1. Introduction :

- La **programmation** recouvre l'ensemble des techniques permettant de résoudre des problèmes à l'aide de **programmes** s'exécutant sur un ordinateur.
- Une étape essentielle consiste en l'écriture d'un texte de programme dans un langage particulier (un langage de programmation).
- Mais **l'écriture du programme**, bien que fondamentale, n'est **qu'une étape du processus de programmation** que l'on peut décomposer de la manière suivante :

a. L'analyse et la spécification du problème,

- Permettent de préciser le problème à résoudre,
- On détermine quelles sont les données et leurs propriétés,
- Quels sont les résultats attendus,
- Ainsi que les relations exactes entre les données et les résultats.

b. La conception (ou modélisation) : il s'agit généralement de déterminer la méthode de résolution du problème (un **algorithme)** et d'en identifier les principales étapes. C'est également dans cette phase que l'on conçoit la manière dont on va mettre en œuvre les fonctionnalités d'un logiciel (celles qui ont été identifiées dans la phase d'analyse).

c. L'implantation (ou codage) dans un ou plusieurs langages de programmation particuliers.

Il s'agit de traduire la méthode et les algorithmes préconisés en un ou plusieurs textes de programmes.
A ce stade, il faut respecter la syntaxe du langage de programmation choisi et préciser tout ce qui était resté dans l'ombre.

2. Généralités

2.1 Langage de programmation : Un code de communication, permettant à un être humain de dialoguer avec une machine en lui soumettant des instructions.

Les langages de programmation représentent l'interface entre le programmeur et l'ordinateur. Ils sont écrits pour exploiter la puissance des ordinateurs et leurs capacités de résolution de problèmes.

2.2 Programme: un programme est une suite d'instructions qui spécifient étape par étape les opérations à exécuter par un ordinateur.

2.3 Classification des langages de programmation

− Le domaine d'application : gestion, éducation, intelligence artificielle
− La technologie visée : réseau, Base de données, web
− La façon d'aborder un problème : utilisation des procédures, des objets,

Selon ce critère :

Les langages de programmation sont classés en paradigmes

- Programmation Impérative (Pascal, Cobol, fortran, C, ….)
- Programmation Déclarative
- Programmation O.O (Java, C++, simula, …)

- **Programmation Impérative** : décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme. Ce type de programmation est le plus répandu parmi l'ensemble des langages de programmation existants.
- **Programmation déclarative :** dans cette forme de programmation en décrivant la tâche que le programme doit accomplir au lieu de décrire comment le programme doit accomplir cette tâche.

  Les requêtes SQL de type SELECT sont déclaratives.

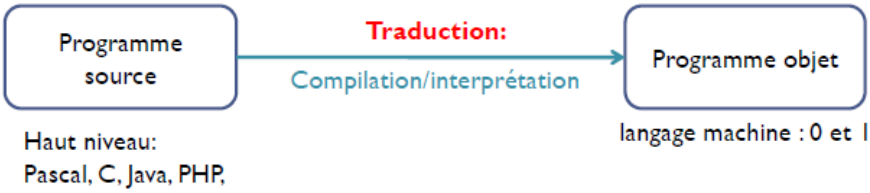
```
CREATE TABLE users (
        id INTEGER,
         name VARCHAR(50),
        PRIMARY KEY(id)
INSERT INTO users (id, name)
VALUES (1, 'Luke');
INSERT INTO users (id, name)
VALUES (2, 'Anakin');
SELECT id, name
FROM users;
```

- Ce code permet de déclarer à la machine que l'on souhaite créer, insérer puis récupérer les champs id et name de la table users, sans pour autant lui décrire **comment** y arriver.

Le développeur ici ne s'intéresse pas au contexte ou à l'état du programme qui enverra le résultat.

- Avec un langage impératif, le développeur aurait besoin d'écrire les instructions pour chercher parmi toutes les tables et trouver les bons résultats.

2.4 Code source vs code machine



— Programme source, code source :

→Programme écrit dans un langage.

— Code machine, code exécutable

→Programme dans un langage machine

→ Directement exécutable par la machine

a. **Langage machine** : utilisé par le processeur, il est constitué d'une suite de 0 et de 1. Le langage machine n'est pas compréhensible par l'être humain.
b. **L'assembleur :** est le premier langage informatique qui ait été utilisé. Celui-ci est très proche du langage machine mais reste compréhensible pour des développeurs.

2.5 **Compilateur** : Logiciel chargé de traduire le code source en langage machine. Un programme écrit dans un langage dit « **compilé** » va être traduit une fois pour toutes par un **compilateur**, afin de générer un nouveau fichier qu'on appelle fichier **exécutable** (programme objet).

Convertir un code source en un code exécutable,

Une fois le code source compilé, le programme " l'exécutable" peut être exécuté autant de fois que nécessaire.

