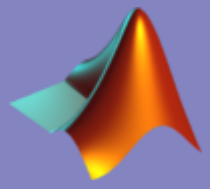
Université Badji Mokhtar – Annaba
Faculté des Sciences de l'Ingéniorat
Département d'Hydraulique
Licence troisième année Hydraulique

Hydro-Informatique

-Initiation à l'utilisation de Matlab-

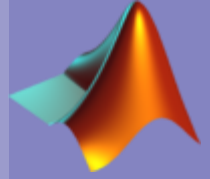
Mail du cours :initiation.matlab@gmail.com

Dr. Hamouda BOUTAGHANE



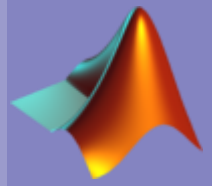
Sommaire

III. Graphisme sous Matlab



Matlab :Matrix Laboratory

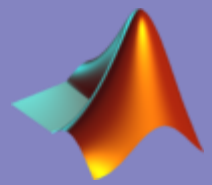
Matlab Un logiciel de calcul numérique produit par MathWorks (voir le site web http://www.mathworks.com/)



Matlab :Matrix Laboratory

Matlab Un logiciel de calcul numérique produit par MathWorks (voir le site web http://www.mathworks.com/)

Matlab est un langage simple et très efficace, optimisé pour le traitement des matrices, d'où son nom. Pour le calcul numérique, ***Matlab*** est beaucoup plus concis que les "vieux" langages (C,Pascal, Fortran, Basic).

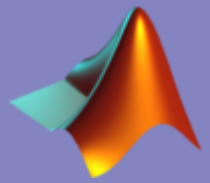


Matlab :Matrix Laboratory

Matlab Un logiciel de calcul numérique produit par MathWorks (voir le site web http://www.mathworks.com/)

Matlab est un langage simple et très efficace, optimisé pour le traitement des matrices, d'où son nom. Pour le calcul numérique, ***Matlab*** est beaucoup plus concis que les "vieux" langages (C,Pascal, Fortran, Basic).

Matlab est enrichi avec des « ***toolbox*** » qui sont des ensembles de fonctions supplémentaires, profilées pour des applications particulières (traitement de signaux, analyses statistiques, optimisation, etc.)



