans l'espace mémoire de MATLAB (le *workspace*), et en utilisant différentes fonctions prédéfinies. Cependant, dans la programmation généralement on s'attaque à des problèmes plus complexes, qui demandent des manipulations de commandes plus structurées ou plus nombreuses.
Nous allons voir dans ce qui suit, comment utiliser MATLAB comme un véritable langage de programmation, en passant des fichiers de commandes (que l'on peut sauvegarder et donc réutiliser), en écrivant nos propres fonctions et en utilisant des structures de contrôle.

I.1. GÉNÉRALITÉS ET NOTIONS DE BASE

ORDINATEUR ET PROGRAMMATION

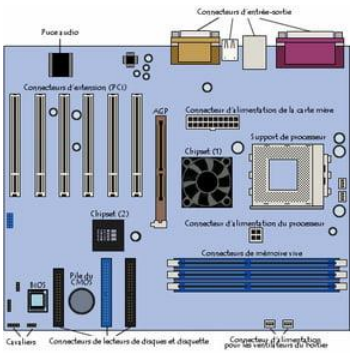
L'informatique (INFORmation autoMATIQUE), comme son nom l'indique, permet d'automatiser l'information que nous manipulons. Cette informatisation permettra de réaliser un gain considérable en temps et en effort.
Aujourd'hui, parmi les applications courantes de l'informatique on peut citer la communication, le calcul scientifique, le multimédia….Ces applications ne sont possibles que grâce à un ordinateur auquel il faut lui fournir un logiciel (ou un programme) adapté pour chaque tache. La programmation est donc une activité fondamentale dans l'automatisation de l'information.
La programmation peut être vue comme l'art de déterminer un algorithme (une démarche) pour résoudre un problème et d'exprimer cet algorithme au moyen d'un langage de programmation.

DÉFINITION ET UNITÉ DE MESURE

Pour mieux expliquer l'utilité d'un ordinateur, prenant comme exemple le processus de réalisation d'une pièce mécanique (le schéma ci-dessous):

Composition d'un ordinateur

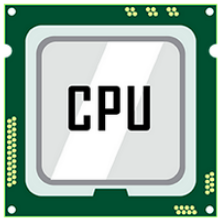
a. Carte mère



Un ordinateur est composé d'un ensemble de composants électroniques reliés entre eux. Ces éléments sont placés sur un circuit appelé carte mère (plaque sur lequel est gravé le **circuit de connexions électriques reliant les composants**) (Figure 1). La carte mère joue un rôle principale dans le fonctionnement d'un ordinateur. Elle permet de relier un **processeur** à de la **mémoire** (tous les échanges de données partent et aboutissent à la carte-mère).

b. Processeur

Un autre élément essentiel dans la composition d'un **ordinateur**, celui qui permet d'effectuer les **calculs** et **opérations** demandées est le **processeur**. Les **opérations**, les **calculs** demandés au **processeur** sont sous **forme binaire**.



Quand le **processeur** effectue une **opération**, le résultat peut être **mémorisé** de façon :

- **Temporaire** : résultat mémorisé un temps défini, il n'existe obligatoirement plus une fois l'**ordinateur** éteint – mis hors tension.
- **Permanente** : résultat mémorisé même après que l'**ordinateur** soit éteint – mis hors tension. On peut donc retrouver le résultat une fois l'**ordinateur** rallumé.

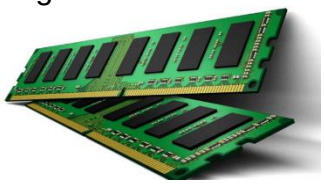
c. Mémoire centrale

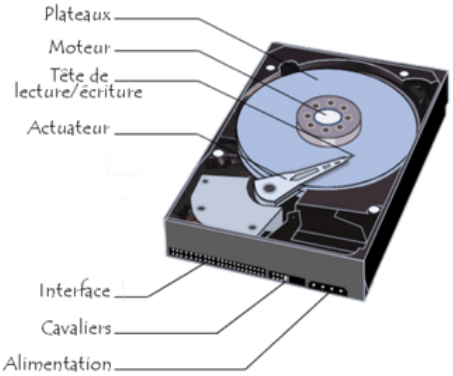
Il existe donc dans l'**ordinateur** deux gestions différentes de la **mémorisation du résultat** :

- La gestion de la **mémoire temporaire** appelée **mémoire vive** de l'**ordinateur** est prise en charge par des **puces de mémoire** qui permettent de **mémoriser temporairement l'information** (**barrette de mémoire vive** – **RAM**).
- La gestion de la **mémoire permanente**, appelée **mémoire morte** de l'**ordinateur** est prise en charge par le **disque dur** (**hard drive**) qui permet de **mémoriser de façon permanente l'information**.

Au cours du déroulement du programme, celui-ci utilise des données, soit les données fournies en entrée, soit des données intermédiaires que le programme utilise pour fonctionner. Ces données sont stockées dans des variables. Physiquement, les variables sont des données binaires dans la mémoire centrale (appelée aussi mémoire **RAM**). Lorsque le processeur effectue un calcul, le programmeur peut indiquer que le résultat de ce calcul doit être mémorisé dans une variable (en **RAM**). La quantité de mémoire **RAM** est mesurée en octets (mégaoctets ou gigaoctets). Les données en mémoire centrale ne sont conservées que pendant le déroulement du programme, et disparaissent lorsque le programme se termine (notamment lorsque l'on éteint l'ordinateur), tandis que les résultats finale d'un calcul est stocké d'une manière permanente dans le disque **dur**.

Les **composants** qui constituaient autrefois la **mémoire vive** et le **disque dur** de l'**ordinateur** étaient de nature très différente. La **mémoire vive** était constituée de **petits circuits imprimés** regroupant et reliant plusieurs **puces de mémoire** (appelées **barrettes de RAM**). Le **disque dur** était composé de **disques magnétiques** superposés et lus par une **tête de lecture** mécanique il est maintenant suplanté par la téchnologie **SSD**.





d. Périphériques

Ce sont des dispositifs connecté à un système informatique (ordinateur) et qui ont pour objectif d'assuré le lien entre la carte mère et l'extérieur. On distingue :

- Les périphériques d'entré : Les périphériques d'entrée transmettent les données dans un seul sens, du périphérique vers la mémoire centrale.
  Exp : clavier, souris, scaner, …etc.

- Les périphériques de sortie : Les périphériques de sortie reçoivent des données dans un seul sens, de la mémoire centrale vers le périphérique.
  Exp : écran, imprimante, …Etc

- Les périphériques d'entrée-sortie : Les périphériques d'entrée-sortie permettent la communication dans les deux sens entre la mémoire centrale et le périphérique.
  Exp : disque dur, port USB, carte réseau.

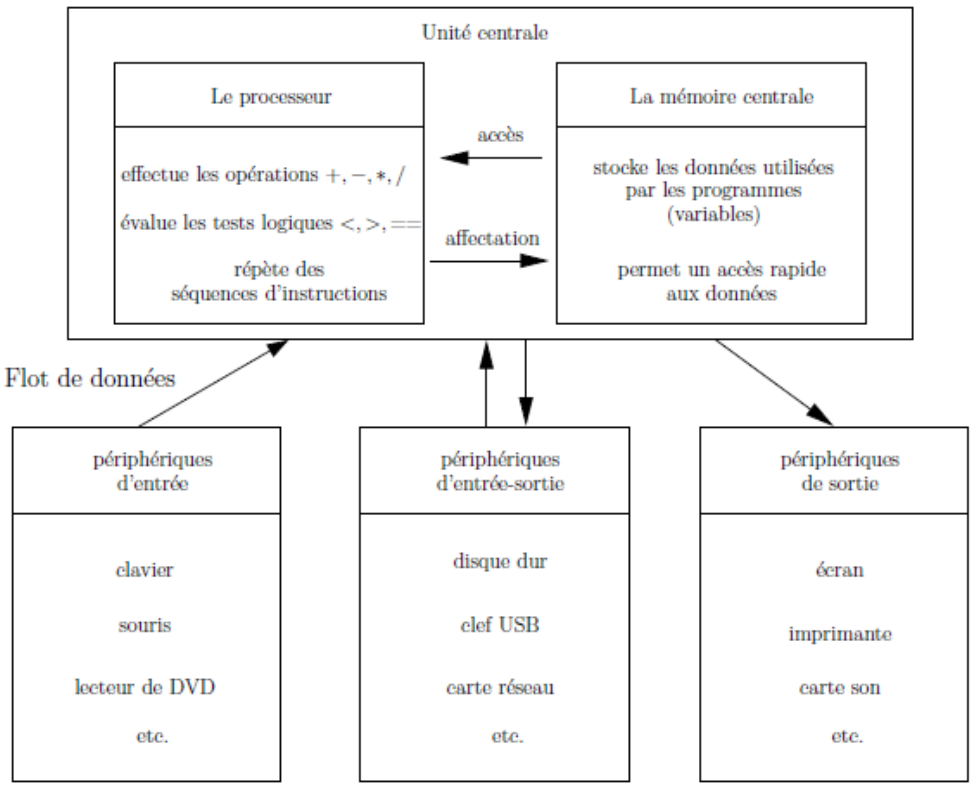


Schéma d'architecture d'un ordinateur.

e. Système d'exploitation

Un **programme informatique** doit recevoir des données pour les traiter, et produire d'autres données. Pour que le programme puisse fonctionner, il faut du matériel (composants électroniques), et il faut une couche logicielle intermédiaire avec le matériel, appelée *système d'exploitation*. Le système assure la communication entre le programme informatique et le matériel, et permet au programme d'agir sur le matériel.

Codage des données

Le fonctionnement d'un système informatique est basé sur une logique à deux états telle que, de façon schématique, le courant passe ou ne passe pas. Dans cette logique les informations sont transformées par impulsion électriques, qui est le seul langage

compris par l'ordinateur. Du point de vue informatique, ces deux états logiques sont conventionnellement notés 1 ou 0, et déterminent une logique dite **binaire**.
Un ordinateur étant basé sur le système binaire, il est utile de connaître les systèmes binaire (base 2), hexadécimal (base 16) et octal (base 8), ainsi que les techniques de conversion entre ces différents systèmes.
Toute information à traiter devra être représentée sous forme binaire.

a. Le système Binaire

C'est la base utilisée en informatique pour la représentation des informations au niveau de la machine. Dans ce système, la base 'b' vaut 2 chiffres, et il y a donc 2 digits 0 et 1 appelé dans ce cas « BIT » (**B**inary dig**IT)**

Exemple

Le nombre 1011 exprimé en binaire signifie

$$(1011)_2 = 1*20+0*21+1*22+1*23$$
$$= 1 + 2 + 8$$
$$= (11)_{10}$$

b. Système Octal

Dans ce système la base de numérotation vaut 8. Une séquence de 8 bits consécutifs s'appelle un octet (en anglais byte ou B avec une majuscule dans les notations). Il permet par exemple de stocker un caractère, tel qu'une lettre ou un chiffre. Ce regroupement de nombres par série de 8 permet une lisibilité plus grande, au même titre que l'on apprécie, en base décimale, de regrouper les nombres par trois pour pouvoir distinguer les milliers. Le nombre « 1 256 245 » est par exemple plus lisible que « 1256245 ».
Dans la base octal il y'a 8 digit : 0, 1, 2, 3, 4, 5, 6 et 7. Dans cette base il n'y a pas de chiffres 8 et 9.

Exemple

Le nombre 275exprimé en Octal dans la base décimal

$$(275)_8 = 5*80+7*81+2*82$$
$$= 5+56+128$$
$$= (189)_{10}$$

c. Système décimal

Dans la base dix, appelé système décimal, il y a dix digits : 0, 1, 2, 3 … 8 et 9.

Exemple

Le nombre 1234 exprimé dans la base décimal

$$(1234)_{10} = 4*100+3*101+2*102+1*103$$
$$= 4+30+200+1000$$
$$= (1234)_{10}$$

NB : c'est le même en base 10

d. Système hexadécimal

Dans ce système, la base B vaut 16, il y a 16 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. les dix premier digits de 0 à 9 sont les chiffres du système décimal et les digits de 10 à 15 sont les premiers lettres majuscules de l'alphabet.

Exemple

Le BAC exprimé Hexadécimal

$(BAC)_{16}$ = C*160+A*161+B*162
= 12*1+10*16+11*256
= 12+160+2816
= $(2988)_{10}$

$(3F9)_{16}$ = 9*160+F*161+3*162
= 9*1+15*16+3*256
= 9+240+768
= $(1017)_{10}$

Pour toute transformation (changement de base : binaire, octal et hexadécimal) il suffit d'effectuer un développement polynomiale de ce nombre dans la même base et de faire la somme par la suite.

e. Correspondances entre bases

Nombre	Binaire	Octal	Décimal	Hexadécimal
0	0000	0	0	0
1	0001	1	1	1
2	0010	2	2	2
3	0011	3	3	3
4	0100	4	4	4
5	0101	5	5	5
6	0110	6	6	6
7	0111	7	7	7
8	1000	10	8	8
9	1001	11	9	9
10	1010	12	10	A
11	1011	13	11	B
12	1100	14	12	C
13	1101	15	13	D
14	1110	16	14	E
15	1111	17	15	F

f. Conversion base b – système décimal

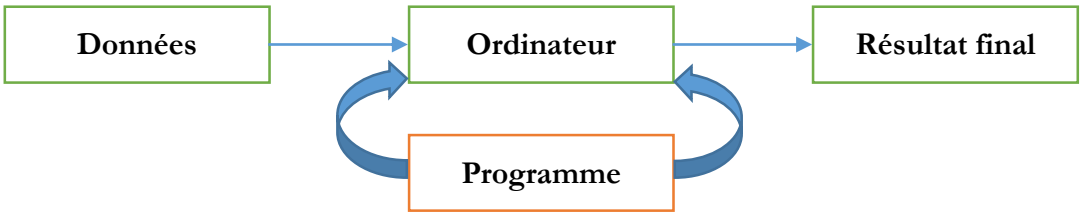
On développe le nombre selon les puissances de la base b.

Exemple

$$(1001110101)_2 = 1 \times 2^0 + 1 \times 2^2 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^6 + 1 \times 2^9 = (629)_{10}$$

Automatisation de l'information

En informatique, l'automatisation de l'information au moyen d'un ordinateur suit le schéma suivant :



Le schéma ci-dessus, montre que le processus de l'automatisation de l'information est pris en charge par l'ordinateur par le biais d'un programme informatique. Le fonctionnement d'un programme informatique nécessite en général des informations en entrées (données) pour livre des résultats en sortie.

L'ordinateur est une super "machine à calculer". Maintenant, supposant que l'on souhaite qu'il calcule les racines d'une équation du second degré en lui soumettant les ordres suivants :

a. Lire les 3 coefficients A, B et C.

b. Calculer la valeur de Δ telle que Δ= 4AB

c. En supposant que Δ est positive, calculer les racines X1 et X2 tel que:

$$x_1 = \frac{-B - \sqrt{\Delta}}{2A} \qquad x_1 = \frac{-B - \sqrt{\Delta}}{2A}$$

Une fois le processus d'automatisation des étapes de calcul précédentes programmé, en utilisant un langage de programmation, des résultats serons affichés. Maintenant nous posons la question suivante :

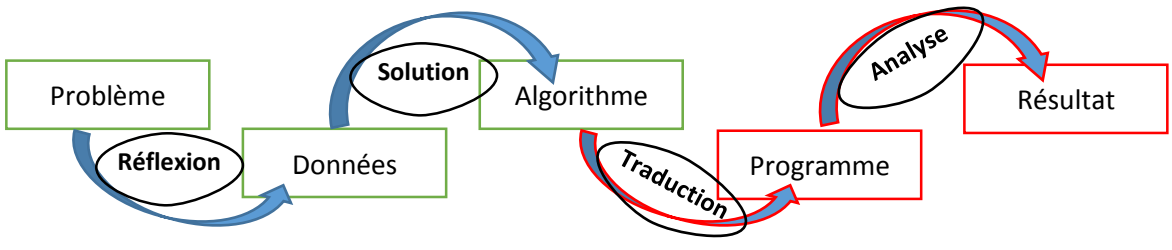
Est-ce que l'ordinateur nous donnera les valeurs justes de X1 et X2 ?.

Vu la formule qui calcul Δ introduite, dans le programme qui 'est fausse ($\Delta = B^2-4AC$), le résultat finale sera faux, Ce qui montre que l'ordinateur ne pourra jamais détecter une formule fausse dans un programme.

Au final, l'ordinateur permet d'automatiser le traitement de l'information en fonction des instructions et des commandes introduites par l'utilisateur, Ainsi, s'il existe dans le programme des anomalies, le résultat, n'est pas garanti.

Résolution d'un problème

La résolution d'un problème passe par une succession d'étapes comme le montre le schéma suivant :



a. **Identification du problème**: cette étape consiste à rassembler les différentes informations et hypothèses (données) et connaître exactement ce qui est attendu comme résultat.
b. **Analyse et réflexion**: l'analyse va déterminer les différentes solutions et actions qui nous mènerons aux résultats attendus. En évaluant leurs avantages, leurs inconvénients ainsi que les moyens nécessaires à mettre en œuvre on pourra alors déterminer la meilleure solution.
c. **ALGORITHME :** appelé aussi plan d'action, après avoir décidé de la meilleure solution a adopté, il va falloir écrire, dans un langage conventionnel, séquentiellement les actions élémentaires que devrait exécuter l'ordinateur.
d. **Programmation** : l'algorithme n'étant pas compréhensible par l'ordinateur, cette étape consistera à traduire l'algorithme en un langage de programmation évolué, tel que le langage C, Fortran, java, Matlab. Le résultat de cette traduction s'appelle un programme.

Une fois ces actions déterminées, il suffit de les traduire dans le langage de programmation choisi.
Durant l'écriture d'un programme, on peut être confronté à 2 types d'erreur :

- **Les erreurs syntaxiques** : observées lors de la compilation et sont le résultat d'une mauvaise écriture dans le langage de programmation.
- **Les erreurs sémantiques** : observées lors de l'exécution et sont le résultat d'une mauvaise analyse. Ces erreurs sont beaucoup plus graves car elles peuvent se déclencher en cours d'exploitation du programme.

I.2. ALGORITHMES

I.2.1. GÉNÉRALITÉS

D'après LAROUSSE, un Algorithme est « un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations. »
L'Algorithme est l'étape la plus importante pour la résolution d'un problème. Elle permet de détailler le processus que doit entreprendre un ordinateur, via un langage de programmation qui assure la traduction de l'algorithme, pour arriver au résultat. Un algorithme représente la méthode de résolution d'un problème. Il est constitué d'un ensemble d'instructions ou de commandes, placées dans un ordre d'exécution particulier, nécessaire à la bonne marche du processus de calcul.
En général la forme d'un algorithme se présente comme suit :

```
Algorithme Nom ;
var
   Déclaration ;
Début
   Traitement ;
Fin.
```

- Un algorithme commence toujours par le mot '**Algorithme**' suivi de son nom. Ce dernier indique, en général, ce que fait cet algorithme.
- la partie **déclaration** est très importante, car elle indique les différentes données, **variables** et **structures** qui seront **traitées**, ainsi que leur type.
- Le mot '**Début**' indique le début du traitement.

- La partie **traitement** est constituée d'un ensemble d'actions, nécessaire à la résolution du problème. Chaque action représente une opération.
- Le mot '**Fin**' suivi d'un point '.' indique la fin du traitement (ou de l'algorithme).

Déclaration

Pour représenter une donnée (non constante) il faut utiliser un nom de variable pour le stockage de la valeur. Le nom de la variable représente l'adresse mémoire où se trouve la donnée correspondante. Elle doit toujours commencer par une lettre alphabétique.

Exemple

x= 6
delta = alpha + b - 10
y = x + z

7y = alpha – delta
Dans ce cas '**7y** ' n'est pas une variable puisqu'elle commence par un chiffre

x + y = z
La variable, où stocker la valeur, se trouve à droite du signe égal.-→ Cette écriture est fausse.

Il faut plutôt écrire :
z = x + y

La variable, où stocker la valeur, se trouve toujours à gauche du signe égal.

Syntaxe

La syntaxe pour déclarer une variable est la suivante:

Syntaxe
Identité
identificateur : Type ;

Pour le cas de plusieurs variables

Syntaxe :
Identité
Ident1, Ident2, Ident3, … : Type ;

Dans ce cas on remarque que les variables sont séparées par une virgule.

Remarque

Identité : représente l'identité de l'identificateur:

- **Var**: pour les variables

- **Const**: pour les constantes
- **Tableau** : pour les tableaux
- etc…
-

Type : représente le type des données :

- Réel.
- Entier.
- Booléen.
- Caractère.
- etc…
-

Traitement

Le traitement est une suite d'actions qui manipulent généralement des expressions. Ces dernières peuvent être une constante, une variable ou la combinaison des deux à l'aide d'opérateurs : arithmétique, relationnel, logique, chaîne (voir chapitre identification des variables).

On distingue en général 5 opérateurs arithmétiques (voir chapitre I):

- L'addition (+).
- La soustraction (-).
- La multiplication (*).
- La division (/).
- La puissance (^).

Dans la programmation, les expressions seront écrites suivant les conventions de l'algèbre courante sur une ligne. Les parenthèses internes sont évaluées en premier en obéissant à la règle des priorités des opérations suivantes :

a- La puissance.
b- La multiplication et la division en commençant par la plus à gauche.
c- L'addition et la soustraction en commençant par la plus à gauche.

Exemple

Soit la fonction suivante :

$$x = N_u.tg\varphi + c.A - F_h$$

La traduction intégrale de cette équation, en informatique, donnera :

$$x = N_u * tg\varphi + c*A - F_h$$

INSTRUCTIONS DE BASE

Instruction d'affectation

Dans un algorithme, on est appelé à sauvegarder les données et les résultats intermédiaires dans des variables. L'attribution d'une valeur à une variable peut se faire par affectation ou par lecture.

Syntaxe

Syntaxe 1 :

Variable : = expression ;

Syntaxe 2 :

Variable expression ;

Le « : = » et « la flèche inverse » représentent le symbole d'affectation. La valeur issue de l'expression qui se trouve à droite est affectée à la variable qui se trouve à gauche. La variable, qui reçoit le résultat de l'expression, doit être toujours à droite du signe d'affectation.

Instruction d'affichage

Dans l'algorithme précédent, la moyenne ne sera pas visualisée sur l'écran. Pour faire, il faut utiliser l'instruction d'affichage.

Syntaxe

ECRIRE (Expressions) ;
L'expression peut être un commentaire, une constante, une variable, une expression ou une combinaison de ces derniers. Le commentaire est une chaîne de caractères délimitée par deux apostrophes ou deux guillemets.

Exemple

```
Algorithme Affichage ;
  var
     Z : Entier ;
Début
      Z <--- 20 ;
     Ecrire (' La valeur de Z = ', Z ) ;
Fin.
```

La valeur de Z = 20

Instruction de lecture

L'instruction lecture permet de lire une liste de variable, séparée par virgule, dont les données sont saisies ou introduire les données, via le clavier par exemple. Cette instruction met en attente de lecture l'ordinateur.

Syntaxe :

LIRE (Liste de variables) ;

Exemple

```
Algorithme Lecture ;
var
   x, y : Réel ;
Début
    Ecrire ( ' Donnez deux valeurs : ' ) ;  *
    Lire ( x , y ) ;                          **
    Ecrire ( ' La somme est : ' , x + y ) ;  ***
Fin.
```

Remarque

***** L'instruction d'affichage « Ecrire » affiche le message (ou le commentaire) «Donnez deux valeurs : »

** L'instruction de lecture « Lire » suspend l'exécution pour donner le temps à l'utilisateur de saisir autant de valeurs qu'il y a de variables entre parenthèses.

*** Après introduction des deux valeurs, il sera affiché le commentaire « La somme est : » suivi de la valeur de l'expression « x + y »

Instruction de condition simple

Elle permet d'exécuter une ou plusieurs actions, suivant une condition. Cette dernière est une expression logique qui utilise les opérateurs relationnels suivants :
< Inférieur, <= Inférieur ou égal, < > Différent, = Egal, > Supérieur, >= Supérieur ou égale

Syntaxe

SI (condition vraie) **ALORS action**;

Ou bien

SI (condition vraie) **ALORS action**;

Remarque

L'action n'est exécutée que si la condition est vraie

Exemple

```
Algorithme Branchement ;
  var
     x,y : Réel ;
Début
       Ecrire ( ' Donnez deux valeurs :' ) ; 1
       Lire ( x , y ) ; 2
Si ( x > y ) Alors
Ecrire ( ' x est plus grand que y ' ) ; 3
Fin
```

Après introduction des deux valeurs, l'instruction « SI » sollicite le processeur pour vérifier si la condition est vraie, c'est-à-dire si x est supérieur à y.