Problème: Un programme compilé sous une plateforme **X ne peut être** exécuté sur toutes les autres plateformes. (Exemple programme écrit en C++).

2.6 **Interpréteur :** est un programme chargé de décoder chaque instruction du langage et d'exécuter les actions correspondantes instruction par instruction.

- Ne génère pas le fichier en code machine.
- Un interpréteur quand à lui, effectue le même travail de vérification du code que le compilateur, mais ne génère pas le code exécutable dans un fichier, il interprète le code ligne par ligne et les exécutent immédiatement.

Un programme écrit dans une plateforme **X peut être exécuté sur toutes les** autres plateformes (Exemple programme écrit en Python, Java)

2.7 Exemple de langages de programmation couramment

Langage	Domaine d'application principal	Compilé/interprété
ADA	**Le temps réel**	Langage compilé
BASIC	Educatif	Langage interprété
C	Programmation système	Langage compilé
C++	Programmation système objet	Langage compilé
Cobol	Gestion	Langage compilé
Fortran	Calcul	Langage compilé
MATLAB	Calcul mathématique (numérique)	Langage interprété
Mathematica	Calcul mathématique (formel	Langage interprété
Pascal	Enseignement	Langage compilé
PHP	Programmation web	Langage interprété
Prolog	Intelligence artificielle	Langage interprété
Perl	Traitement de chaînes de caractères	Langage interprété

3. **MATLAB** (**MAT**rix **LAB**oratory) :

Matlab est un environnement de programmation interactif pour le calcul scientifique, la programmation et la visualisation des données.

MATLAB comprend :

- De nombreuses fonctions graphiques,
- Un système puissant d'opérateurs s'appliquant à des matrices,
- Un langage de programmation extrêmement simple à utiliser.
- En termes de vitesse d'exécution, les performances sont inférieures à celles obtenues avec un langage de programmation classique. L'emploi de MATLAB devrait donc être restreint à des problèmes peu gourmands en temps calcul, mais dans la plupart des cas, il présente une solution élégante et rapide à mettre en œuvre.

3.1 Domaines d'application

- Domaines d'ingénierie
- Domaines de la recherche scientifique,
- Etablissements d'enseignement supérieur.

3.2 Point fort du matlab

- forte et simple interaction avec l'utilisateur
- Sa richesse fonctionnelle :
  - Réaliser des manipulations mathématiques complexes en écrivant peu d'instructions.
  - Evaluer des expressions, dessiner des graphiques et exécuter des programmes classiques.
  - Permet l'utilisation directe de plusieurs milliers de fonctions prédéfinie.
- La simplicité de son langage de programmation : un programme écrit en MATLAB 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.

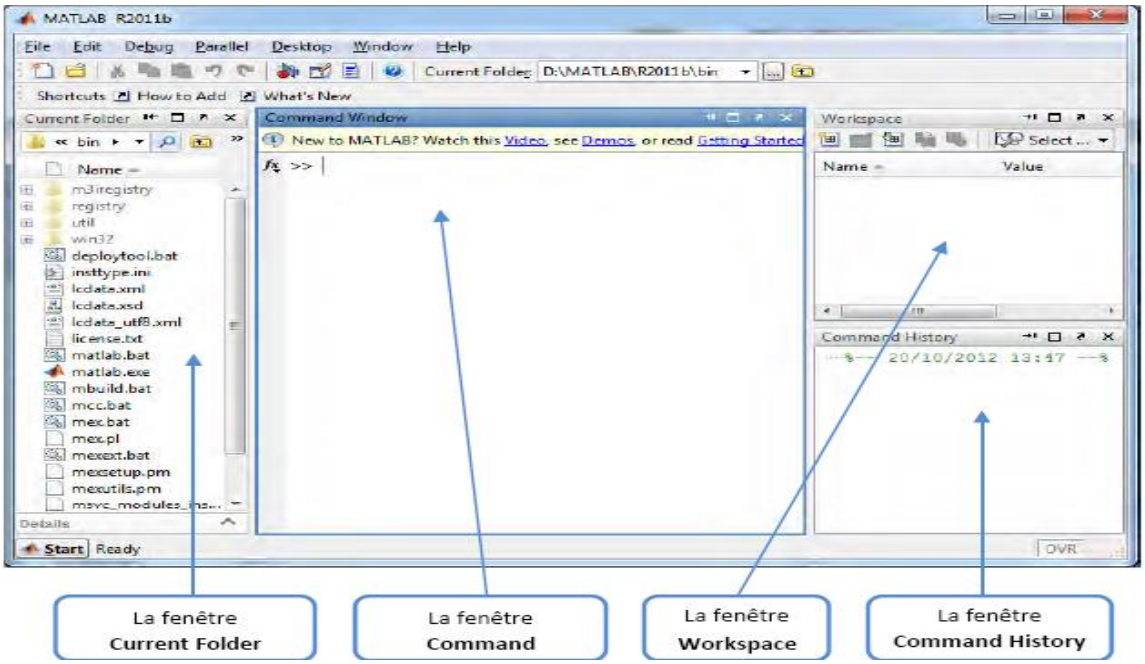
3.3 Histoire du Matlab

A l'origine MATLAB était conçu pour faire principalement des calculs sur les vecteurs et les matrices d'où son nom '**Matrix Laboratory',** mais par la suite **il a était amélioré et augmenté** pour pouvoir traiter beaucoup plus de domaines.