Lancer / Arrêter

Pour Lancer

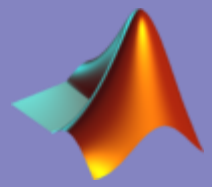
Sous Windows selectionner

`Start-> tous les Programmes-> MATLAB > MATLAB R2009a`

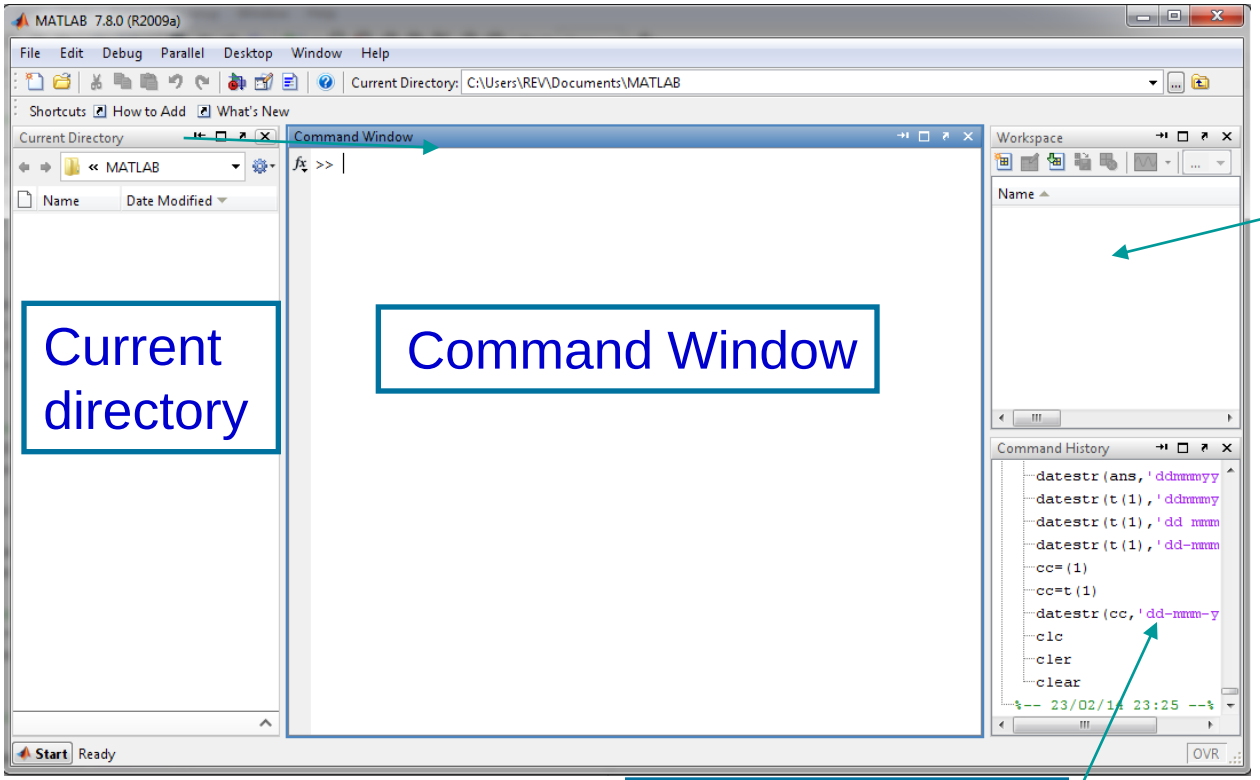
Pour Arreter

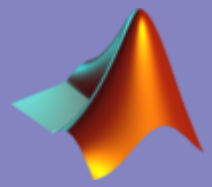
Selectioner `File -> Exit`

Ou taper `quit` dans le menu MATLAB command window

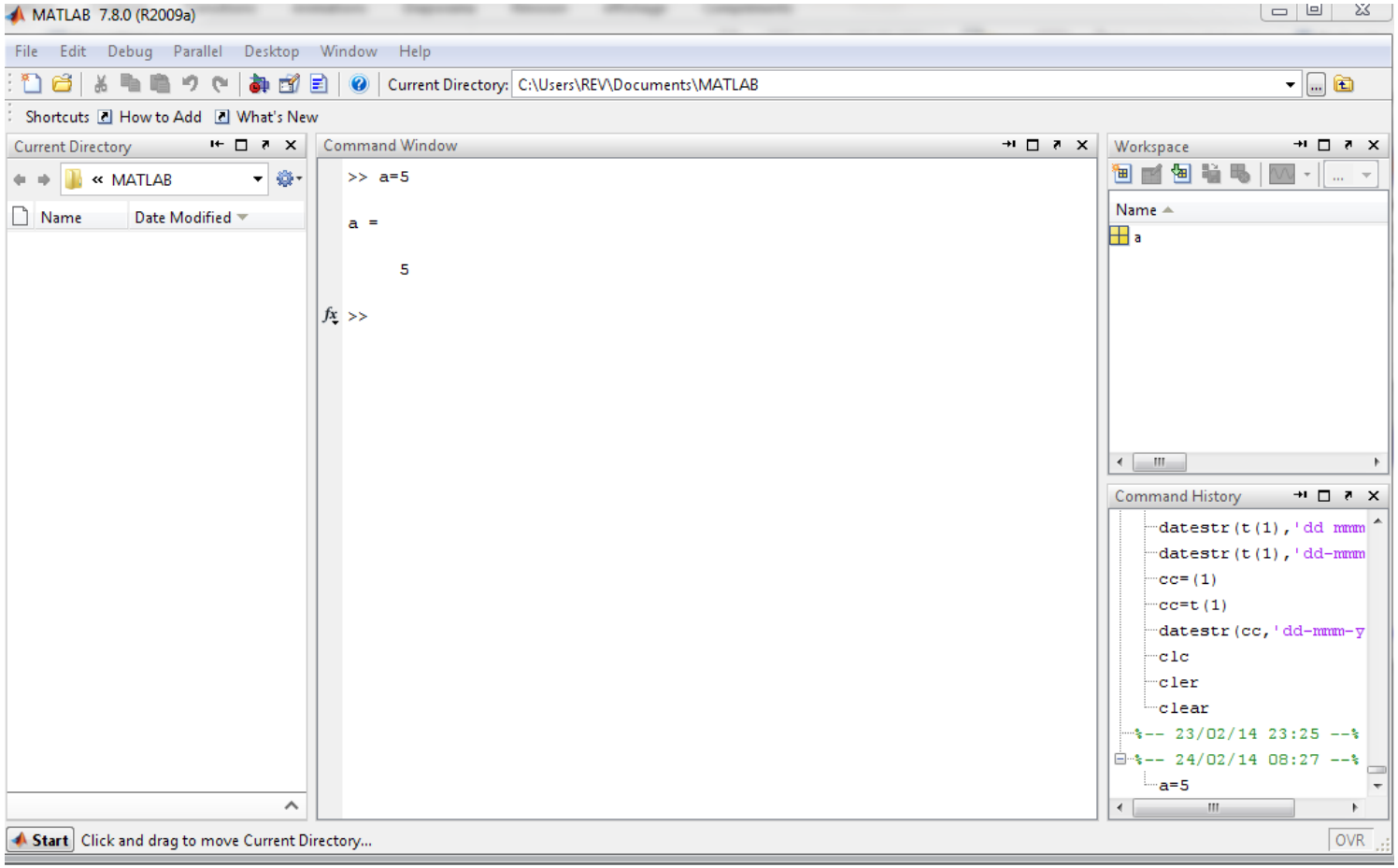


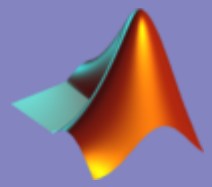
Interface Matlab





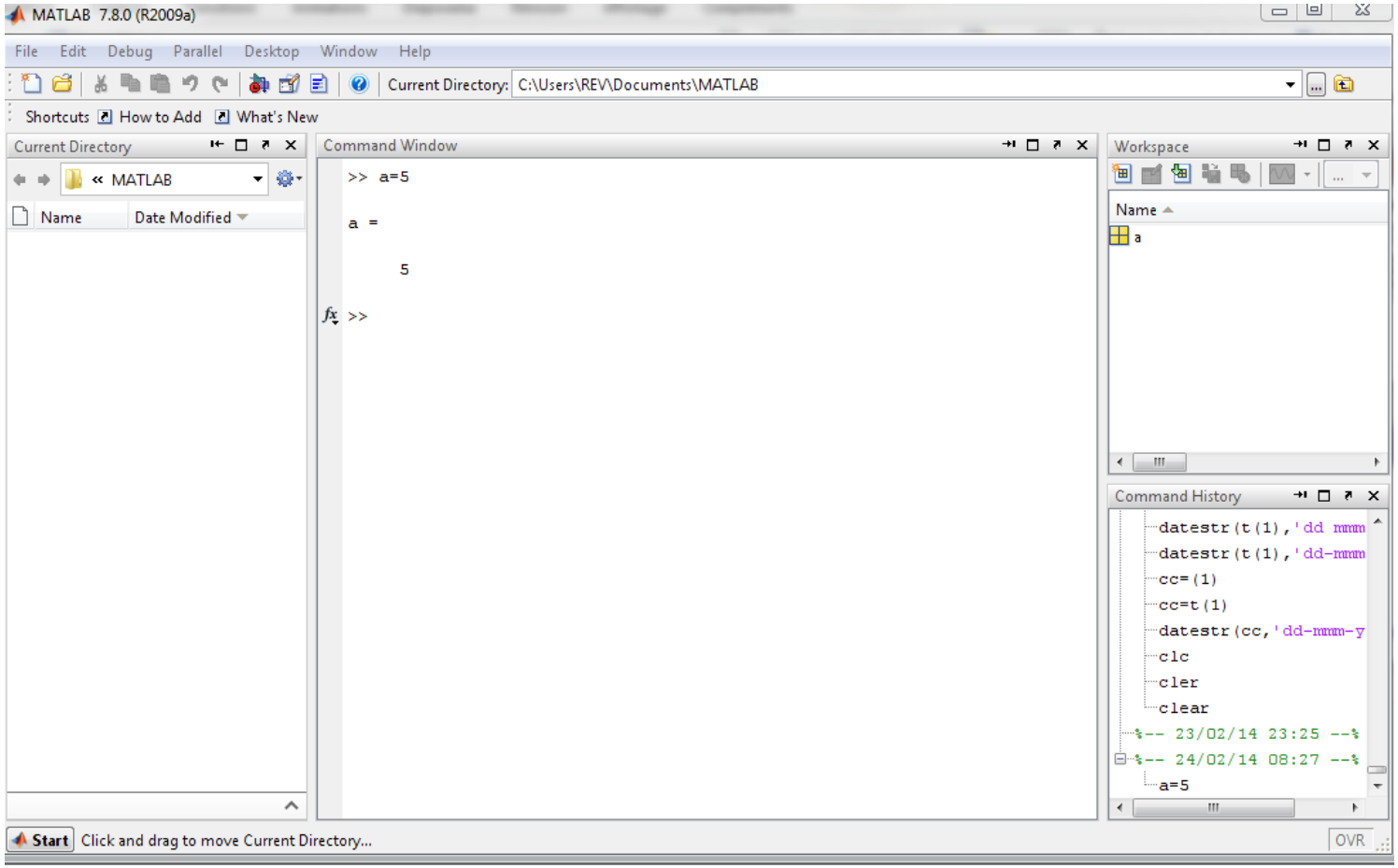
Interface Matlab

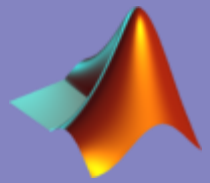




1. Initiation à l'utilisation de Matlab

Interface Matlab



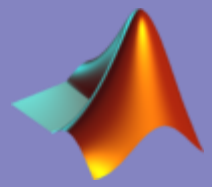


Variables, Vecteurs et Matrices

Création de variables

- **Nom de la variable**
  - ✓ Peut être en n'importe quelle chaîne de lettres majuscules et minuscules aussi que des chiffres et des caractères de soulignement, mais il doit toujours commencer par une lettre
  - ✓ Les noms réservés sont IF, WHILE, ELSE, END, SUM, etc
- **Valeur**
  - ✓ sont les données associées à la variable;
  - ✓ l'accès aux données se fait en utilisant le nom de la variable.

- **Les variables ont le type de la dernière donnée qui leur est attribuée**
  - ✓ L'affectation se fait sans avertissement-il n'y a pas de mises en garde si vous écrasez une variable avec quelque chose d'un type différent.



Variables, vecteurs et matrices

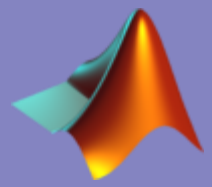
Création de variables: Variable simple

Pour affecter une valeur à une variable utiliser le symbol égale '='

>> A = 32

Pour trouver la velur d'une variable : tapez son nom

```
>> A = 32

A =

    32

>>

>> A

A =

    32

>>
```



