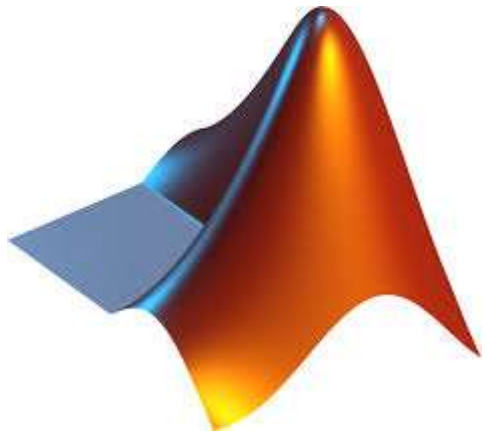
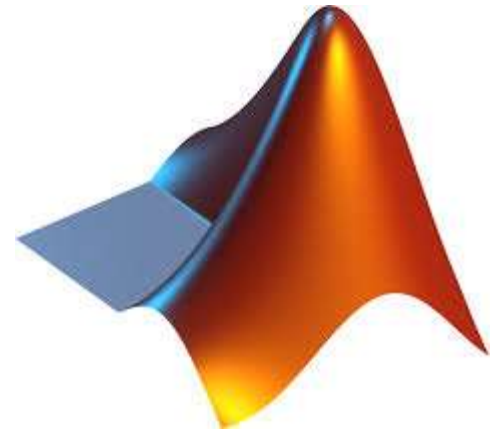
Outils de Programmation Mathématique 2



Contenu de la matière :

Chapitre 1 : Prise en Main

Démarrage et aide variable – Variables - Répertoire de travail - Sauvegarde de l'environnement du travail - Fonctions et commandes.



Chapitre 2 : Les nombre en Matlab avec licence ou Scilab

Entiers naturels - Représentation des réelles - Nombres complexe.

Chapitre 3 : Vecteurs et Matrices

Opérations sur les vecteurs et les Matrices - Fonctions mathématiques élémentaires.

Chapitre 4 : Eléments de programmation

Script – Fonction - Boucle de contrôle - Instruction conditionnelle.

Chapitre 5 : Polynômes

Polynômes en Matlab avec licence ou Scilab - Zéros d'un polynôme - Opérations sur les polynômes.

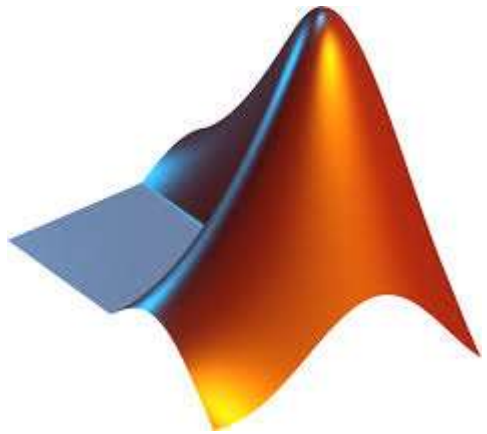
Chapitre 6 : Graphisme en Matlab avec licence ou Scilab

Affichage des courbes en dimension deux et dimension trois - Graphe d'une fonction - Surface Analytique.

Chapitre 7 :Calcul symbolique

Appel de la toolboxsymbolic - Développement et mise en fonction d'une expression - Dérivée et primitive d'une fonction - Calcul du développement limité d'une fonction.

Prise en Main

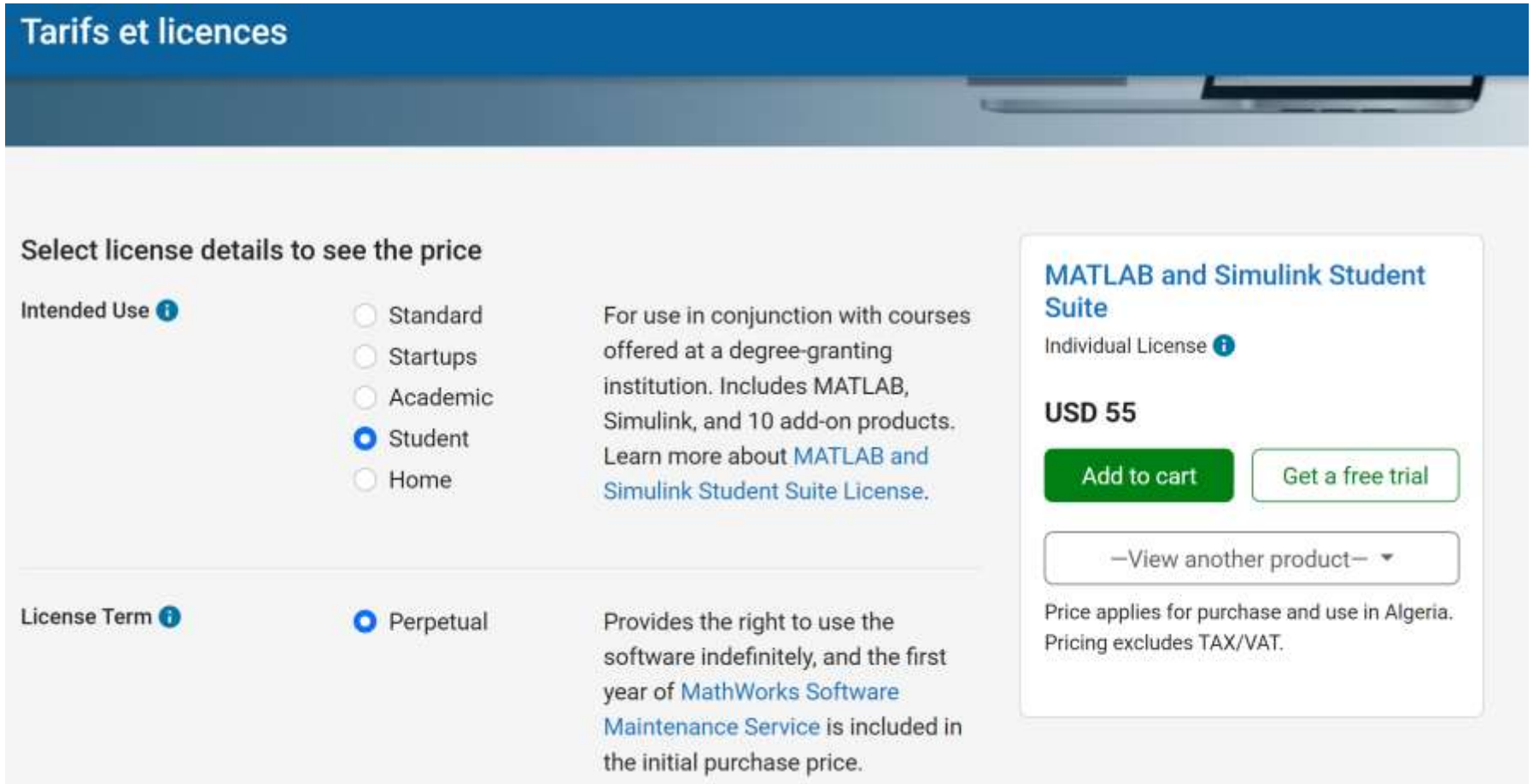


Introduction

- Le nom **Matlab** est la contraction du terme anglais Matrix Laboratory.
- Ce logiciel est spécialement conçu pour le calcul scientifique et la manipulation de vecteurs et de matrices.
- **Matlab** est à la fois un langage de programmation et un environnement de développement développé et commercialisé par la société américaine **MathWorks**.
- **Matlab** est utilisé dans les domaines de l'éducation, de la recherche et de l'industrie pour le calcul numérique mais aussi dans les phases de développement de projets.

Introduction

Prix de Matlab



Introduction

Alternatives de Matlab

1. Scilab est un logiciel open-source sous licence GPL, développé depuis 1990 par des chercheurs de l'**INRIA**(Institut national de recherche en informatique et automatique), il est maintenant maintenu par la fondation de coopération scientifique **Digiteo**.

- Il est disponible sur les plateformes Windows, Mac OS X, Linux et BSD.

- Pour plus d'informations et pour télécharger le logiciel:

http://www.scilab.org/

Introduction

2. Octave est également un logiciel open-source sous licence GPL.
Son développement a commencé au début des années 90 par
John W.Eaton dans le cadre du projet **GNU**.

- Sa syntaxe est proche de celle de **Matlab**.

- Il est disponible sur les plateformes Windows, Mac OS X, Linux et BSD.

- Pour plus d'informations et pour télécharger le logiciel:
http://www.gnu.org/software/octave/

Introduction

- Matlab est un **interpréteur**: les instructions sont interprétées et exécutées ligne par ligne.
- Matlab fonctionne dans plusieurs environnement tels que Windows, Macintosh, UNIX, Linux.
- Il existe deux modes de fonctionnement:

1. **Mode interactif:** Matlab exécute les instructions au fur et à mesure qu'elles sont données par l'utilisateur.
2. **Mode exécutif**: Matlab exécute ligne par ligne un '***fichier M***' (programme en langage Matlab).

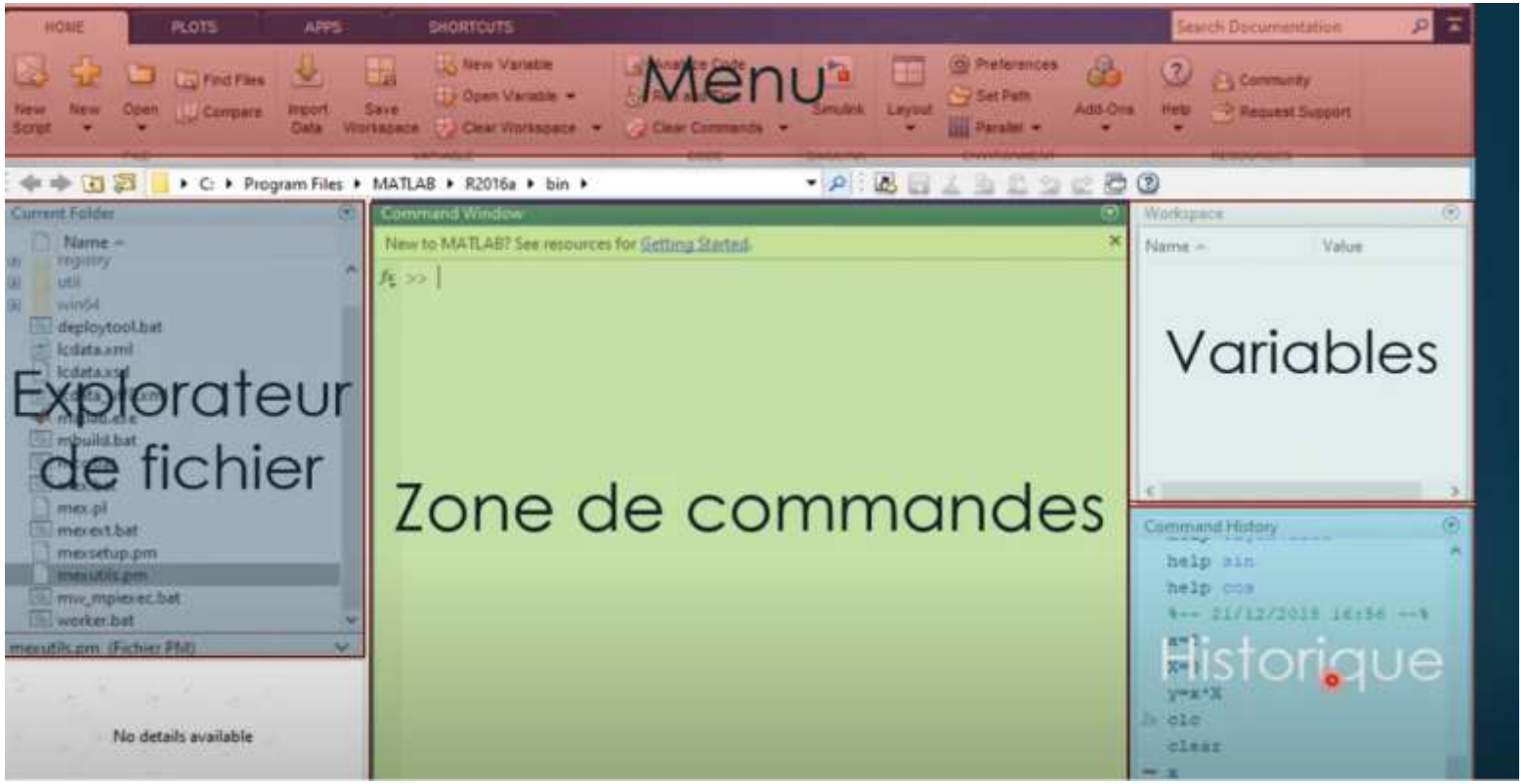
Démarrage de Matlab

- Pour démarrer Matlab en mode graphique entrez

  matlab

- Sur Windows, il suffit de cliquer sur l'icône Matlab 

Explication de l'interface du Matlab



Explication de l'interface du Matlab

- **Le Menu** regroupe les commandes de base de Matlab: enregistrer, afficher, préférences etc.
- **L’exploiteur de fichier** permet de visualiser les fichiers scripts et de les ouvrir pour édition.
- **La zone de commandes** permet d’écrire les commandes et de visualiser le résultat d’exécution.
- **La zone de variables** permet de visualiser toutes les variables en mémoire à l’instant présent (les noms et les contenus).
- **L’historique** permet de visualiser l’historique des commandes précédemment exécutées.