- MATLAB n'est pas le seul environnement de calcul scientifique
- Existent d'autres dont les plus importants sont **MAPLE** et **MATHEMATICA**.
- Il existe même des logiciels libres qui sont des clones de MATLAB comme **SCILAB** et **OCTAVE**.

3.4 L'environnement MATLAB

Au démarrage il affiche plusieurs fenêtres (selon la version)



- La fenêtre **CurrentFolder** : indique le répertoire courant ainsi que les fichiers existants.
- La fenêtre **Command** : Nous l'utilisons pour formuler nos expressions et interagir avec MATLAB, par l'utilisateur.
- La fenêtre **Workspace :** indique toutes les variables existantes avec leurs types et valeurs.
- La fenêtre **Command History** : garde la trace de toutes les commandes entrées.

3.5 Première interaction avec MATLAB

Le moyen le plus simple d'utiliser MATLAB est d'écrire directement dans la fenêtre de commande (Command Window) juste après le curseur (prompt) >> :
Pour calculer une expression mathématique il suffit de l'écrire comme ceci :
>> 5+6 Puis on clique sur la touche Entrer pour voir le résultat
ans =
11

Il est bien sûr possible de conserver un résultat de calcul et de le stocker dans des variables '**variable = définition**'. Gros avantage sur les langages classiques : on ne déclare pas les variables. Leur type (entier, réel, complexe) s'affectera automatiquement en fonction du calcul effectué. Pour affecter une variable, on dit simplement à quoi elle est égale. **Exemple** :

>> a=1.2

a =

1.2000

- On peut maintenant inclure cette variable dans de nouvelles expressions mathématiques, pour en définir une nouvelle :

>> b = 5*a^2+a

b =

8.4000

et ensuite utiliser ces deux variables :

>> c = a^2 + b^3/2

c =

297.7920

- On a maintenant trois variables a, b et c. Ces variables ne sont pas affichées en permanence à l'écran. Mais pour voir le contenu d'une variable, rien de plus simple, on tape son nom :

>> b

b =

8.4000

ou on double-click sur son nom dans le workspace.

Si nous voulons qu'une expression soit calculée mais sans afficher le résultat, on ajoute un point-virgule ';' à la fin de l'expression comme suit :

>> 5+6 ;

>>

Il est possible d'écrire plusieurs expressions dans la même ligne en les faisant séparées par des virgules ou des points virgules. Par exemple :

>> 5+6, 2*5-1, 12-4

ans =

11

ans =

9

ans =

8

>> 5+6; 2*5-1, 12-4;

ans =

9

>>

3.6 Les variables en Matlab : le nom d'une variable ne doit contenir que des caractères alphanumériques ou le symbole '_', et doit commencer par un alphabet. Nous devons aussi faire attention aux majuscules car le MATLAB est sensible à la casse (**A** et **a** sont deux identifiants différents).

3.7 Les opérations de base dans une expression :

Les opérations de base dans une expression sont résumées dans le tableau suivant :

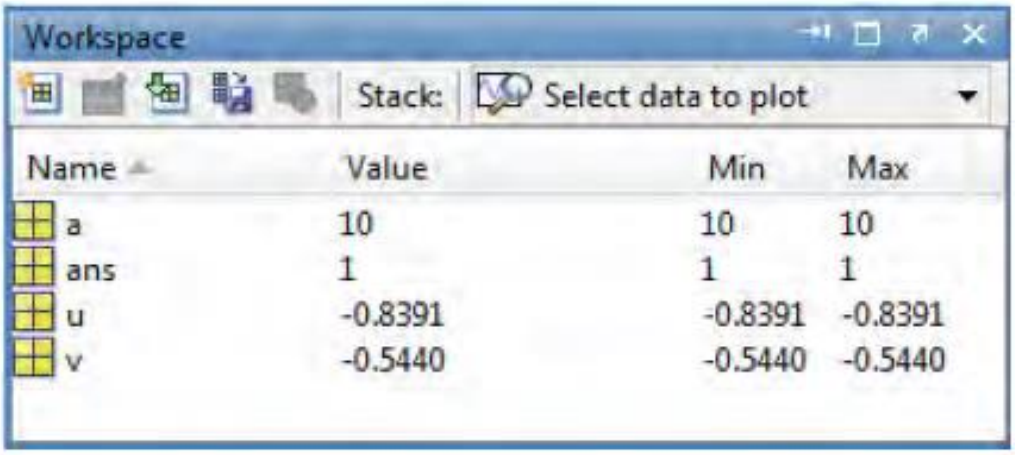
L'opération	La signification
+	L'addition
-	La soustraction
*	La multiplication
/	La division
\	La division gauche (ou la division inverse)
^	La puissance
'	Le transposé
(et)	Les parenthèses spécifient l'ordre d'évaluation