Variables, vecteurs et matrices

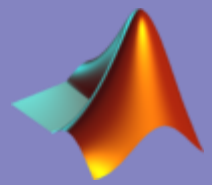
Création de variables: Variable simple

Pour crier une autre variable égal à celle déjà existante

```
>> B = A
```

```
>> B = A

B =

    32

>> A = 15

A =

    15

>> B

B =

    32

>>
```

La nouvelle variable n'est pas mise à jour lorsque vous modifiez la valeur d'origine



Variables, vecteurs et matrices

Création de variables: Variable simple

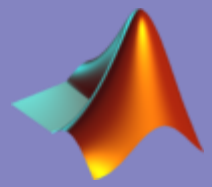
La valeur de deux variables peuevent etre additionner et le résultat sera visionner sur écran

```
>> A = 10
>> A + A
```

…ou le résulat peut etre stocker dans une autre variable

```
>> A = 10
>> B = A + A
```

```
>> A = 10

A =

     10

>> A + A

ans =

     20

>> B = A + A

B =

     20

>>
```



Variables, vecteurs et matrices

Vecteur

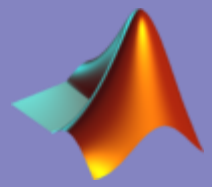
Le vecteur contient plusieurs nombres

Utiliser les crochets [] pour contenir les nombres

	A	B	C	D	E	F	G	H	I
1									
2	Test Results:	0.5	0.3	0.54	0.7	0.89	0.2	1	
3									
4									

Pour crier un vecteur ligne utiliser ',' pour séparer les contenants

```
>> Test_Results = [0.5, 0.3, 0.54, 0.7, 0.89, 0.2, 1]

Test_Results =

    0.5000    0.3000    0.5400    0.7000    0.8900    0.2000    1.0000

>>
```



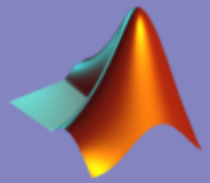
Variables, vecteurs et matrices

Vecteur

Pour crier un vecteur colonne on utilise ';' to separate the content

	A	B
1	Test Results	
2	0.5	
3	0.3	
4	0.54	
5	0.7	
6	0.89	
7	0.2	
8	1	
9		

```
>> Test_Results = [0.5; 0.3; 0.54; 0.7; 0.89; 0.2; 1]

Test_Results =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    0.2000
    1.0000

>>
```