Aide Matlab

- L'aide en ligne s'obtient en tapant la commande ***help***.
- Une longue liste de sujets, pour lesquels l'aide est disponible, apparaît alors.

```
>> help

HELP topics:

matlab/general       -  General purpose commands.
matlab/ops           -  Operators and special characters.
matlab/lang          -  Programming language constructs.
matlab/elmat         -  Elementary matrices, matrix manipulation.
matlab/elfun         -  Elementary math functions.
matlab/specfun       -  Specialized math functions.
matlab/matfun        -  Matrix functions, numerical linear algebra.
matlab/datafun       -  Data analysis and Fourier transforms.
matlab/audio         -  Audio support.
matlab/polyfun       -  Interpolation and polynomials.
matlab/funfun        -  Function functions and ODE solvers.
matlab/sparfun       -  Sparse matrices.
matlab/graph2d       -  Two dimensional graphs.
```

- Après, il suffit d'exécuter

>>help sujet

pour avoir l'aide sur le sujet concerné.

- Pour avoir la description d'une fonction MATLAB, il suffit d'exécuter la commande *help* avec le nom de la fonction

>>help fonction

Par exemple, si on veut la description de la fonction *plot*, qui permet de tracer des courbes 2D, on tape simplement:

```
>>help plot

 PLOT   Linear plot.
    PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
    then the vector is plotted versus the rows or columns of the matri
    whichever line up.  If X is a scalar and Y is a vector, length(Y)
    disconnected points are plotted.

    PLOT(Y) plots the columns of Y versus their index.
    If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
    In all other uses of PLOT, the imaginary part is ignored.

    Various line types, plot symbols and colors may be obtained with
    PLOT(X,Y,S) where S is a character string made from one element
    from any or all the following 3 columns:

           y     yellow          .     point              -     solid
           m     magenta         o     circle             :     dotted
           c     cyan            x     x-mark             -.    dashdot
           r     red             +     plus               --    dashed
           g     green           *     star
           b     blue            s     square
           w     white           d     diamond
           k     black           v     triangle (down)
                                 ^     triangle (up)
                                 <     triangle (left)
                                 >     triangle (right)
                                 p     pentagram
                                 h     hexagram

    For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus
```

- Il est possible de retrouver toutes les fonctions contenant un mot clé. Pour cela on utilise la commande *lookfor*.
- Par exemple, si on recherche les fonctions relatives à la fonction d'affichage *disp*

```
>> lookfor disp
imagemodel                  - Access to properties of an image relevant to its d
celldisp                    - Display cell array contents.
cellplot                    - Display graphical depiction of cell array.
enumeration                 - Display class enumeration member and names.
events                      - Display class event names.
javaMethodEDT               - Invoke a Java method from the Event Dispatch Threa
javaObjectEDT               - Invoke a Java object constructor and subsequent me
methods                     - Display class method names.
properties                  - Display class property names.
superclasses                - Display superclass names.
depdir                      - Start to display DEPDIR's deprecation warning in 1
echo                        - Display statements during function execution.
details                     - Display array details
```

- Si dans l’aide en ligne, les fonctions et commandes MATLAB apparaissent en lettres majuscules c’est seulement pour les distinguer des autres mots. Les fonctions et commandes Matlab doivent être codées en minuscules.
- Les commandes d’éditions de Matlab sont simples:

↑ remonter dans l’historique de commandes;

↓ descendre dans l’historique de commandes;

←, → déplacement sur la ligne; Backspace, Delete modifications sur la ligne de commande.

Variables

Les noms des variables MATLAB sont des mots de 63 caractères maximum, sans espace et sans symbole de ponctuation. Le nom d'une variable doit obligatoirement commencer par une lettre de l'alphabet. Il y a une distinction entre majuscules et minuscules pour les noms de variables.

```
>> x1=3.8675, X1=pi^2
x1 =
    3.8675

X1 =
    9.8696
```

Comme tous les langages, MATLAB dispose de mots-clés. Certains mots-clés ne peuvent être utilisés comme variables. Ce sont les mots réservés du langage. La liste des mots-clés de MATLAB est donnée dans le tableau 1.1.

```
>> for=2.16
??? for=2.16
        |
Error: "identifier" expected, "=" found.
```

La liste de de ces mots réservés est fournie par la fonction `iskeyword`. D'autres mots-clés, non réservés, représentent des variables ou des constantes prédéfinies, *cf.* tableau 1.2. Ces mots-clés peuvent être utilisés comme variables définies par l'utilisateur.

```
>> pi, realmax
ans =
    3.1416
```

`ans`	*answer* variable MATLAB par défaut
`eps`	précision numérique relative, `eps=2.220446049250313e-016`
`inf`	l'infini mathématique ∞, *e.g.* `1.0/0.`
`nan`	littéralement *Not a Number*, *e.g.* `0/0`
`pi`	constante $\pi = 3.14159265358979$
`i` ou `j`	nombre complexe $\sqrt{-1}$
`nargin`	nombre d'arguments d'entrée d'une fonction
`nargout`	nombre d'arguments de sortie d'une fonction
`realmin`	plus petit réel utilisable
`realmax`	plus grand réel utilisable
`bitmax`	plus grand entier utilisable, stocké en double précision
`varargin`	nombre variable d'arguments d'entrée d'une fonction
`varargout`	nombre variable d'arguments de sortie d'une fonction

```
ans =
     1.7977e+308

>> pi=278, realmax=.2568
pi =
    278

realmax =
         0.2568
```

break case catch continue else elseif end for function
global if otherwise persistent return switch try while

TABLE 1.1 – Listes de mots réservés

Répertoire de travail

- Avant de commencer à travailler, il faut indiquer à Matlab le répertoire dans lequel on veut travailler.
- Sous Windows, le répertoire de travail par défaut s'appelle *works* et se trouve dans le répertoire d'installation de Matlab. Pour le changer il suffit d'entrer le nom de votre répertoire de travail dans la fenêtre *Current Directory* de la barre de menu.

Sauvegarde de l'environnement de travail

La fonction `save` permet de sauvegarder (par défaut en binaire) tout (par défaut) ou partie de l'espace de travail, qui se récupère grâce à la commande `load`. Par exemple si on veut sauvegarder les variables `X` et `Y` dans le fichier `FichXY` il suffit d'exécuter la ligne de commande

```
>> save FichXY X Y
```

Par défaut le fichier de sauvegarde est en binaire et comporte l'extension `.mat`.

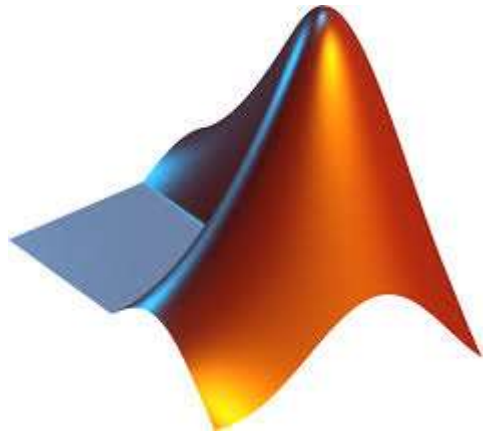
La fonction `load` est l'inverse de la fonction `save`. Par exemple, pour récupérer les variables `X` et `Y` sauvegardées dans le fichier `FichXY`, il suffit de faire

```
>> load FichXY X Y
```

La commande `diary` permet de sauvegarder l'intégralité d'une session de travail. Elle est très utile lorsqu'on tâtonne. On démarre la sauvegarde avec `diary on` et on arrête avec `diary off`. Dans tous les cas lorsqu'on tape `diary`, on passe d'un état à un autre, *i.e.* on passe de `diary on` à `diary off` ou *vice versa*. Il est possible d'utiliser la forme fonctionnelle suivante

```
>> diary('nomfich')
```

Alors toute la session sera sauvegardée dans le fichier `nomfich`.



Les nombres en Matlab avec licence ou Scilab

En Matlab :

• Toute variable est une matrice d'éléments d'un type donné.

• On ne définit pas de manière explicite le type d'une variable.

• Les principaux types de données utilisés par Matlab sont réel, entier, complexe, chaîne de caractères et logique.

• Le type logique a deux valeurs : vrai (representé par 1) et faux (representé par 0).

• Les fonctions suivantes permettent de déterminer le type d'une variable :

ischar(a) : retourne 1 si a est de type chaîne de caractères et 0 sinon.

islogical(a) : retourne 1 si a est de type logique et 0 sinon.

isreal(a) : retourne 1 si a est réel ou de type chaîne de caractères ou logique et 0 sinon (a est complexe à partie imaginaire non nulle ou n'est pas une matrice de valeurs réelles ou de caractères ou des éléments logiques).

2.1 Les nombres en Matlab

2.1.1 les entiers naturels

• Pour définir ou convertir un nombre en entier on utilise uint ou int pour les entiers naturel ou relatifs suivi du nombre de bits.

Exemple :

uint16 signifie un entier naturel codé sur 16 bits.

int32 signifie un entier relatif codé sur 32 bits.

• Matlab supporte les entiers jusqu'à 64 bits.

• Les fonctions d'arrondis :

round(x) entier le plus proche de x.

floor(x) arrondi par défaut de x.

ceil(x) arrondi par excés de x.

Chacune de ces trois fonctions retourne un resultat ayant le même type que le paramètre d'entrée.

Autres fonctions d'arithmétiques :

rem(a,b) : reste de la division entière de a par b.

lcm(a,b) : plus petit multiple commun de a et b.

gcd(a,b) : plus grand diviseur commun de a et b.

factor(a) : décomposition en facteurs premiers de a.

2.1.2 les nombres réels

Opérateurs arithmétiques

• Pour efféctuer les opérations arithmétiques, Matlab propose les opérateurs suivants :

∧	Puissance	.^	Puissance élément par élément
*	Multiplication	.*	Multiplication élément par élément
/	Division à droite (ordinaire)	./	Division à droite élément par élément
\	Division à gauche (left division)	.\	Division à gauche élément par élément
+	Addition	-	Soustraction

• Ces opérateurs suivent les règles usuèlles de priorité.

• Les parenthèses peuvent être utilisés pour indiquer la priorité.