- SI la condition est vraie, le commentaire « x est plus grand que y » sera affiché.
- Dans le cas contraire, le commentaire ne sera pas affiché.

Il se peut que la condition « Si » concerne plus d'une action. Dans ce cas, sa syntaxe est la suivante

Syntaxe

```
SI ( condition vraie ) ALORS
  Début
        action 1 ;
        action 2 ;
         .....
Fin;
```

Remarque

Pour indiquer à l'ordinateur que la condition « Si » concerne plus d'une action, il faut mettre toutes les actions entre « Début…..Fin ; »

Exemple d'application

Problème
Soit à calculer la moyenne des étudiants, où chaque étudiant possède 3 notes d'examen (X1, X2 et X3).

Analyse
Le problème peut-être décomposé comme suit

A- Collecter les trois notes d'un étudiant. Par exemple :
X1 =
X2 =
X3 =

B- Calculer la moyenne de cet étudiant en utilisant la formule suivante :

$$\text{Moy} = \frac{X_1 + X_2 + X_3}{3}$$

C- Afficher la moyenne

Algorithme

Application N°01

```
1       Algorithme Moyenne ;
2         var
3               X1, X2, X3, Moy : Réel ;
4       Début
5               X1  ← 10.5 ;
6               X2  ← 15.3 ;
7              X3   ←20 ;
8              Moy ←(X1+ X2 + X3) / 3 ;
9              Ecrire ( ' La moyenne = ' , Moy );
10      Fin.
```

Application N°02

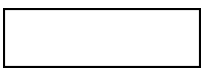
```
1       Algorithme Moyenne ;
2         var
3               X1, X2, X3, Moy : Réel ;
4       Début
5               Ecrire ( 'Donnez les trois notes' );
6               Lire (‘X1, X2, X3) ;
7              Moy ←( X1 + X2 + X3 ) / 3 ;
8              Ecrire ( ' La moyenne = ' , Moy );
9       Fin.
```

Application N°03

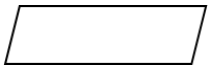
```
1       Algorithme Moyenne ;
2         var
3               X1, X2, X3, Moy : Réel ;
4       Début
5               Ecrire ( 'Donnez les trois notes' );
6               Lire (‘X1, X2, X3) ;
7              Moy ←( X1 + X2 + X3 ) / 3 ;
8        Si Moy>=10
9              Ecrire ( ' La moyenne = ' , Moy );
10             Ecrire ( ' cet étudiant est admis' );
11             Ecrire (‘fin du test’) ;
12      Fin.
```

REPRÉSENTATION GRAPHIQUE OU ORGANIGRAMME

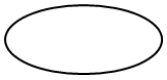
La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l’inconvénient de consommer une place importante. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin. C'est ainsi qu'on a :



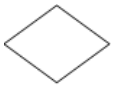
Utilisé pour représenter les calculs



Utilisé pour représenter la lecture et l'affectation



Utilisé pour mentionné le début et la fin



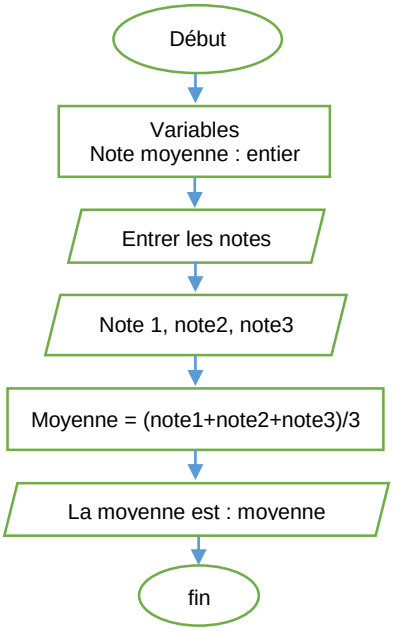
Utilisé pour représenter les Testes

Exemple

Créer un algorithme pour calculer la moyenne de 3 notes.

Solution

- Algo moyenne
  - Variables note, moyenne : entier
  - Début
  - Ecrire (entrer les notes)
  - Lire (note1, note2, note3)
  - Moyenne= (note1+note2+note3)/3
  - Ecrire (la moyenne est : moyenne)
  - Fin



I.1.3. LES STRUCTURES DE CONTRÔLE

On appel structure de contrôle toute action qui permet d'effectuer un groupe d'actions sous condition(s), et permet donc d'orienter le déroulement d'un programme suivant la réalisation de ces conditions.

ACTION CONDITIONNELLE OU ALTERNATIVE

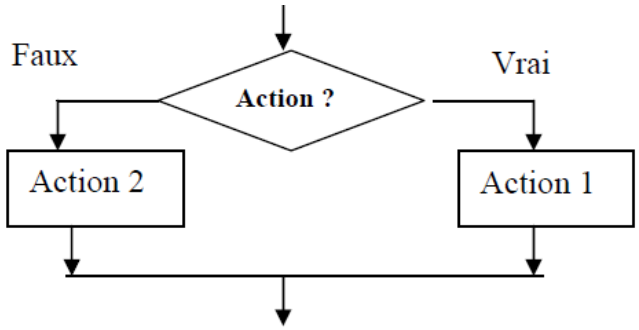
Syntaxe

Si <Condition> **alors** <Action 1>
Sinon <Action 2 >
Fin si

Cette action s'exécute de la manière suivante :

- Si la condition définie dans <**Condition**> est vérifiée il faut exécuter l'action (*ou le groupe d'action*) définie dans <**Action 1**>, puis les actions qui suivent le fin si
- Si cette condition n'est pas vérifiée il faut exécuter l'action (*ou le groupe d'actions*) définie dans <**Action 2**>, puis les actions qui suivent le fin si.

Organigramme

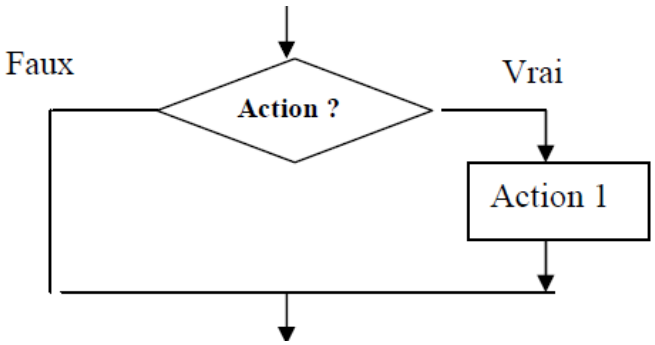


Il existe une seconde forme de cette action conditionnelle :
Si <Condition> **alors** <Action 1> **Fsi**

Et dans ce cas :

- Si la condition <Condition> est vérifiée, l'action (*ou le groupe d'actions*) <**Action 1**> est exécutée puis les actions qui suivent le fsi.
- Par contre si la condition n'est pas vérifiée, seules les actions qui suivent le fsi seront exécutées.

Organigramme



ACTION RÉPÉTITIVE 'Tant que'

Syntaxe

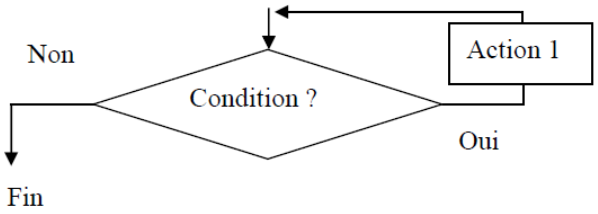
Tant que <Condition> **faire**
<Action>
Fin tant que ;

Cette action spécifie qu'une action (*ou groupe d'actions*) doit être répétée tant que la condition reste vérifiée.

Remarque

L'action <Action> étant répétée jusqu'à ce que la condition devienne fausse, il faut donc que la partie action soit capable de modifier les paramètres intervenant dans la condition **condition** afin de sortir de tant que.

Organigramme



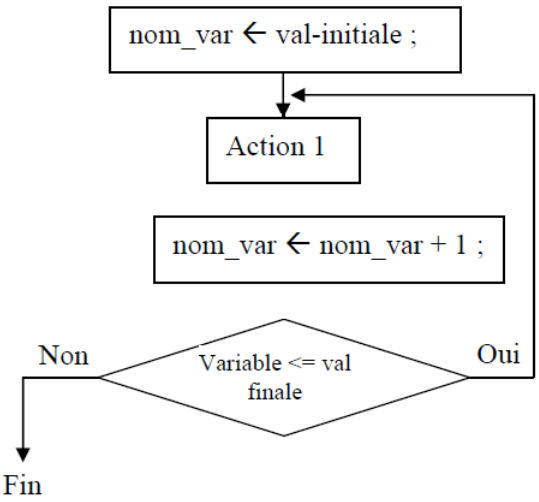
ACTION RÉPÉTITIVE 'POUR'

Syntaxe

Pour <nom_var> **allant de** <val_initiale> **à** <val_finale> **faire**
<Action>
Fait ;

Cette action permet de répéter une action (*ou groupe d'actions*) un nombre de fois déterminé par les valeurs initiale et finale de paramètre <nom_var>. La partie action est exécutée la première fois en affectant au paramètre <nom_var> que l'on appelle aussi compteur, la valeur <val_initiale>. Apres chaque exécution on rajoute au compteur la valeur 1. L'action est ré exécuté jusqu'à ce que la valeur du compteur dépasse la valeur spécifiée par <val_finale>.

Organigramme



I.1.4. UTILISATION DES PROCÉDURES ET FONCTION

Les fonctions et les procédures sont appelés aussi les sous-programmes, elles fournissent aux programmeurs un moyen simple d'abstraction en lui permettant de nommer une séquence d'instructions et de l'appeler autant de fois qu'il sera nécessaire au cours d'un même programme. Un sous-programme (*fonction ou procédure*) à la même structure qu'un programme, les seules différences importantes sont les notions de paramètres et de variables locales.

LES FONCTION

Une fonction est un ensemble d'instructions qui forment un sous-programme, les fonctions en algorithmique (*programmation*) ressemblent à celles de mathématique, chaque fois que l'on appelle elle renvoie au programme appelant une valeur qui est le résultat du traitement effectués par des instructions de la fonction. Une fonction peut renvoyer n'importe quel type de base.

DÉCLARATION DES FONCTIONS

La structure générale d'une fonction est comme suit :

Syntaxe algorithmique

Fonction NomFonction (Var1 : **type1**, Var2 : **type2**, …VarN : **typeN**) : **Type** ;
{l'entête de la fonction}
Var variable1, …VariableN : **type** ; {déclaration des variables propres à la fonction}
Begin
Instructions ; {corps de la fonction}
End ;
Où :

- Nomfonction est le nom de la fonction (Identificateur)
- ***Type1, Type2, …,TypeN*** sont les type des paramètres de la fonction
- **Type** est le type de résultat retourné.
- Instructions est le calcul effectué par la fonction.

Plus de détail voir chapitre suivants.

Remarque

UTILISATION DES FONCTIONS

Une fonction, une fois déclarée, pourra être appelée depuis le programme principal, ou une autre fonction.

Exemple

```
1    Algorithme CalculVal ;
2    Var Val, n, p : entiers
3
4    Fonction fact (n : entier) : entier
5    Var t, i : entiers ;
6
7    Debut
8    T ← 1 ;
9    Pour i allant de 1 à n faire
10   Debut
11   T←t*i ;
12   Fin ;
13   Fact←p ; {le résultat doit être affecté au
     nom de la fonction}
14   Fin ;
15   (Pour appeler la fonction dans le
     programme principale)
16
17   Debut
18   Ecrire ('donner la valeur de n et p') ;
19   Lire (n,p) ;
20   Val← fact(n)/(fact(n-p)*fact(p)) ;
21   Ecrire ('Val=', Val);
22   Fin.
```

APPEL D'UNE FONCTION

À la fin d'une fonction, il faut affecter le résultat du traitement au nom de la fonction.

Pour appeler une fonction, on fait apparaître le nom de la fonction à droite d'une affectation de variable souvent appelée variable de retour.

Syntaxe

Variable _ Nom_fonction (paramètres) ;

EXERCICE D'APPLICATION

Écrire un algorithme (*programme*) qui permet de calculer la moyenne et l'écart type des éléments d'un tableau à une dimension donné.

```
1    Program Examp;
2    Var moy, ecrt, :real;
3    Taille, j: integer;
4    Tab: array[1..100] of real;
5    Procedure moy_ecart(taille1, var moy1: real, var
     ecrt1:real);
6    Var s1,s2:real;
```

```
7    i:integer
8    Begin
9    S1:=0;
10   For i:=1 to taille1 do
11   S1:=s1+tab[i];
12   Moy1:=s1/taille1;
13
14   S2:=0;
15   For i:=1 to taille1 do
16   S2:=s2+(tab[i]-moy1)* (tab[i]-moy1);
17   S2:=SQRT(s2);
18   ecrt1:=s2/SQRT(taille1);
19   end;
20
21   begin
22   write (‘donner la taille du tableau’);
23   read (taille) ;
24   write (‘remplissage du tableau’) ;
25   for j :=1 to taille do
26   read(tab[j]) ;
27   moy_ecart(taille, moy, ecrt);
28   write (‘la moyenne des éléments du tableau est’, moy) ;
29   write (‘l’’ecart type des éléments du tableau est’, ecrt) ;
30   end.
```

CHAPITRE II : INITIATION À MATLAB

Le nom MATALAB provient de la combinaison des deux mots MATrix-LABoratory. L'objectif initial était de fournir un accès simplifié aux bibliothèques de fonctions des projets LINPACK et EISPACK (dédiées au calcul matriciel et l'algèbre linéaire). Le logiciel MATALAB est pourvu d'une interface interactive et conviviale, et permet avec une grande flexibilité d'effectuer des calculs numériques, symboliques et des visualisations graphiques de très haute qualité.

MATALAB a de nombreux avantages par rapport aux langages de programmation traditionnels (tel que le C++, FORTRAN…). la popularité de MATALAB est due principalement à sa forte et simple interaction avec l'utilisateur mais aussi aux points suivants :

♦ ***Sa richesse fonctionnelle*** : avec MATALAB, il est possible de réaliser des manipulations mathématiques complexes en écrivant peu d'instructions. Il peut évaluer des expressions, dessiner des graphiques et exécuter des programmes classiques. Et surtout, il permet l'utilisation directe de plusieurs milliers de fonctions prédéfinie.

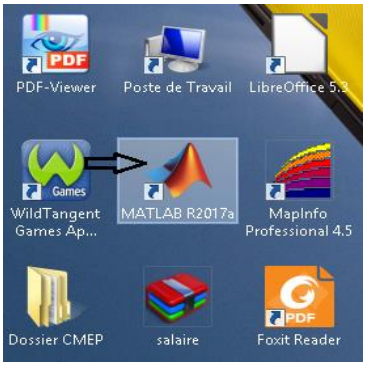
♦ ***Les fonctions préprogrammées*** : la bibliothèque de MATALAB fournit de nombreuses fonctions préprogrammées regroupées en boîtes à outils (toolbox) pour de nombreux domaines (par ex : statistique, calcul probabiliste optimisation, ...). Cette option rend La structure de données de base est le tableau ne nécessitant pas de dimensionnement comme les anciens programmes.

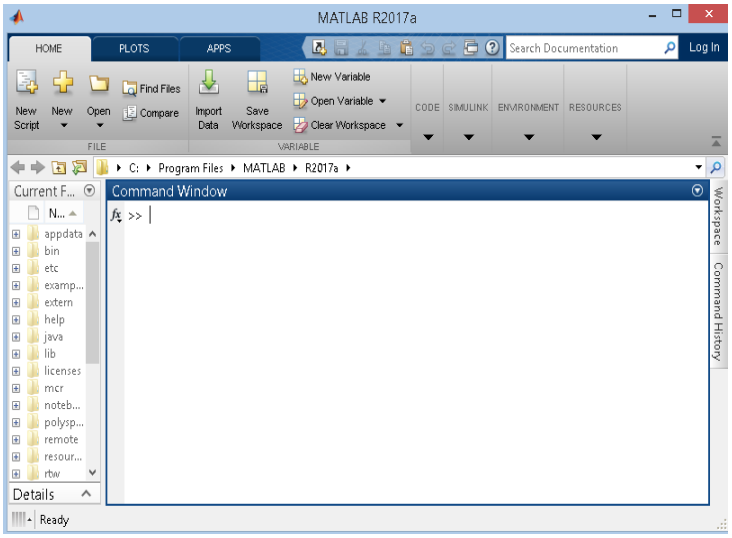
♦ ***La simplicité de son langage de programmation*** : un programme écrit en MATALAB est plus facile à écrire et à lire comparé au même programme écrit en C ou en PASCAL.

♦ ***Sa manière de tout gérer comme étant des matrices***, ce qui libère l'utilisateur de s'occuper de typage de données et ainsi de lui éviter les problèmes de transtypage.

II.1. DÉMARRAGE ET QUITTER

Pour lancer le programme, double clic sur raccourci de MATALAB qui apparait sur le bureau ou bien tapez **MATALAB** dans une fenêtre de commandes. Une fenêtre logo fait une brève apparition, puis dans la fenêtre de commandes, le **symbole >>** apparaît : c'est l'invité de MATALAB qui attend vos commandes.





Pour quitter la session MATLAB, écrire la commande **quit** sur la commande Windows**.**

II.2. INTERFACE GRAPHIQUE

Au lancement de MATALAB pour la première fois, l'écran ressemble à celui de la Figure 1. Le 'bureau' MATALAB est une fenêtre contenant d'autres sous-fenêtres, qui sont :

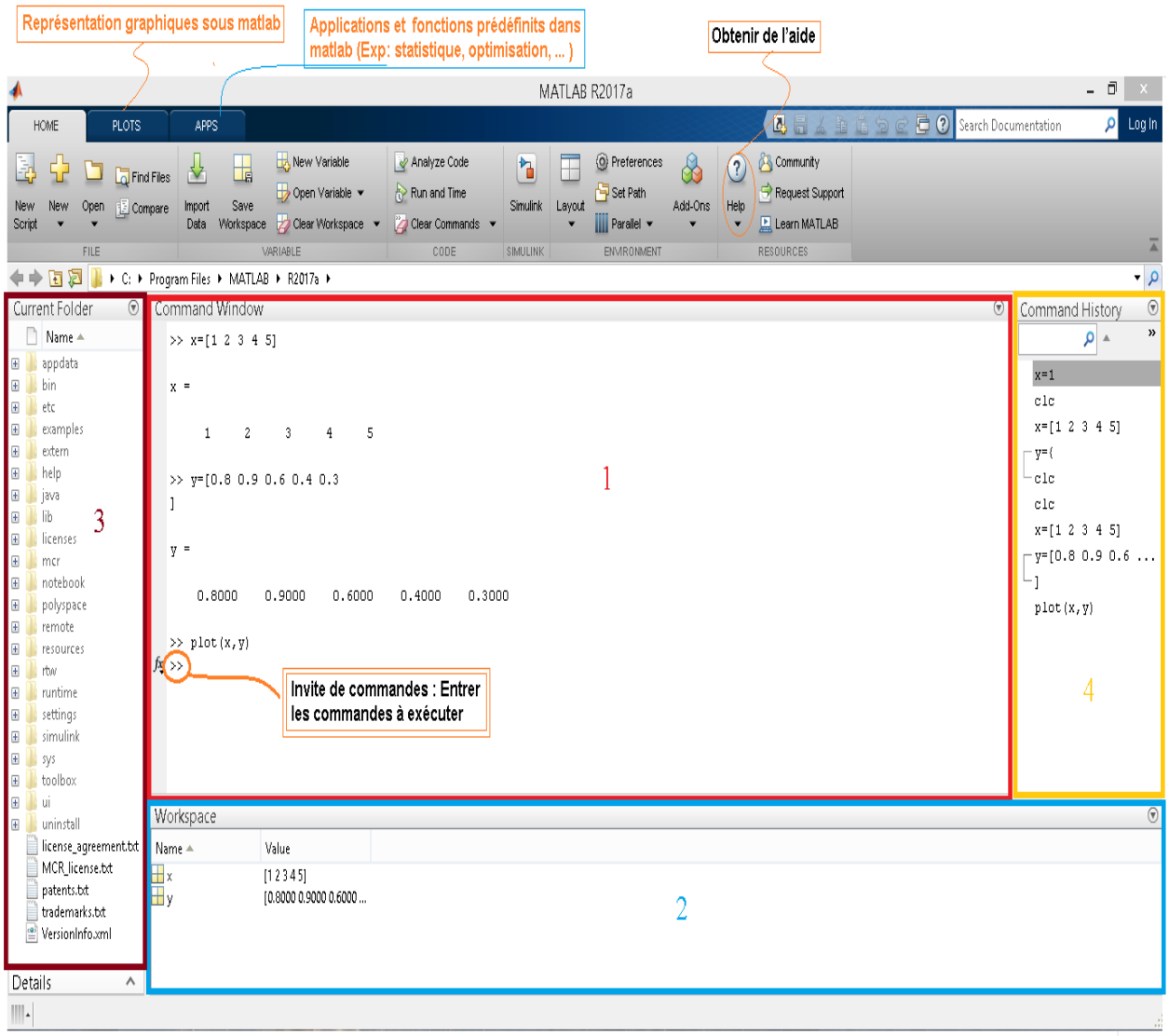
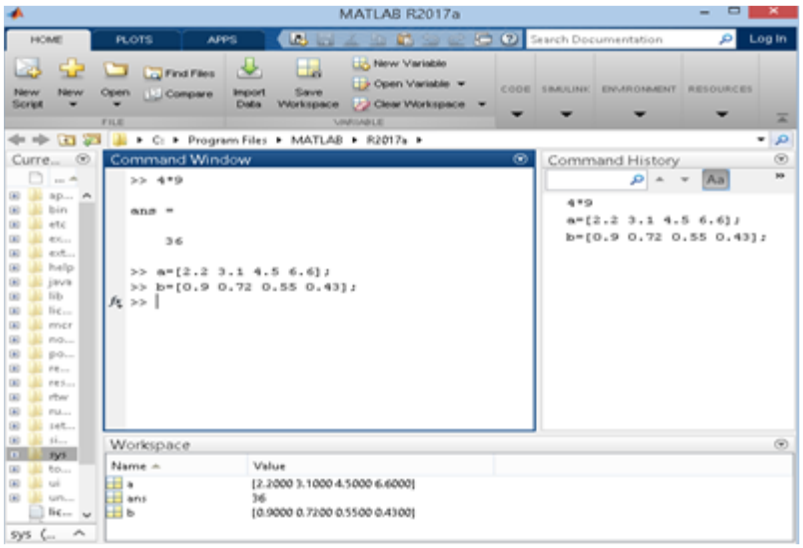


Figure II.1. Interface MATALAB.

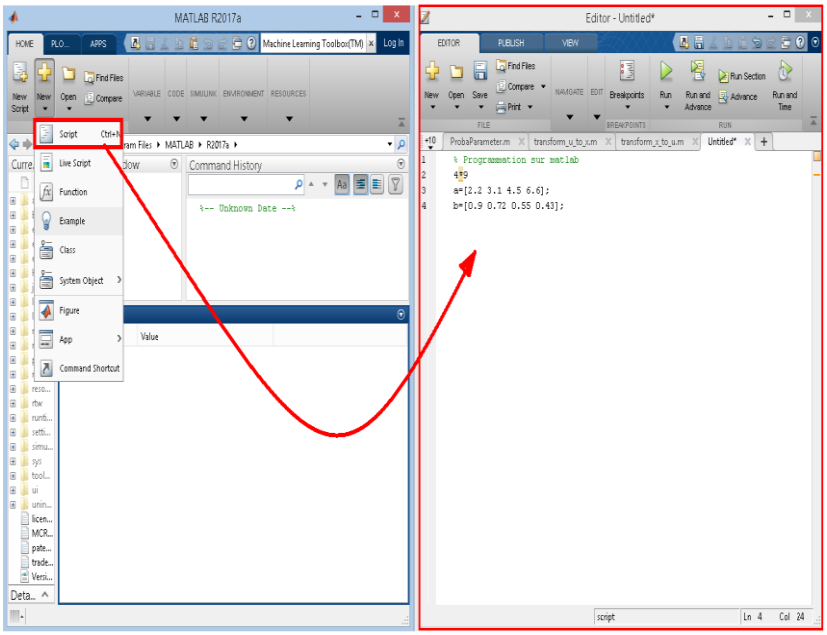
1. **COMMAND WINDOW** : est la fenêtre de centrale de l'interface (invité de commande). À partir de la command window, l'utilisateur pourra lancer les commandes, taper des instructions, appeler des scripts et d'exécuter des fonctions MATALAB. Le principe est simple, il est basé sur la connaissance des fonctions et le respect de leur syntaxe. dans cette commande, le signe (≫, invité de commande) indique que MATALAB est prêt à recevoir les commandes pour réaliser des calculs.
2. **WORKSPACE** : il affiche le nom, le type ainsi que la taille des variables exécutées/en mémoire, il permet également de parcourir graphiquement le contenu des variables.