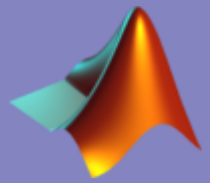


Variables, vecteurs et matrices

Vecteur

Un vecteur ligne peut etre converté en vecteur colonne en utilisant l'opératuer de transposition ‘

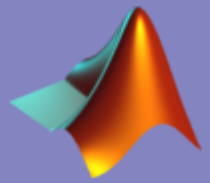
```
>> Test_Results = [0.5, 0.3, 0.54, 0.7, 0.89, 0.2, 1]

Test_Results =

    0.5000    0.3000    0.5400    0.7000    0.8900    0.2000    1.0000

>> Test_Results = Test_Results'

Test_Results =

    0.5000
    0.3000
    0.5400
    0.7000
    0.8900
    0.2000
    1.0000

>>
```



Variables, vecteurs et matrices

Matrices

- Une matrice MATLAB est un réseau rectangulaire de nombres
  - ✓ Scalaires et les vecteurs sont considérés comme des cas particuliers de matrices
  - ✓ MATLAB vous permet de travailler avec toute une gamme à la fois

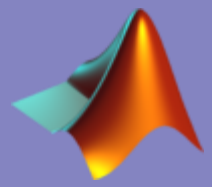


Variables, vecteurs et matrices

Matrices

- Vous pouvez créer des matrices (tableaux) de toute taille en utilisant une combinaison de méthodes de création des vecteurs

- Dressez la liste des numéros à l'aide ',' pour séparer chaque colonne, puis ';' pour définir une nouvelle ligne

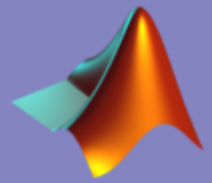


Variables, vecteurs et matrices

Matrices

	A	B	C	D	E
1	Results				
2					
3	Subject	Test 1	Test 2	Test 3	
4	1	0.5	0.6	0.3	
5	2	0.3	0.33	0.75	
6	3	0.54	0.2	0.99	
7	4	0.7	0.6	0.1	
8	5	0.89	0.73	0.3	
9	6	0.2	0.9	0.94	
10	7	1	0.4	0.3	
11					

```
>> Results = [1, 0.5, 0.6, 0.3; 2, 0.3, 0.33, 0.75;
              3, 0.54, 0.2, 0.99; 4, 0.7, 0.6, 0.1;
              5, 0.89, 0.73, 0.3; 6, 0.2, 0.9, 0.94;
              7, 1, 0.4, 0.3]

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000    0.9900
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```



Variables, vecteurs et matrices

Matrices

Vous pouvez également utiliser des fonctions intégrées pour créer une matrice

>> A = zeros(2, 4)

créer une matrice appelée A avec 2 lignes et 4 colonnes contenant la valeur 0

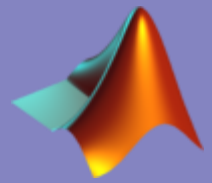
>> A = zeros(5) or >> A = zeros(5, 5)

créer une matrice appelée A avec 5 lignes et 5 colonnes

Vous pouvez également utiliser

>> ones(lignes, colonnes)

>> rand(lignes, colonnes)



Variables, vecteurs et matrices

Matrices

Vous pouvez également utiliser des fonctions intégrées pour créer une matrice

>> A = zeros(2, 4)

crée une matrice appelée A avec 2 lignes et 4 colonnes contenant la valeur 0

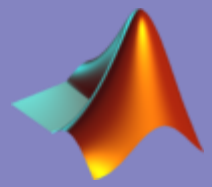
>> A = zeros(5) or >> A = zeros(5, 5)

crée une matrice appelée A avec 5 lignes et 5 colonnes

Vous pouvez également utiliser

>> ones(i,j)
>> rand(i,j)
>> magic(n)

Remarque: MATLAB se réfère toujours à la première valeur que le nombre de lignes puis la seconde que le nombre de colonnes



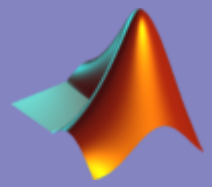
Variables, vecteurs et matrices

Effacez les variables

Vous pouvez utiliser la commande "clear all" pour éffacer toutes les variables présente dans le workspace

Vous pouvez éffacer une variable en spécifiant son nom:

```
>> clear Variable_Name
```



Variables, vecteurs et matrices

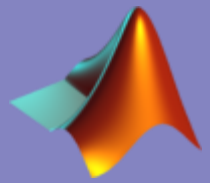
Accès aux éléments des matrices

- Un élément est un numéro unique au sein d'une matrice ou d'un vecteur;

- Pour accéder aux éléments d'un type de matrice: Le nom de matrices suivie par des parenthèses contenant une référence au numéro de ligne et de colonne:

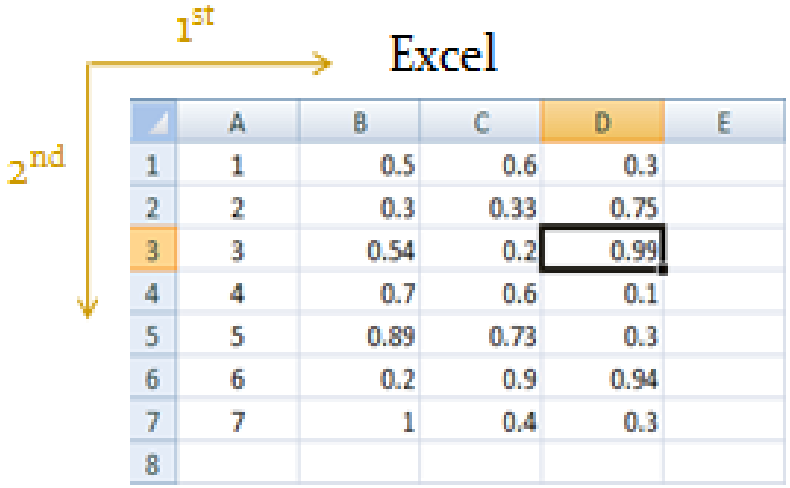
  >> Nom_Variable (Numero_Ligne , Numero_Colonne)

NOTEZ: Dans Excel les valeurs sont référencées par le numéro de la colonne puis celui de la ligne. Dans Matlab c'est l'inverse.

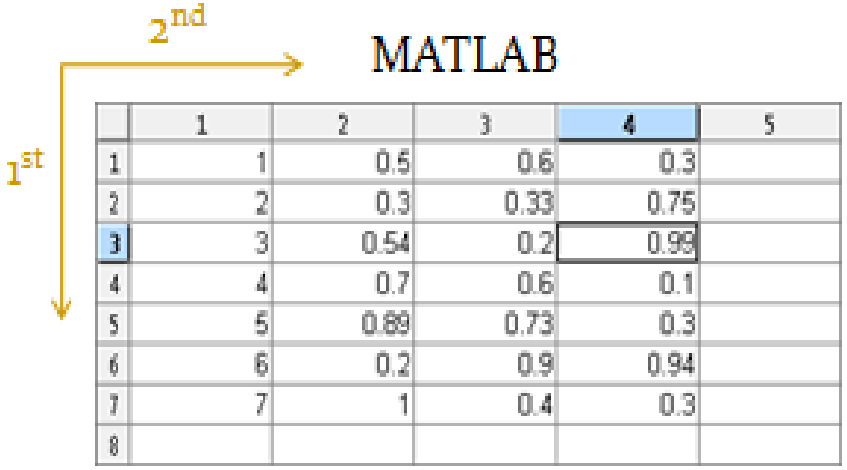


Variables, vecteurs et matrices

Accès aux éléments des matrices



	A	B	C	D	E
1	1	0.5	0.6	0.3	
2	2	0.3	0.33	0.75	
3	3	0.54	0.2	0.99	
4	4	0.7	0.6	0.1	
5	5	0.89	0.73	0.3	
6	6	0.2	0.9	0.94	
7	7	1	0.4	0.3	
8					

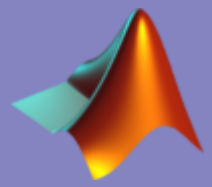


	1	2	3	4	5
1	1	0.5	0.6	0.3	
2	2	0.3	0.33	0.75	
3	3	0.54	0.2	0.99	
4	4	0.7	0.6	0.1	
5	5	0.89	0.73	0.3	
6	6	0.2	0.9	0.94	
7	7	1	0.4	0.3	
8					

Pour acceder au résultat de l'exemple

Dans Excel (Colonne, Ligne): D3

Dans MATLAB (Ligne, Colonne): >> results(3, 4)



Variables, vecteurs et matrices

Modification des éléments d'une matrices

L'élément référencé peut également être modifié

```
>> results(3, 4) = 10
```

or

```
>> results(3,4) = results(3,4) * 100
```

```
>> Results(3,4) = 10

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000   10.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

```
>> Results(3,4) = Results(3,4) * 100

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000  198.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

Variables, vecteurs et matrices

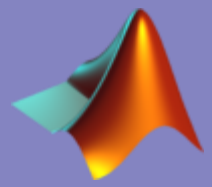
Accès aux lignes d'une matrice

- Vous pouvez également accéder à plusieurs valeurs à partir d'une matrice en utilisant le symbole **:**
  - Pour accéder à toutes les colonnes d'une ligne tapez:

    >> Nom_Variable (Numero_ligne,:)

	A	B	C	D	E
1	Subject	Test 1	Test 2	Test 3	
2	1	0.5	0.6	0.3	
3	2	0.3	0.33	0.75	
4	3	0.54	0.2	0.99	
5	4	0.7	0.6	0.1	
6	5	0.89	0.73	0.3	
7	6	0.2	0.9	0.94	
8	7	1	0.4	0.3	
9					

```
>> Results(4, :)

ans =

    4.0000    0.7000    0.6000    0.1000

>>
```



Variables, vecteurs et matrices

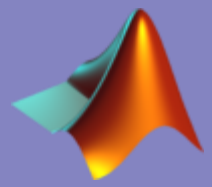
Accès aux colonnes d'une matrice

Pour acceder a toutes les lignes d'une colonne tapez

>> Nom_Variable (:, Numero_Colonne)

	A	B	C	D	E
1	Subject	Test 1	Test 2	Test 3	
2	1	0.5	0.6	0.3	
3	2	0.3	0.33	0.75	
4	3	0.54	0.2	0.99	
5	4	0.7	0.6	0.1	
6	5	0.89	0.73	0.3	
7	6	0.2	0.9	0.94	
8	7	1	0.4	0.3	
9					

```
>> Results(:, 3)

ans =

    0.6000
    0.3300
    0.2000
    0.6000
    0.7300
    0.9000
    0.4000

>>
```



Variables, vecteurs et matrices

Modification des lignes ou colonnes d'une matrice

Ces méthodes de référence peuvent être utilisées pour modifier les valeurs de multiples éléments d'une matrice

- Pour modifier toutes les valeurs d'une ligne ou d'une colonne à la valeur zéro

>> results(:, 3) = 0

```
>> Results(:,3) = 0

Results =

    1.0000    0.5000         0    0.3000
    2.0000    0.3000         0    0.7500
    3.0000    0.5400         0    0.9900
    4.0000    0.7000         0    0.1000
    5.0000    0.8900         0    0.3000
    6.0000    0.2000         0    0.9400
    7.0000    1.0000         0    0.3000

>>
```

>> results(:, 5) = results(:, 3) + results(:, 4)

```
>> Results(:, 5) = Results(:, 3) + Results(:, 4)