Fonctions mathématiques

• Matlab propose aussi plusieurs fonctions prédéfinies pour le calcule arithmétique. Voici quelques exemples :

abs(x)	Valeur absolue$\|x\|$	**log(x)**	Logarithme naturel(ln)
sign(x)	Signe 1 ou -1	**log2(x)**	Logarithme en base 2
sqrt(x)	Racine carrée : $\sqrt{x}$	**log10(x)**	Logarithme en base 10
exp(x)	Puissance de e : e^x	**sin(x), cos(x), tan(x)**	Fonctions trigonométriques
exp(x,y)	$x.\hat{}y$	**asin(x), acos(x), atan(x)**	Fonctions trigonométriques

• Ces fonctions s'appliquent aux scalaires, aux vecteurs et aux matrices en agissant sur des éléments d'une manière independante.

• Elles s'appliquent aussi à des données de type complexe.

2.1.3 les nombres complexes

Le type complexe s'obtient en utilisant l'unité imaginaire `i` ou `j` (i.e. $\sqrt{-1}$).

```
>> z1=2+3i
z1 =
   2.0000 + 3.0000i
>> z2=1+pi*j
z2 =
   1.0000 + 3.1416i
```

On peut aussi écrire un nombre complexe sous sa forme polaire (*ie.* $e^{i\theta}$)

```
>> z=exp(i*pi/6)
z =
   0.8660 + 0.5000i
```

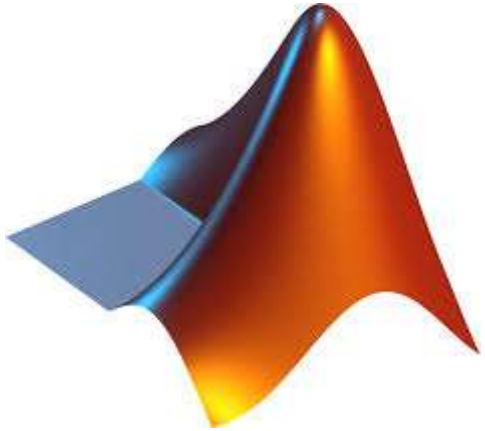
Les fonctions suivantes sont associées au type complexe:

`imag(z)` partie imaginaire de `z`;

`real(z)` partie réelle de `z`;

`abs(z)` module de `z`;

`angle(z)` argument de `z`.



Vecteurs et Matrices

Introduction

Comme son nom l'indique, MATLAB ne travaille qu'avec des matrices, au sens large. Ainsi, un scalaire est pour MATLAB une matrice 1×1 et un vecteur de taille n, une matrice $n \times 1$ ou $1 \times n$ suivant que le vecteur est en colonne ou en ligne. En plus des opérations usuelles sur les matrices (somme, soustraction, produit) MATLAB dispose d'opérateurs qui agissent élément par élément, permettant de se passer des boucles.

1. Création d'un vecteur ou d'une matrice

1.1. Vecteurs

Pour définir un vecteur (ligne ou colonne) il suffit de donner la liste de ses éléments entre crochets ([]). Pour un vecteur ligne, les éléments sont séparés au choix par un espace ou une virgule (,) .

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

>> v=[3,6,8,1.765]
v =
    3.0000    6.0000    8.0000    1.7650
```

Pour un vecteur colonne, les éléments sont séparés au choix par un point-virgule (;) ou un retour-chariot.

```
>> w=[1; 2; 3]
w =
     1
     2
     3

>> w=[3.56
      2.9
      6.3456
      4.5]
w =
```

Définir un vecteur ligne

- On définit un vecteur ligne en donnant la liste de ses éléments entre crochets ([ ]).

```
>> x=[1,5,9,5,6]

x =

     1     5     9     5     6
```

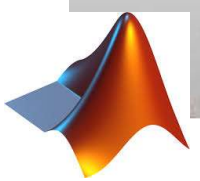
- Les éléments sont séparés au choix par des espaces ou par des virgules.
- On peut obtenir la longueur d'un vecteur donné grâce à la fonction length()

Définir un vecteur colonne

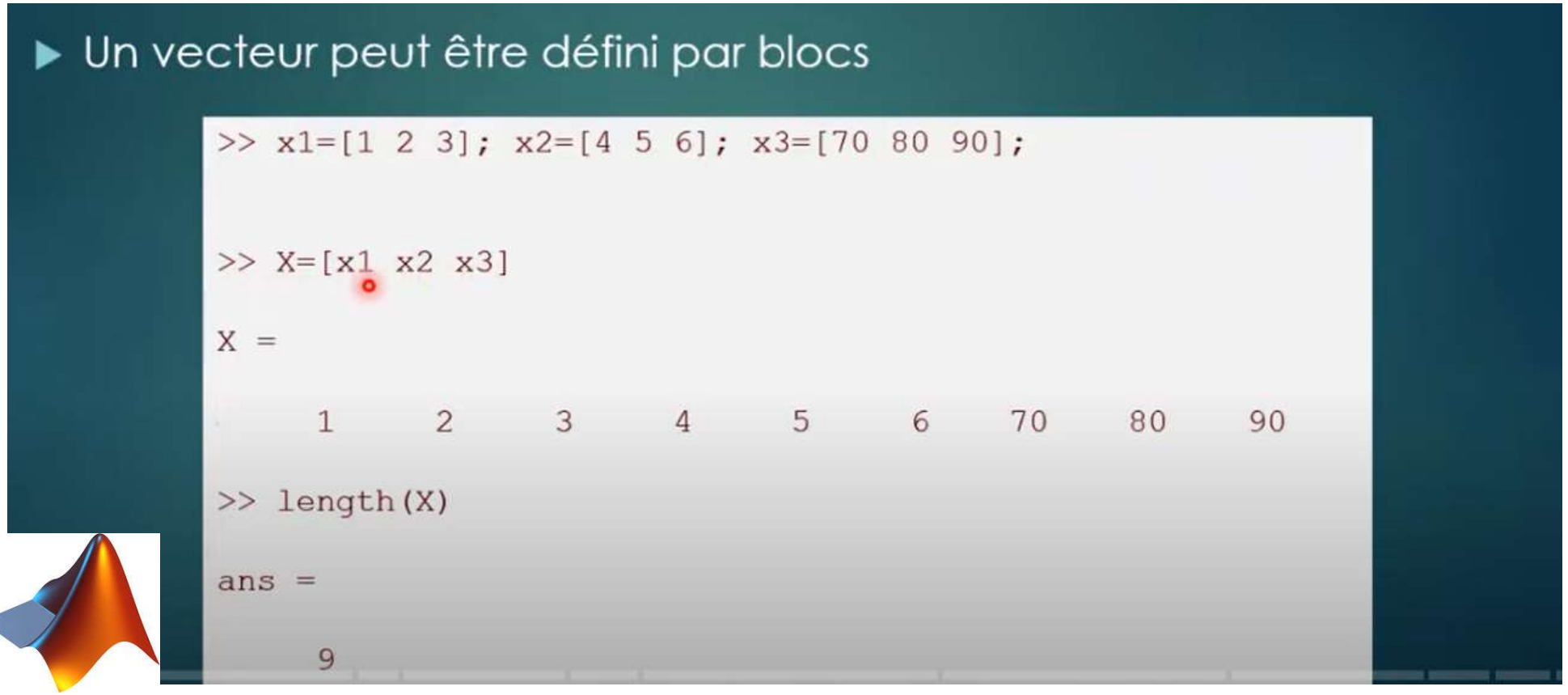
- On définit un vecteur colonne en donnant la liste de ses éléments séparés au choix par des points virgules (;) ou par des retours chariots (touche Entrée/Enter).
- On peut transformer un vecteur ligne x en un vecteur colonne et réciproquement en tapant x' (' est le symbole de transposition).

```
>> z=[9;6;8;7;9;5]
z =
     9
     6
     8
     7
     9
     5
```

```
>> w=z'

w =

     9     6     8     7     9     5
```



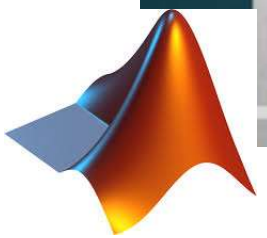
Définir un vecteur par blocs



Manipuler les éléments d'un vecteur

- Les éléments d'un vecteur peuvent être manipulés grâce à leur indice dans le tableau,
- Le $k^{ième}$ élément du vecteur x est désignée par x(k).
- Le premier élément d'un vecteur a obligatoirement pour indice 1.

```
>> x1=[1 2 3]; x2=[4 5 6]; x3=[70 80 90];
>> X=[x1 x2 x3];
>> X(8)
ans =
    80
>> X(0)
Subscript indices must either be real positive integers or logicals.
>> X(1)
ans =
     1
```



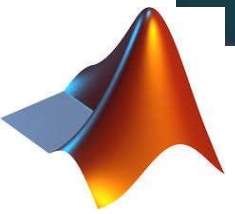
Manipuler les éléments d'un vecteur

- Il est possible de manipuler plusieurs éléments d'un vecteur simultanément.
  Ainsi les éléments de k à l du vecteur x sont désignés par **`x(k :l)`**.
- On peut également manipuler facilement les éléments d'un vecteur dont les indices sont en progression arithmétique. Ainsi si l'on souhaite extraire Les éléments k, k +p, k +2p, · · ·, k +Np = l on écrira x(k :p :l).
- Plus généralement, si K est un vecteur de valeurs entières, **`X(K)`** retourne les éléments du vecteur X dont les indices sont les élémenis du vecteur K.

Les vecteurs spécieux

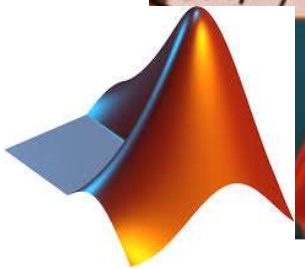
Les commandes **ones**, **zeros** et **rand** permettent de définir des vecteurs dont les éléments ont respectivement pour valeurs 0, 1 et des nombres générés de manière aléatoire.

- ones(1,n) : vecteur ligne de longueur n dont tous les éléments valent 1
- ones(m,1) : vecteur colonne de longueur m dont tous les éléments valent 1
- zeros(1,n) : vecteur ligne de longueur n dont tous les éléments valent 0
- zeros(m,1) : vecteur colonne de longueur m dont tous les éléments valent 0
- rand(1,n) : vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1
- rand(m,1) : vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1



Opérations et fonctions portant sur les vecteurs

x	**x(1)**	**x(2)**	…	**x(i)**	…	**x(n)**
y	**y(1)**	**y(2)**	…	**y(i)**	…	**y(n)**
K * x (k dans **R**)	K * x(1)	K * x(2)	…	K * x(i)	…	k * x(n)
U = sqrt(x)	U(1)= sqrt(x(1))	U(2)= sqrt(x(2))	…	U(i)= sqrt(x(i))	…	U(n)= sqrt(x(n))
x + y	x(1) + y(1)	x(2) + y(2)	…	x(i) + y(i)	…	x(n) + y(n)
x – y	x(1) - y(1)	x(2) - y(2)	…	x(i) - y(i)	…	x(n) - y(n)
x .* y	x(1) * y(1)	x(2) * y(2)	…	x(i) * y(i)	…	x(n) * y(n)
x ./ y	x(1) / y(1)	x(2) / y(2)	…	x(i) / y(i)	…	x(n) / y(n)

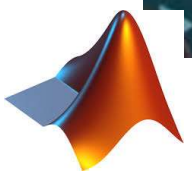


Attention :

- Il faut que les vecteurs aient la même dimension.
- Ne pas oublier le point pour la multiplication et la division

Opérations et fonctions portant sur les vecteurs

- `cross(x,y)` : calcule le produit vectoriel des deux vecteurs x et y.
- `sum(x.*y)` : calcule le produit scalaire des deux vecteurs x et y.
- `sum(x)` : somme des éléments du vecteur x,
- `prod(x)` : produit des éléments du vecteur x,
- `max(x)` : plus grand élément du vecteur x,
- `min(x)` : plus petit élément du vecteur x,
- `mean(x)` : moyenne des éléments du vecteur x,
- `sort(x)` : ordonne les éléments du vecteur x par ordre croissant,
- `fliplr(x)` : renverse l'ordre des éléments du vecteur x.

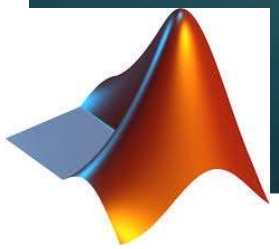


Les Matrices

Définir une matrice

- D'une façon générale, on définit une matrice en donnant la liste de ses éléments entre crochets.
- Les éléments d'une ligne de la matrice peuvent être séparés au choix par un blanc ou bien par une virgule (,).
- Les lignes quant à elles peuvent être séparées au choix par le point-virgule (;) ou par un retour chariot.