3. **CURRENT FOLDER** : indique le répertoire courant ainsi que les fichiers existants. Cette commande permet aussi d'affiché le répertoire de travail courant avec son chemin (path en anglais)
4. **COMMAND HISTORY** : garde la trace de toutes les commandes entrées par l'utilisateur.

La programmation sur MATALAB fait appel à des commandes qui créent des variables et des appels de fonction, qui seront afficher dans la fenêtre commande windows et automatiquement ajouter dans le Worksace.



Dans la programmation en MATALAB, pour plus de flexibilité il serait recommandé d'écrire les instructions directement dans l'éditeur de texte intégré dans MATALAB. Cet éditeur est lance soit :



♦ En tapant la commande ***edit*** dans la fenêtre des commandes.

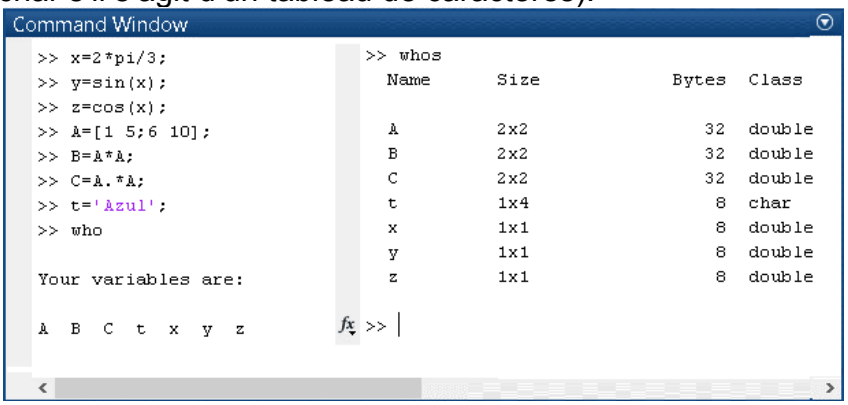
♦ En cliquant l'icône new M-file. Ainsi, on utilisera l'expression script MATALAB.

A l'enregistrement, le fichier portera l'extension **nomFichie.m**. les fichiers dont le nom compte un espace (**nom fichier**) ou des (-, /,%) ne sera pas pris en considération par MATALAB.

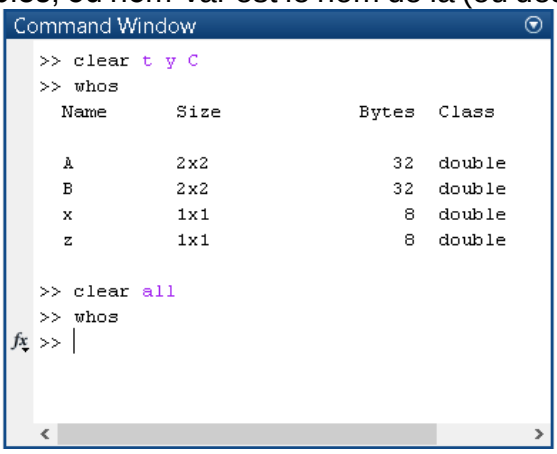
Dans les fichiers M-file, tout ce qui est écrit après le signe pourcentage (%) est un commentaire, dont MATALAB ne tiendra pas compte lors de l'exécution du programme.

II.3. ESPACE DE TRAVAIL

Dans le langage MATALAB, les variables sont définies au fur et à mesure que l'on donne leurs noms et leurs valeurs numériques ou leurs expressions mathématiques. MATALAB ne nécessite pas de déclaration de type ou de dimension pour une variable. Les variables sont stockées dans l'espace de travail (ou **workspace**) et peuvent être utilisées dans les calculs subséquents.
Pour obtenir la liste des variables actives de l'espace de travail on dispose des commandes **who** et **whos**. La commande **who** affiche le nom des variables actives. La commande **whos** donne plus d'informations : le nom, la taille du tableau (nombre de lignes et de colonnes) associé, l'espace mémoire utilisé (en Bytes) et la classe des données (principalement double array s'il s'agit d'un tableau de valeurs réelles ou complexes et char s'il s'agit d'un tableau de caractères).



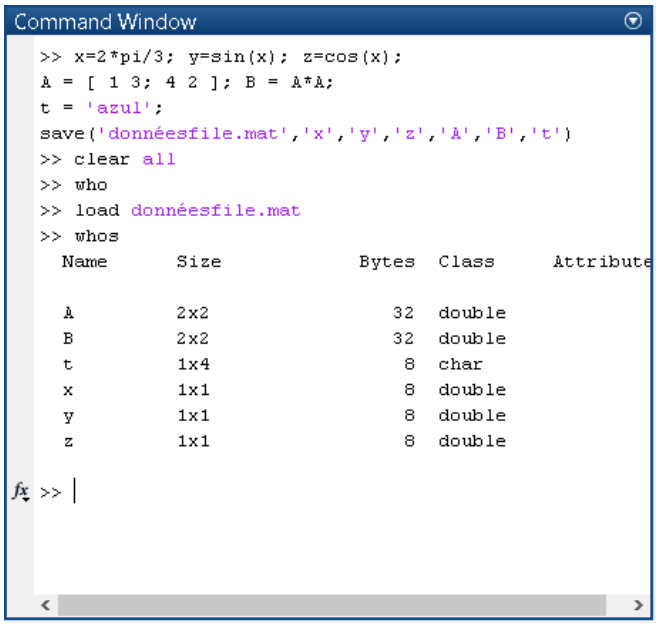
Il est possible de nettoyer l'espace de travail, pour cela la commande **clear** est utilisée. Cette commande permet de nettoyer tous l'espace de travail avec la syntaxe **clear all**, ou toutes les variables sont détruites. La syntaxe **clear nom-var** permet de détruire qu'une partie des variables, où nom-var est le nom de la (ou des) variable(s) à détruire.



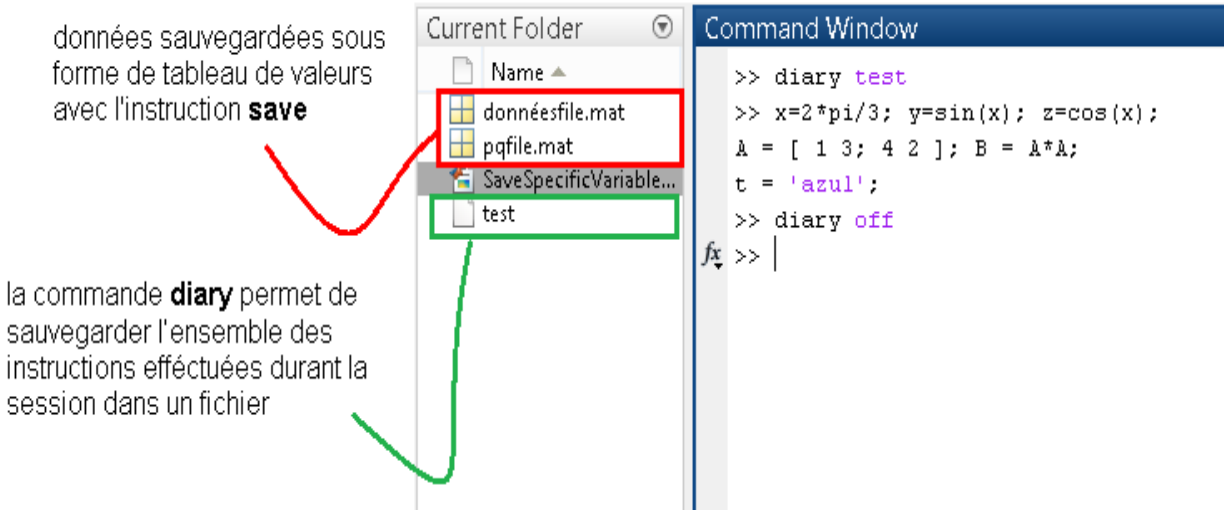
En termes de sauvegarde, MATALAB offre la possibilité de sauvegarde une session pour une utilisation ultérieure, ou l'instruction **save nom-fichier** est utilisée. Cette instruction permet d'enregistré dans un fichier d'extension **nom-fichier,** toutes les variables de l'espace de travail. Seul le contenu des variables est sauvegardé et non pas l'ensemble des instructions effectuées.

Pour ramener dans l'espace de travail les variables sauvegardées dans le fichier **nom-fichier.mat**, taper **load nom-fichier**.

Dans l'exemple suivant, On sauve ces résultats dans un fichier donnéesfile.mat. Après le nettoyage de l'espace de travail (ou bien après avoir quitté MATALAB), on tape la syntaxe **laod donnéesfile.mat**, on peut vérifier que les variables (x y z A B t) existent avec les valeurs précédentes (figure suivante).

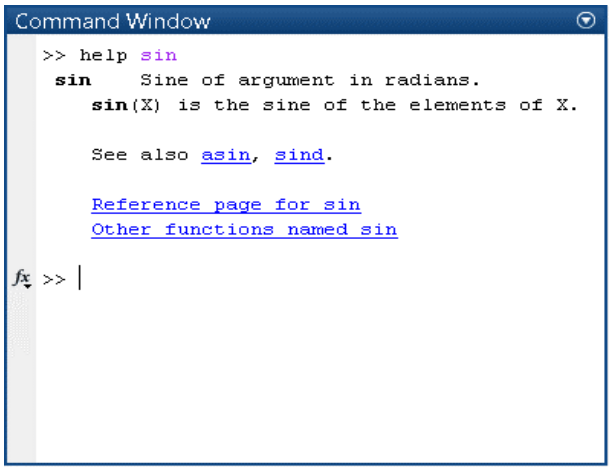


La commande **diary** permet de sauvegarder l'ensemble d'une session dans un fichier ''**ascii** '' pour un traitement ultérieur (insertion dans un document, impression, . . .). Le fichier de sauvegarde par défaut a pour nom **diary**. On provoque la sauvegarde dans le fichier **nom-fic**, par l'instruction **diary nom-fic**. Attention, c'est la commande **diary** qui déclenche le début de sauvegarde ; il est impossible de sauvegarder la partie de la session précédent son appel. Pour arrêter la sauvegarde, taper **diary off**.



II.4. AIDE ET DOCUMENTATION EN LIGNES

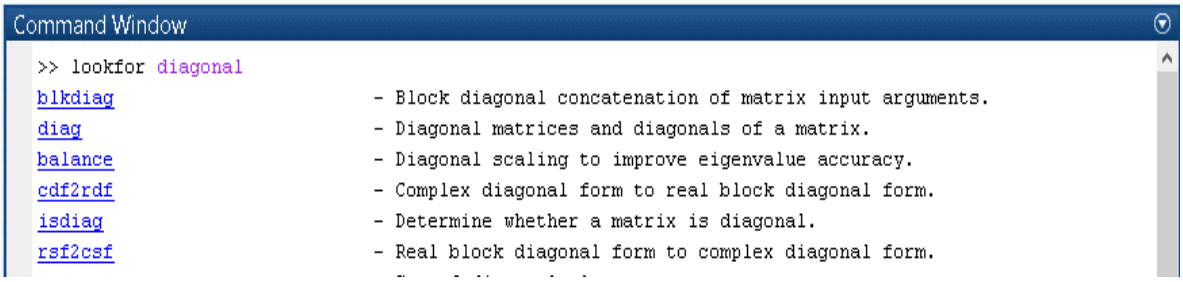
Le HELP BROWSER est un navigateur permettant de parcourir l'aide de MATLAB. L'aide est un outil précieux pour trouver les fonctions et apprendre leur fonctionnement (notamment le format des données à fournir en entrée ainsi que les valeurs renvoyées par la fonction). Pour avoir de l'aide sur sur une commande MATLAB il suffit de tape dans la command window **>> help nom-commande** (fig).



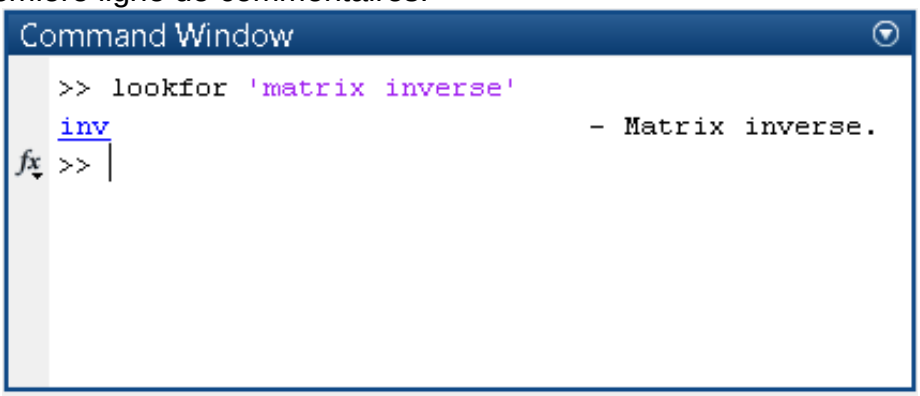
Les commandes MATLAB doivent être tapées en minuscules pour être reconnues.
De la même manière, l'aide de MATLAB est peut être obtenu par le biais de la commande ***doc*** qui donne accès à une documentation complète au L'environnement de MATLAB , sous format HTML.

Une autre syntaxe qui facilite la recherche dans la documentation matlab est noté lookfor. Cette commande permet de rechercher un mot-clé, rédigé en anglais, parmi les lignes de commentaires en entête des fonctions MATLAB. Cette commande s'écrit :

- ***lookfor mot clé :*** recherche le mot-clé dans la première ligne de commentaire de toutes les fonctions MATLAB.
- ***lookfor mot clé-all*** : recherche le mot-clé dans toutes les lignes de commentaires en entête des fonctions.



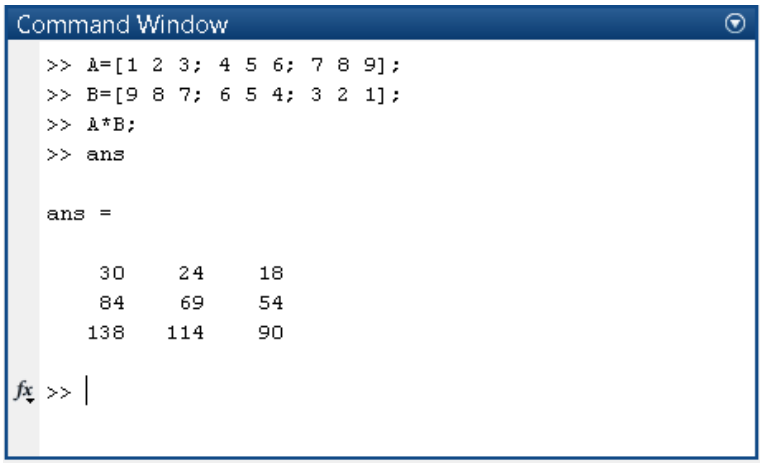
- ***lookfor ' '*** : utilise une phrase comme mot clés, cette écriture permet de recherche le nom de la fonction MATLAB (ou des fonctions) où le mot-clé figure dans la première ligne de commentaires.



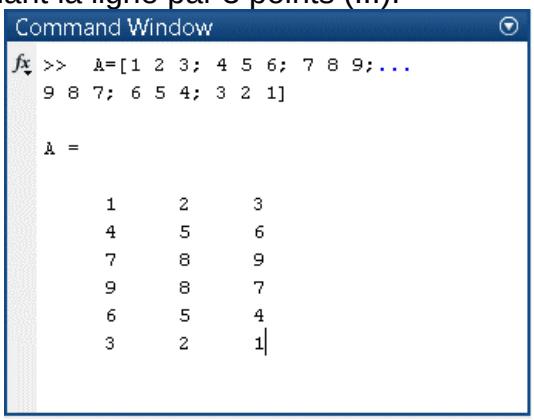
MATLAB dispose d'une aide accessible en ligne en consultant le site de la société MathWorks, à l'adresse https://fr.mathworks.com/help/.

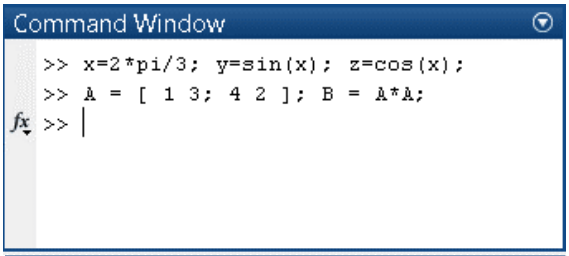
II.5. SYNTAXE D'UNE LIGNE D'INSTRUCTIONS

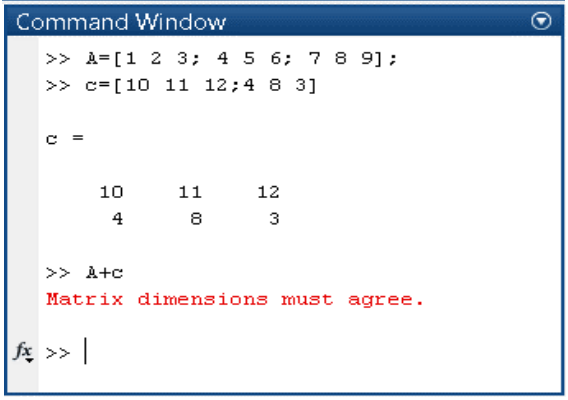
Si une instruction MATLAB est suivie d'un point-virgule, le résultat de cette instruction n'est pas affiché. Pour réafficher un résultat contenu dans une variable, il suffit de taper le nom de la variable. Le résultat de la dernière instruction exécutée peut être rappelé par la commande ans



Plusieurs instructions MATLAB peuvent figurer sur une même ligne. Il faut alors les séparer par une virgule ou par un point-virgule. D'autre part, si une commande est trop longue pour tenir sur une ligne, il est possible de poursuivre sur la ligne suivante en terminant la ligne par 3 points (...).







Si la syntaxe de l'instruction soumise est erronée ou si on cherche à exécuter une instruction illégale (qui n'a pas de sens mathématique par exemple), MATLAB affiche un message d'erreur, ou on trouve les sources d'erreurs possibles qui permettent de corriger rapidement l'erreur commise.

II.6. SYNTAXE DES PRINCIPALES COMMANDES PRÉDÉFINIT EN MATLAB

La bibliothèque de matlab est assez riche en matière de commandes et fonctions, ce qui simplifie la programmation.

II.6.1. LES COMMANDES

MATLAB nous offre une bibliothèque de commande très riche pour l'interaction avec l'utilisateur, les commandes de l'ordre général sont mentionnées dans le tableau suivant. Les commandes spécifiques pour certaine tache seront exposées au fur et à mesure de l'avancement du cours.

Commandes	Signification
Who	Affiche le nombre de variables utilisées
Whos	Affiche des informations sur les variables utilisées
Clear , clear all	Supprimer tout
Clear A C	Supprimer les variables A et C
Clc	Effacer l'écran
Exit, quit	Fermer l'environnement MATLAB
Format	Définit le format d'affichage des valeurs numériques (format long : affiche 14 chiffres après la virgule) (format short : affiche 4 chiffres après la virgule) (format bank : affiche avec 2 chiffres après la virgule) (Format rat : affiche les nombres sous forme d'un ratio (a/b))
Close, close all	Ferme tout
Load	Télécharge un fichier
Save	Enregistre les données
Help	Demande de l'aide

II.6.3. OPÉRATEURS ARITHMÉTIQUES

Les opérateurs arithmétiques s'appliquent aux scalaires, aux vecteurs et aux matrices. Il faut juste vérifier que les opérandes soient compatibles

Opérateurs	Signe
Addition	+
Soustraction	-
Division	/
Multiplication	*
Exponentiation	^

II.6.4. OPÉRATEURS LOGIQUE ET DE COMPARAISON

Opérateurs	Descriptions
<	Inférieur à
<=	inférieur ou égal à
>	supérieur
>=	supérieur ou égal à
==	exactement égale à
!=	Différent
x|y	x ou y
&	ET
isTRUE(x)	Teste si x est vrai

CHAPITRE III : VARIABLES ET FONCTIONS DANS MATALAB

III.1. NOTION DE VARIABLES ET AFFECTATION DE VALEUR

Dans le langage MATLAB, les calculs complexes sont scindés en plusieurs étapes, et les résultats intermédiaire sont stocker dans des variables, afin de pouvoir les réutiliser dans d'autres calculs.
Pour stocker une valeur ou le résultat d'un calcul dans une variable, il suffit d'affecter cette valeur ou ce résultat à la variable. Pour affecter une variable, on déclare simplement à quoi elle est égale.

```
>> x=5
x =
    5
```

Dans cette instruction, on affecte à la variable x la valeur de 5.
Dans MATLAB, une fois une valeur ou un résultat est affecté à une variable, celle–ci est automatiquement créée si elle n'existe pas déjà, afin d'y stocker cette valeur ou ce résultat. Si elle existe déjà, le contenu de cette variable sera remplacé par celui indiqué à la droite du signe égal.
Contrairement à la plupart des langages comme Fortran, l'affectation d'une variable est immédiate, et s'écrit en une seule étape. Il n'est pas nécessaire de déclarer les variables avant de les utiliser, ni de les typer. Leur type (entier, réel, complexe, chaîne de caractères, tableaux, *etc*) s'affectera automatiquement en fonction du calcul effectué ou de la valeur affectée.

Remarques

Dans la programmation en matlab, il faut faire attention aux points suivants :

- matlab différencie majuscules et minuscules. Ainsi X1 et x1 désignent deux variables distinctes.
- Les variables sont définies au fur et à mesure que l'on donne leurs noms (identificateur) et leurs valeurs numériques ou leurs expressions mathématiques.
- L'utilisation de variables avec matlab ne nécessite pas de déclaration de type ou de dimension.
- Pour matlab toute variable est considéré comme étant un tableau d'éléments d'un type donné. trois formes de tableaux sont prises en considération par matlab :
  - les scalaires : qui sont des tableaux à une ligne et une colonne.
  - Les vecteurs : qui sont des tableaux à une ligne ou une colonne.
  - Les matrices : Les matrices qui sont des tableaux ayant plusieurs lignes et colonnes.

III.2. LES VARIABLES DANS LES CALCULS SIMPLE

Introduisant la variable x=5, que l'on vient de déclarer dans de nouvelle expression mathématiques, pour en définir une nouvelle variable :

```
>> y=8*x+x^2/2
y =
  52.5000
```

Ensuite utiliser ces deux variables :

```
>> y=2*x+y/2
y =
  16.5625
```

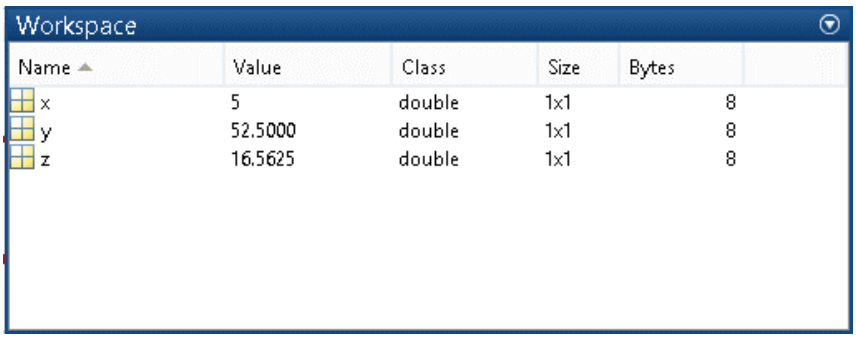
Les variables x, y et z ne sont pas affichés en permanence à l'écran. Mais pour voir le contenu d'une variable, rien de plus simple, il suffit de taper son nom.

III.3. PRINCIPAUX TYPES DE VARIABLES

Comme indiqué précédemment, il n'est pas nécessaire de prédéfinir le type d'une variable, car MATLAB définit automatiquement le type de variables nécessaire lors de son affectation. La fenêtre **Workspace** montre les variables définies et donne accès à un ensemble d'informations.