Results =

    1.0000    0.5000    0.6000    0.3000    0.9000
    2.0000    0.3000    0.3300    0.7500    1.0800
    3.0000    0.5400    0.2000    0.9900    1.1900
    4.0000    0.7000    0.6000    0.1000    0.7000
    5.0000    0.8900    0.7300    0.3000    1.0300
    6.0000    0.2000    0.9000    0.9400    1.8400
    7.0000    1.0000    0.4000    0.3000    0.7000

>>
```

Variables, vecteurs et matrices

Modification des lignes ou colonnes d'une matrice

- Pour remplacer une ligne ou une colonne avec de nouvelles valeurs

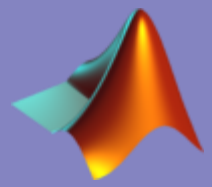
  >> results(3, :) = [10, 1, 1, 1]

  >> results(:, 3) = [1; 1; 1; 1; 1; 1; 1]

```
>> Results(3, :) = [10, 1, 1, 1]

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
   10.0000    1.0000    1.0000    1.0000
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000

>>
```

```
>> Results(:, 3) = [1; 1; 1; 1; 1; 1; 1]

Results =

    1.0000    0.5000    1.0000    0.3000
    2.0000    0.3000    1.0000    0.7500
    3.0000    0.5400    1.0000    0.9900
    4.0000    0.7000    1.0000    0.1000
    5.0000    0.8900    1.0000    0.3000
    6.0000    0.2000    1.0000    0.9400
    7.0000    1.0000    1.0000    0.3000

>>
```

NOTE: Si vous écrasez une valeur unique, les données saisies doivent être de la même taille que la partie de la matrice à écrasé.



Variables, vecteurs et matrices

Accès a plusieurs lignes et colonnes

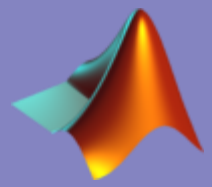
- Pour accéder à des lignes ou colonnes consécutives utiliser: avec des points de début et de fin:

✓ Plusieurs Lignes:
>> Nom_Variable(start:end, :)

✓ Plusieurs Colonnes:
>> Nom_Variable(:, start:end)

```
>> Results(1:2, :)

ans =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500

>>
```

```
>> Results(:, 1:2)

ans =

    1.0000    0.5000
    2.0000    0.3000
    3.0000    0.5400
    4.0000    0.7000
    5.0000    0.8900
    6.0000    0.2000
    7.0000    1.0000

>>
```



Variables, vecteurs et matrices

Accès a plusieurs lignes et colonnes

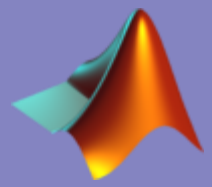
Pour accéder à plusieurs lignes ou colonnes non consécutives utiliser un vecteur d'indices (utilisant les crochets [])

- ✓ Plusieurs Lignes:
  >>Nom_Variable([index1, index2, etc.], :)

- ✓ Multiple Columns:
  >>Nom_Variable(:, [index1, index2, etc.])

```
>> Results([1,3], :)

ans =

    1.0000    0.5000    0.6000    0.3000
    3.0000    0.5400    0.2000    0.9900

>>
```

```
>> Results(:, [1,3])

ans =

    1.0000    0.6000
    2.0000    0.3300
    3.0000    0.2000
    4.0000    0.6000
    5.0000    0.7300
    6.0000    0.9000
    7.0000    0.4000

>>
```



Variables, vecteurs et matrices

Modification de plusieurs lignes, colonnes

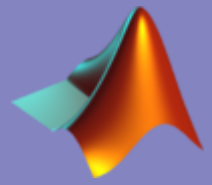
Le même référencement peut être utilisé pour modifier plusieurs lignes ou colonnes

>> results([3,6], :) = 0

```
>> Results([3,6], :) = 0

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
         0         0         0         0
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
         0         0         0         0
    7.0000    1.0000    0.4000    0.3000

>>
```

>> results(3:6, :) = 0

```
>> Results(3:6, :) = 0

Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    7.0000    1.0000    0.4000    0.3000

>>
```



Variables, vecteurs et matrices

L'Opérateur colon (:)

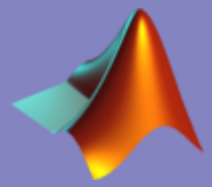
Colon : est en fait un opérateur, qui génère un vecteur ligne . Ce vecteur ligne peut être considérée comme un ensemble d'indices lors de l'accès d'un des éléments d'une matrice

L'expression générale est : [start:stepsize:end]

```
>> [11:2:21]
   11    13    15    17    19    21
>>
```

Stepsize n'est pas obligatoirement un nombre entier (ou positive)

```
>> [22:-2.07:11]
   22.00    19.93    17.86    15.79    13.72    11.65
>>
```



Variables, vecteurs et matrices

Sauvegarde et chargement des données

Les variables qui sont actuellement dans l'espace de travail peuvent être sauvegardés et chargés en utilisant les commandes de sauvegarde et de chargement

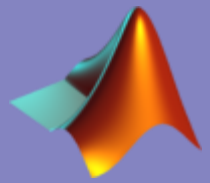
MATLAB enregistre le fichier dans le répertoire courant

Pour enregistrer les variables utilisez

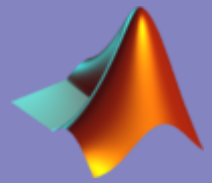
>> save Nom_Ficher [variable variable …]

Pour charger une variable utilisez

>> load Nom_Fichier [variable variable …]



Plus d' Opérateurs



Plus d' Opérateurs

Opérateurs Mathématiques

Add: +

Subtract: -

Divide: ./

Multiply: .*

Power: .^ (e.g. .^2 means squared)

Vous pouvez utiliser des parenthèses pour indiquer l'ordre dans lequel les opérations seront effectuées

Notez:qui précède le symbole / ou * ou ^ ou par un "." signifie que l'opérateur est appliquée entre des paires d'éléments de vecteurs de matrices

Plus d' Opérateurs

Opérateurs Mathématiques

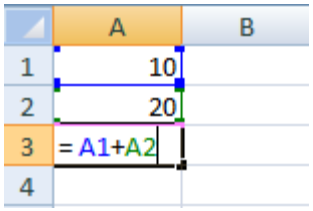
Les opérations mathématiques simples sont faciles avec MATLAB

La structure de la commande est:

>> Variable_Resultat = Nom_Variable1 operator Nom_Variable2

Exemple: Pour additionner deux nombres:

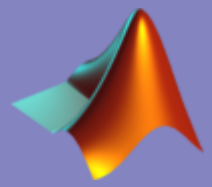
Excel:



MATLAB:

>> C = A + B

>> C = (A + 10) ./ 2



Plus d' Opérateurs

Opérateurs Mathématiques

Vous pouvez appliquer des valeurs uniques pour toute une matrice

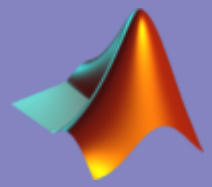
Exemple