```
>> x=[1,5;9,6]

x =

     1     5
     9     6
```



Manipuler les éléments d'une matrice

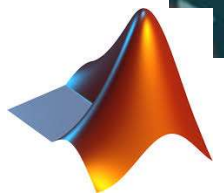
- Un élément d'une matrice est référencé par ses numéros de ligne et de colonne. A(i,j) désigne le **j^{ème}** élément de la ligne **i** de la matrice A.
- Ainsi A(2,1) désigne le premier élément de la deuxième ligne de A,

```
>> B=[1,3;4,2];
>> B(2,1)

ans =

     4
```

Les matrices spéciales

Certaines matrices se construisent très simplement grâce à des commandes dédiées. Citons les plus utilisées :

- **`eye(n)`** : la matrice identité dans $R^{n,n}$
- **`ones(m,n)`** : la matrice à m lignes et n colonnes dont tous les éléments valent 1
- **`zeros(m,n)`** : la matrice à m lignes et n colonnes dont tous les éléments valent 0
- **`rand(m,n)`** : une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1
- **`magic(n)`** qui permet d'obtenir une matrice magique de dimension n



Extraction des éléments d'une matrice

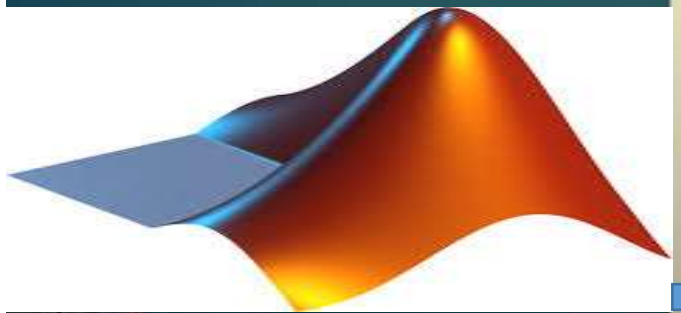
- Le symbole deux-points (:) permet d'extraire simplement des lignes ou des colonnes d'une matrice. Le $j^{ième}$ vecteur colonne de la matrice A est désigné par `A( :,j)`.
- il suffit de traduire le symbole deux-points (:) par « tout ». Ainsi `A( :,j)` désigne toutes les lignes et la $j^{ième}$ colonne de la matrice A.
- Bien entendu, la $i^{ième}$ ligne de la matrice A est désignée par `A(i,:)`

```
>> A=magic(5)
A =
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9
>> A(:,3)
ans =
    1
    7
   13
   19
   25
>> A(4,:)
ans =
   10   12   19   21    3
```

Manipuler les Matrices

- Que c'est-il passé sur la matrice A

```
>> A=magic(5)
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>> v=A(:,2); A(:,2)=A(:,3); A(:,3)=v;
>> A
A =
    17     1    24     8    15
    23     7     5    14    16
     4    13     6    20    22
    10    19    12    21     3
    11    25    18     2     9
```

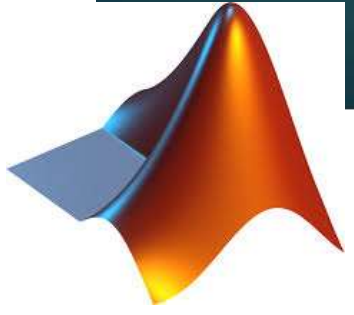


Extraction des éléments d'une matrice

On peut également extraire plusieurs lignes ou colonnes simultanément.

- Si J est un vecteur d'entiers, **`A(:,J)`** est la matrice issue de A dont les colonnes sont les colonnes de la matrice A d'indices contenus dans le vecteur J.
- De même **`A(J,:)`** est la matrice issue de A dont les lignes sont les lignes de la matrice A d'indices contenus dans le vecteur J.

```
A =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
>> L = [1 3 5]; C = [3 4];
>> A(L,C)
ans =
     1     8
    13    20
    25     2
>> A(1:2:5,3:4)
ans =
     1     8
    13    20
    25     2
```



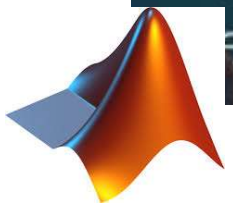
Extraction des éléments d'une matrice

Il existe des commandes matlab permettant de manipuler globalement des matrices.

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

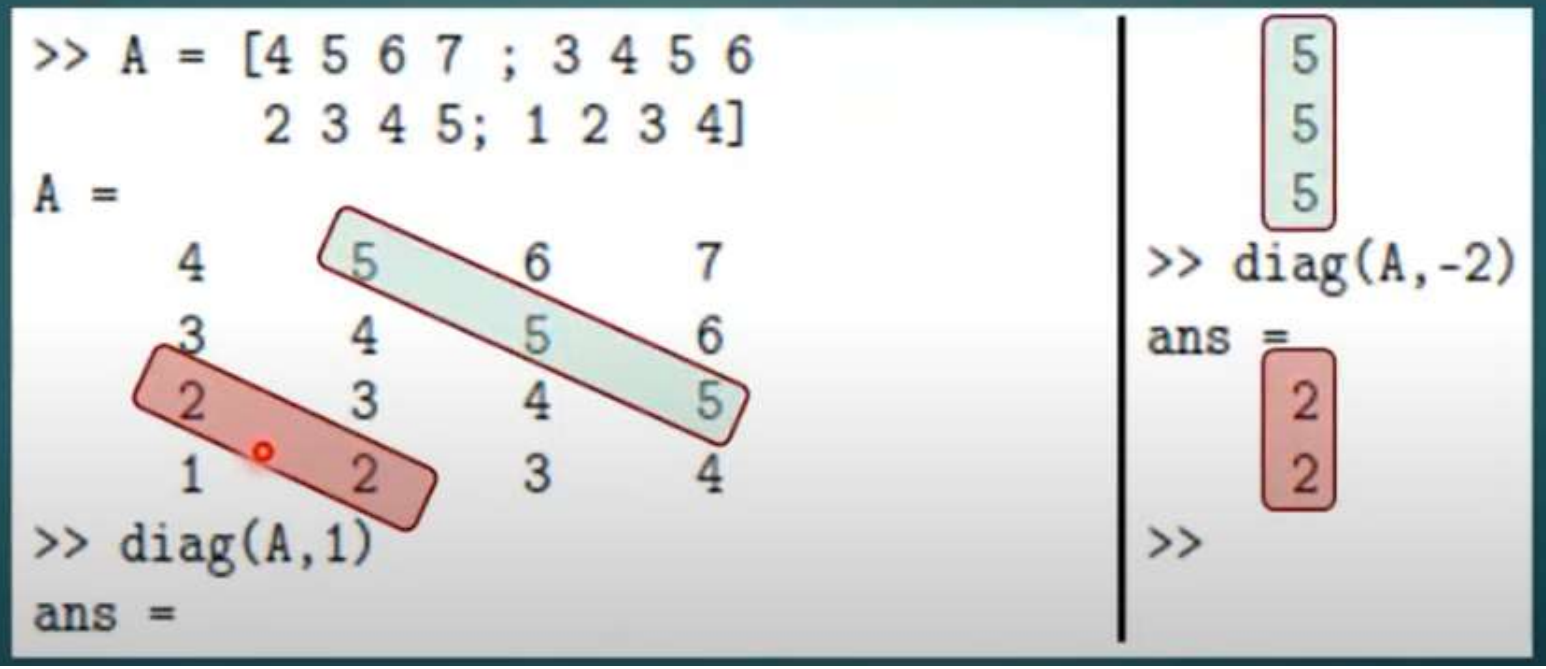
```
>> A=eye(3); diag(A)
ans =
     1
     1
     1
>> v=[1:3]
v =
     1 2 3
```

```
>> diag(v)
 ans =
     1 0 0
     0 2 0
     0 0 3
>>
```



Extraction des éléments d'une matrice

- La commande **diag** admet un second paramètre k pour désigner la $k^{ième}$ sur-diagonale (si k > 0) ou la $k^{ième}$ sous-diagonale (si k < 0).

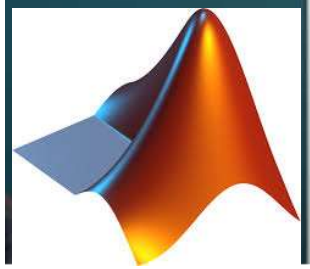


Extraction des éléments d'une matrice

- On dispose également de la commande **tril** permet d'obtenir la partie triangulaire inférieure (l pour lower) d'une matrice. La commande **triu** permet d'obtenir la partie triangulaire supérieure (u pour upper) d'une matrice.

```
>> A = [ 2 1 1 ; -1 2 1 ; -1 -1 2]
A =
     2     1     1
    -1     2     1
    -1    -1     2
>> triu(A)
ans =
     2     1     1
     0     2     1
     0     0     2
```

```
>> tril(A)
ans =
     2     0     0
    -1     2     0
    -1    -1     2
>>
```

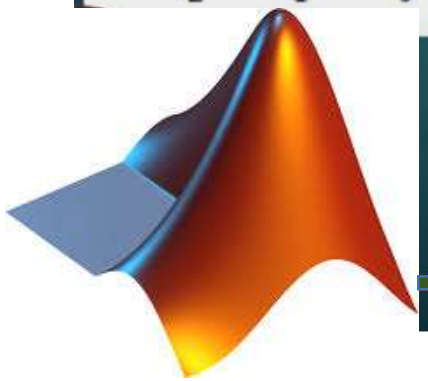


Transposée d'une matrice

- On obtient la transposée de la matrice A à coefficients réels en tapant A'. Si la matrice est à coefficients complexes, A' retourne la matrice adjointe de A.