Le **Workspace** affiche toutes les variables présentes en mémoire, telles que :

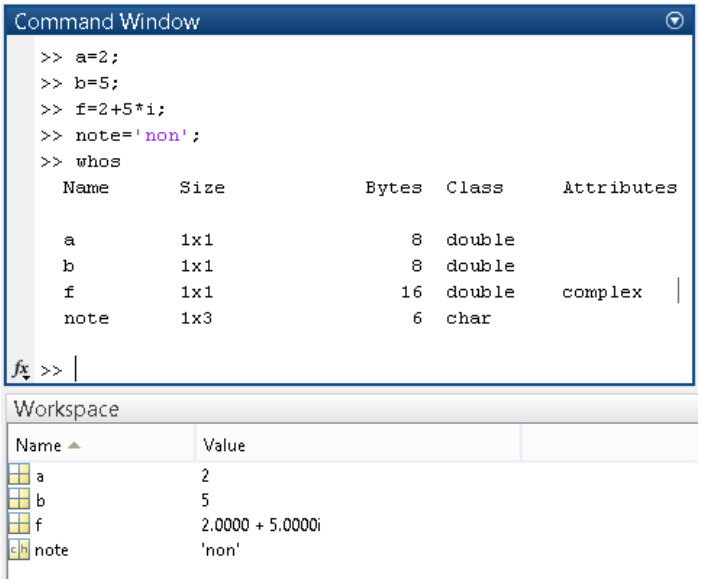
- le nom de la variable ;
- sa valeur ;
- son type (*class* en anglais) ;
- sa taille (*size*) ;
- son occupation mémoire (en *Bytes*, 1 *byte* valant 8 bits, comme un octet en français).



Le type principal qu'utilise MATLAB est le double. Un double est un nombre stocké en double précision, c'est-à-dire sur 64bits (ou 32bits, selon l'architecture matériel de l'ordinateur que vous utilisez).

Cela signifie que contrairement à la plupart des langages, MATLAB code de la même manière l'entier 1, et le réel 1.0.

En matlab, une variable peut être de type **réel, complexe, chaîne de caractères ou logique** établi automatiquement à partir des valeurs affectées à la variable. Le type logique est associé au résultat de certaines fonctions.



Remarque

Il existe dans matlab des commandes qui permet de déterminer le type de la variable, nous pouvons citer :

- **ischar(identificateur-variable)** : retourne 1 si x est de type chaîne de caractères et 0 sinon.
- **isreal(identificateur-variable)** : est à utiliser avec discernement : elle retourne 1 si x est réel ou de type chaîne de caractères et 0 sinon (x est complexe à partie imaginaire non nulle ou n'est pas un tableau de valeurs réelles ou de caractères).
- **islogical(identificateur-variable)** : affiche 1 si x est de type logique et 0 sinon.

```
>> a=2;
>> b=5;
>> f=2+5*i
>> note='non';

>> ischar(a)        ans =
                       logical
                                  0
>> isreal(a)       ans =
                      logical
                                   1

>> isreal(note)     ans =
                      logical
                                   1
>> islogical(f)     ans =
                      logical
                                  0
```

III.3.1. TYPE COMPLEXE

En mathématiques, l'ensemble des **nombres complexes** est créé comme extension de l'ensemble des **nombres** réels, contenant en particulier un **nombre** imaginaire, désigné par i ou j. Tout **nombre complexe** peut s'écrire sous :

- la forme cartésienne a **+ i b**, **a + i*b**, **a+b*i**, **a+bi**
- la forme polaire **r*exp(it)** et **r*exp(i*t)**

Avec **a, b, r** et **t** des variables de type réel.

```
>> a=[2+3i, 5, 8i]
a =
  Columns 1                 through 2              Column 3
   2.0000 + 3.0000i      5.0000 + 0.0000i      0.0000 + 8.0000i
```

```
>> b=[2+3i, 5, 8i]
b =
  Columns 1            through 2          Columns 3          through 4          Column 5
5.0000+0.0000i   0.0000+1.0000i   5.0000+0.0000i   8.0000+0.0000i   0.0000+1.0000i
```

Note

Il faut faire attention à ne pas laisser d'espace autour de l'unité imaginaire afin d'éviter de mauvaises interprétations des données dans certains cas.
Considérant Z un nombre de type complexe, ainsi :

- **imag(Z)** et **real(Z)** retournent la
- partie imaginaire et la partie réelle respectivement de Z.
- **abs(Z)** retournent le module
- **angle(Z)** retournent l'argument de Z

```
>> z=2+3i          z =
                        2.0000 + 3.0000i

>> real(z)        ans =    2

>> imag(z)        ans =    3

>> abs(z)         ans =     3.6056

>> angle(z)       ans =     0.9828

>> conj(z)        ans =     2.0000 - 3.0000i
```

Les commandes **imag**, **real**, **abs**, **angle** permettent aussi de passer aisément de la forme polaire à la forme cartésienne et réciproquement.

Remarque

Il est possible que des variables de noms i ou j aient été redéfinies au cours d'un calcul antérieur et soient toujours actives, alors on peut soit :

- Détruire ces deux variables (**clear i,j**), et j redeviennent alors l'unité imaginaire
- Ré-affecter à i ou à j la valeur unité imaginaire par l'instruction i=sqrt(-1). Avec cette instruction on se méfiera donc des boucles d'indices i et j dans lesquelles on manipule des variables complexes.

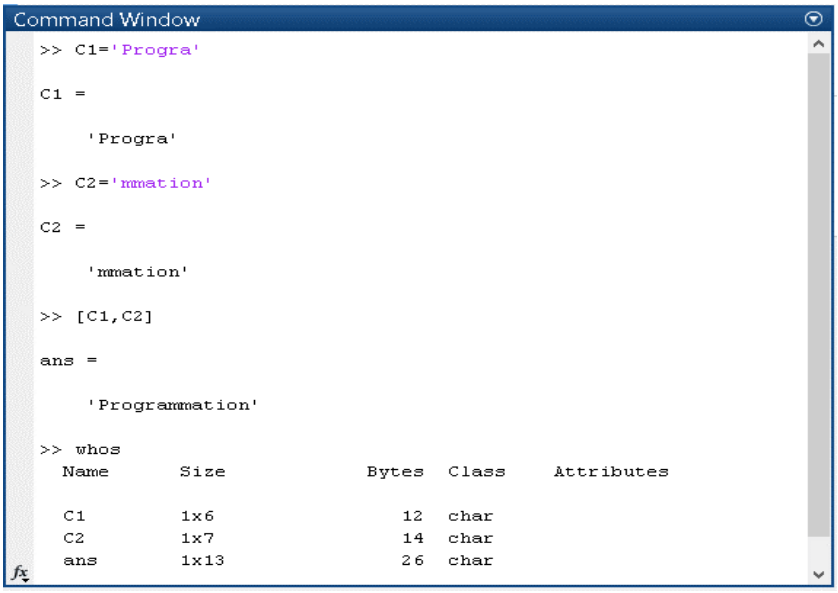
III.3.2. TYPE CHAINE DE CARACTÈRES

Une chaîne de caractères est un tableau de caractères. Une donnée de type chaîne de caractères (char) est représentée sous la forme d'une suite de caractères encadrée d'apostrophes simples (').
On peut effectuer différentes manipulations d'une chaîne de caractères :

Remarque

Si une chaîne de caractères doit contenir le caractère apostrophe (') celui-ci doit être doublé dans la chaîne (ainsi pour affecter le caractère apostrophe (') à une variable on devra écrire "", soit 4 apostrophes).

```
>> C3='Aujourd'huit'
 C3='Aujourd'huit'
          ↑
Error: Unexpected MATLAB expression.
>> C3='Aujourd''huit'
C3 =
   'Aujourd'huit'
```

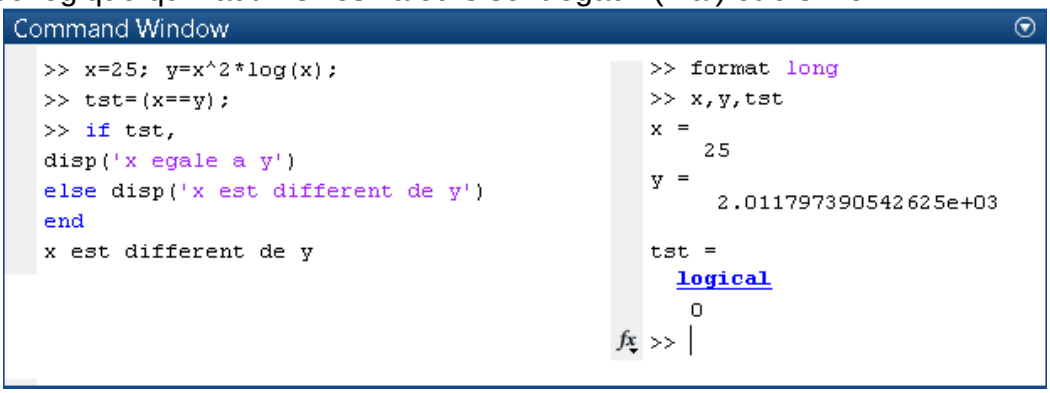


III.3.3. TYPE LOGIQUE

Une variable de type logique (logical) possède 2 formes : 0 pour faux et 1 pour vrai. Ce type de variable est utilisé par certaines fonctions ou dans le cas de certains tests (test logique de la méthode de Monté Carlo). Considérant deux variables x et y. la variable x contient la valeur 25, la variable y est définit par :

$$Y=x^2 * \log(x)$$

Il est demandé de test si les variables x et y contiennent les mêmes valeurs. Pour résoudre ce problème la commande tst est utilisé, cette commande est une variable de type logique qui vaut 1 si les valeurs sont égaux (vrai) et 0 si non.



III.3.4. AUTRES TYPES DE VARIABLES, SPÉCIFIQUES À MATLAB

Jusqu'ici, nous avons vu que nous pouvions stocker une valeur dans une variable. Par extension, nous verrons par la suite que l'on peut stocker un tableau de valeurs numériques dans une matrice, et les manipuler par des opérations terme à terme par exemple.

Les tableaux (**array** en anglais) permettent de stocker un ensemble de valeurs, à condition qu'elles soient toutes homogènes en type. Cela signifie que l'on peut avoir un tableau de doubles, où toutes les valeurs seront de type double, par exemple ; ou un tableau de nombres complexes, où toutes les valeurs seront de type complexe.

Il existe des types de variables plus complexes, que MATLAB a introduit au fil du temps, pour permettre la gestion d'un ensemble de données pouvant être hétérogènes, ou constitué d'éléments de types autres que numériques.

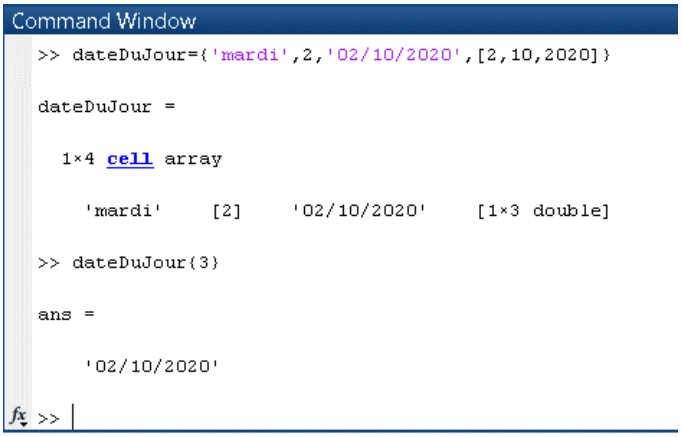
cell : type parfait pour gérer des tableaux de données hétérogènes

Une variable *cell* (cellule) est un tableau où chaque cellule peut être d'un type spécifique. On peut donc stocker dans une même variable un ensemble de données hétérogènes en type.

Syntaxe

Les accolades **{}** permettent de créer des ***cell*** :

Application



struct : type parfait pour créer des variables structurées

Une variable de type ***struct*** est un tableau structuré. Ce type de donnée permet de regrouper des données dans des contenants que l'on appelle des champs (***fields*** en anglais). Chaque champ peut être de n'importe quel type.

Syntaxe

La fonction **struct** permet de définir un élément de structure en précisant ses champs et ses valeurs, la syntaxe de la commandes struct:

nomDeLaStructure = **struct**('nomDuChamp1',valeur1,'nomDuChamp2',valeur2, ...)

Application

```
>> q=struct('NomCommun','Rorcqual commun','Longueur',50,'Poids',10e3)
q =
  struct with fields:
    NomCommun: 'Rorcqual commun'
     Longueur: 50
        Poids: 10000
```

table : le type parfait pour des données hétérogènes tabulées

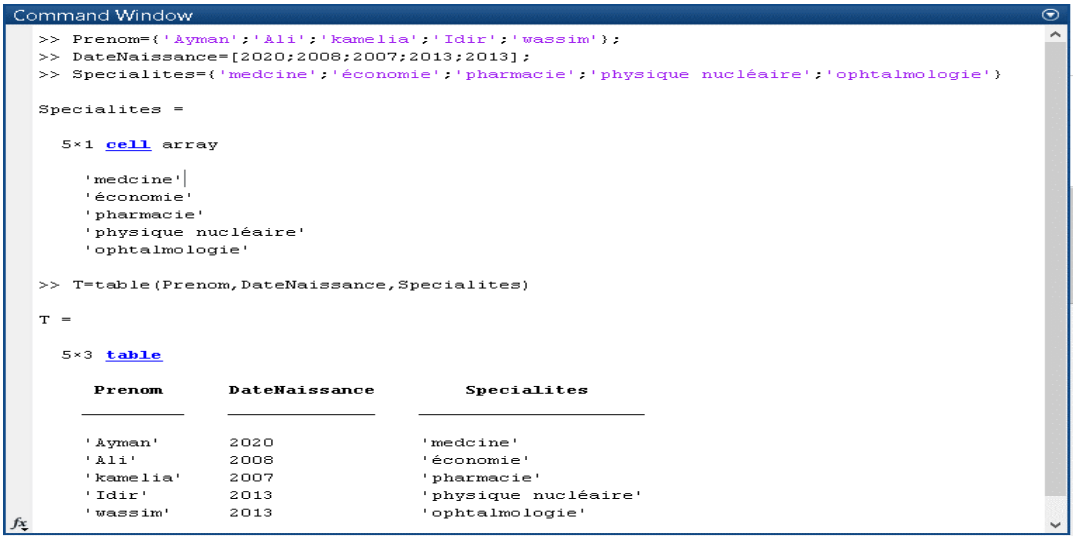
La variable *table* permet de regrouper plusieurs variables de types différents dans un container unique, et de les visualiser aisément sous forme de données tabulées.

Syntaxe

La fonction **table** permet de définir une nouvelle variable de type *table*, en précisant le contenu de chacune de ses colonnes. La fonction table s'ecrit alors :

nomDeLaTable = **table**(valeursColonne1, valeursColonne2, ...)

Application

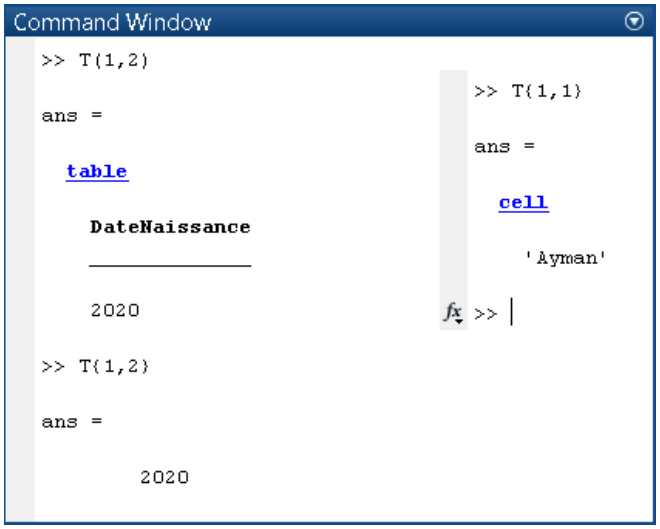


Comme il est montré dans l'exemple précédent, il est obligatoire que le nombre d'éléments dans chacune des variables soit identique.
L'accès à un élément de la table peut se faire de différentes manières :

- **T(1,1)** renverra la valeur Ayman sous le type table
- **T{1,1}** renverra la valeur Ayman sous le type du contenu, ici avec le type double.

De la même manière,

- **T(1,2)** renverra la valeur '2020' sous le type table
- **T{1,2}** renverra la valeur '2020' sous le type du contenu, ici avec le type cell.



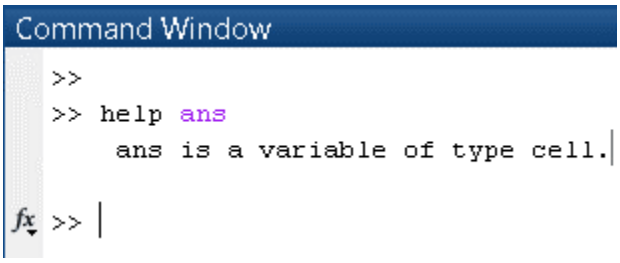
III.4. VARIABLES ET FONCTIONS PRÉDÉFINIES

III.4.1. VARIABLES PRÉDÉFINIES

En matlab, existe un certain nombre de variables pré-existantes, comme ‘’**ans**’’ qui contient le dernier résultat de calcul si celui-ci n'a pas été affecté à une autre variable, ainsi que i et j qui représentent le complexe $\sqrt{-1}$. Il existe aussi la variable **pi**, qui représente **π**, et quelques autres constantes de ce genre.
La variable NaN (*Not a Number*) représente un résultat d'un calcul qui n'est pas numérique. Aussi, la variable **Inf** représente quant à elle la valeur infinie.

Remarque

Toutes les variables prédéfinie, ont une entrée dans l'aide de MATLAB, car toutes les ces variables sont en fait des fonctions, et possèdent une documentation spécifique.



III.4.2. FONCTIONS PRÉDÉFINIES

Les fonctions prédéfinit dans la bibliothèque de matlab sont des fonctions standards qui peuvent s’applique dans le calcul vectoriels ou matriciel. Le langage matlab offre, par le script **fuction**, la possibilité de programmer des fonctions selon le besoin de l’utilisateur. Parmi les fonctions les plus utilisées, on peut noter les suivantes :

Fonctions	Signification
abs(x)	La valeur absolue de x
acos(x)	L’arc cosinus de l’angle x (en radian)
asin(x)	L’arc sinus de l’angle x (en radian)
atan(x)	L’arc tangent de l’angle x (en radian)
cos(x)	Le cosinus de l’angle x (en radian)

exp(x)	Exponentiel de x
log(x)	Logarithme népérien de x (ln(x)=$\log_e(x)$)
log10(x)	Logarithme décimale de x (log à base de 10)
sin(x)	Le sinus de l'angle x (en radian)
sqrt(x)	Calcul la racine carrée de la valeur de x
tan(x)	Le tangent de l'angle x (en radian)
X^	La puissance de x

Remarques

- On retiendra que pour appliquer une fonction à une valeur ou une variable, **il faut mettre cette dernière entre parenthèses**.
- En MATLAB, la fonction logarithme népérien ou naturel, que l'on note souvent **ln** en mathématiques, s'appelle **log** . Tandis que le logarithme commun, en base 10, que l'on note souvent **log** s'appelle **log10**.

CHAPITRE IV : VECTEURS ET MATRICES

Dans cette partie, nous allons aborder un point important : les matrices ! C'est important, parce qu'historiquement MATLAB a été développé pour manipuler des matrices, et la plupart des fonctions tiennent compte du fait que potentiellement elles puissent être appliquées à des matrices.
Rappelons qu'une matrice est un objet mathématique représentant des données numériques sous forme d'un tableau. Une matrice est donc un tableau de données ayant un sens mathématique spécifique. À l'inverse, un tableau correspondant à tout ensemble de données tabulées.
Le concept de tableau est important, car il est à la base du tracé de courbes : typiquement pour une courbe de n points, on définira un tableau de n abscisses et un tableau de n ordonnées. Mais on peut aussi définir des tableaux rectangulaires à deux indices pour définir des matrices au sens mathématique du terme, puis effectuer des opérations sur ces matrices.
Pour MATLAB, en termes de traitement, il n'y a aucune différence entre ces deux notions. Mais il est important de garder en tête que ces deux notions représentent deux objets conceptuellement différents.

IV.1. DÉFINITION D'UN TABLEAU

Syntaxe

Les tableaux sont définis en utilisant la syntaxe suivante :

```
>> U = [1 2 3]
```

Où

- les crochets ([]) sont utilisés pour définir le début et la fin d'un tableau ou d'une matrice.
- Les éléments sont entrés ligne par ligne ;
- les éléments appartenant à la même ligne sont séparés par des espaces (ou par des virgules) ;
- les déférentes lignes qui doivent posséder le même nombre d'éléments, sont séparées par des points-virgules.

Exemple

```
>> u1=[1 2 3]
u1 =
     1     2     3

>> u2=[1,2,3]
u2 =
     1     2     3

>> u3=[1;2;3]
u3 =
     1
     2
     3

>> u4=[1 4 5;2 6 7 ;3 8 9]
u4 =
     1     4     5
     2     6     7
     3     8     9
```

Remarque

Dans la suite, on appellera :

- **Vecteur** : un tableau de format (n, 1) i.e. ne comportant qu'une seule colonne ;
- **Liste ou encore vecteur-ligne** : un tableau de format (1, n) i.e. ne comportant qu'une seule ligne ;
- **Tableau :** un tableau au sens commun du terme, c'est à dire une structure organisée en lignes et colonnes

IV.2. LES MATRICES ET VECTEURS

Une matrice est un tableau rectangulaire avec m lignes et n colonnes et contenant des éléments numérique de même type.

Exemple

Nous allons créer 3 variables x, y et z représentant respectivement un scalaire, un vecteur et une matrice.

```
>> x = 50;
>> y = [50 60];
>> z = [5 0; 1 50; 0 60];
>> size (x)

ans =
    1     1

>> size (y)

ans =
    1     2

>> size (z)

ans =
    3     2
```

On constate que les 3 variables possèdent toutes 2 dimensions :

- Le scalaire x a pour dimension 1×1 ;
- Le vecteur y a pour dimension 1×2 ;
- La matrice z a pour dimension 3×2.

Remarque

Sous Matlab, les scalaires, les vecteurs et les matrices sont tous construits sur une base de matrice 2D.

IV.3.1. DÉCLARATION DE VECTEURS

On définit un vecteur ligne en donnant la liste de ses éléments entre crochets ([ ]). Les éléments sont séparés au choix par des espaces ou par des virgules. On définit un vecteur colonne en donnant la liste de ses éléments séparés au choix par des points virgules (;) ou par des retours chariots (touche Entrée/Enter).

Syntaxe

Un vecteur est définit on donnant la liste de ses éléments

```
>> u= [1 2 3]
u=
    1  2  3
```

```
>> u1 = [1, 2, 3]
u1 =
    1  2  3
```

```
>> u2= [1 ; 2 ; 3]
U2=
    1
    2
    3
```

Ou on utilisant l'opérateur « deux points », qui permet de créer un vecteur régulièrement espacé.

Xi : inc : Xf

- *Xi*, élément de départ du vecteur
- *inc*, valeur de l'incrément
- *Xf*, critère de fin du vecteur

Les éléments de ce vecteur sont les résultats de la suite d'incrémentation partant de Xi et ne dépassant pas Xf.