Pour voir la liste des variables utilisées, soit on regarde à la fenêtre **'Workspace'** soit on utilise les commandes '**whos**' ou '**who**'. **whos** donne une description détaillée (le nom de la variable, son type et sa taille), par contre **who** donne juste les noms des variables.

Par exemple, on a utilisé 3 variables **a, u et v :**

```
>> who
      Your variables are:
      a    ans  u    v

>> whos
      Name      Size            Bytes  Class     Attributes
      a         1x1                 8  double
      ans       1x1                 8  double
      u         1x1                 8  double
      v         1x1                 8  double
```

L'utilisation de ces deux commandes peut être omise car des informations sur les variables sont visibles directement dans la fenêtre workspace.



3.8 Les nombres en MATLAB

- Une notation décimale conventionnelle, avec un point décimal facultatif '.'
- Le signe '+' ou '–' pour les nombres signés.
- La notation scientifique utilise la lettre 'e' pour spécifier le facteur d'échelle en puissance de 10.
- Les nombres complexes utilise les caractères 'i' et 'j' (indifféremment) pour designer la partie imaginaire.

Le type	Exemples	
Entier	5	-83
Réel en notation décimale	0.0205	3.1415926
Réel en notation scientifique	1.60210e-20	6.02252e23 (1.60210×10^{-20} et 6.02252×10^{23})
Complexe	5+3i	-3.14159j

MATLAB utilise toujours les nombres réels (double précision) pour faire les calculs, ce qui permet d'obtenir une précision de calcule allant jusqu'aux 16 chiffres significatifs.

La commande	Signification
format short	Affiche les nombres avec 4 chiffres après la virgule
format long	Affiche les nombres avec 15 chiffres après la virgule
format bank	Affiche les nombres avec 2 chiffres après la virgule
format rat	Affiche les nombres sous forme d'une ration (a/b)

```
>> format long

>> 8/3
       ans =
              2.66666666666667
>> format bank

>> 8/3
       ans =
              2.67
>> format short

>> 8/3
       ans =
              2.6667



>> 7.2*3.1
       ans =
              22.3200

>> format rat

>> 7.2*3.1
       ans =
              558/25

>> 2.6667
ans =
   26667/10000
```

La fonction **vpa** peut être utilisé afin de forcer le calcule de présenter plus de décimaux significatifs en spécifiant le nombre de décimaux désirés.

Exemple :

```
>> sqrt(2)
       ans =
             1.4142

>> vpa(sqrt(2),50)
       ans =
             1.4142135623730950488016887242096980785696718753769
```

3.9 Les principales constantes, fonctions et commandes

3.9.1 Les principales constantes

La constante	Sa valeur
pi	π=3.1415...
exp(1)	e=2.7183...
i	$=\sqrt{-1}$
j	$=\sqrt{-1}$
Inf	∞
NaN	Not a Number (Pas un numéro)
eps	$\varepsilon \approx 2 \times 10^{-16}$.

3.9.2 Les principales fonctions

Parmi les fonctions fréquemment utilisées :

La fonction	Sa signification
sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian)
sqrt(x)	la racine carrée de x → $\sqrt{.}$
abs(x)	la valeur absolue de x → $\|x\|$
exp(x)	$= e^x$
log(x)	logarithme naturel de x → $\ln(x)=\log_e(x)$
log10(x)	logarithme à base 10 de x → $\log_{10}(x)$
imag(x)	la partie imaginaire du nombre complexe x
real(x)	la partie réelle du nombre complexe x
round(x)	arrondi un nombre vers l'entier le plus proche
floor(x)	arrondi un nombre vers l'entier le plus petit → $\min\{n \mid n \le x, n \text{ entier}\}$
ceil(x)	arrondi un nombre vers l'entier le plus grand → $\max\{n \mid n \ge x, n \text{ entier}\}$

3.9.3 Les principales commandes

MATLAB offre beaucoup de commandes. Nous nous contentons pour l'instant d'un petit ensemble:

La commande	Sa signification
who	Affiche le nom des variables utilisées
whos	Affiche des informations sur les variables utilisées
clear x y	Supprime les variables x et y
clear, clear all	Supprime toutes les variables
clc	Efface l'écran des commandes
exit, quit	Fermer l'environnement MATLAB
format	Définit le format de sortie pour les valeurs numériques `format long` : affiche les nombres avec 14 chiffres après la virgule `format short`: affiche les nombres avec 04 chiffres après la virgule `format bank` : affiche les nombres avec 02 chiffres après la virgule `format rat` : affiche les nombres sous forme d'une ration (a/b)

3.10 La priorité des opérations dans une expression

L'évaluation d'une expression s'exécute de gauche à droite en considérant la priorité des opérations indiquée dans le tableau suivant :

Les opérations	La priorité (1=max, 4=min)
Les parenthèses (et)	1
La puissance et le transposé ^ et '	2
La multiplication et la division * et /	3
L'addition et la soustraction + et -	4

Par exemple:

5+2*3 = 11 et 2*3^2 = 18

Exercice récapitulatif : créer une variable x et donnez-la la valeur 2, puis écrivez les expressions suivantes :

- $3x^3-2x^2+4x$
- $\frac{e^{1+x}}{1-\sqrt{2x}}$
- $|\sin^{-1}(2x)|$
- $\frac{\ln(x)}{2x^3} - 1$

La solution :

```
>> x=2 ;
>> 3*x^3-2*x^2+4*x ;
>> exp(1+x)/(1-sqrt(2*x)) ;
>> abs(asin(2*x)) ;
>> log(x)/(2*x^3)-1 ;
```

4. Les vecteurs et les matrices

4.1 Les vecteurs

- Un vecteur est une liste ordonnée d'éléments.
- Un vecteur ligne: les éléments sont arrangés horizontalement.
- Un vecteur colonne: les éléments sont arrangés verticalement.

- Pour créer un **vecteur ligne** il suffit d'écrire la liste de ses composants entre crochets **[ ]**, et de les séparés par des **espaces** ou des **virgules** comme suit :

```
>> u=[1 4 7 9]
u =
   1    4    7    9
>> u=[1,4,7,9]
u =
   1    4    7    9
```

- Pour créer un **vecteur colonne** il est possible d'utiliser une des méthodes suivantes :

```
>> U = [ 4 ; -2 ; 1 ]                    % Création d'un vecteur colonne U
      U =
            4
           -2
            1
```

2. écrire verticalement le vecteur :

```
>> U = [
    4
   -2
    1
]
      U =

            4
           -2
            1
```

3. calculer le transposé d'un vecteur ligne :

```
>> U = [ 4 -2 1 ]'                       % Création d'un vecteur colonne U
      U =
            4
           -2
            1
```

- Si les composants d'un vecteur X sont ordonnés avec des valeurs consécutives, nous pouvons le noter avec la notation suivante :

```
X = premier_élément : dernier_élément
```

(Les crochets sont facultatifs dans ce cas)

Par exemple :

```
>> X = 1:8                               % on peut aussi écrire colon(1,8)
      X =
            1     2     3     4     5     6     7     8
>> X = [1:8]
      X =
            1     2     3     4     5     6     7     8
```

- Si les composants d'un vecteur **X sont ordonnés avec des valeurs consécutives** mais avec un pas (d'incrémentation/décrémentation) **différente de 1**, nous pouvons spécifier le pas avec la notation :

  X=premier_élément : le pas : dernier_élément

```
>> X=[0:2:10]
X =
    0    2    4    6    8   10
>> X=[-4:2:6]
X =
   -4   -2    0    2    4    6
>> X=0:0.2:1
X =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

  On peut écrire des expressions plus complexes comme :

```
>> V=[1:2:5,-2:2:1]
V =
    1    3    5   -2    0
>> A=[1 2 3]
A =
    1    2    3
>> B=[A,4,5,6]
B =
    1    2    3    4    5    6
```

- **La fonction linspace : l**a création d'un vecteur dont les composants sont ordonnés par intervalle régulier et avec un nombre d'éléments bien déterminé peut se réaliser avec la fonction **: linspace (début, fin, nombre d'éléments)**

  Le pas d'incrémentation est calculé automatiquement par MATLAB selon la formule :

$$le\ pas = \frac{fin - debut}{nombre\ 'éléments - 1}$$

Par exemple :

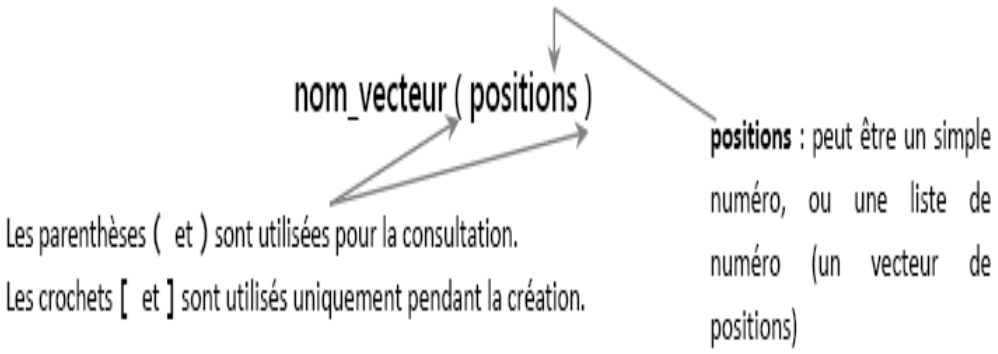
```
>> X = linspace(1,10,4)          % un vecteur de quatre élément de 1 à 10
      X =
            1     4     7    10