```
>> A = [0 1 2; -1 0 1; -2 -1 0]
A =
     0     1     2
    -1     0     1
    -2    -1     0
```

```
>> A'
ans =
     0    -1    -2
     1     0    -1
     2     1     0
```



```
>> C=[0 6-6i 8+3i ; 5+9i 0 9+7i; 8-8i 6+9i 0]

C =

   0.0000 + 0.0000i   6.0000 - 6.0000i   8.0000 + 3.0000i
   5.0000 + 9.0000i   0.0000 + 0.0000i   9.0000 + 7.0000i
   8.0000 - 8.0000i   6.0000 + 9.0000i   0.0000 + 0.0000i

>> C'

ans =

   0.0000 + 0.0000i   5.0000 - 9.0000i   8.0000 + 8.0000i
   6.0000 + 6.0000i   0.0000 + 0.0000i   6.0000 - 9.0000i
   8.0000 - 3.0000i   9.0000 - 7.0000i   0.0000 + 0.0000i
```

Exercice 01

- Ecrire un script qui déclare la variable R contenant la valeur 20. Déclarer 3 variables D, P et S et affecter respectivement à ces variables les valeurs du diamètre, du périmètre et de la surface d'un cercle dont le rayon est R. On affichera à l'écran le contenu de ces différentes variables selon le format suivant :

```
Un cercle de rayon WW a pour diamètre XX, pour
circonférence YY et pour surface ZZ.
```

Exercice 02

- La fonction rand(n) renvoie une matrice carrée d'ordre n dont les éléments sont des nombres aléatoires compris entre 0 et 1. Ecrire un code MATLAB pour générer une matrice carrée avec une distribution uniforme de nombres aléatoires dans un intervalle spécifique [a,b].

- Pour se faire, multiplier la sortie de la fonction rand() par (b-a) puis ajouter a

La structure Sparse

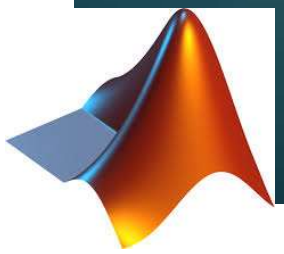
- On appelle matrice creuse (anglais « sparse matrix ») une matrice comportant une forte proportion de coefficients nuls.
- Une structure creuse permet la réduction de la place mémoire (on ne stocke pas les zéros), en plus de la réduction du nombre d'opérations (on n'effectuera pas les opérations portant sur les zéros).
- Par défaut dans matlab une matrice est considérée comme pleine (ou « full» en anglais), c'est-à-dire que tous ses coefficients sont mémorisés.
- La commande **`sparse(M)`** permet d'obtenir la matrice M stockée sous la forme sparse.
- La commande **`full(M)`** permet de rendre la matrice M stockée sous la forme ordinaire.

Opérations et fonctions portant sur les matrices

- Si les opérandes sont des matrices, les opérations + (addition), - (soustraction), * (multiplication), ^ (exponentiation), sont alors les opérations matricielles usuelles. Ainsi A*B désigne le produit de la matrice A par la matrice B, A+B désigne la somme de ces deux matrices et A^2 le carré de la matrice A

```
>> A=[1 2 3; 4 5 6]
A =
     1 2 3
     4 5 6
>> B = [1 1; 2 2; 3 3]
B =
     1 1
     2 2
     3 3
```

```
>> C = A*B
C =
     14 14
     32 32
>> C^2
ans =
     644  644
    1472 1472
>>
```



Opérations et fonctions portant sur les matrices

- Si les dimensions des matrices A et B sont incompatibles avec l'opération matricielle, matlab renvoi un message d'erreur :

```
>> A+B

??? Error using ==> + Matrix dimensions must agree.
>>
```

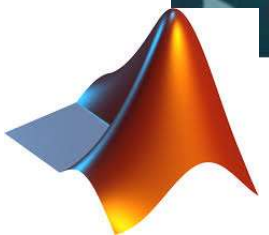
Opérations et fonctions terme à terme sur les matrices

- il est possible d'effectuer des opérations entre matrices «élément par élément». Pour cela, il faut faire précéder l'opérateur d'un point (.).
- Ainsi si A et B sont 2 matrices de même dimension, on obtient la matrice dont le terme d'indices (i, j) est le produit des deux termes d'indices (i, j) des matrices A et B par la commande A.*B.
- De même la commande A.^2 fournit la matrice dont les termes sont les carrés des termes de la matrice A.
- les commandes A.+B et A+B donnent le même résultat.

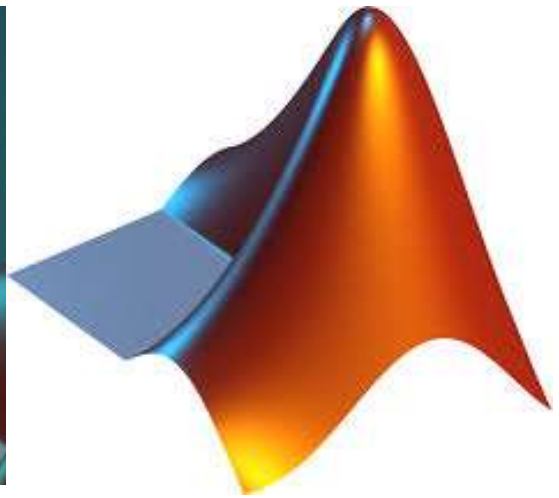
```
>> A=[1 2 3; 4 5 6]
A =
    1 2 3
    4 5 6
>> B=[1 2 3; 1 2 3]
B =
    1 2 3
    1 2 3
>> A.*B
ans =
    1 4 9
    4 10 18
>> A.^2
ans =
     1     4     9
    16    25    36
>>
```

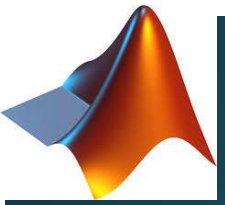
Les fonctions matricielles courantes

- **`det(A)`** : renvoie le déterminant de la matrice carrée A.
- **`eig(A)`** : renvoie les valeurs propres (eigenvalues) de la matrice carrée A. Si l'on souhaite également les vecteurs propres on exécutera **`[V,D] = eig(A)`** qui renvoie une matrice diagonale D formée des valeurs propres de A et une matrice V dont les vecteurs colonnes sont les vecteurs propres correspondant.
- **`poly(A)`** : renvoie les coefficients du polynôme caractéristique associé à la matrice carrée A;
- **`inv(A)`** : renvoie l'inverse de la matrice carrée A.
- **`rank(A)`** : renvoie le rang de la matrice carrée A.
- **`trace(A)`** : renvoie la trace de la matrice A.
- **`expm(A)`** : renvoie l'exponentielle matricielle de A.



Programmer sous Matlab





Scripts et fonctions

- Pour des calculs complexes et répétitifs, il est préférable (ou plutôt indispensable) de rassembler l'ensemble des commandes dans un fichier qui constituera le programme à exécuter.
- Il est possible d'enregistrer une séquence d'instructions dans un fichier (appelé un «M-file ») et de les faire exécuter par Matlab.
- Un tel fichier doit obligatoirement avoir une extension de la forme **.m** (d'où le nom M-file) pour être considéré par Matlab comme un fichier d'instructions.
- On distingue 2 types de M-file, **les fichiers de scripts** et **les fichiers de fonctions**.

Scripts et fonctions

Editer un fichier .m

- Bien que l'environnement de Matlab propose son propre éditeur (fenêtre Editor), ces fichiers sont de simples fichiers textes avec une extension .m. Vous pouvez donc utiliser votre éditeur de texte préféré pour créer vos programmes (sans oublier de modifier l'extension).
- A partir de Matlab, un m-file est créé ou ouvert, soit depuis le menu Fichier (New > M-File), soit depuis l'invite en tapant :

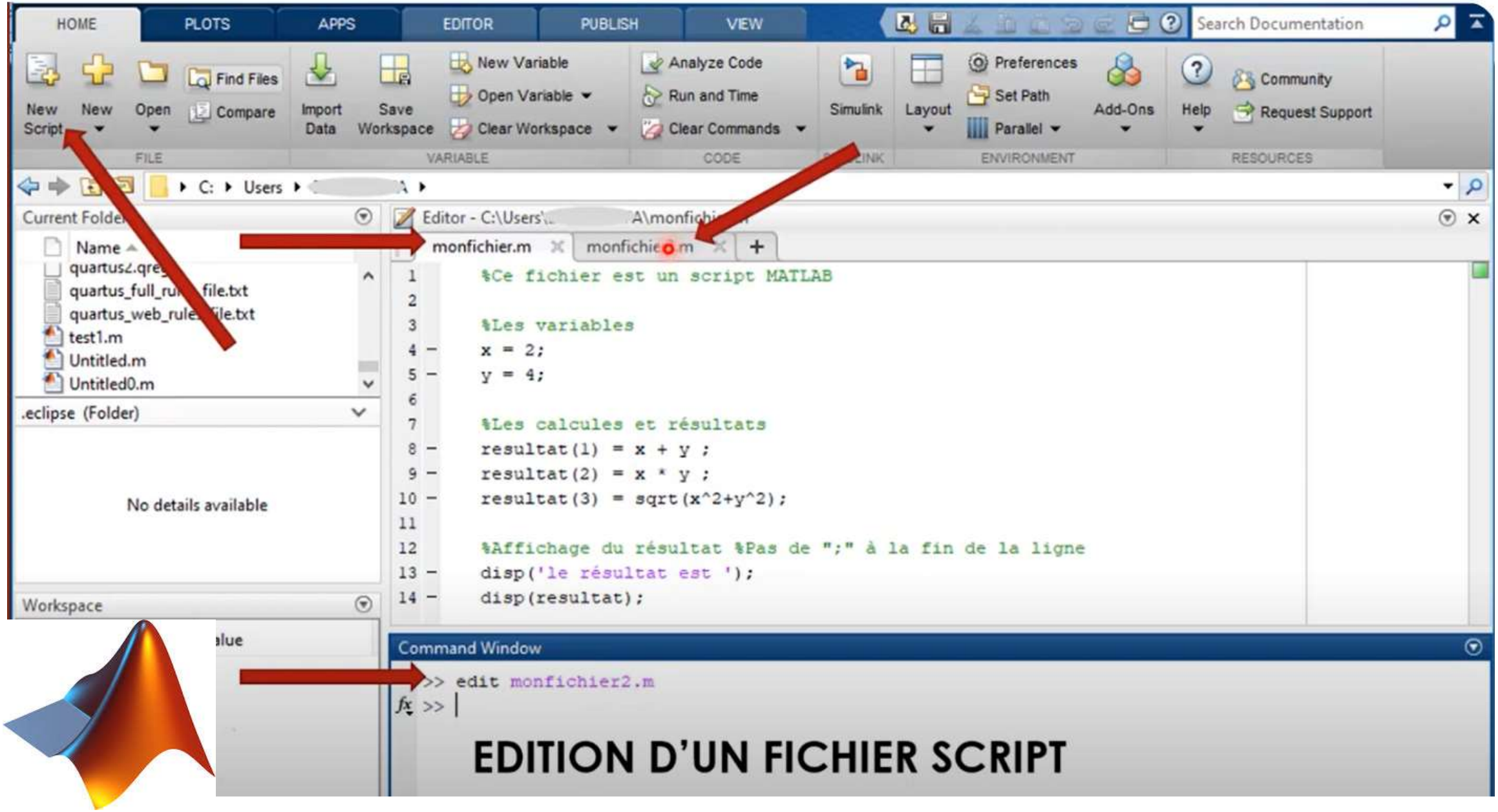
```
>> edit monfichier.m
```

- Un **m-file** est reconnu, et donc exécutable, s'il se trouve dans le répertoire courant (current directory) ou si le répertoire contenant est spécifié dans le **PATH**.

Fichier script

- Un fichier script permet de regrouper des séries de commandes Matlab.
- A son lancement, les instructions qu'il contient s'exécutent séquentiellement comme si elles étaient lancées depuis l'invite de commande.

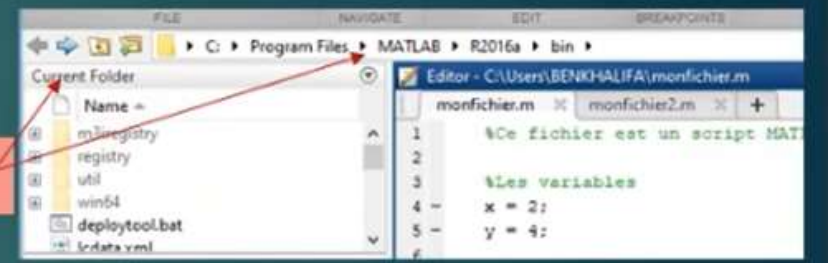
- Un script stocke ses variables dans le workspace, lequel est partagé par tous les scripts. Ainsi, toutes les variables créées dans les scripts sont visibles depuis la command window et vice versa.
- Lorsque Matlab détecte une erreur, le programme s'arrête et un message d'erreur s'affiche à l'écran (avec le numéro de la ligne où l'erreur est détectée).


HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Simulink
Layout
Preferences
Set Path
Parallel
Add-Ons
Help
Community
Request Support
FILE
VARIABLE
CODE
ENVIRONMENT
RESOURCES
C: ▸ Users ▸
Current Folder
Name
quartus_full_rule_file.txt
quartus_web_rule_file.txt
test1.m
Untitled.m
Untitled0.m
.eclipse (Folder)
No details available
Workspace
Editor - C:\Users\...\monfichier.m
monfichier.m
monfichier2.m
1 %Ce fichier est un script MATLAB
2
3 %Les variables
4 x = 2;
5 y = 4;
6
7 %Les calcules et résultats
8 resultat(1) = x + y ;
9 resultat(2) = x * y ;
10 resultat(3) = sqrt(x^2+y^2);
11
12 %Affichage du résultat %Pas de ";" à la fin de la ligne
13 disp('le résultat est ');
14 disp(resultat);
Command Window
>> edit monfichier2.m
fx >>
EDITION D'UN FICHIER SCRIPT

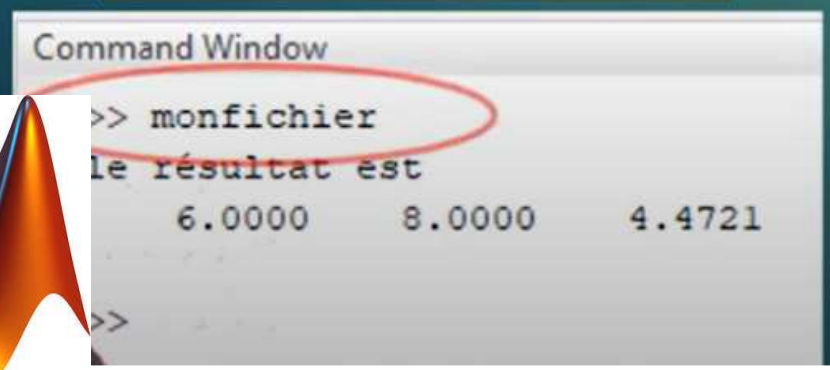
Exécuter un fichier script

Pour exécuter un script !

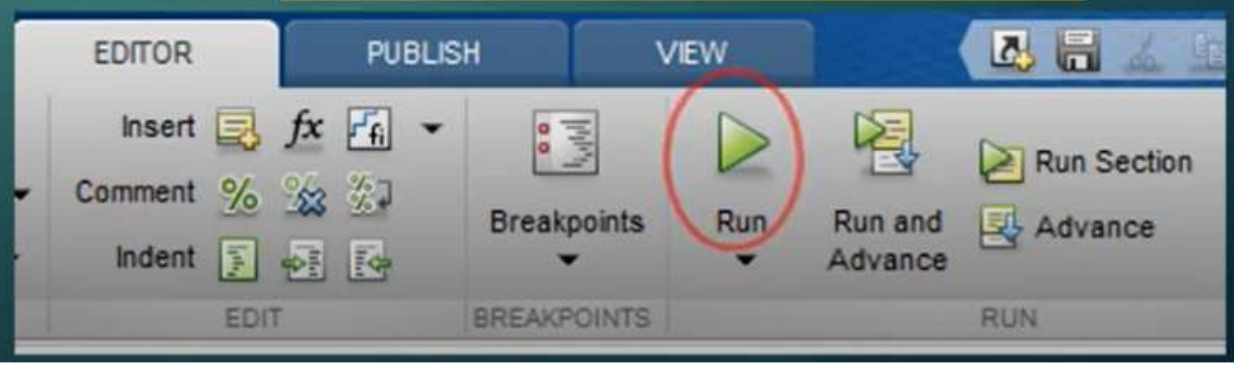
Il faut que le fichier soit enregistré dans le répertoire actif



Commande window



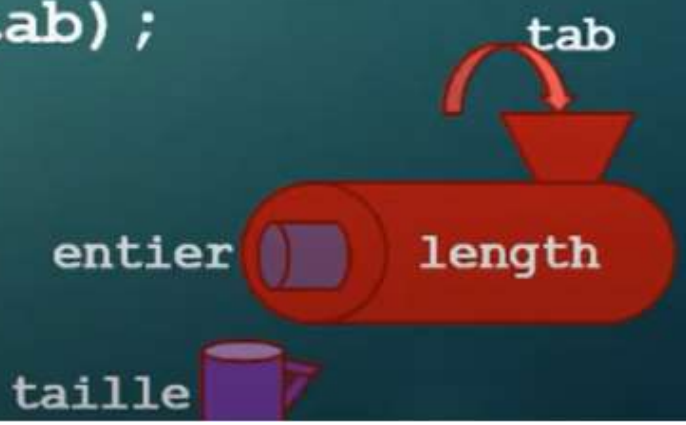
Menu EDITOR, commande Run



Fichier fonction

- Le principe d'une fonction est d'effectuer des opérations à partir d'une ou plusieurs entrées et fournir une ou plusieurs sorties (résultat).
- Les variables d'entrées sont des paramètres à spécifier en argument de la fonction, tandis que les variables de sorties sont des valeurs quelle renvoie.
- Un m-file fonction est tout à fait semblable aux fonctions intégrées de Matlab. Par exemple, la fonction **`length()`** renvoie la taille du tableau entré en argument.