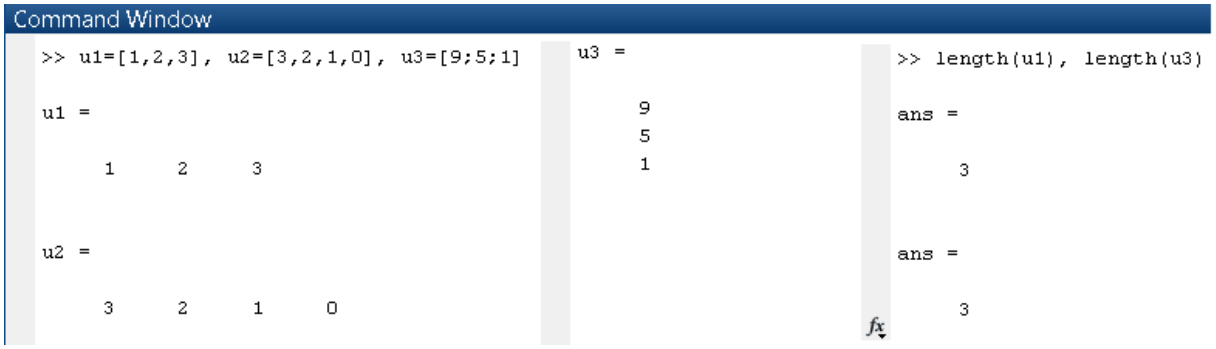
- L'incrément peut être négatif
- L'incrément peut être omis s'il vaut 1
- Le vecteur peut être vide selon les valeurs de Xi et Xf

```
>> u=2 :0.5 :5
u=
   2.000  2.5000  3.0000  3.5000  4.0000 4.5000  5.0000
```

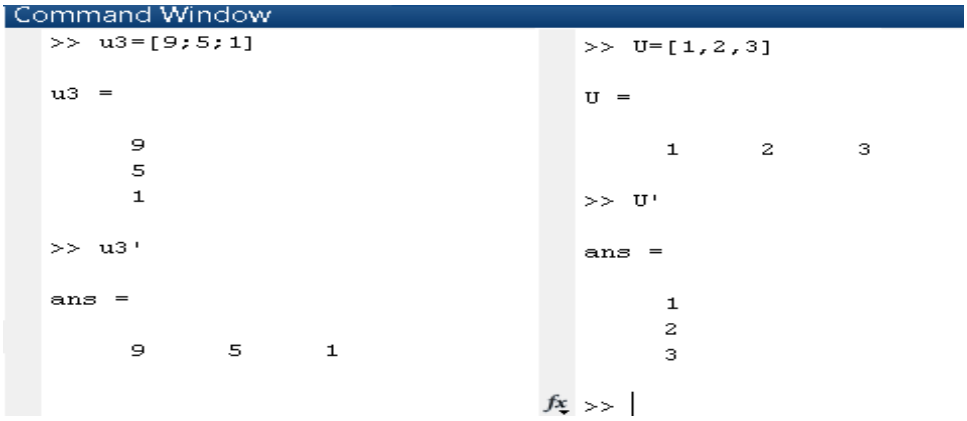
On peut obtenir la longueur d'un vecteur donné grâce à la commande **length.**

Un vecteur peut également être défini « par blocs » selon la même syntaxe.

Exemple



On peut transformer un vecteur ligne x en une vecteur colonne et réciproquement en tapant x' (' est le symbole de transposition).



Certaines fonctions de Matlab s'appliquent à l'ensemble d'un vecteur. Le tableau suivant décrit le résultat de quelques-unes de ces fonctions lorsqu'elles sont appliquées à un vecteur x :

x=rand(1,100)	vecteur aléatoire de 100 réalisations
max(x)	maximum
min(x)	minimum
sort(x)	tri par ordre croissant
[y, i] = sort(x)	retourne en plus les indices des éléments de x
find(x)	retourne les indices non nuls de x
[y, i] = find(x)	retourne des lignes (dans le vecteur i) et des colonnes (dans le vecteur j) des éléments non nuls du x
sum(x)	somme des éléments de 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 de x

IV.3.2. DÉCLARATION DE MATRICES

Les matrices sont l'objet fondamental de MATLAB et sont particulièrement importantes dans notre cours. Les matrices peuvent être créées dans MATLAB de plusieurs façons.

Syntaxe

Le moyen le plus simple de saisir une matrice est d'entrer ses éléments entre deux crochets « **[ ]** », en séparant les lignes la touche «Entrée».

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

On peut utiliser le symbole, (virgule) qui sert de séparateur de colonne et ; (point-virgule) de séparateur de ligne. Ainsi pour définir la matrice précédente on aurait pu taper :

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

Nous pouvons également générer des matrices par des fonctions prédéfinies, telles que des matrices aléatoires

```
>>  rand(3)

ans=
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

Les matrices rectangulaires peuvent être obtenues par spécification du nombre de lignes et de colonnes, comme dans

```
>> rand(2,3)
ans =
    0.9649    0.9706    0.4854
    0.1576    0.9572    0.8003
```

CRÉATION DE MATRICES ÉLÉMENTAIRES

Ci-dessous, des commandes qui nous permettent de générer facilement des matrices utiles dans la vie de tous les jours.

- **ones** : matrice de 1 de la taille voulue;
- **zéros** : matrice de 0 de la taille voulue ;
- **eye** : matrice identité de la taille voulue ;
- **linspace** : vecteur de N valeurs linéairement réparties entre 2 valeurs ;
- **logspace** : vecteur de N valeurs logarithmiquement réparties entre 2 valeurs ;

- **rand** : matrice de la taille voulue dont les éléments sont le résultat d'une distribution aléatoire uniforme ;
- **randn** : matrice de la taille voulue dont les éléments sont le résultat d'une distribution aléatoire normale ;
- **magic** : carré magique de taille quelconque

Application

Créer une matrice remplie de 0

```
>> M = zeros(2,3)
M =
    0    0    0
    0    0    0

>> M = zeros(3)
M =
    0    0    0
    0    0    0
    0    0    0

>> M(4,2) = 0
M =
    0    0    0
    0    0    0
    0    0    0
    0    0    0
```

Créer une matrice remplie de 1

```
>> M = ones(2,4)
M =
    1    1    1    1
    1    1    1    1

>> M = ones(4)
M =
    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1
```

Créer une matrice diagonale

```
>> M = eye(5)
M =
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1

>> M = eye(5,2)
M =
    1    0
    0    1
    0    0
    0    0
    0    0
```

```
>> M = eye(5);

>> M = M(:, end:-1:1)
M =
    0    0    0    0    1
    0    0    0    1    0
    0    0    1    0    0
    0    1    0    0    0
    1    0    0    0    0
```

IV.3.3. OPÉRATIONS SUR LES MATRICES

ACCÉDER À UN ÉLÉMENT D'UNE MATRICE

Pour accéder à un élément particulier d'un tableau ou d'une matrice, il suffit d'entrer le nom du tableau ou de la matrice suivi entre parenthèses du ou des indices dont on veut lire ou écrire la valeur. Considérant la matrice A

Syntaxe

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

>> A(2,4)
ans =
    7

>> A(11)
ans =
    1
```

ACCÉDER À PLUSIEURS ÉLÉMENTS D'UNE MATRICE : EXTRACTION D'UNE SOUS-MATRICE

Comme nous venons de le voir, l'opérateur : (*colon* en anglais) permet de créer facilement des listes en indiquant les valeurs de ses bornes, et optionnellement le pas. Cet opérateur est très souvent utilisé pour définir une liste d'indices de ligne ou de colonne, pour définir un sous ensemble de cellules sur lesquelles on veut réaliser un calcul ou un traitement.
Ce que l'on appelle l'extraction d'une sous-matrice issue d'une matrice de plus grande taille est en fait un cas spécifique d'utilisation de cet opérateur, et de la syntaxe d'accès à un élément que nous avons vu précédemment.
Ainsi, si l'on veut extraire une sous-matrice, il faut spécifier les indices du bloc concerné, en indiquant les bornes de début et de fin des indices.

Syntaxe

La syntaxe générale est donc la suivante (pour un tableau à deux dimensions) :

tableau(début:fin, début:fin)

Exemple

Considérant la matrice A

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

A(1 :3, 2 :3)
ans  =
      7    3
      9    4
      6    8
```

Si l'on utilise l'opérateur (*:*) seul, cela veut dire que l'on considère tous les indices possibles. C'est bien pratique pour extraire des lignes ou des colonnes spécifiques d'une matrice.
Par exemple, si l'on veut extraire les deux dernières colonnes, mais conserver toutes les lignes, on écrira

```
>> A(:,2:4)

ans =

    7    3    5
    9    4    7
    6    8    4
    6    7    9
```

Ou pour obtenir une ligne de la matrice A :

```
>> A(3,:)

ans =

    2    6    8    4
```

Pour extraire un vecteur d'une matrice, il est souvent utile de connaître l'indice de la dernière ligne ou de la dernière colonne. Or, bien souvent, on ne le connaît pas explicitement.

Dans ce contexte, le mot clé **end** peut être utilisé pour faire référence au dernier indice (sous-entendu le plus grand indice possible).

- Pour un vecteur ligne :

```
>> A(1,1:end)

ans =

    5    7    3    5
```

- Pour un vecteur colonne

```
>> A(1:end,1)
ans =
    5
    2
    2
    3
```

OPÉRATIONS ARITHMÉTIQUES SUR LES TABLEAUX

Addition et soustraction

Les deux opérateurs sont les mêmes que pour les scalaires, à savoir + et -. À partir du moment où les deux tableaux concernés ont la même taille, le tableau résultant est obtenu en ajoutant ou soustrayant les termes de chaque tableau.

Multiplication, division et puissance terme à terme

Ces opérateurs sont notés.*, / et ^ (**attention à ne pas oublier le point**).
Ils sont prévus pour effectuer des opérations terme à terme sur deux tableaux de même taille.
Ces opérations sont fondamentales lorsque l'on veut tracer des courbes, et nous le reverrons spécifiquement dans ce cas d'usage, ou plus généralement lorsque l'on souhaite effectuer ces opérations arithmétiques sur un ensemble de données tabulées.

Distinction entre opérations terme à terme et opérations matricielles

Il est fondamental de bien distinguer l'opération de multiplication (et par voie de conséquence de division et de puissance) définie terme à terme de celle définie pour les matrices.
Dans bien des cas (et notamment si les matrices sont carrées et de même dimensions), les deux types d'opérations peuvent être réalisés, sans produire d'erreur de syntaxe pour le langage MATLAB, mais produiront des résultats numériques totalement différents.
Pour bien montrer la différence entre les opérateurs **.*** et *****, prenons un exemple suivant :

```
>> M=[2 4; 6 8]
M =
    2    4
    6    8

>> N=[1 0; 0 1]
N =
    1    0
    0    1
```

- Voici la multiplication au sens des matrices :

```
>> x=N*M
x =
    2    4
    6    8
```

- Et maintenant regardons la multiplication terme à terme :

```
>> y=N.*M
y =
    2    0
    0    8
```

En plus de ces trois opérations, pour tableaux ou matrices, il existe ‘autres opération qu’on résumé dans le tableau suivant :

opérations terme à terme	
A.*B	times(A,B)
A./B	rdivide(A,B)
A.^B	power(A,B)
.*, .^	multiplication, puissance terme à terme
./	division terme à terme
opérations matricielles	
A’	Transposé de A
rank(A)	Rang de A
expm(A)	Exponentielle de A
inv(A)	Inverse de A
det(A)	Déterminant de A
trace(A)	Trace de A
poly(A)	Polynome caractéristique de A
eig(A)	Valeurs propres de A
[u,D]=eig(A)	Vecteurs propres et valeurs propres de A
+, -	Addition, soustraction
***, ^**	Multiplication, puissance (amtritielles)
A*B	mtimes(A,B)
A\b	Solution de Ax=b
b/A	Solution de xA=b
A/B	mrdivide(A,B)
A^B	mpower(A,B)

Opérations relationnels

Les opérateurs relationnels permettent de réaliser des comparaisons logiques entre des valeurs numériques. Dans ce type d’opération la valeurs logique est désignée par ‘’VRAIE’’, noté true représenté aussi par 1, ou ‘’FAUX’’, noté false représenté par 0.

Les opérateurs relationnels permettent la comparaison de deux valeurs entre-elles. Le tableau suivant synthétise les syntaxes des différents opérateurs relationnels disponibles :

Syntaxe des opérateurs relationnels

Opération relationnelle	Syntaxe MATLAB	
A égal à B	A == B	eq(A,B)
A différent de B	A ~= B	ne(A,B)
A supérieur à B	A > B	gt(A,B)
A supérieur ou égal à B	A >= B	ge(A,B)
A inférieur à B	A < B	lt(A,B)
A inférieur ou égal à B	A <= B	le(A,B)

```
>> 2 == 2

ans =
  logical
   1

>> 2.4 > 3.2
ans =
  logical
   0
```

Considérant la matrice de l'exemple précédent

```
>> M=[2 4; 6 8]

>> M>4

ans =

  2×2 logical array

   0   0
   1   1
```

Les termes de M supérieurs à 4 donnent 1 (true), les autres 0 (faux).
Prenons maintenant les deux matrice M et N de même dimension.

```
>> M=[2 4; 6 8]

>> N=[1 0; 0 1]

>> H=M>N

H =

  2×2 logical array

   1   1
    1   1
```

Remarque

Les valeurs logiques peuvent être utilisées comme valeurs numériques (elles sont alors automatiquement sur-typées en réel double). Ceci est particulièrement utile pour définir des fonctions par morceaux.

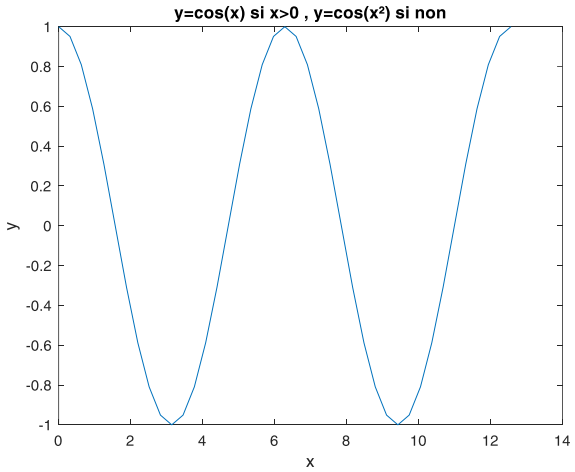
Application

Considérons la fonction f(x), définie par morceaux (ces valeurs dépends d'un test logique), suivante :

$$f(x)=\begin{cases}\cos(x) & \text{si } x>0\\ \cos(x^2) & \text{si non}\end{cases}$$

```
>> x=0:pi/10:4*pi;
>> y=cos(x).*(x>0)+cos(x.^2).*not(x>0);
```

Si on trace le résultat



La syntaxe précédente permet d'écrire facilement une fonction définie par morceaux, mais cela ne permet pas d'extraire ou de modifier des valeurs selon un test logique. Pour cela, on peut utiliser la fonction **find**.
On fait appel à la fonction **find** pour identifier simplement les éléments non nuls d'un tableau, et par extension, d'identifier les valeurs vérifiant un critère logique donné.

Syntaxe

```
k=find(X)
```

Cette écriture renvoie dans la variable k la liste des indices des éléments non nuls du tableau X.

```
k=find(opération logique sur X)
```

Cette écriture renvoie dans la variable k la liste des indices vérifiant l'opération logique du tableau X.

Application

Soit un tableau de valeurs suivant :

```
>> X=[-2.6 -1.5 0 1 1.9 2.6 3.8]
X =
  -2.6000   -1.5000         0    1.0000    1.9000    2.6000    3.8000
```

La fonction find permet d'identifier les éléments comportant des valeurs non nulles :

```
>> k=find(X);
>> k
k =
     1     2     4     5     6     7
```

Pour trouver tous les éléments correspondant à un critère logique :

```
>> k=find(X>0)
k =
     4     5     6     7
```

Dans cet **exemple**, on a cherché tous les éléments positifs du vecteur X. nous pouvons ensuite extraire facilement le sous-tableau contenant les éléments positifs de X en écrivant :

```
>> u=X(k)
u =
    1.0000    1.9000    2.6000    3.8000
```

TP N°01

Exercice IV.1

Exécutez l'introduction de Matlab, chargez l'aide …, faites un tour sur les commandes essentiels selon le tableau ci-dessous...

Solution

intro lance une introduction à Matlab
help produit une liste de toutes les commandes par thèmes
demo démonstration donnant une représentation des fonctionnalités de bases de Matlab
info information sur la boite à outils disponibles
helpwin ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations
help donne la liste de toutes les commandes par thermes
help *nom* décrit la fonction nom.m
lookfor *nom* recherche une instruction à partir du mot clé nom

Exercice IV.2

1- Tapez la commande **a=1:7 ;**
2- Tapez les commandes **a, who et whos**.
- Utilisez « ↑ et → ou ← » pour modifier a : **a=1:2;.**
- Tapez la commande **b=a+2;**.
- Ré-exécutez les commandes **who et whos** en utilisant « ↑ ».

Tapez **clear et b**.

Exercice IV.3

Entrez les différents vecteurs et matrices et donnez la longueur et la taille de chacun (Utilisez help pour trouver les fonctions qui donnent longueur et taille).

Solution

```
>> x1=1:5          %(Un vecteur contenant les entiers de 1 à 5)
>> x2=1:1:5
>> x3=10:-1:1
>> x4=1:0.3:pi
>> x1(2)=x3(3)
>> x4(3:5)=[1,2,3]
>> x4(3:5)=[]
>> x5=linspace(1,5,5)
>> help linspace
>> who
>> whos
>> clear l1 l2 l3 l5
>> who
>> clc              % (efface le contenu de la fenêtre de commande)
>> clear
```

Exercice IV.4

Construire :
1. une suite de **-8** à **-5** avec un pas de **0.25**.
2. une suite décroissante d'entiers de 15 à 3.
3. une suite de longueur 100 de –pi à pi.
4. recherche la longueur de x en utilisant une commande (a recherche dans le help)

Solution

```
1-  X1=-8 :0.25 :-5
2-  X2=15 :-1 :3
3-  X3=linspace(-pi,pi,100)
-
4-  N=numel(x)
```

Exercice IV.5

Extraction de composantes.
Entrez la matrice

```
>> A=[0:2 ;5 :7 ;8 :-1 :6]
```

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 IV.6

1. Créez des matrices particulières en utilisant les commande zeros, eye,…(voir page 44)
2. Exemple de création d'une matrice par blocs

```
H=[A,zeros(3,2) ; zeros(2,3), eye(2)]
H =
    0    1    2    0    0
    5    6    7    0    0
    8    7    6    0    0
    0    0    0    1    0
    0    0    0    0    1
```

Exercice IV.7

Soit les matrices A

$$A=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix},\quad C=\begin{bmatrix}2 & 9 & 4 & 9 & 3\\ 5 & 1 & 3 & 6 & 4\\ 5 & 4 & 3 & 8 & 7\\ 4 & 7 & 3 & 6 & 5\end{bmatrix}$$

Essayez des fonctions sur la matrice A. Par exemple :

- quels sont ses valeurs et vecteurs propres ?
- déclarez une matrice C de même taille que A. Essayez :
  - A+C,
  - A*C,
  - A.*C.

Ensuite, définissez la matrice B comme étant la matrice A à laquelle on a ajouté le vecteur
colonne [1; 2; 3].

- Calculez le déterminant de la matrice B (manuellement et avec matlab)
- Calculez l'inverse de B (manuellement et avec matlab).

Exercice IV.8

Résolution d'un système sous-dimensionné :

$$4x_1 - \frac{2}{3}x_2 + 6x_3 = 1$$

$$5x_1 - 2x_2 + x_3 = 2$$

5- Écrivez le système sous la forme matricielle Ax = b, et calculez le rang de la matrice A.
6- Définissez la matrice B comme étant la matrice A à laquelle on a ajouté le vecteur colonne b.
7- Calculez le rang de la matrice B. Conclusion ?
8- Résolvez l'équation Ax = b.

NB :

- A\b est équivalent à inv(A)*b si A est inversible.
- Éléments de réponse (voir page 49)

Remarque

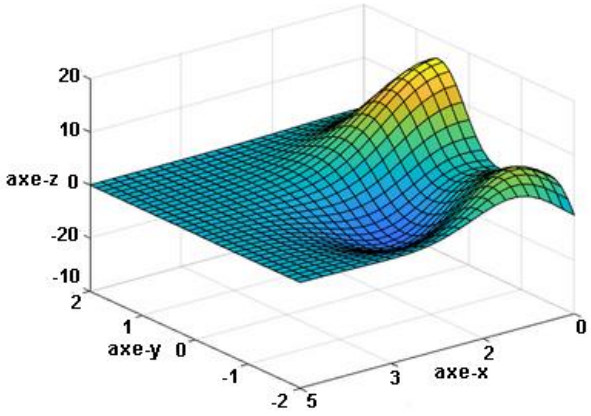
La résolution des exercices 5, 6, 7 et 8 ce fera en séances TP.

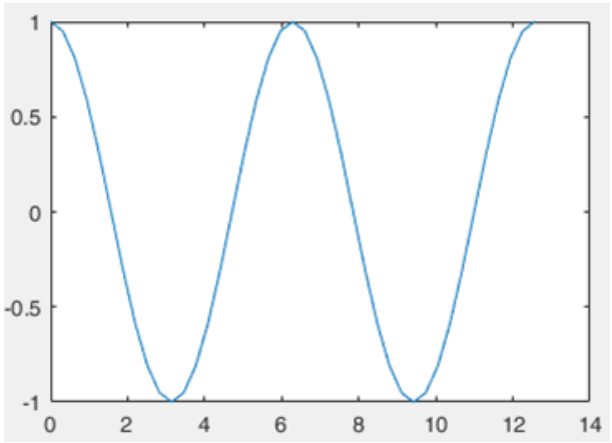
CHAPITRE V. REPRÉSENTATIONS GRAPHIQUES SOUS MATLAB

Nous avons vu dans les chapitres précédents que des tableaux pouvaient contenir des données numériques, issue de différentes manipulations. En sciences de l'ingénieur, il souvent demandé de représenté ces données, pour différentes raisons :

- Analyser la cohérence des données ;
- Vérifier ou déterminer la tendance de celle-ci ;
- Observer l'évolution des données selon un axe ;
- Visualiser l'existence de points particuliers

Pour une meilleure lecture des résultats stocker sus forme de données, il faut être capable de les représenter sous forme graphique. Le logiciel, MATLAB peut produire des graphiques impressionnants de couleurs en 2D et 3D. Il fournit aussi les outils et moyens de personnaliser et de modifier pratiquement tous leurs aspects.





V.1. GESTION DES FENÊTRES GRAPHIQUES

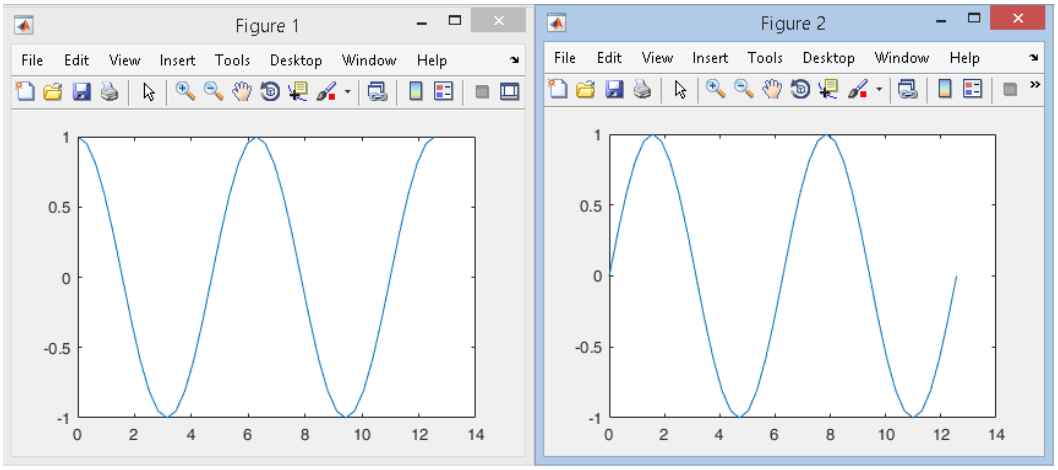
Une instruction graphique ouvre une fenêtre dans laquelle est affiché le résultat de cette commande. Par défaut, une nouvelle instruction graphique sera affichée dans la même fenêtre et écrasera la figure précédente, pour résoudre ce problème est affiche en plusieurs fenêtre, on fait appelle à la commande **figure** qui nous permet d'ouvrir une nouvelle fenêtre graphique pour chaque joue de données. Chaque fenêtre se voit affecter un numéro. Ce numéro est visible dans le bandeau de la fenêtre sous forme d'un titre.

Le résultat d'une instruction graphique est par défaut affiché dans la dernière fenêtre graphique ouverte qui est la fenêtre graphique active. Pour manipuler une figure, plusieurs options sont offertes par matlab :

- On rend active une fenêtre graphique précédemment ouverte en exécutant la commande **figure(n)**, où **n** désigne le numéro de la figure.
- La commande **close** permet de fermer la fenêtre graphique active.
- On ferme une fenêtre graphique précédemment ouverte en exécutant la commande **close(n)**, où **n** désigne le numéro de la figure.
- Il est également possible de fermer toutes les fenêtres graphiques en tapant **close all**.

Exemple