>> Y = linspace(13,40,4)         % un vecteur de quatre élément de 13 à 40
      Y =
           13    22    31    40
```

La taille d'un vecteur (le nombre de ses composants) peut être obtenue avec la fonction **length** comme suit :

```
>> length(X)              % la taille du vecteur X
      ans =
            4
```

4.1.1 Référencement et accès aux éléments d'un vecteur

L'accès aux éléments d'un vecteur se fait en utilisant la syntaxe générale suivante



Exemples :

```
>> V =[1:2:12,-1:1:2]
V =
 1    3    5    7    9   11   -1    0    1    2
>> V([1,3,4])
Ans=
    1    5    7
>> V(1)=8
Ans=
 8    3    5    7    9   11   -1    0    1    2
>> V(12)=-5
Ans=
 8    3    5    7    9   11   -1    0    1    2    0    -5
>> V(2) =[]
V =
 8    5    7    9   11   -1    0    1    2   0   -5
%supprimer le deuxième élément
>> V(3:5)=[]
Ans=
    8    5   -1   0   1   2   0   -5
%supprimer du 3eme au  5eme élément
```

4.1.2 Les opérations élément-par-élément pour les vecteurs

Avec deux vecteurs $\vec{u}$ et $\vec{v}$, il est possible de réaliser des calcules élément par élément en utilisant les opérations suivantes :

L'opération	Signification	Exemple avec : >> u = [-2, 6, 1] ; >> v = [3, -1, 4] ;
+	Addition des vecteurs	>> u+2 ans = 0 8 3 >> u+v ans = 1 5 5
-	Soustraction des vecteurs	>> u-2 ans = -4 4 -1 >> u-v ans = -5 7 -3
.*	Multiplication élément par élément	>> u*2 ans = -4 12 2 >> u.*2 ans = -4 12 2 >> u.*v ans = -6 -6 4
./	Division élément par élément	>> u/2 ans = -1.0000 3.0000 0.5000 >> u./2 ans = -1.0000 3.0000 0.5000 >> u./v ans = -0.6667 -6.0000 0.2500
.^	Puissance élément par élément	>> u.^2 ans = 4 36 1 >> u.^v ans = -8.0000 0.1667 1.0000

L'écriture d'une expression tel que : **u*u** génère une erreur car cette expression réfère a une multiplication de matrices (u*u doit être réécrite u*u' ou u'*u pour être valide).

4.2 Les matrices

Une matrice est un tableau bidimensionnel d'éléments.

Les vecteurs sont des matrices avec une seule ligne ou une seule colonne (monodimensionnels).

Pour insérer une matrice, il faut respecter les règles suivantes :

• Les éléments doivent être mises entre des crochets **[et]**

• Les **espaces** ou les **virgules** sont utilisés pour s**éparer** les éléments dans la **même ligne**

• Un **point-virgule** (ou la touche **entrer)** est utilisé pour séparer **les lignes**

➢ Pour illustrer cela, considérant la matrice suivante:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Cette Matrice peut être écrite en Matlab avec une des syntaxes suivantes:

>> A=[1,2,3,4;5,6,7,8;9,10,11,12];

>> A=[1 2 3 4;5 6 7 8;9 10 11 12];

>> A=[1 2 3 4

5 6 7 8

9 10 11 12];

>> A=[[1;5;9],[2;6;10],[3;7;11],[4;8;12]];

- Le nombre d'éléments dans chaque ligne (nombre de colonnes) doit être identique dans toutes les lignes de la matrice, sinon une erreur sera signalée par MATLAB.
  Exemple:
  >> X=[1 2;3 4 5]
  Error using vertcat
  Dimensions of matrices being concatenated are not consistent.

Une matrice peut être générée par des vecteurs comme le montre les exemples suivants :

```
>> x = 1:4                          % création d'un vecteur x
      x =
            1     2     3     4
>> y = 5:5:20                       % création d'un vecteur y
      y =
            5    10    15    20
>> z = 4:4:16                       % création d'un vecteur z
      z =
            4     8    12    16
```