```
>> taille = length(tab);
```

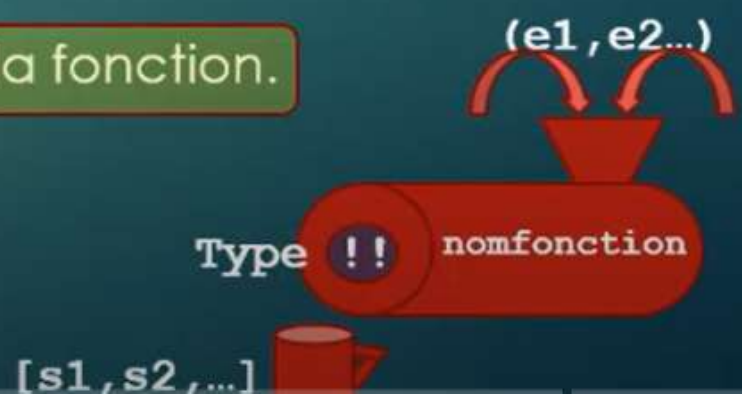


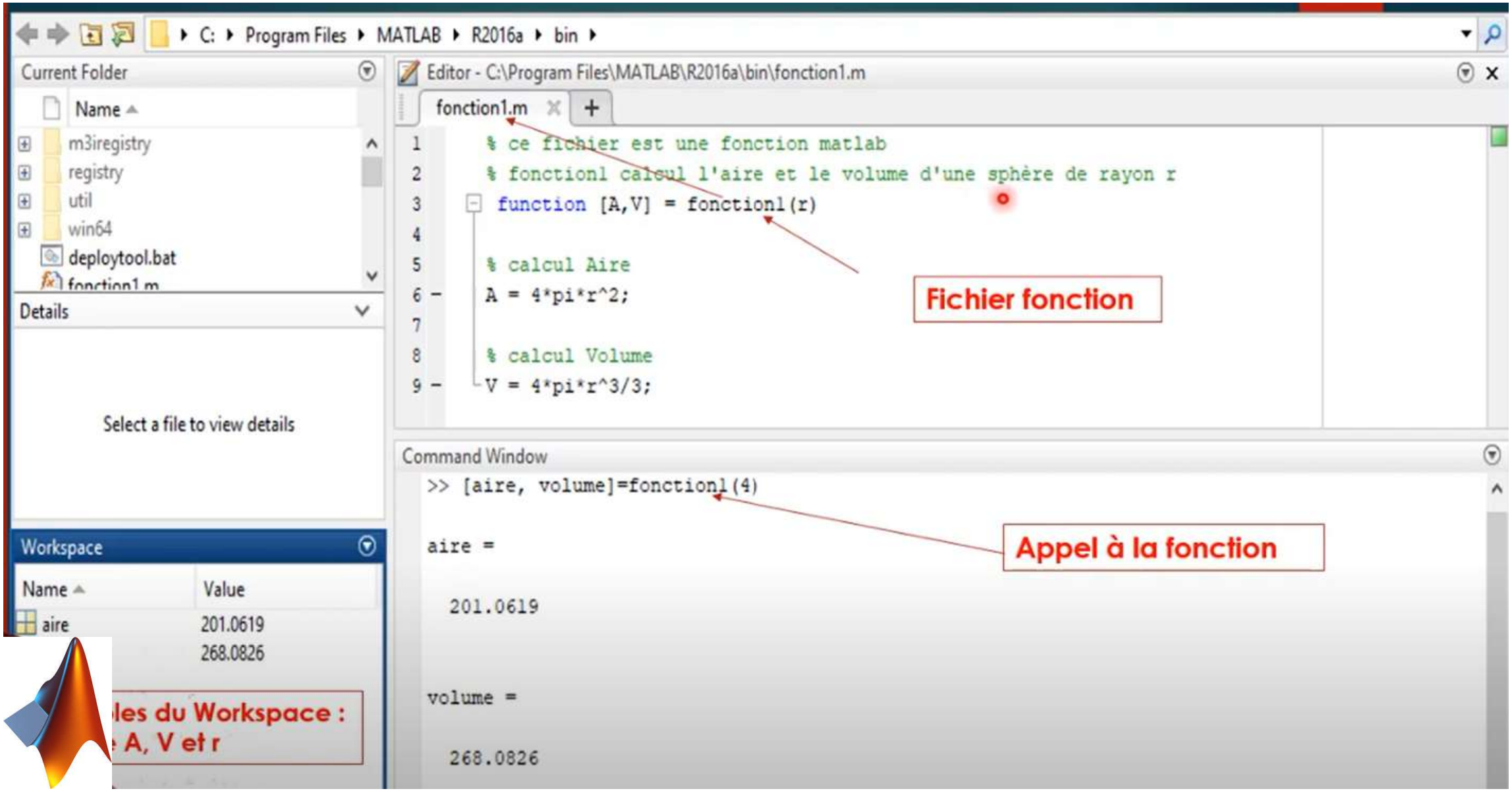
Créer un fichier fonction

- Un m-file est définit comme une fonction en plaçant en tête du fichier le mot clé **`function`** suivit de son prototype.
- Un prototype est de la forme