```
x=0:pi/10:4*pi;
>> y=cos(x);
>>z=sin(x) ;
>> figure
>> plot(x,y)
>> figure
>> plot(x,z)
```



Sur la figure deux fenêtres graphiques sont représenté à savoir,

- la fenêtre graphique 1 où est affiché le graphe de la fonction **cosinus** entre 0 et **pi**.
- la fenêtre graphique 2 où est affiché le graphe de la fonction sinus entre 0 et **pi**.

V.2. COURBES ET REPRÉSENTATIONS 2D

Si l'on dispose de tableaux de dimension égale à 2, c'est-à-dire un vecteur ligne, un vecteur colonne ou un tableau de taille **n×m**, on peut représenter ces données sous forme d'une courbe **2D**. Dans le concept de **2D**, une courbe est définie par un ensemble de points, chacun défini par une abscisse et une ordonnée. Ces points peuvent être reliés ou non par des segments de droite, pour donner l'illusion d'une courbe continue.

V.2.1. TRACER LE GRAPHE D'UNE FONCTION

La commande plot

La commande **fplot** permet de tracer le graphe d'une fonction sur un intervalle donné. La syntaxe est :

Syntaxe

plot(x,y)
ou x = [x1, x2, x3,…xn] est le vecteur d'abscisses ; et y = [y1, y2, y3, …yn] est le vecteur d'ordonnées

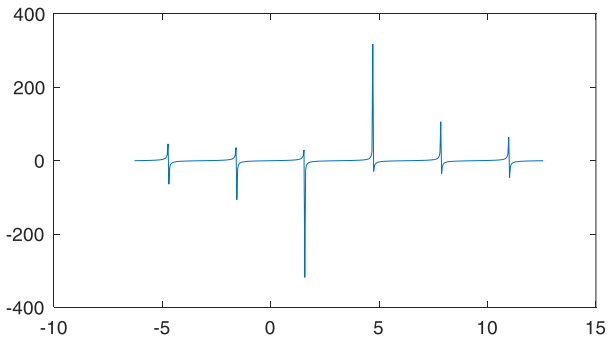
Remarque

- Les vecteurs peuvent être indifféremment des lignes ou des colonnes,
- Les vecteurs doivent tous deux avoir la même longueur.

Exemple

Pour créer un graphique simple à l'aide de la commande plot, nous allons utiliser la fonction linspace pour définir l'axe des abscisses **x** comme un vecteur de 500 valeurs linéairement espacées entre -2π et 4π et l'axe des ordonnées **y** comme la fonction tangente évaluée aux valeurs de **x.**

```
>> x=linspace(-2*pi, 4*pi,500) ;
>> y=tan(x) ;
plot(x,y)
```



V.2.2. TRACER UN GRAPHIQUE DANS UNE NOUVELLE FENÊTRE

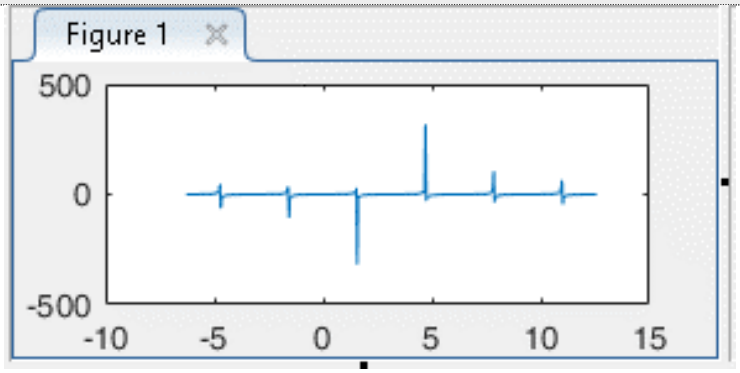
Pour tracer un graphique dans une nouvelle fenêtre, au lieu de le tracer dans la figure en cours, nous faisons appel à la commande **figure,** qui permet de d'ouvrir une nouvelle fenêtre graphique. Si on déclare pas une nouvelle fenêtre graphique, par défaut, MATLAB efface les graphiques et traces sur la figure encours.

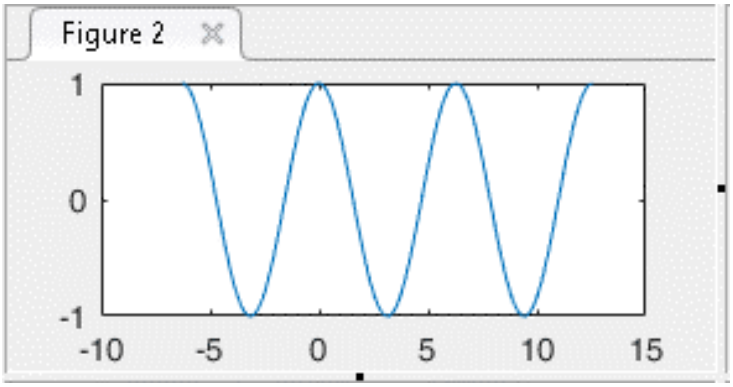
Exemple

Nous allons créer un graphique en escalier de **z** en fonction de **v, ou :**

```
>> x=linspace(-2*pi, 4*pi,500) ;
y=tan(x) ;
figure
plot(x,y)

z=linspace(-2*pi, 4*pi,500) ;
v=cos(z) ;
figure
plot(z,v)
```



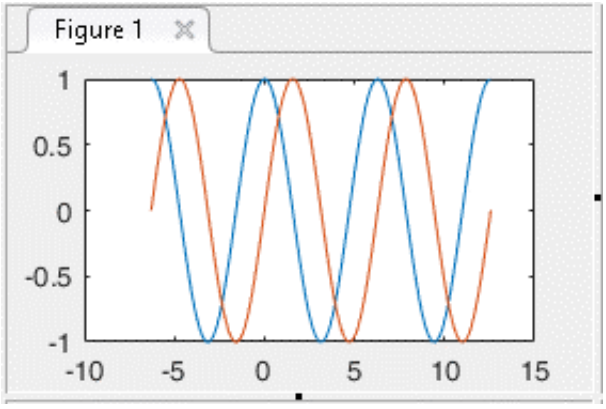


V.2.3. SUPERPOSER PLUSIEURS COURBES SUR UNE FIGURE

Pour tracer plusieurs courbes sur le même graphique, il suffit de spécifier autant de couples de vecteurs (abscisses, ordonnées) qu'il y a de courbes à tracer.

Syntaxe

```
>> x=linspace(-2*pi, 4*pi,500) ;
y=cos(x);
z=sin(x);
figure
plot(x,y,x,z)
```

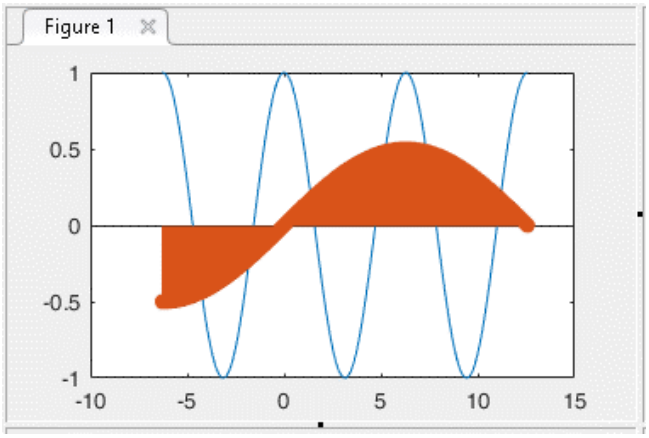


Cette syntaxe fonctionne même si les abscisses des différentes courbes ne sont pas les mêmes.

Pour ajouter une courbe à un graphique existant, MATLAB contient une autre commande, noté **hold on.** Cette commande à pour objet de conserver la courbe existante et ajouter un nouveau tracé au graphe.

Syntaxe

```
 >> x=linspace(-2*pi, 4*pi,500) ;
y=cos(x);
z=0.5*sin(x/4);
figure
plot(x,y), hold on
>> plot(x,z)
>> hold off
```



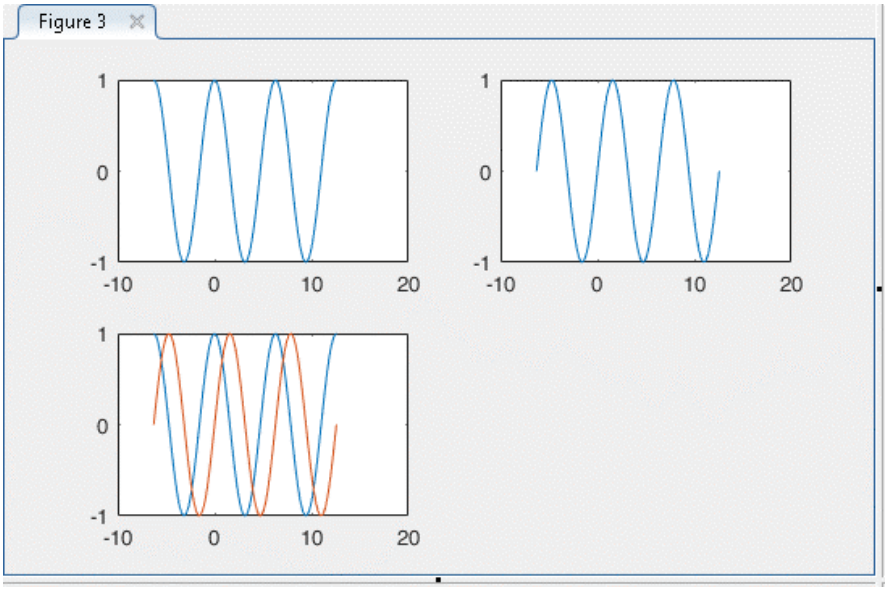
V.2.4. CRÉER UNE FIGURE AVEC PLUSIEURS GRAPHIQUES

Syntaxe

Pour créer une figure contenant plusieurs graphiques, nous faisons appel à la fonction **subplot**. La commande, **subplot (m, n, p)**, divise la figure en une grille de **m** lignes et **n** colonnes dans l'emplacement de la grille **p**-ème.

Exemple

```
>> x=linspace(-2*pi, 4*pi,100) ;
y=cos(x);
z=sin(x);
figure
subplot(2,2,1)
plot(x,y)
subplot(2,2,2)
plot(x,z)
subplot(2,2,3)
plot(x,y,x,z)
```



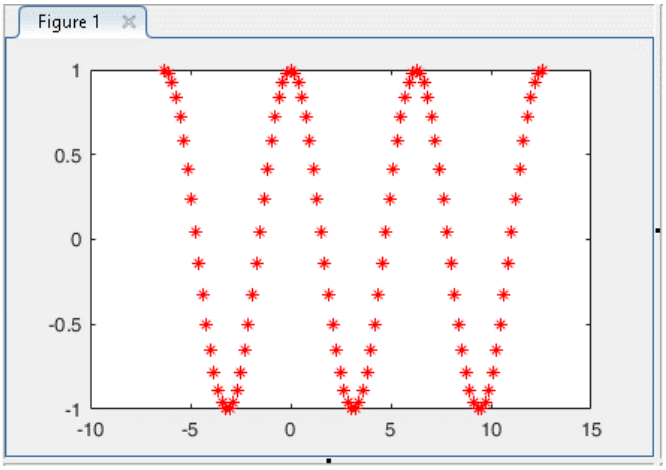
V.2.5. TRACER UNIQUEMENT LES POINTS DE DONNÉES

Dans la représentation des données expérimentales l'utilisateur est appelé dans certains cas de figure à tracer uniquement les points de données en omettant l'option de style de trait.

Syntaxe

Définissez les données x et y. Tracer les données et afficher un marqueur d'étoile à chaque point de données.

```
>> x=linspace(-2*pi, 4*pi,500) ;
>> y=cos(x);
>> figure
>> plot(x,y,':r'), hold on
```

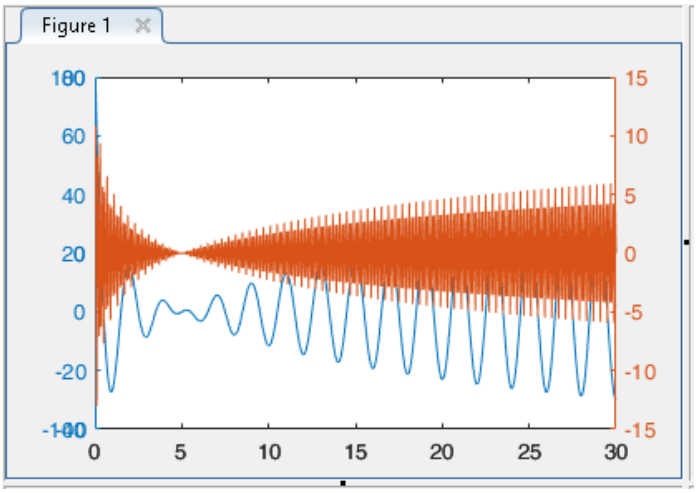


V.2.6. PLOT AVEC DEUX AXES D'ORDONNÉES

Lorsque l'on veut représenter deux courbes dans le même domaine de définition, mais ayant des amplitudes très différentes, la bibliothèque de Matlab nous offre une fonction noté **plotyy** qui gère de manière automatique et indépendante les échelles des ordonnées des deux tracés.

Syntaxe

```
>>x= 0 :0.05 :30 ;
>> y1=cos(x.*pi).*((50/3)*(log(5./x)));
>>y2=sin(3.*x.*(5*pi)).*((10/3)*(log(5./(x))));
>> plotyy(x,y1,x,y2)
```



V.2.7. ATTRIBUTS DE COURBES

Dans les exemples précédents MATLAB attribue des couleurs, style de marqueurs et de ligne par défaut aux courbes.

Il est possible de modifier la couleur, le style du trait et celui des marqueurs de points, en spécifiant après chaque couple (abscisse, ordonnée) une chaîne de caractères (entre guillemets simples et droits ', single-quotes) pouvant contenir les codes suivants (issus de l'aide obtenue par help plot) :

V. 2.8. COULEUR

Couleurs et désignation sur MATLAB	
b	Bleu (blue)
g	Vert (green)
r	Rouge (red)
c	Cyan
m	Magenta
Y	Jaune (yellow)
k	Noir (black)
w	Blan (white)

V.2.9. STYLE DE MARQUEURS

Style de marqueurs et désignation sur MATLAB	
.	Point
o	Cercle
X	Marque x
+	Plus
*****	Etoile
s	Carré (square)
d	Losange (diamond)
v	Triangle pointe en bas
^	Triangle pointe en haut
<	Triangle pointe à gauche
>	Triangle pointe à droite
p	pentagone
h	hexagone

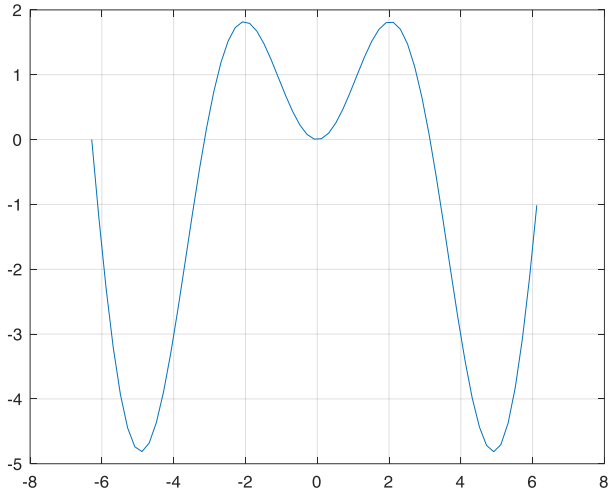
V.2.10. STYLE DE LIGNE

Désignation sur MATLAB	
-	Trait plein
--	Pointillés longs sans ligne
-.	Pointillés mixte
:	Points courts

Remarques

1- Tous ces codes peuvent être combinés entre eux.
2- Les valeurs par défaut sont c = b, s = . et t = - ce qui correspond à un trait plein bleu reliant les points entre eux. Il n'est pas obligatoire de spécifier chacun des trois caractères. On peut se contenter d'en spécifier un ou deux. Les autres seront les valeurs par défaut. La commande **grid** permet d'obtenir un quadrillage de la figure.

```
>> x=-2*pi:0.2:2*pi;
>> y=x.*sin(x);
>> plot(x,y)
>> grid on
```

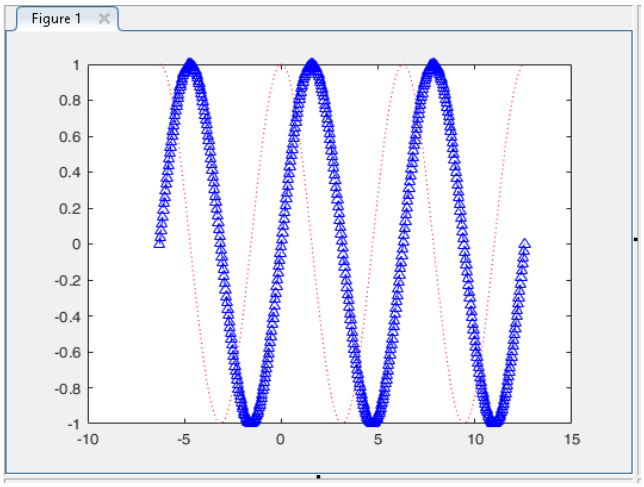


V.2.11. SPÉCIFIER LE STYLE DE TRAIT, LA COULEUR ET LES MARQUEURS, POUR PLUSIEURS COURBES

Cet exemple montre comment tracer deux ondes sinusoïdales avec différents styles de traits en ajoutant une spécification de courbe à chaque plot.
Tracer la première onde avec une courbe en pointillée long en rouge en utilisant « **:r** » puis tracer la deuxième onde sinusoïdale avec une courbe pointillée court bleu et marqueurs en Triangle pointe en haut (voir tableau) en utilisant **« -.b^»**.

Syntaxe

```
>> x=linspace(-2*pi, 4*pi,500) ;
>> y=cos(x);
>> z=sin(x);
>> figure
>> plot(x,y,':r'), hold on
>> plot(x,z,'b-.^')
```



Remarque

- Lorsque l'on utilise seulement un style de marqueurs, MATLAB ne trace plus de segments de droite entre les points successifs, mais seulement les points eux-mêmes. Ceci peut être pratique par exemple pour présenter des résultats expérimentaux.
- On peut choisir tout type de styles et de couleurs pour les courbes, selon l'inspiration de chacun. Cependant, lorsque l'on représente des données scientifiques, il est d'usage de différencier :
  - les données expérimentales, par nature un ensemble discontinu de valeurs, que l'on représente sans style de ligne, avec des marqueurs de points ;

- les données issues d'un modèle ou d'une équation continue, que l'on représente par un style de ligne au choix, avec ou sans marqueurs.

V.2.12. AUTRES FORMES DE REPRÉSENTATION

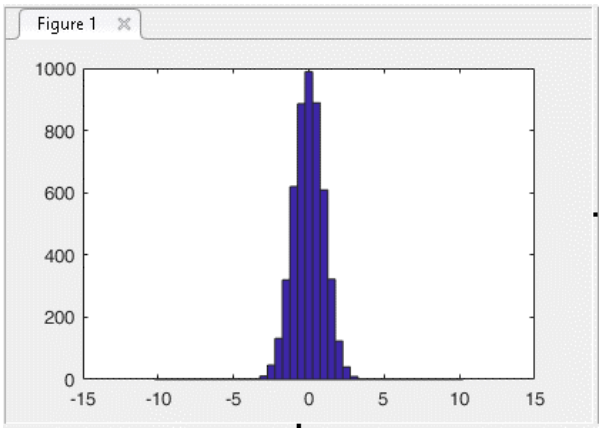
Dans le domaine des sciences de l'ingénieur, certaines fonctions, notamment celles de distribution, demande une autre forme de représentation sous forme d'histogramme ou de barres plutôt que de points ou de traits.

REPRÉSENTATION EN HISTOGRAMME

Dans les études probabiliste la représentation d'une loi de distribution (normal, log normal, Gumbel,…) est de grande importance. Sous MATLAB, la fonction **Hist** permet le tracé des histogrammes de distributions.

Syntaxe

```
>> x=randn(5000,1) ;
>> y=-10 :0.5 :10 ;
>> hist(x,y)
```



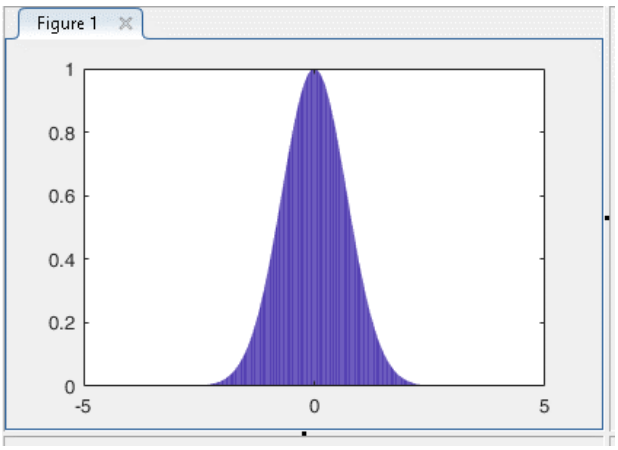
Dans cette écriture, La fonction **randn** permet de définir un ensemble de valeurs aléatoires selon une distribution normale

REPRÉSENTATION SOUS FORME DE BARRES

Dans cette représentation on fait appel à la commande **bar.**

Syntaxe

```
>> x=-5:0.01:5;
>> y=exp(-x.^2);
>> bar(x,y)
```

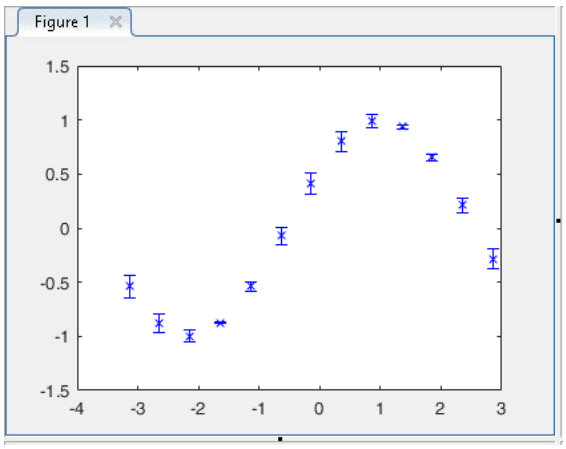


Tracé de données avec barres d'erreur

Il peut être utile de tracer des données (expérimentales) en matérialisant les barres d'erreur, qu'elles soient connues ou estimées, constantes ou non. La fonction **errorbar** permet ce type de tracé.

Syntaxe

```
>> x=-pi:0.5:pi;
y=cos(1-x);
e=0.1*cos(x);
errorbar(x,y,e,'bx')
```



V.2.13. GESTION DE LA FENÊTRE GRAPHIQUE

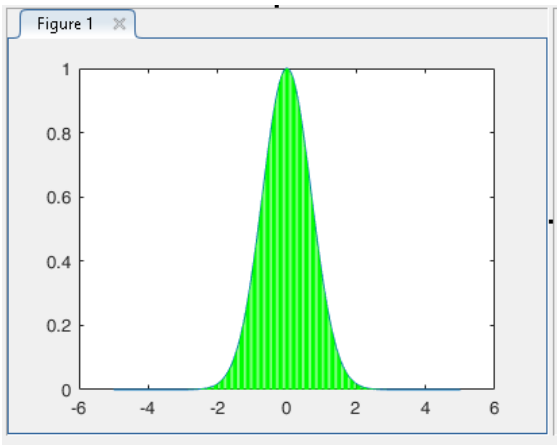
Nous avons vus que lorsqu'on fait appelle à la fonction plot, une fenêtre graphique est créée. Si on utilise à nouveau la fonction plot, MATLAB efface cette fenêtre pour effectuer le nouveau tracé. Pour résoudre ce problème MATLAB dispose de commandes permettant de gérer ou d'interagir avec cette fenêtre graphique.

SUPERPOSITION DE GRAPHIQUES

Nous avons vu qu'il est possible de tracer plusieurs courbes lors d'un appel de la fonction **plot**. Mais dans certains cas, il peut être utile de tracer des graphiques de natures différentes (avec **plot**, **bar**, **hist**, etc), dans une même fenêtre graphique.
Par défaut, les fonctions de tracés effacent systématiquement le graphique précédent. Il est parfois utile de le conserver et de venir le surcharger avec le nouveau graphique. Pour cela on utilise la commande **hold**. Pour démarrer le mode surcharge, tapez **hold on**, pour revenir en mode normal, **hold off**.
Il est conseillé de ne pas abuser de cette commande.

Syntaxe