```
>> A = [x ; y ; z]                % A est formée par les vecteurs lignes x, y et z
      A =
            1     2     3     4
            5    10    15    20
            4     8    12    16

>> B = [x' y' z']                 % B est formée par les vecteurs colonnes x, y et z
      B =
            1     5     4
            2    10     8
            3    15    12
            4    20    16

>> C = [x ; x]                    % C est formée par le même vecteur ligne x 2 fois
      C =
            1     2     3     4
            1     2     3     4
```

4.2.1 Référencement et accès aux éléments d'une matrice

L'accès aux éléments d'une matrice se fait en utilisant la syntaxe générale suivante :
nom_matrice (positions_lignes , positions_colonnes)
Positions : peut être un simple numéro, ou une liste de numéro (un vecteur de positions)
Les parenthèses **(et) sont utilisées pour la consultation.**
Les crochets **[et] sont utilisés uniquement pendant la création.**

Notes:

- L'accès à un élément de la ligne i et la colonne j se fait par **A(i,j)**
- L'accès à toute la ligne numéro i se fait par **: A(i,:)**
- L'accès à toute la colonne numéro j se fait par **: A(:,j)**

Exemples :

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]      % création de la matrice A
      A =
            1     2     3     4
            5     6     7     8
            9    10    11    12

>> A(2,3)                          % l'élément sur la 2ème ligne à la 3ème colonne
      ans =
            7

>> A(1,:)                          % tous les éléments de la 1ère ligne
      ans =
            1     2     3     4

>> A(:,2)                          % tous les éléments de la 2ème colonne
      ans =
            2
            6
            10
```

```
>> A(2:3,:)                    % tous les éléments de la 2ème et la 3ème ligne
      ans =
            5     6     7     8
            9    10    11    12

>> A(1:2,3:4)                  % La sous matrice supérieure droite de taille 2x2
      ans =
            3     4
            7     8

>> A([1,3],[2,4])              % la sous matrice : lignes(1,3) et colonnes (2,4)
      ans =
            2     4
           10    12

>> A(:,3) = []                 % Supprimer la troisième colonne
      A =
            1     2     4
            5     6     8
            9    10    12
```

```
>> A(2,:) = []                      % Supprimer la deuxième ligne
       A =
             1     2     4
             9    10    12
```

```
>> A = [A , [0;0]]                  % Ajouter une nouvelle colonne avec A(:,4)=[0;0]
      A =
            1     2     4     0
            9    10    12     0
```

```
>> A = [A ; [1,1,1,1]]              % Ajouter une nouvelle ligne avec A(3,:)=[1,1,1,1]
      A =
            1     2     4     0
            9    10    12     0
            1     1     1     1
```

- La fonction **size**: permet d'acquise les dimensions d'une matrice**.** Son résultat pour une matrice A de dimension m × n est un vecteur de deux composants, une pour **m** et l'autre pour **n**.

```
>> d = size(A)
      d =
            3     4
```

Pour obtenir les dimensions séparément on peut utiliser la syntaxe :

```
>> d1 = size (A, 1)                 % d1 contient le nombre de ligne (m)
      d1 =
            3
```

```
>> d2 = size (A, 2)                 % d2 contient le nombre de colonne (n)
      d2 =
            4
```

4.2.2 Génération automatique des matrices

Il existe des fonctions qui permettent de générer automatiquement des matrices particulières :

La fonction	Signification
zeros(n)	Génère une matrice n × n avec tous les éléments = 0
zeros(m,n)	Génère une matrice m × n avec tous les éléments = 0
ones(n)	Génère une matrice n × n avec tous les éléments = 1
ones(m,n)	Génère une matrice m × n avec tous les éléments = 1
eye(n)	Génère une matrice identité de dimension n × n
magic(n)	Génère une matrice magique de dimension n × n
rand(m,n)	Génère une matrice de dimension m × n de valeurs aléatoires

4.2.3 Les opérations de base sur les matrices

L'opération	Signification
+	L'addition
-	La soustraction
.*	La multiplication élément par élément
./	La division élément par élément
.\	La division inverse élément par élément
.^	La puissance élément par élément
*	La multiplication matricielle
/	La division matricielle (A/B) = ($A*B^{-1}$)

Remarque : Les opérations élément par éléments sur les matrices sont les mêmes que ceux pour les vecteurs (la seule condition nécessaire pour faire une opération élément par élément est que les deux matrices aient les mêmes dimensions).