```
>> data = rand(5,1)
>> A = 10
>> results = data + A
```

```
>> data = rand(5, 1)

data =

    0.0967
    0.8181
    0.8175
    0.7224
    0.1499

>>
```

```
>> A = 10

A =

    10
```

```
>> data + A

ans =

   10.0967
   10.8181
   10.8175
   10.7224
   10.1499

>>
```



Plus d' Opérateurs

Opérateurs Mathématiques

Ou, si deux matrices / vecteurs ont la même taille, vous pouvez effectuer ces opérations entre eux

>> results = [1:5]'

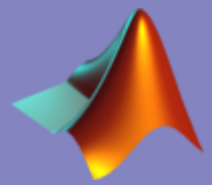
>> results2 = rand(5,1)

>> results3 = results + results2

```
>> results = [1:5]'

results =

     1
     2
     3
     4
     5
```

```
>> results2 = rand(5, 1)

results2 =

    0.6596
    0.5186
    0.9730
    0.6490
    0.8003
```

```
>> results3 = results + results2

results3 =

    1.6596
    2.5186
    3.9730
    4.6490
    5.8003

>>
```



Plus d' Opérateurs

Opérateurs Mathématiques

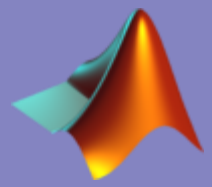
En combinant cela avec les méthodes d'accès aux éléments de matrice , ca donne lieu à des opérations plus utiles

```
>> results = zeros(3, 5)
>> results(:, 1:4) = rand(3, 4)
>> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) +  results(:, 4)
```

or

```
>> results(:, 5) = results(:, 1) .* results(:, 2) .* results(:, 3) .*  results(:, 4)
```

NOTEZ: Il existe un moyen simple de le faire en utilisant les fonction Sum et Prod. On les verra plus tard.



Plus d' Opérateurs

Opérateurs Mathématiques

>> results = zeros(3, 5)

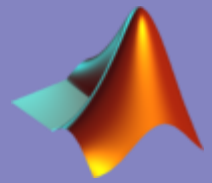
>> results(:, 1:4) = rand(3, 4)

>> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) + results(:, 4)

```
>> results = zeros(3, 5)

results =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

```
>> results(:, 1:4) = rand(3, 4)

results =

    0.4228    0.4177    0.7011    0.6981         0
    0.5479    0.9831    0.6663    0.6665         0
    0.9427    0.3015    0.5391    0.1781         0
```

```
>> results(:, 5) = results(:,1) + results(:,2) + results(:,3) + results(:,4)

results =

    0.4228    0.4177    0.7011    0.6981    2.2398
    0.5479    0.9831    0.6663    0.6665    2.8638
    0.9427    0.3015    0.5391    0.1781    1.9615

>>
```



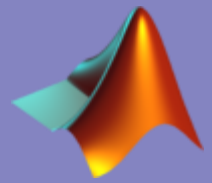
Plus d' Opérateurs

Opérateurs Logiques

Vous pouvez utiliser l'indexation logique pour trouver des données qui sont conforme à certaines limites

Opérateurs logiques :

- Greater Than: >
- Less Than: <
- Greater Than or Equal To: >=
- Less Than or Equal To: <=
- Is Equal: ==
- Not Equal To: ~=



Plus d' Opérateurs

Opérateurs Booleans

Opérateurs Booleans :

AND: &

OR: |

NOT: ~

Connecte deux expressions logiques ensemble

En utilisant une combinaison d'opérateurs logiques et booléens , nous pouvons sélectionner des valeurs qui entrent dans une limite inférieure et supérieure

```
>> r = results(:,1)
>> ind = r > 0.2 & r <= 0.9
>> r(ind)
```

Variables, vecteurs et matrices

Stockage des matrices en mémoire

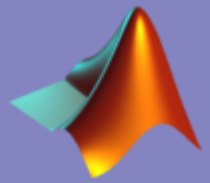
Une matrice se représente visuellement comme un tableau 2D composé de lignes et de colonnes. Elle est toujours stockée sous la forme d'un vecteur, colonne par colonne, avec chaque élément mis bout à bout.

1	4	7
2	5	8
3	6	9

Exemple de matrice 3x3

1
2
3
4
5
6
7
8
9

Stockage d'une matrice sous forme de vecteur



Variables, vecteurs et matrices

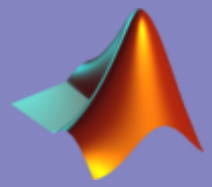
Méthodes d'indexage

Les méthodes d'indexage servent à accéder aux éléments de la matrice à l'aide d'indices.

Indexage classique (ligne,colonne)

La méthode d'indexage classique consiste à spécifier la position d'un élément en fonction de l'indice de la ligne et de l'indice de la colonne

M(idx ligne, idx colonne).