```
>> x=-5:0.01:5;
>> y=exp(-x.^2);
>> plot(x,y), hold on
>> bar(x,y,'g'), hold off
```



SAISIE D'UN POINT À LA SOURIS

La commande **ginput**(N) permet de cliquer N points dans la fenêtre graphique, et d'en récupérer les coordonnées. La fonction renvoie un tableau, dont la première colonne correspond aux abscisses, la seconde aux ordonnées des points sélectionnés.
Utilisée sans le paramètre N, la fonction tourne en boucle jusqu'à ce que la touche «Entrée» soit tapée.

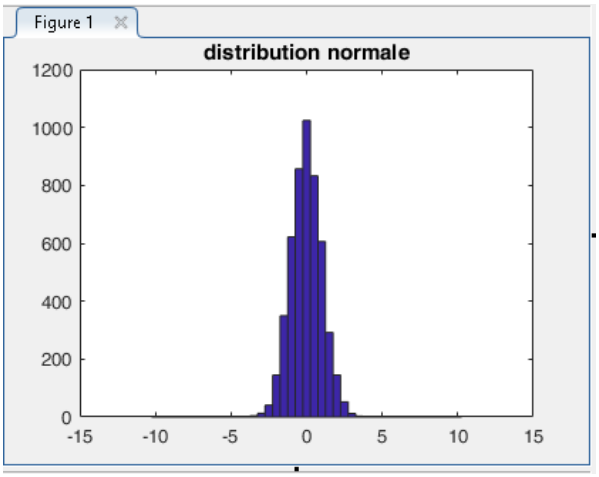
AJOUTER UN TITRE, DES ÉTIQUETTES ET UNE LÉGENDE

En programmation si on cherche à produire des graphiques pour présenter des données numériques, il est nécessaire d'ajouter des éléments pour faciliter leur lecture. Ces éléments sont le titre, les labels des axes ou les légendes, et plus généralement tout ce qui ajoute de l'information facilitant la compréhension du graphique.

Ajouter un titre

Le premier élément pour la lecture d'un graphe est le titre, cet élément est ajouter par l'instruction ***title, à laquelle il faut fournir une chaine de caractères.*** Le titre apparaît simplement en haut de la fenêtre graphique :

Syntaxe



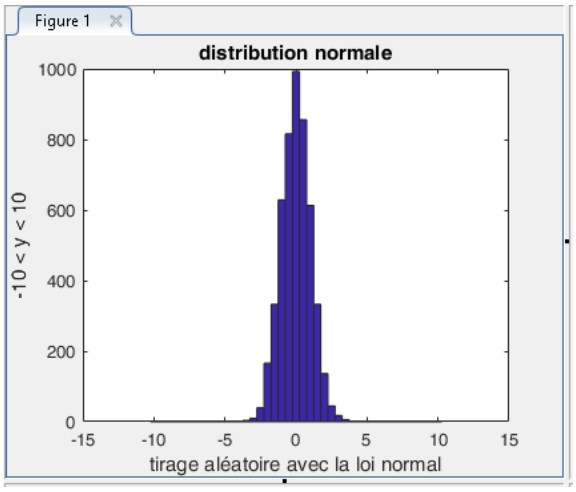

```
>> x=randn(5000,1) ;
>>y=-10 :0.5 :10 ;
>>hist(x,y)
>> title('distribution normale')
```

Ajouter des étiquettes aux axes

Il s'agit d'afficher du texte sous les abscisses (x) et à côté de l'axe des ordonnées (y) utilisant les fonctions **xlabel** et **ylabel**, afin de spécifier de quoi il s'agit :

Syntaxe

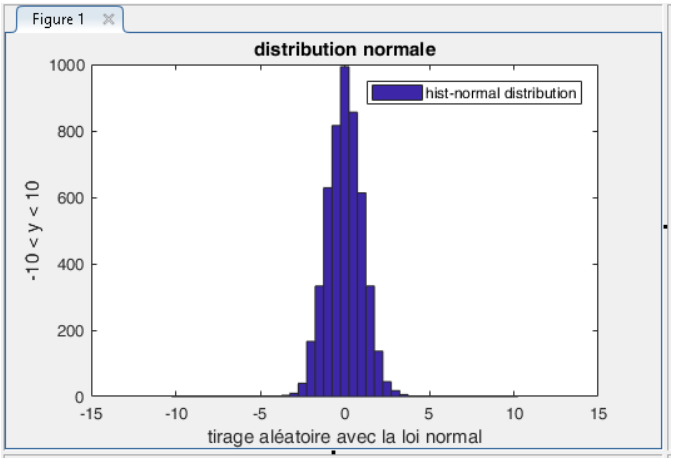
```
>> x=randn(5000,1) ;
>> y=-10 :0.5 :10 ;
>> hist(x,y)
>> title('distribution normale')
>> xlabel('tirage aléatoire avec la loi normal')
>> ylabel('-10 < y < 10')
```



Ajouter une légende

Dès que l'on trace plusieurs courbes sur le même graphique, il devient indispensable d'ajouter une légende, pour spécifier à quoi correspond chacune des courbes.
On MATLAB, L'instruction **legend** permet d'ajouter cet élément. Il faut lui communiquer autant de chaînes de caractères que de courbes tracées. Un cadre est alors ajouté sur le graphique dont l'emplacement sur le graphique est spécifie en utilisant la propriété position de la légende.

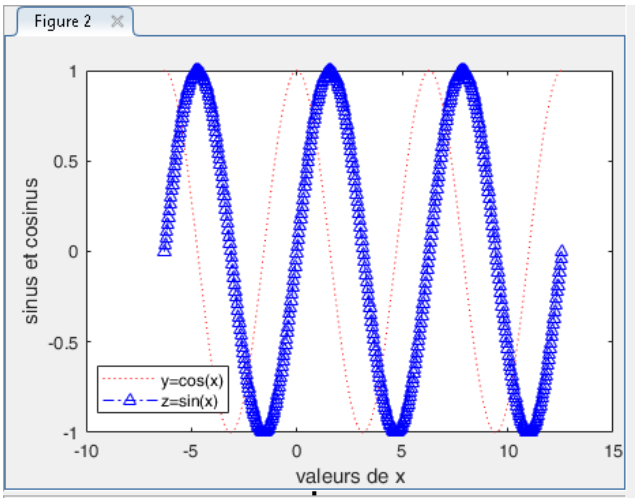
Syntaxe




```
>> x=randn(5000,1) ;
>> y=-10 :0.5 :10 ;
>> hist(x,y)
>> title('distribution normale')
>> xlabel('tirage aléatoire avec la loi normal')
>> ylabel('-10 < y < 10')
>> legend('hist-normal
distribution','location','northeast')
```

(northwest, southeast, southwest)

```
>> x=linspace(-2*pi, 4*pi,500) ;
y=cos(x);
z=sin(x);
figure
plot(x,y,':r'), hold on
plot(x,z,'b-.^')
>> xlabel('valeurs de x')
>> ylabel('sinus et cosinus')
>> legend('y=cos(x)','z=sin(x)','location','southwest')
```



Adaptation des graphiques

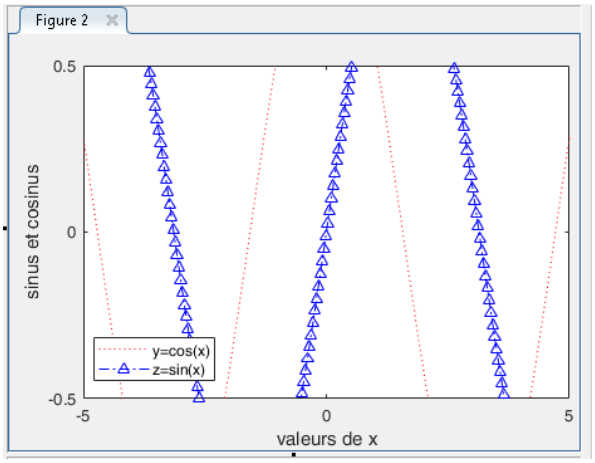
les axes

Dans les exemples précédents lorsque l'on tracé de courbes, on ne spécifie pas la zone graphique qui nous intéresse, alors MATLAB détermine par défaut cette zone, à partir des valeurs minimales et maximales des abscisses et des ordonnées.
Pour spécifier la zone graphique souhaité afficher la commande ***axis*** est utilisée.

Syntaxe

La commande axis prend pour argument un vecteur donnant les valeurs extrêmes des abscisses et ordonnées [xmin, xmax, ymin, ymax].

```
>> x=linspace(-2*pi, 4*pi,500) ;
y=cos(x);
z=sin(x);
figure
plot(x,y,':r'), hold on
plot(x,z,'b-.^')
>> xlabel('valeurs de x')
>> ylabel('sinus et cosinus')
>> legend('y=cos(x)','z=sin(x)','location','southwest')
>>axis([-5,5,-0.5,0.5))
```

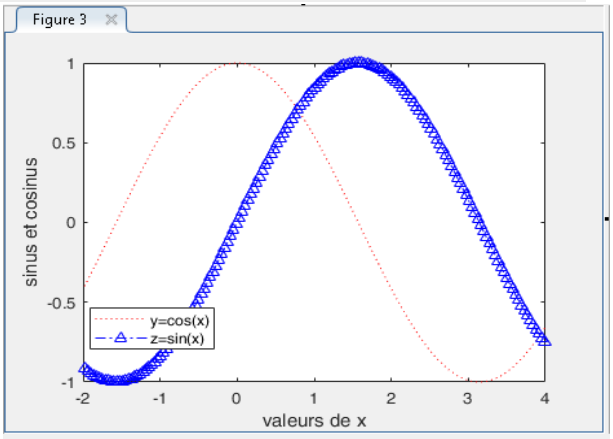


Choix de l'intervalle d'abscisses ou d'ordonnées : axes limites

Les commandes **xlim** et **ylim** permettent de spécifier dans quel intervalle, en abscisses ou en ordonnées, il faut tracer le graphique.

Syntaxe

```
>> x=linspace(-2*pi, 4*pi,500) ;
y=cos(x);
z=sin(x);
figure
plot(x,y,':r'), hold on
plot(x,z,'b-.^')
xlabel('valeurs de x')
ylabel('sinus et cosinus')
legend('y=cos(x)','z=sin(x)','location','southwest')
>> xlim([-2,4])
```



Ajout de texte sur un graphique

Il est possible d'ajouter du texte sur une fenêtre graphique en choisissant sa position. La fonction gtext(strTexte) placera le texte contenu dans la variable strTexte à l'emplacement indiqué par un clic de souris.

SAUVEGARDER UNE FIGURE

Syntaxe

La commande print permet de sauvegarder la figure d'une fenêtre graphique dans un fichier sous divers formats d'images. La syntaxe de la commande print est:

print -f<num> -d<format> <nomfic>

où

- <num> désigne le numéro de la fenêtre graphique. Si ce paramètre n'est pas spécifié, c'est la fenêtre active qui est prise en compte.
- <nomfic> est le nom du fichier dans lequel est sauvegardée la figure. Si aucune extension de nom n'est donnée, une extension par défaut est ajoutée au nom du fichier en fonction du format choisi (.ps pour du PostScript, .jpg pour du jpeg, par exemple).
- <format> est le format de sauvegarde de la figure. Ces formats sont nombreux. On pourra obtenir la liste complète en tapant help plot. Les principaux sont:

ps	PostScript noir et blanc
psc	PostScript couleur
eps	PostScript Encapsulé noir et blanc
epsc	PostScript Encapsulé couleur
jpeg	Format d'image JPEG
tiff	Format d'image TIFF

V.3. REPRÉSENTATION 3D

V.3.1.COURBES DANS UN ESPACE 3D

Comme dans le cas de tracé en 2D, est définie par une liste de doublets (x,y), on peut définir une courbe en 3D par une liste de triplets (x,y,z). Puisque l'instruction **plot** attendait deux arguments, le vecteurs des abscisses, et celui des ordonnées, l'instruction **plot3** en attend trois : le vecteur des abscisses x, le vecteur des ordonnées y et le vecteur des cotes z.

Syntaxe

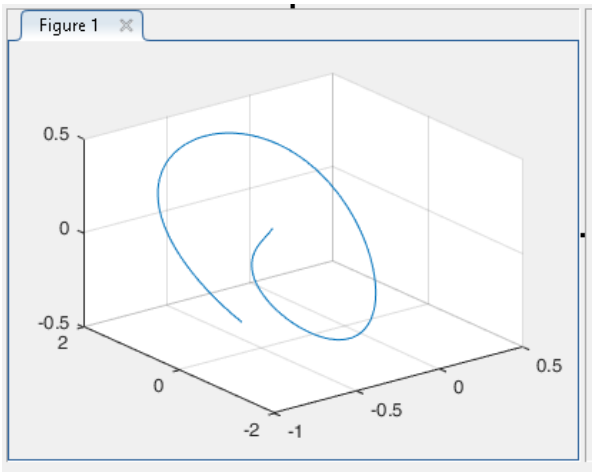
plot3(x, y, z) ;

Exemple

Soit x, y et z toirs fonction definis dans un intervalle [-2,2] avec un pas de 0.01.

Ecrire un programme qui permet de tracer la courbe 3D de la liste des triplets (x,y,z).

```
>> f= -2 : 0.01 : 2 ;
>> x=cos(2+f).*sin(f./2) ;
>> y=sin(f.^2)+cos(f-2);
>> z=cos(f).*sin(f);
>> plot3(x,y,z)
>> grid ;
```



V.3.1. TRACÉ DE SURFACES EN 3D

LA COMMANDE MESHGRID POUR LA GÉNÉRATION D'UN MAILLAGE

En mathématique, une surface est définit par un ensemble de triplets (x,y,z). dans une représentation en 2D, les point (x,y) forment, généralement, dans le plan un maillage régulier dont la seul contrainte à satisfaire est que le nombre de points soit le produit de deux entiers **m×n**.

Dans une représentation en 3D, si l'on a **m×n** points, cela signifie qu l'on a **m×n** valeurs de x, de y et de z. cette écriture signifie que les abscisses, les ordonnées et les cotes des points de la surface peuvent être stockées dans des tableaux de taille **m×n**.

Syntaxe

Soit Mx la matrice d'abscisses, **My** la matrice d'ordonnées et **Mz** la matrice des côtes définit comme suit :

$$M_x = \begin{bmatrix} x_{11}\ x_{12}\ x_{13}\dots x_{1n} \\ x_{21}\ x_{22}\ x_{23}\dots x_{2n} \\ \dots\dots\dots\dots\dots \\ x_{m1}\ x_{m2}\ x_{m3}\dots x_{mn} \end{bmatrix} \quad M_y = \begin{bmatrix} y_{11}\ y_{12}\ y_{13}\dots y_{1n} \\ y_{21}\ y_{22}\ y_{23}\dots y_{2n} \\ \dots\dots\dots\dots\dots \\ y_{m1}\ y_{m2}\ y_{m3}\dots y_{mn} \end{bmatrix} \quad M_z = \begin{bmatrix} z_{11}\ z_{12}\ z_{13}\dots z_{1n} \\ z_{21}\ z_{22}\ z_{23}\dots z_{2n} \\ \dots\dots\dots\dots\dots \\ z_{m1}\ z_{m2}\ z_{m3}\dots z_{mn} \end{bmatrix}$$

Considérons une surface définies par :

$$z = y^2 - x^2$$

Pour tracé une surface représentative sur un intervalle [-1,1]x[-3,3], il faut définir une suite de valeurs $\mathbf{x_1,\dots,x_m}$ pour x et une suite de valeurs $\mathbf{y_1,\dots,y_n}$ pour **y**.

Exemple

```
1 | >>  x = -1:0.1:1;
2|  >>  y = -5:0.1:5;
```

Ensuite, en combinant toutes ces valeurs de **x** et **y**, on obtient **m×n** points dans le plan (**x,y**). Il faut maintenant construire deux tableaux, l'un contenant les **m×n** abscisses de ces points l'autre les **m×n** ordonnées, soit :

$$X = \begin{bmatrix} x_1\ x_2\ x_3\dots x_m \\ x_1\ x_2\ x_3\dots x_m \\ \dots\dots\dots\dots\dots \\ x_1\ x_2\ x_3\dots x_m \end{bmatrix} \quad Y = \begin{bmatrix} y_1\ y_1\ y_1\dots y_1 \\ y_2\ y_2\ y_2\dots y_2 \\ \dots\dots\dots\dots\dots \\ y_n\ y_n\ y_n\dots y_n \end{bmatrix}$$

Ce type de tableaux est construit facilement a l'aide de la commande meshgrid de matlab, dont la syntaxe est donné comme suit :

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

Une fois les vectrices abscisses et ordonné sont définis, il reste à calculer $z = y^2 - x^2$. À ce niveau 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 ^, comme montré dans le chapitre précédent.

```
1 | >>   Z = Y.^2-X.^2 ;
```

LES TRACÉS 3D

Plusieurs commandes dans matlab donnent la possibilité d'effectuer un tracé de surface en 3D, nous pouvons citer mesh, et surfc.

Syntaxe

mesh(X,Y,Z)

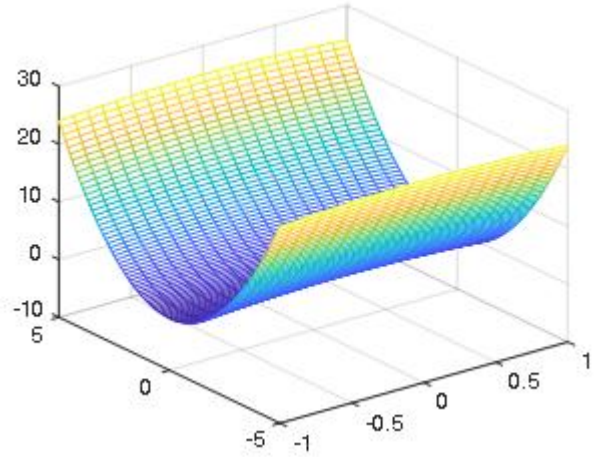
Exemple

Soit x, y deux vecteurs définit comme suit :
x= [-1,1]

y= [-5,5]
Les composantes de x et y forment une suite arithmétique de pas h=0.1.

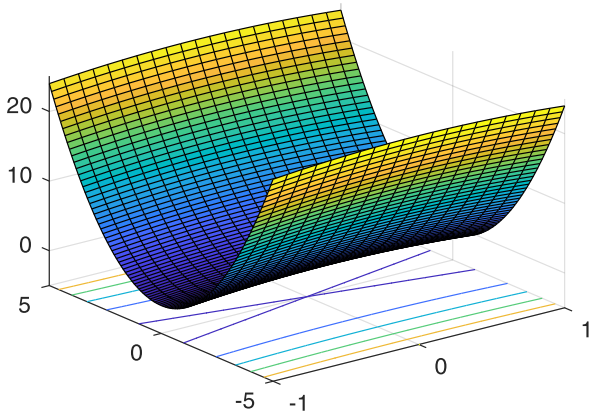
- En utilisant la commande mesh, écrire un programme qui permet de tracé la surface dans un espace 3D.

```
1| >> x=-1:0.1:1;
2| >> y=-5:0.1:5;
3| >> [X,Y]=meshgrid(x,y);
4| >> Z = Y.^2 - X.^2;
5| >> mesh (X,Y,Z);
```



- En utilisant la commande surfc, écrire un programme qui permet de tracé la surface dans un espace 3D

```
1 | >> x=-1:0.1:1;
2| >> y=-5:0.1:5;
3| >> [X,Y]=meshgrid(x,y);
4| >> Z = Y.^2 - X.^2;
5| >> surfc(X,Y,Z);
```



Tracé d'une surface avec surfc, et avec la palette de couleurs par défaut.

V.3.2. ADAPTATION DES TRACÉS 3D

PALETTE DE COULEURS

Une fois la surface est tracé, la commande colormap définit la palette de couleur qui permet de peindre les surfaces. On peut définir ses propres tableaux de couleurs ou bien utiliser des palettes prédéfinies (par exemple 'bone', 'winter', 'hot', *etc*).

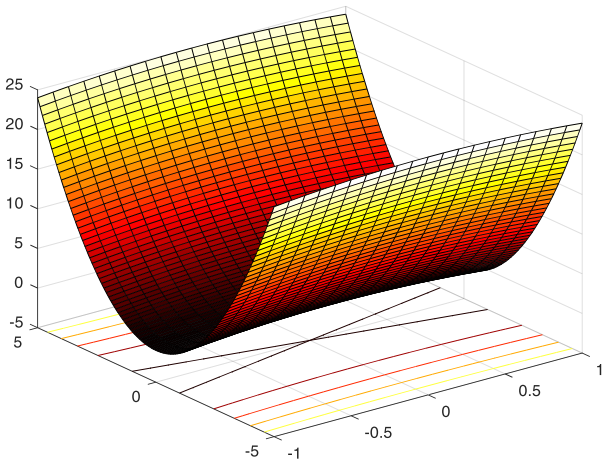
Syntaxe

```
colormap('hot');
```

Exemple

Considérant l'exemple précédent avec la commande surfc.

```
1 | >> x=-1:0.1:1;
2| >> y=-5:0.1:5;
3| >> [X,Y]=meshgrid(x,y);
4| >> Z = Y.^2 - X.^2;
5| >> surfc(X,Y,Z);
6| >>colormap('hot')
```



Tracé d'une surface avec **surfc**, et avec la palette de couleurs **'hot'**

INTERPOLATION

Sur les figures tracées nous constatant que les surfaces sont tracées avec des facettes qui correspondent aux mailles du maillage, ou chaque point correspondant à un nœud.
Sur matlab doc, existe des commandes, comme **shading**, qui permet de lisser ces surfaces.

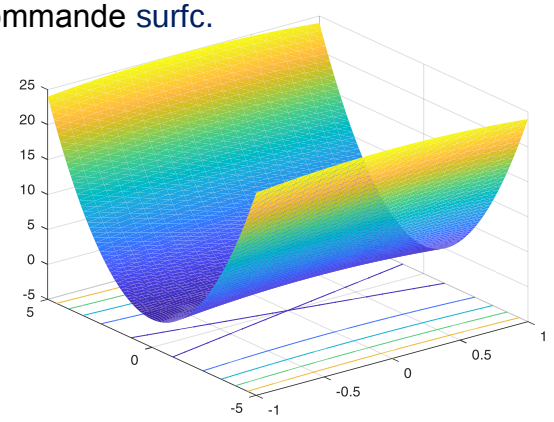
Syntaxe

shading('inetrp');

Exemple

Reprenant l'exemple précédent avec la commande surfc.