```
[s1,s2,…] = nomfonction(e1,e2,…)
```

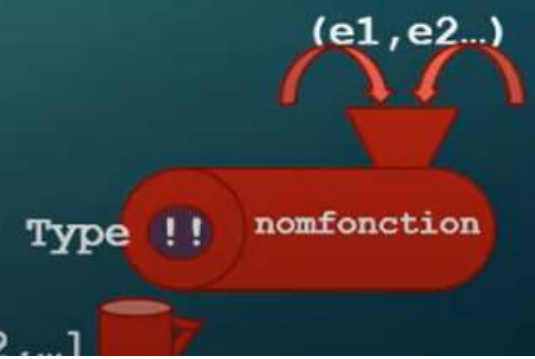
- Le membre de gauche regroupe les sorties renvoyées par la fonction et les variables entres parenthèses sont les entrées.
- Le nom du fichier doit être identique au nom de la fonction.

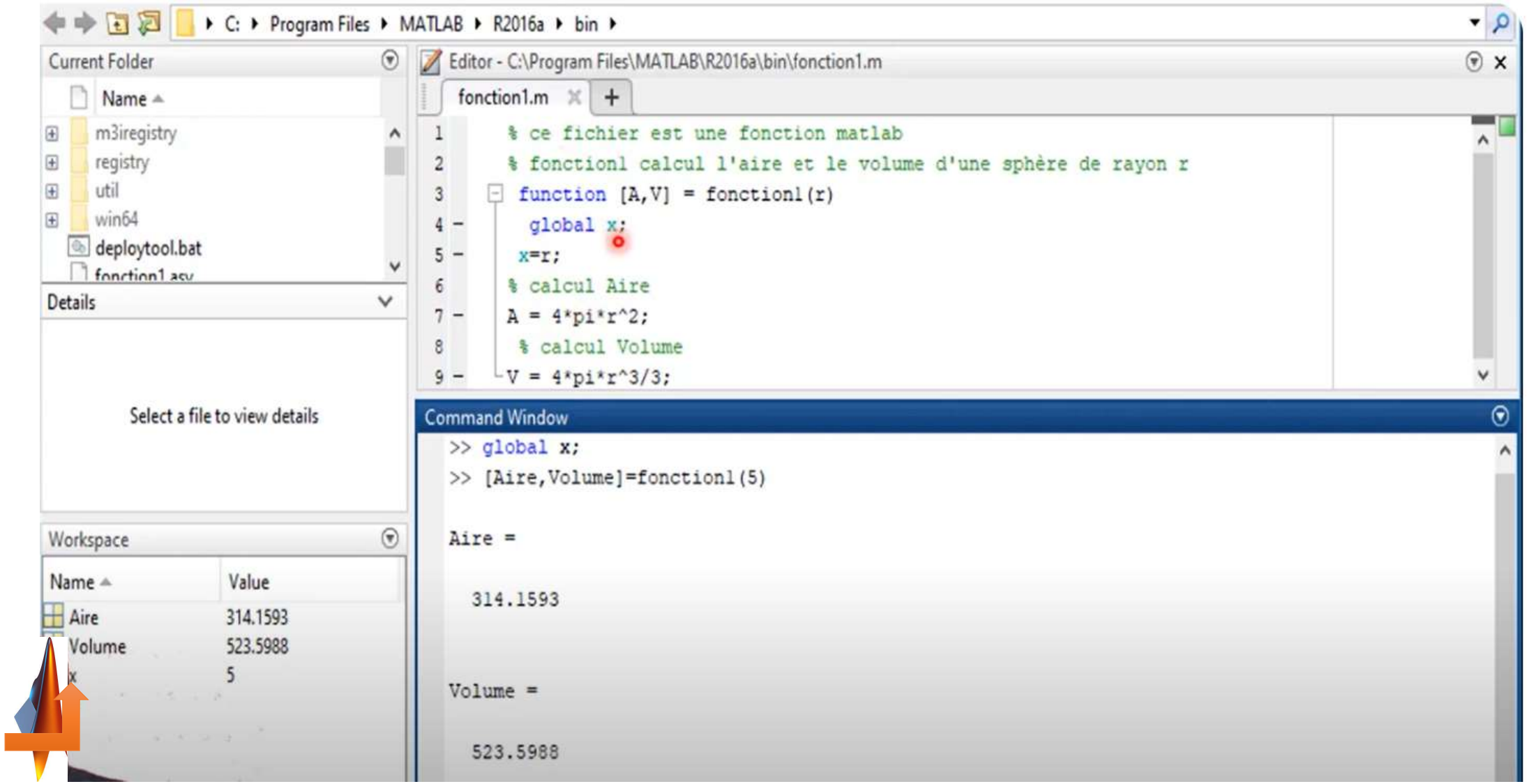



C: ▸ Program Files ▸ MATLAB ▸ R2016a ▸ bin ▸
Current Folder
Name
m3iregistry
registry
util
win64
deploytool.bat
fonction1.m
Details
Select a file to view details
Workspace
Name
Value
aire
201.0619
268.0826
Editor - C:\Program Files\MATLAB\R2016a\bin\fonction1.m
fonction1.m
1 % ce fichier est une fonction matlab
2 % fonction1 calcul l'aire et le volume d'une sphère de rayon r
3 function [A,V] = fonction1(r)
4
5 % calcul Aire
6 A = 4*pi*r^2;
7
8 % calcul Volume
9 V = 4*pi*r^3/3;
Fichier fonction
Command Window
>> [aire, volume]=fonction1(4)
aire =
201.0619
volume =
268.0826
Appel à la fonction
les du Workspace :
A, V et r

Les variables d'une fonction

- Le point fondamental qui différencie une fonction d'un script est le fait que les variables internes soient locales, c'est-à-dire que les variables définies dans une fonction n'existent que dans celle-ci.
- De plus, les variables du workspace ne sont pas visibles depuis une fonction.
- Pour pouvoir utiliser une variable partagée par le workspace et une (voire des) fonction(s), celle-ci doit être déclarée comme global à la fois dans la command window et dans la (les) fonction(s).




C: ▸ Program Files ▸ MATLAB ▸ R2016a ▸ bin ▸
Current Folder
Name
m3iregistry
registry
util
win64
deploytool.bat
fonction1.asv
Details
Select a file to view details
Workspace
Name
Value
Aire
314.1593
Volume
523.5988
x
5
Editor - C:\Program Files\MATLAB\R2016a\bin\fonction1.m
fonction1.m
1 % ce fichier est une fonction matlab
2 % fonction1 calcul l'aire et le volume d'une sphère de rayon r
3 function [A,V] = fonction1(r)
4 global x;
5 x=r;
6 % calcul Aire
7 A = 4*pi*r^2;
8 % calcul Volume
9 V = 4*pi*r^3/3;
Command Window
>> global x;
>> [Aire,Volume]=fonction1(5)
Aire =
314.1593
Volume =
523.5988

Opérateurs relationnels et logiques

- Matlab possède des opérateurs qui permettent d'établir des expressions renvoyant en résultat une valeur logique, c'est-à-dire 0 ou 1.
- Ces expressions logiques sont généralement utilisées dans les structures de contrôle (if, while, switch, for).

Opérateurs relationnels		Opérateurs logiques	
==	Égale à	&	Et
~=	Différent de	|	Ou
<	Strictement inférieur à	~	Non
<=	Inférieur ou égal à	xor	Ou exclusif
>	Strictement supérieur à	any(x)	retourne 1 si un des éléments de x est non nul
>=	Supérieur ou égale à	all(x)	retourne 1 si tous les éléments de x sont nuls

- Le résultat vaut 1 si le test est vrai et 0 s'il est faux.

Opérateurs relationnels et logiques

```
>> 10 > 9
ans =
     1

>> 2 == 3
ans =
     0

>> 4 ~= 7
ans =
     1

>> [1 4 ; 7 3] <= [0 6 ; 7 2]
ans =

     0     1
     1     0
```

```
>> x = [0 1 0 1];
>> y = [0 0 1 1];
>> x & y
ans =
     0     0     0     1

>> x | y
ans =
     0     1     1     1

>> ~x
ans =
     1     0     1     0

>> xor(x,y)
ans =
     0     1     1     0

>> 0 | 3
ans =
     1
```

Structures de contrôle

Branchement conditionnel (`if … elseif … else`)

Cette structure permet d'exécuter un bloc d'instructions en fonction de la valeur logique d'une expression.

```
if expression
instructions …
end
```

- L'ensemble des instructions '*instructions*' est exécuté seulement si *expression est vraie*.
- Plusieurs tests exclusifs peuvent être combinés.

Plusieurs **`elseif`** peuvent être concaténés. Leur bloc est exécuté si l'expression correspondante est vraie et si toutes les conditions précédentes n'ont pas été satisfaites. Le bloc ***instruction3*** *associé* au **`else`** est quant à lui exécuté si aucune des conditions précédentes n'a été réalisées.

```
if expression1
 instructions1 …
elseif expression2
 instructions2 …
else
 instructions3 …
end
```

C: ▸ Program Files ▸ MATLAB ▸ R2016a ▸ bin ▸

Current Folder

Name

- m3iregistry
- registry
- util
- win64
- deploytool.bat
- fonction1.m

Details

Select a file to view details

Workspace

Name	Value
x	1

Editor - C:\Program Files\MATLAB\R2016a\bin\ifstatement.m

ifstatement.m

```
1 -    x=0;
2 -    x=input('donnez une valeur pour X :');
3 -    if x > 0
4 -        disp ('X est positif');
5 -    elseif x == 0
6 -        disp('X est null');
7 -    else x= 1;
8 -    end
```

Command Window

```
>> ifstatement
donnez une valeur pour X :25
X est positif
>> ifstatement
donnez une valeur pour X :0
X est null
>> ifstatement
donnez une valeur pour X :-8
fx >>
```

Structures de contrôle

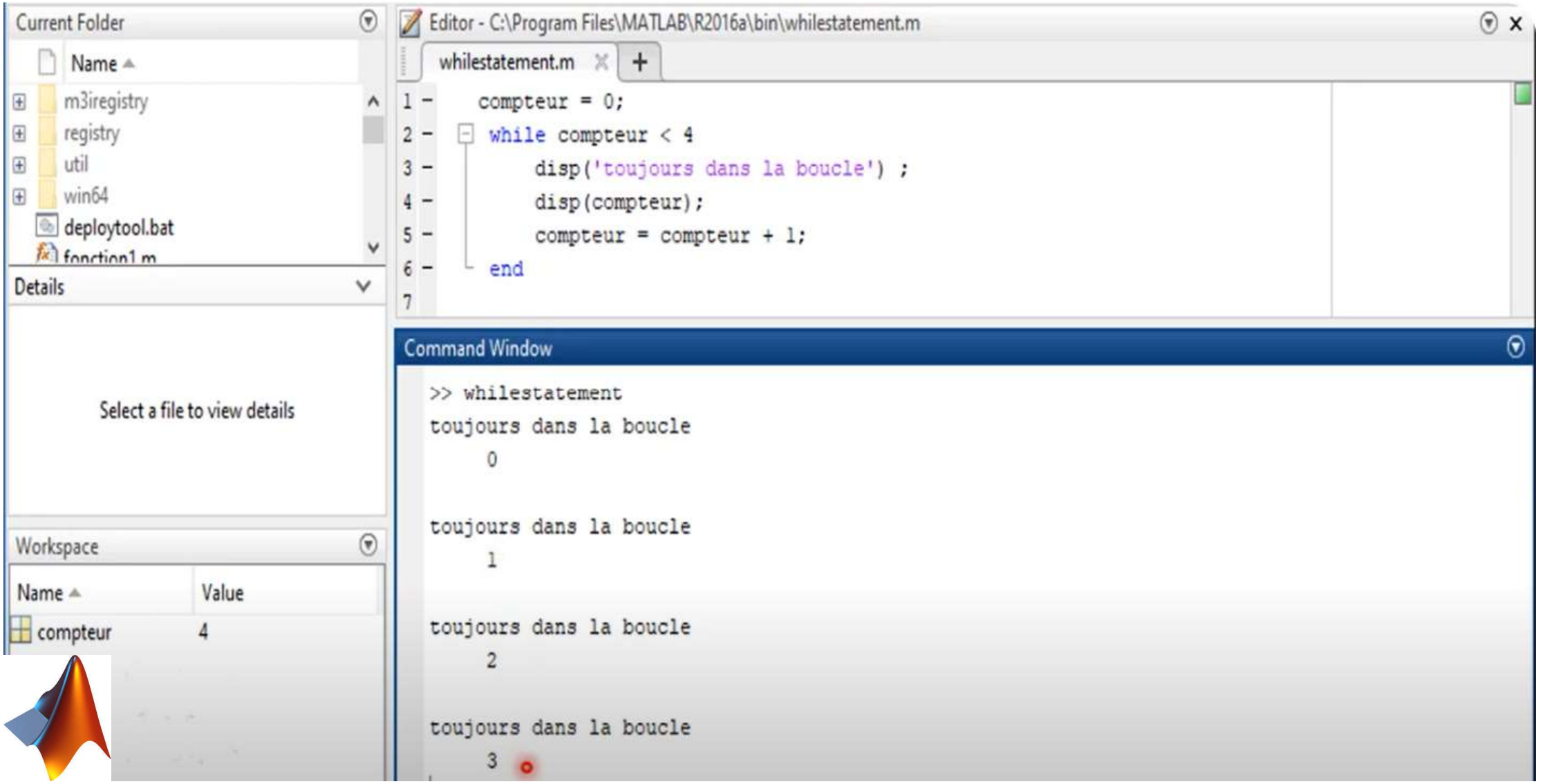
Boucle conditionnel (while…)

Ce mécanisme permet de répéter une série d'instructions tant qu'une condition est vérifiée.