Exemple :

```
>> A=ones(2,3)
      A =
            1     1     1
            1     1     1
>> B=zeros(3,2)
      B =
            0     0
            0     0
            0     0
>> B=B+3
B =
     3     3
     3     3
     3     3
>> A*B
      ans =
            9     9
            9     9
>> B=[B , [3 3 3]']            % ou bien B(:,3)=[3 3 3]'
      B =
            3     3     3
            3     3     3
            3     3     3
>> B=B(1:2,:)                  % ou bien B(3,:)=[]
      B =
            3     3     3
            3     3     3
>> A=A*2
      A =
            2     2     2
            2     2     2
>> A.*B
      ans =
            6     6     6
            6     6     6
>> A*eye(3)
      ans =
            2     2     2
            2     2     2
```

4.2.4 Fonctions utiles pour le traitement des matrices

La fonction	L'utilité	Exemple d'utilisation

det	Calcule de déterminant d'une matrice	`>> A = [1,2;3,4] ;`
		`>> det(A)`
		`ans =`
		`-2`

inv	Calcule l'inverse d'une matrice	`>> inv(A)`
		`ans =`
		`-2.0000 1.0000`
		`1.5000 -0.5000`

rank	Calcule le rang d'une matrice	`>> rank(A)`
		`ans =`
		`2`

trace	Calcule la trace d'une matrice	`>> trace(A)`
		`ans =`
		`5`

eig	Calcule les valeurs propres	`>> eig(A)`
		`ans =`
		`-0.3723`
		`5.3723`

dot	Calcule le produit scalaire de 2 vecteurs	`>> v = [-1,5,3];`
		`>> u = [2,-2,1];`
		`>> dot(u,v)`
		`ans =`
		`-9`

norm	Calcule la norme d'un vecteur	`>> norm(u)`
		`ans =`
		`3`

cross	Calcule le produit vectoriel de 2 vecteurs	`>> cross(u,v)`
		`ans =`
		`-11 -7 8`

diag	Renvoie le diagonal d'une matrice	`>> diag(A)`
		`ans =`
		`1`
		`4`

diag(V)	Crée une matrice ayant le vecteur V dans le diagonal et 0 ailleurs.	`>> V = [-5,1,3]`
		`>> diag(V)`
		`ans =`
		`-5 0 0`
		`0 1 0`
		`0 0 3`

tril	Renvoie la partie triangulaire inferieure	**>> B=[1,2,3;4,5,6;7,8,9** **B =** **1 2 3** **4 5 6** **7 8 9** **>> tril(B)** **ans =** **1 0 0** **4 5 0** **7 8 9** **>> tril(B,-1)** **ans =** **0 0 0** **4 0 0** **7 8 0** **>> tril(B,-2)** **ans =** **0 0 0** **0 0 0** **7 0 0**
triu	Renvoie la partie triangulaire supérieure	**>> triu(B)** **ans =** **1 2 3** **0 5 6** **0 0 9** **>> triu(B,-1)** **ans =** **1 2 3** **4 5 6** **0 8 9** **>> triu(B,1)** **ans =** **0 2 3** **0 0 6** **0 0 0**

4.2.5 Comparaison des matrices

La comparaison des vecteurs et des matrices diffère quelque peu des scalaires, d'où l'utilité des deux fonctions '**isequal' et 'isempty' (qui permettent de donner une réponse** concise pour la comparaison).

La fonction	Description
isequal	teste si deux (ou plusieurs) matrices sont égales (ayant les mêmes éléments partout). Elle renvoie **1** si c'est le cas, et **0** sinon.
isempty	teste si une matrice est vide (ne contient aucun élément). Elle renvoie **1** si c'est le cas, et **0** sinon.

Exemple :

```
>> A = [5,2;-1,3]            % Créer la matrice A
       A =
             5     2
            -1     3
>> B = [5,1;0,3]             % Créer la matrice B
       B =
             5     1
             0     3
```

```
>> A==B                    % Tester si A=B ? (1 ou 0 selon la position)
      ans =
            1     0
            0     1
>> isequal(A,B)            % Tester si effectivement A et B sont égales (les mêmes)
      ans =
            0
>> C=[] ;                  % Créer la matrice vide C
>> isempty(C)              % Tester si C est vide (affiche vrai = 1)
      ans =
            1
>> isempty(A)              % Tester si A est vide (affiche faux = 0)
      ans =
            0
```

……..Suite partie 2

Références :

[1] Cours Outils de programmation pour les mathématiques, Y.BAZIZ, Centre Universitaire RELIZANE, Institut des Sciences et Technologies, Spécialité : Mathématique et Informatique, LMD 1ére année 2013/2014.
[2] Tutoriel MATLAB, Introduction aux Méthodes Numériques 2013-2014.
[3] poly introduction Matlab, Introduction à Matlab
[4] Introduction à MATLAB, André Casadevall, Université Paris-Dauphine, Département MIDO, mars 2013
[5] Outils Mathématiques et utilisation de Matlab, Quentin Glorieux , Cours 2013-2014, Université Pierre et Marie Curie - Paris VI, Licence Professionnelle (L3) Instrumentation Optique et Visualisation,.