```
1 | >> x=-1:0.1:1;
2| >> y=-5:0.1:5;
3| >> [X,Y]=meshgrid(x,y);
4| >> Z = Y.^2 - X.^2;
5| >> surfc(X,Y,Z);
6| >> colormap('default')
7| >> shading('interp')
```



CHAPITRE VI : PROGRAMMATION MATLAB

VI.1. LES FICHIERS SCRIPT ET FUNCTIONS

VI.1.1. LES SCRIPTS

Un Script est un fichier de commandes MATLAB portant une extension '.m'. Ce type de fichier est un simple fichier texte, qui comporte des commandes MATLAB, identiques à celles que l'on peut employer directement dans la fenêtre de commandes de MATLAB. L'avantage évident est que ces commandes sont stockées dans un fichier, il est donc aisé de les rappeler pour les ré-exécuter, même dans une nouvelle session de MATLAB (c'est-à-dire après fermeture et ré-ouverture de MATLAB).

Exemple 1

```
%% script matrice

A=rand(2,3) ;
M=1984*A ;
vecteur=M(:,2)
```

Exemple 2

```
%% CourbeExp.m
%% ceci est un script qui trace la courbe y=a*x.*cos(2*x);
a=exp(5);
x=0:0.05:1;
y=a*x.*cos(2*x);
plot(x,y);
xlabel('x'); ylabel('y');title('y = ax')
disp('Tracé de la courbe y = ax*sin(2x)')
```

Remarque

Pour lancer le script, il suffit de taper dans la fenetre commande window le nom du fichier contenant le programme (sans le l'extension **.m).**

VI.1.2. LES FONCTION SUR MATLAB

En plus de fonctions offertes par la bibliothèque de MATALB, on peut définir de nouvelle fonction spécifique à un problème donnée et l'utiliser de la même manière que les autres fonctions prédéfinies.

Pour définir une nouvelle fonction sous Matlab, on écrit la définition de la fonction dans un fichier avec une extension **.m** (fonction M-File). Le nom du fichier doit être le nom de la première fonction définie (la seule visible). On l'exécute en tapant le nom de la fonction avec entre parenthèses la liste des arguments.

La première ligne du fichier de la fonction doit suivre la syntaxe suivante :

Syntaxe

function arguments de sortie = nom(arguments d'entrée)

```
Function[arg_S1, arg_S2,…,arg_Sn]=nom_fonction(arg_E1,arg_E2,…,arg_En]
 % arg=argument de sortie et d'entré.
% le corps de la fonction
…….
arg_S1 =      % la valeur de sortie arg_S1
arg_S2 =     % la valeur de sortie arg_S2
arg_Sn =     % la valeur de sortie arg_Sn
end             % pour ce type de fichier end est facultatif
```

Avec : arg_S1, arg_S2,…, arg_Sn sont les valeurs recherchées, et arg_E1, arg_E2,…, arg_En sont les valeurs d'entrée ou arguments d'entrée.

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

D'autre part :

```
Function[arg_S1, arg_S2]=ma_fonction(arg_E1,arg_E2]

Function [arg1, arg2]=fonction_aux (argE1, argE2)
Function [arg3, arg4]=fonction_aux2 (argE3, argE4)
```

On doit donc déclarer les valeurs calculées par la fonction dans les arguments de sortie, et les paramètres de la fonction dans les arguments d'entrée (qui ne sont pas modifiés).

Remarque

- Attention, contrairement au script, les variables d'une fonction sont des variables locales, qui ne sont pas accessibles en dehors de la fonction. Pour passer des informations à une fonction, il faut donc utiliser les paramètres d'entrée ou de sortie. En fait on peut utiliser des variables globales (mot clé global) sous Matlab, mais cela est à éviter dans le cadre d'une bonne programmation.
- Il est conseillé de donner au fichier dans lequel on enregistre la fonction le **nom de cette fonction.**
- La fonction principale doit toujours être placée au début.
- A part la fonction principale, toutes les autres fonctions (appelé auxiliaires) ne sont pas accessibles que dans le fichier dans lequel elles se situent, c.-à-d., dans l'exemple précédant, a partir d'un fichier (en mode script ou fonction) on pourra accéder à la fonction **ma_fonction,** mais pas aux **fonction_aux.**

Exemple

```
Function[S1,S2]=surface(R1,R2)

S1=pi*(R1)^2 ;
S2=pi*(R2)^2 ;
```

On obtient sur la fenêtre de la commande Windows :

```
>> [S1,S2]=surface(6,8)
S1 =
113.0973
S2 =
201.0619
```

VI.2. LES ENTRÉES / SORTIES

VI.1.2.1. AFFICHAGE

Lors de la programmation sur MATLAB, toute commande exécutée produit un résultat, qui, par défaut sera apparait dans la fenêtre d'exécution (fenêtre Windows) à la suite de la commande. Cependant, on peut empêcher l'affichage des résultats en insérant « **;** » à la fin de cheque commande.

Exemple

```
H=[1 2 3]

H =
     1  2  3                    %( résultat affiché)

H=[1 2 3] ;                     %( résultat affiché)
```

Remarque

En programmation, l'option **d'affichage**, est utilisé essentiellement, dans le but d'afficher les résultats de la programmation ainsi que pour détecter les erreurs liées à cette dernier. Dans le but de réduire le temps et la mémoire, **il est important de ne pas oublier de mettre un « ; »** à la fin de chaque ligne de **commande**.

ENTRÉE AU CLAVIER (LECTURE)

La lecture de données est une opération de grande importance lors de la programmation. Pour lire des données avec MATLAB, on peut procéder de deux manières, soit :

- En interrompant l'exécution du programme et en demandant à l'utilisateur d'introduire les données au clavier.
- En lisant un fichier contenant des données (exemple : spectre d'accélération sismique).

Pour interrompre l'exécution d'un programme est demander une valeur, la commande **input** est utilisée.

La commande « input » permet dans un programme de demander à l'utilisateur de fournir des données (exemple : dimension, valeur de contrainte, température,…).

Syntaxe

Var= input ('une question (phrase)')

Après exécution, la phrase introduite est afficher est MATLAB attend que le programmeur introduit une donnée au clavier. Pour aérer le programme, il est possible de provoquer des sauts de ligne en utilisant le symbole « \n » de la manière suivante :

Var= input ('\n une question (phrase) : \n')

SORTIE À L'ÉCRAN

Syntaxe

Pour afficher un résultat à l'écran, plusieurs commandes peuvent être utilisées, à savoir **: disp, fprintf, sprintf.**

Exemple

```
>> A= [1 2 3] ;
>> Disp(A) ;
      1 2 3
```

Pour l'affichage avec commande **fprintf**, il faut utiliser la syntaxe suivante :

fprintf(format, var1,var2,..)

SORTIE À L'ÉCRAN DIRIGÉ PAR FORMAT

Pour personnaliser l'affichage, il est possible d'utiliser la fonction **sprintf** qui permet de construire des chaînes de caractères complexes, en formatant l'affichage des variables.

```
>> str = sprintf('La valeur de sin(4/pi) est %8.6f',a) ;
>> disp(str)
La valeur de sin(4/pi) est 0.95605
```

La chaîne de caractère %8.6f est ce que l'on appelle un format. Un format commence toujours par le caractère %, et indique comment on doit représenter la variable qui suit. Le format est substitué par la variable formatée. Ici, le format signifie que l'on veut

représenter la valeur de a comme un réel, sur 8 chiffres maximum, avec une précision de 6 chiffres après la virgule.

Le format est une chaine de caractères décrivant le format d'écriture ou d'affichage des variables que l'on souhaite afficher. Les principaux types de format sont :

Types	Description	exemple
%d	Entier	%3d :entier de taille 3, expl :897
%f	Réel	%6.4f :réel de taille 6 avec 4 chiffres a prés la virgule expl : 54.6598
%e	Exponentiel	%4.6 e :
%g	Réel double précision	Mode automatique de détection entre e et f
%s	Chaine de caractère	

VI.3. STRUCTURES DE CONTRÔLES (BOUCLES / TESTS) SUR MATLAB

La programmation fait souvent appel à des tests conditionnels ou de boucles, qu'on appelle structures de contrôle. Dans la programmation, Les structures de contrôle définissent l'ordre d'exécution des instructions d'un programme. Les structures de contrôle disponible sous MATLAB sont très proches de celles offertes par d'autres langages de programmation.

VI.3.1. L'INSTRUCTION FOR

Afin d'exécuter une séquence d'instructions de manière répéter un nombre de fois fixé (connu), nous somme appelé à utiliser une boucle pour les valeurs d'un indice, incrémenté à chaque itération, qui varie entre deux bornes [a, b] bien définit. Ce processus est mis en œuvre par une boucle incrémentale appelé la boucle « **for** ».

Syntaxe

```
For variable (indice)= valeurs_inf :valeurs_sup
    Instruction
End
```

Ou :

- **Indice** : est une variable appelée l'indice de la boucle ;
- **Var_inf et var_sup** : sont deux constante réelles appelées paramètres de la boucle ;
- **Séquence d'instruction** : est le traitement à effectuer pour les valeurs d'indice variant entre var_inf et var_sup avec un pas d'incrémentation. Ce que en appel corps de la boucle.

Interprétation

1- La **variable** (**indice**) prend la **valeur_inf**
2- La **séquence d'instructions** est exécutée

3- **L'indice** voit sa valeur augmentée de 1

4- Les étapes **3** et **4** sont **répétées** en **boucle** jusqu'à ce que **l'indice dépasse** la **valeur_sup de la borne.**

Remarque

- Cette boucle est répéter tant de fois qu'il y a de valeurs (fonction des bornes et du pas)
- Pour chaque valeur il aura un nouveau passage de la boucle, comme suit :
  - La variable sera affectée avec une valeur consécutive (définit par **valeur** dans la boucle).
  - La variable est appelée compteur car c'est avec elle qu'on compte les passages.
  - Les instructions seront exécutées pour cette valeur de la variable
- En MATLAB valeurs, doit être juste un vecteur quelconque contenant les valeurs consécutives avec lesquelles la variable doit être affectée.

Exemple

```
For i=1 :8 ;
        J=i*3;
        disp(j) ;
end
```

```
For i=1 :2 :8 ;
        J=i*3 ;
        disp(j) ;
end
```

```
For i=[1,3,6,9]  ;
        J=i*3 ;
        disp(j)
end
```

- Dans cet exemple, nous allons essayer de calculer la somme pondérée des éléments de deux vecteurs de même taille. On commence par déclarer les deux vecteurs et une variable qui servira au calcul :

```
x = rand(1,1000);
y = rand(1,1000);
s=0;
```

- Ensuite, on utilise une boucle **for** afin de parcourir les deux vecteur **x** et **y**, et un indice pour parcourir **s** :

```
k=1
```

```
for i=x
        s=s+i*y(k) ;
        k=k+1 ;
end
```

- Ce programme est sensé afficher tous les éléments de « s **»** à travers la variable « **k»** dans la boucle.

Pour réduire le temps de calcul, on pourrait parcourir les deux vecteurs x et y en meme temps, en les fusionnant dans une matrice de deux lignes :

```
For i=[x ;y] ;
      s=s+i(1)*i(2) ;
      disp(s)
end
```

VI.3.2. L'INSTRUCTION While

La boucle **while** est une boucle qui exécute un bloc d'instructions tant qu'une condition logique est vraie (vaut **1** ou **true**).

Syntaxe

```
while condition logique
   instructions
end
```

Interprétation

La boucle while répète l'exécution de l'instruction pour un nombre indéterminer de fois selon la valeur d'une condition logique donnée.

Le fonctionnement dans ce cas est classique. Un while est une boucle dont on ne connaît pas *a priori* le nombre de termes, est la boucle tourne tan que l'expression de while est évaluée a vrai (true =1)... et cet ensemble de termes peut être infini.

Exemple

Faire un programme sous matlab qui nous permet de calculer la somme suivante :

S= (1/2 !) + (3/3 !)+….

```
>> s=1 ; i=1 ; f=1 ;
>> while s<=2.5
        i=i+1 ;
```

```
            f=f*i ;
            s= s+i/f ;
        end
            disp  (i) ;
            disp  (s) ;
```

VI.3.3. L'INSTRUCTION CONDITIONNELLE if

L'instruction indiquée par le mot clé **if** est principalement suivi d'une condition logique, d'où cette instruction indique le début d'une structure conditionnelle. Cette instruction permet d'orienter l'exécution du programme en fonction de la valeur logique d'une condition.

Syntaxe

```
if (condition logique)
   instructions 1
   instructions 2
     …
   instructions
end
```

Interprétation

Avec la boucle **if**, le bloc d'instructions sera exécuté si et seulement si la condition logique est évaluée à **vrai (true)**.

Si la condition est évaluée à **faux** (**false**), les instructions entre **if** et **end** ne seront pas exécutées, par contre, si un **else** existe, les instructions entre le **else** et **end** seront exécutées normalement.

Dans certains cas, ou la condition n'est vérifier, on peut définir d'autres blocs conditionnels avec un autre mot clé appelé – **elseif**- (comportant nécessairement une autre condition logique).

Syntaxe

```
if (condition logique-1)
   Instructions 1
elseif (condition logique-2)
   Instruction 2
  . . .
elseif (condition logique-N)
   Instruction N
```

```
else
   Ensemble d'instructions si toutes les expressions étaient fausses
end
```

Exemple 1

Ecrire un programme qui peut nous indique en fonction de l'heure la période dans a journée :

```
heure = input('Quelle heure est-il ?\n');


if (heure<=12) && (heure>=0)
   disp('   C''est le matin !');
elseif (heure>12) && (heure<=24)
   disp('   C''est l''après-midi !');
else
   disp('   Ce n''est pas possible...');
end
```

Exemple 2

Ecrire un programme qui peut nous nous résoudre le problème suivant:

- a=x si x = 0
- $a=x^3$ si x > 0
- a=25 si x < 0

```
x = input('entrez la valeur de x=');

if  x = 0
      a = x;
end

if  x > 0
      a = x^3 ;
end

if  x < 0
      a = 25;
end

disp('la valeur de a est ='), a
```

```
x = input('entrez la valeur de x=');

if  x = 0
     a = x;

elseif  x > 0
      a=x3 ;

 else  x < 0
       a = 25;
 end

 disp('la valeur de a est ='), a
```

Exemple 3

Créons un programme qui trouve les racines d'une équation de deuxième degré désigné par :

$$ax^2 + bx + c = 0$$

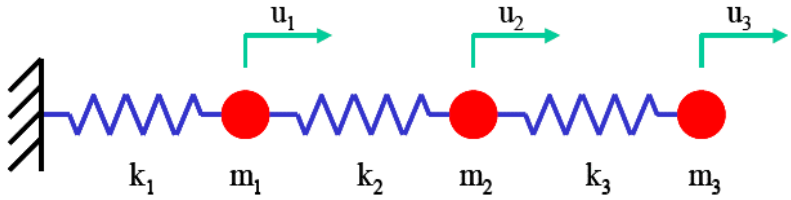
```
% Programme de résolution de l'équation de deuxième degré
a=input('entrez la valeur de a=') ;
b=input('entrez la valeur de b=') ;
c=input('entrez la valeur de c=') ;
delta = b^2 - 4*a*c ;
%% évaluation des racines
If delta < 0
Disp('Pas de solutions')
Elseif delta > 0
Disp('deux solution distinctes :')
x1 = (-b + sqrt(delta)) / (2*a)
x2 = (-b - sqrt(delta)) / (2*a)
Else
Disp('solution double :')
x = -b / (2*a)
end
%%  enregistrer le fichier avec le nom 'RacineEq2deg.m'
%%    solution
%%    pour l'exécution du programme, tapez le nom du fichier sur MATLAB
>> RacineEq2deg
Entrez la valeur de a=-5
Entrez la valeur de b=10
Entrez la valeur de c=12
```

CHAPITRE VII : EXEMPLES D'APPLICATION

EXERCICE VII.1.

Énoncé

On considère le système composé de trois masses m1, m2 et m3, et de trois ressorts de traction compression k1, k2 et k3. Chaque masse est repérée par sa position le long d'un axe horizontal, les paramètres de déplacement des masses sont notés u1, u2 et u3.



$k_1 = 10^4\ \mathrm{N/m}$ $k_2 = 5.10^4\ \mathrm{N/m}$ $k_3 = 3.10^4\ \mathrm{N/m}$

$m_1 = 100\ \mathrm{kg}$ $m_2 = 75\ \mathrm{kg}$ $m_3 = 50\ \mathrm{kg}$

$F_0 = 100\ \mathrm{N}$ $\xi = 3\ \%$ $V_0 = 1\ \mathrm{m/s}$

Déterminer

Les fréquences propres et les vecteurs propres du système ; les modes seront normés en composante maximale unitaire.

Solution

Étude théorique

Un rapide calcul donne :

$$2E_C = m_1\dot{u}_1^2 + m_2\dot{u}_2^2 + m_3\dot{u}_3^2 \qquad \text{soit :} \quad \mathbb{M} = \begin{bmatrix} m_1 & & \\ & m_2 & \\ & & m_3 \end{bmatrix}$$

$$2E_D = k_1u_1^2 + k_2(u_2 - u_1)^2 + k_3(u_3 - u_2)^2 \qquad \text{soit :} \quad \mathbb{K} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix}$$

Les pulsations propres et les vecteurs propres du système sont donc déterminés à partir de ces deux matrices.

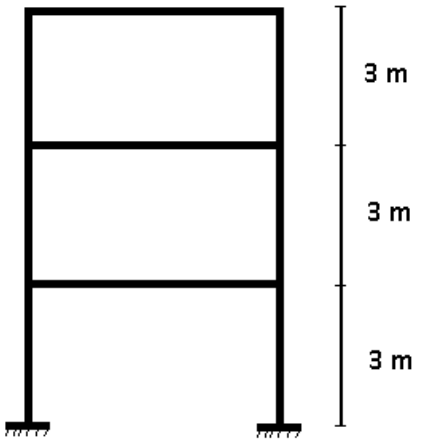
Les vecteurs propres seront normés en composante maximale unitaire (plus grande valeur absolue égale à 1).

Mise en œuvre sous MATLAB :

```
clear all
close all
clc
% SAISIE DES DONNEES
% raideurs
k1 = 1e4 ;
k2 = 5e4 ;
k3 = 3e4 ;
% masses
m1 = 100 ;
m2 = 75 ;
m3 = 50 ;
% MATRICES DU PROBLEMES
% matrice de masse
matm = diag([m1 m2 m3]) ;
% matrice de raideur
matk = [k1+k2 -k2 0 ; ...
-k2 k2+k3 -k3 ; ...
0 -k3 k3 ] ;
% RECHERCHE DES MODES PROPRES
[V,D]=eig(matk,matm) ;
% fréquences propres
fqce = sqrt(diag(D))/(2*pi)
% vecteurs propres normés en composante maximale unitaire
V1 = V(:,1)/max(abs(V(:,1)))*sign(V(find(abs(V(:,1))==max(abs(V(:,1)))),1))
V2 = V(:,2)/max(abs(V(:,2)))*sign(V(find(abs(V(:,2))==max(abs(V(:,2)))),2))
V3 = V(:,3)/max(abs(V(:,3)))*sign(V(find(abs(V(:,3))==max(abs(V(:,3)))),3))
```

EXERCICE VII.2.

La figure ci-contre illustre un portique en 2D de 3 niveaux. Il est constitué de poteaux de section carrés et de poutres infiniment rigides.



Données

- Sections poteaux (50x50) Cm²
- Hauteur niveau H=3m
- Module d'élasticité E= 2 . 10^6 KN/m².
- Moment d'inertie

$$I = \frac{bh^3}{12}$$

- Rigidité d'un poteau

$$K_p = \frac{12EI}{h^3}$$

- Ecrire un m-file pour la fonction moment d'inertie.
- Ecrire un programme qui permet de calculer la rigidité globale de chaque étage.

Solution

M-fille function	Programme	Résultats
function [I]=inertie(b,h) I=b*h^3/12	disp('--------- Données ------ --------') b=0.5; % metre h=0.5; % metre H=3; % metre E=2*10^6; % Kn/m² disp('-- Moment d"inertie --- ') I=inertie(0.5,0.5) disp('-- Rigidité -------') K=12*E*I/h^3 Kglo=2*K	I = 0.0052 K = 1000000 Kglob = 2000000

EXERCICE VII.3.

Soit une structure a 1DDl représentée par un système masse-ressort ou :

- Le moment d'inertie I= 0.0015, la masse M=1.5kg, et la hauteur de la structure H=4m.

Ce système est excité par une vitesse initial v0=1m/s et un déplacement initial x0=0.

1- Ecrire un fichier fonction qui permet de calculer la rigidité du système
Avec $K = \frac{12*E*I}{H^3}$

2- En utilisant la fonction précédente, écrire un programme qui permet de calculer la réponse vibratoire du système, donnée par la relation:

$$x = A * \sin(w * t + Phi)$$

Avec :

- $t \in [0,10]$s % temps de variation E. I. K
- $w = \sqrt{\frac{k}{m}}$ % Pulsation da la structure
- $A = \sqrt{\frac{(v_0^2+(x_0*w)^2)}{w^2}}$ % Amplitude de vibration
- $Phi = atan2(x_0 * w, v_0)$ % déphasage
- La réponse du système est donnée par la relation

3- Tracer la courbe de variation da la réponse du système dans l'intervalle [0,10], avec un pas de variation de 0.2s.

Solution

Programme

a- M-file fonction rigidité

```
function [K]=rigidite(I,E,H)
K=12*E*I/H^3;
```

b- Programme de calcul

```
clear all
close all
clc

 %%
 %%%%%%%%% Matrice Masse
m=10^4;
 %%%%%%%%% moment d'inertie
%  I=0.0015
 %%%%%%%%% hauteur H
% H=4
%%% module de young
% E=3*10^10
%vitesse initial
v0=1;
%deplacement initial
x0=0;
%temps d'observation
t=0:.2:10;

%%%%%%%%% Matrice rigidité
K=rigidite(0.0015,3*10^10,4)

%calcul des pulsations
w=sqrt(K/m)

%l'amplitude
A=sqrt(((v0)^2+(x0*w)^2)/w^2)

phi=atan2(x0*w,v0);
```

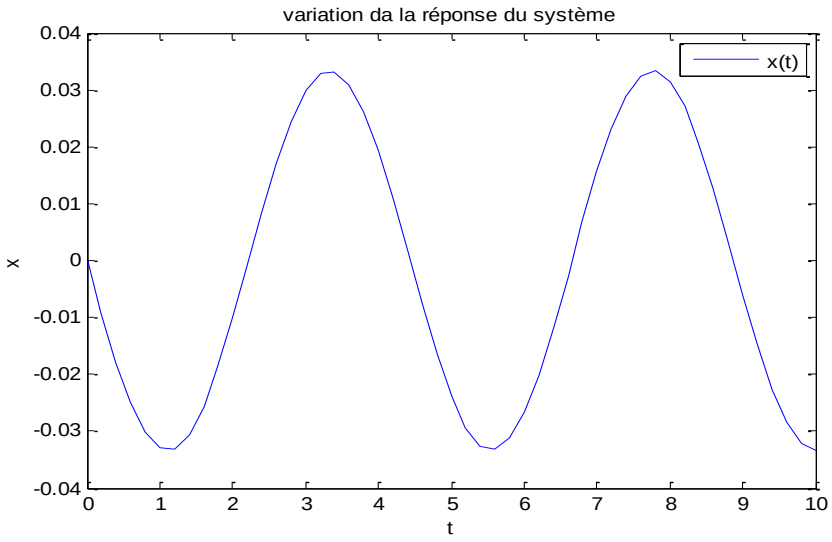
```
x=A*sin(w*t+phi);

% Graphe
plot(t,x)
title('variation da la réponse du système')
xlabel('t')
ylabel('x')
legend ('x(t)')
```

Résultats

```
K =
    9000000
w =
   30          pour (m=1.5)
A =
   0.0333      pour (m=1.5)
```

Graphique



EXERCICE VII.4.

Soit la fonction $f(x)=x^3+2x-1$. On cherche à estimer $x\in[0,1]$ tel que $f(x)=0$ avec une tolérance $\varepsilon=10^{-3}$.

Solution

Paramètres d'entrées :

f : fonction continue sur un intervalle I

a et b deux réels de I tels que *f*(a)**f*(b)<0

ξ : tolérance de précision

Paramètres d'entrées : c, n

c approximation du zero de la fonction *f*

n tolérance de précision

Algorithme

a0← a , b0←b
tant que (bn-an) > ξ faire
si f(an)*f(cn)>0 alors
an+1 ← cn, bn+1 ← bn
cn+1← (cn+bn)/2
sinon
an+1 ← an, bn+1 ← cn
cn+1← (cn+an)/2
fin si
fin tant que

```
clear all
close all
clc
% la mèthode de dichotonomie dans matlab
d=input('entrer la fonction f(x):','s');
f=inline(d);
a=input('entrer la valeur de a:');
b=input('entrer la valeur de b:');
tol=input('entrer la valeur de la tolérance:');
if f(b)*f(a)<0
else
    fprintf('f(b)*f(a)>0 valeur de a & b incorrectes! entrer nouvelle
valeurs \n');
    a=input('entrer la nouvelle valeur de a:\n');
    b=input('entrer la nouvelle valeur de b:\n');
end
for i=2:1000
    c=(b+a)/2;
    if f(b)*f(c)<0
        a=c;
    else
        b=c;
    end
   if f(a)*f(c)<0
        b=c;
    else
        a=c;
    end
    alpha(1)=0;
```

```
    alpha(i)=c;
    if abs(alpha(i)-alpha(i-1))<tol
        break
    end
end
sprintf('le nombre d ittération n=%d',i)
sprintf('la solution de f(x)=0 est c=%12.12f',c)

f(c)
```

RÉFÉRENCES BIBLIOGRAPHIQUES

- *BOUMAHRAT, M. , GOURDIN, A. (1993). Méthodes numériques appliquées. OPU, Alger.*
- Hunt, B. R., Lipsman, R. L., & Rosenberg, J. M. (2014). *A guide to MATLAB: for beginners and experienced users*. Cambridge university press.
- Varga, R. S. (1962). Matrix iterative analysis: Englewood Cliffs. *XIV–NY: Prentice-Hall*.
- Bestougeff, H., Guilpin, C., & Jacques, M. (1975). *La technique informatique: Algorithmes numériques et non numériques* (Vol. 2). Masson.
- Rajasekaran, S. (2009). *Structural dynamics of earthquake engineering: theory and application using MATHEMATICA and MATLAB*. Elsevier.
- Kwon, Y. W., & Bang, H. (2018). *The finite element method using MATLAB*. CRC press.
- Mathews, J. H., & Fink, K. D. (2004). *Numerical methods using MATLAB* (Vol. 4). Upper Saddle River, NJ: Pearson prentice hall.