```
while expression
      instructions …
  end
```

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

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


Current Folder
Name
m3iregistry
registry
util
win64
deploytool.bat
fonction1.m
Details
Select a file to view details
Workspace
Name
Value
compteur
4
Editor - C:\Program Files\MATLAB\R2016a\bin\whilestatement.m
whilestatement.m
1 - compteur = 0;
2 - while compteur < 4
3 - disp('toujours dans la boucle') ;
4 - disp(compteur);
5 - compteur = compteur + 1;
6 - end
7
Command Window
>> whilestatement
toujours dans la boucle
0
toujours dans la boucle
1
toujours dans la boucle
2
toujours dans la boucle
3

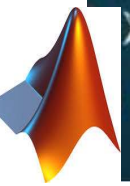
Structures de contrôle

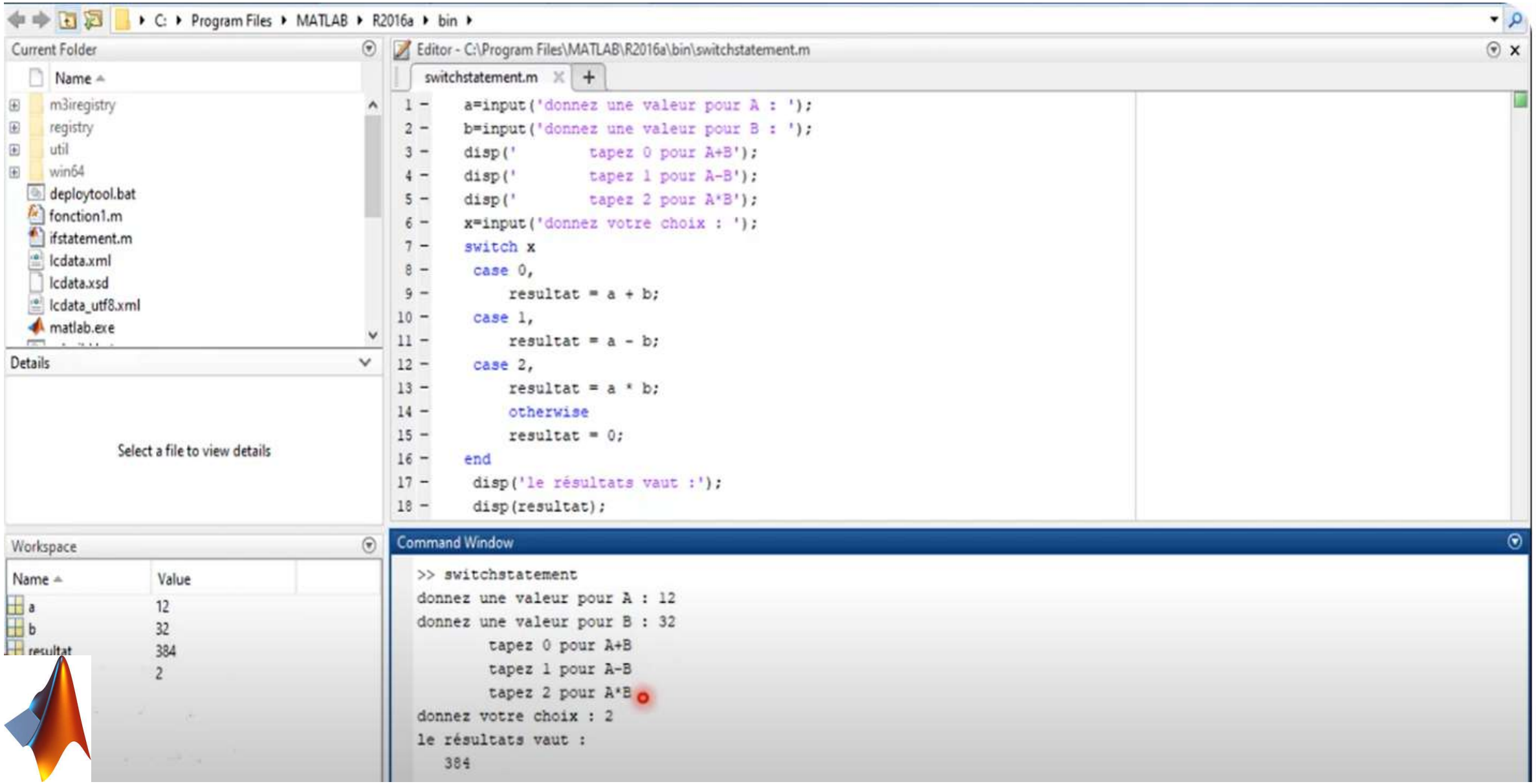
Branchements multiples (`switch ... case`)

- Dans cette structure, une expression numérique est comparée successivement à différentes valeurs. Dès qu'il y a identité, le bloc d'instructions correspondant est exécuté.

```
switch expression
case valeur1,
     instructions1 …
case valeur2,
     instructions2 …
case valeur3,
     instructions3 …
 ……
 otherwise
      instructions …
 end
```

- L'expression testée, ***expression***, *doit être un scalaire ou une chaîne de caractère.*
- *Une fois qu'un* bloc ***instructions(i)*** *est exécuté, le flux d'exécution sort de la structure et reprend après le end.*
- *Si* aucun case ne vérifie l'égalité, le bloc qui suit **otherwise** est exécuté.




C: ▸ Program Files ▸ MATLAB ▸ R2016a ▸ bin ▸
Current Folder
Name
m3iregistry
registry
util
win64
deploytool.bat
fonction1.m
ifstatement.m
lcdata.xml
lcdata.xsd
lcdata_utf8.xml
matlab.exe
Details
Select a file to view details
Workspace
Name
Value
a
12
b
32
resultat
384
2
Editor - C:\Program Files\MATLAB\R2016a\bin\switchstatement.m
switchstatement.m
1 - a=input('donnez une valeur pour A : ');
2 - b=input('donnez une valeur pour B : ');
3 - disp('tapez 0 pour A+B');
4 - disp('tapez 1 pour A-B');
5 - disp('tapez 2 pour A*B');
6 - x=input('donnez votre choix : ');
7 - switch x
8 - case 0,
9 - resultat = a + b;
10 - case 1,
11 - resultat = a - b;
12 - case 2,
13 - resultat = a * b;
14 - otherwise
15 - resultat = 0;
16 - end
17 - disp('le résultats vaut :');
18 - disp(resultat);
Command Window
>> switchstatement
donnez une valeur pour A : 12
donnez une valeur pour B : 32
tapez 0 pour A+B
tapez 1 pour A-B
tapez 2 pour A*B
donnez votre choix : 2
le résultats vaut :
384

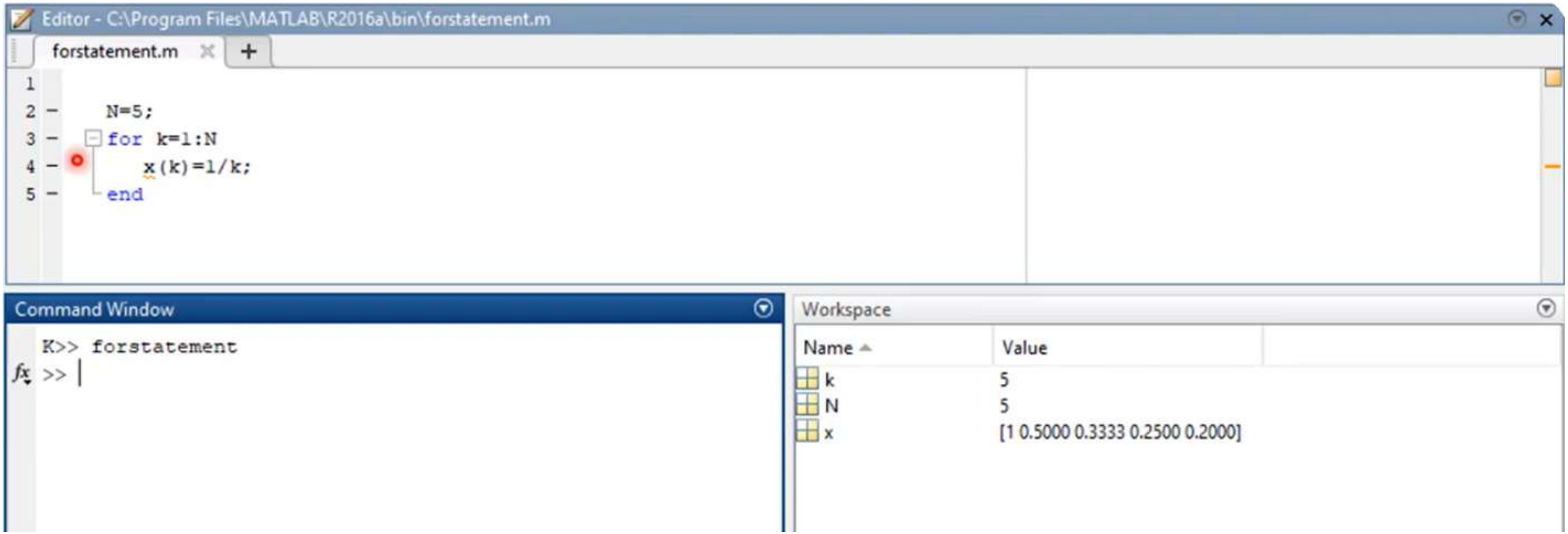
Structures de contrôle

Boucle itérative (`for`)

- Cette boucle exécute le bloc interne autant de fois que spécifié par une variable jouant un rôle de compteur.

```
for variable = debut : increment : fin

    instructions ...
end
```

Le compteur **`variable`** *est initialisé à la valeur* **`debut`** *et évolue jusqu'à la valeur fin par pas de* **`increment`**.
A chaque itération, le bloc **`instructions`** *est exécuté.*
Généralement, **`variable`** *est* un scalaire, et souvent un entier.


Editor - C:\Program Files\MATLAB\R2016a\bin\forstatement.m
forstatement.m
1
2 - N=5;
3 - for k=1:N
4 - x(k)=1/k;
5 - end
Command Window
K>> forstatement
>>
Workspace
Name
Value
k
5
N
5
x
[1 0.5000 0.3333 0.2500 0.2000]

Optimisation des calculs

tic … toc

- Les calculs sont accélérés de façon spectaculaire en utilisant des opérations vectorielles en lieu et place de boucles. Comparons les deux fonctions suivantes (la commande tic déclenche un chronomètre ; toc arrête le chronomètre et retourne le temps écoulé depuis tic en seconde) :

```
function [t,b] = test1(n)
% détermine le temps mis pour créer la liste
% des racines carrées des entiers compris
% entre 1 et n
    tic ;
    for k = 1 : 1 : n
        b(k) = sqrt(k) ;
    end
    t = toc ;
```

```
function [t,b] = test2(n)
% détermine le temps mis pour créer la
% liste des racines carrées des entiers
% compris entre 1 et n
    tic ;
    a = 1 : 1 : n ;
    b = sqrt(a) ;
    t = toc ;
```

Command Window

```
>> test2(99999)

ans =

    0.0077

>> test1(99999)

ans =

    0.0176
```

Les résultats obtenus montrent que test2 est plus efficace que test1.