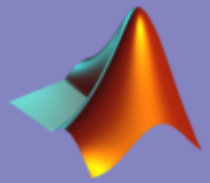


Variables, vecteurs et matrices

Méthodes d'indexage

Indexage classique (ligne,colonne)

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> M(4,3)

ans =

    15
```



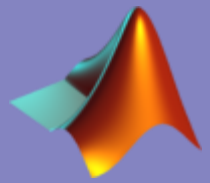
Variables, vecteurs et matrices

Méthodes d'indexage

Indexage linéaire

MATLAB ne stocke pas les matrices dans la mémoire sous forme de tableaux à 2 dimensions, mais sous forme de vecteurs (colonne par colonne) dont chaque élément est mis bout à bout.

Ce type de stockage permet d'utiliser une autre technique d'indexage, que l'on appelle indexage linéaire. On ne localise plus un élément d'une matrice par le couple d'indices ligne-colonne, mais directement par la position de l'élément dans le vecteur stocké en mémoire.



Variables, vecteurs et matrices

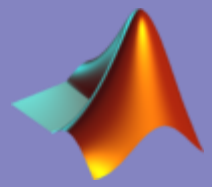
Méthodes d'indexage

Indexage linéaire

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
16
 5
 9
 4
 2
11
 7
14
 3
10
 6
15
13
 8
12
 1
```

```
>> M(4,3) % indexage classique

ans =

    15

>> M(12) % indexage linéaire

ans =

    15
```



Variables, vecteurs et matrices

Méthodes d'indexage

Indexage linéaire

- La relation de passage entre l'indexage classique et l'indexage linéaire est :

  M(i,j) => M(i+(j-1)*size(M,1))

  Soit dans l'exemple précédent M(4,3) => M(4+(3-1)*4) => M(12)

- De la même manière, la relation de passage entre l'indexage linéaire et l'indexage classique est :

  M(k) => M(k-ceil(k/size(M,1))*size(M,1)+size(M,1),ceil(k/size(M,1)))

Soit dans l'exemple précédent

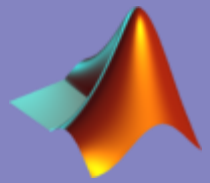
M(12) => M(12-(ceil(12/4))*4+4,ceil(12/4)) => M(12-3*4+4,3) => M(4,3)

Variables, vecteurs et matrices

Méthodes d'indexage

Indexage logique

- ✓ Il est basée sur les conditions logiques. On l'utilise principalement avec les opérateurs relationnels et les opérateurs logiques.
- ✓ Ce type d'indexage permet souvent d'améliorer l'efficacité des codes en évitant l'utilisation de fonctions supplémentaires (comme la fonction **find**).
- ✓ L'indexage logique est souvent utilisé avec les fonctions **any** et **all**.



Variables, vecteurs et matrices

Méthodes d'indexage

Indexage logique

Par exemple, on souhaite trouver toutes les valeurs supérieures à 3 (soit 8 et 7) dans la matrice suivante :

```
>> x = [8 1 ; 2 7]

x =

     8     1
     2     7
```

```
>> [i,j] = find(X>3)

i =

     1
     2


j =

     1
     2
```

```
>> X(i(1),j(1))

ans =

     8

>> X(i(2),j(2))

ans =

     7
```

Variables, vecteurs et matrices

Méthodes d'indexage

Indexage logique

ou pour obtenir les indices linéaires comme ceci :

```
>> idx = find(X>3)

idx =

     1
     4

>> X(idx)

ans =

     8
     7
```

Variables, vecteurs et matrices

Méthodes d'indexage

Indexage logique

L'indexage logique consiste simplement à se passer de la fonction **find** comme ceci :

```
>> X = [8 1 ; 2 7]

X =

     8     1
     2     7
```

```
>> idx = X>3 % indexage logique

idx =

  1  0
  0  1
```

```
>> X(idx)

ans =

     8
     7
```

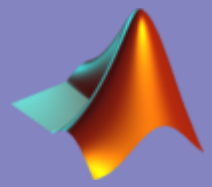
Variables, vecteurs et matrices

Les outils d'indexage

MATLAB possède plusieurs outils qui permettent de simplifier ou d'optimiser l'indexage des matrices

L'opérateur end

end est un mot-clé de MATLAB. Il sert à fermer les structures itératives (**for-end** par exemple) ou les structures conditionnelles (**if-end**, par exemple). Mais ce mot-clé peut aussi être employé comme opérateur d'indexage.



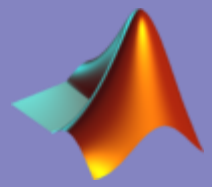
Variables, vecteurs et matrices

Les outils d'indexage

L'opérateur end

Dans le cas d'un vecteur, le dernier élément est retourné :

```
>> M = [4  7  2]

M =

     4     7     2

>> M(end)

ans =

     2
```



Variables, vecteurs et matrices

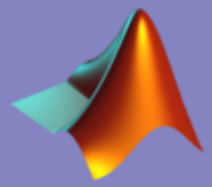
Les outils d'indexage

L'opérateur end

Dans le cas d'une matrice, il renvoie le dernier élément d'une des dimensions

```
>> X = [8 1 ; 2 7]

X =

     8     1
     2     7
```

```
>> X(end,1) % Dernier élément de la première colonne

ans =

     2

>> X(end,2) % Dernier élément de la seconde colonne

ans =

     7

>> X(1,end) % Dernier élément de la première ligne

ans =

     1
```



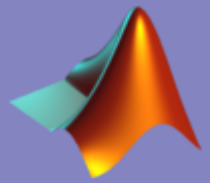
Variables, vecteurs et matrices

Les outils d'indexage

L'opérateur end

```
>> X = [8 1 ; 2 7]

X =

     8     1
     2     7
```

```
>> X(2,end) % Dernier élément de la seconde ligne

ans =

     7

>> X(end,end) % Dernier élément de la matrice (indexage classique)

ans =

     7

>> X(end) % Dernier élément de la matrice (indexage linéaire)

ans =

     7
```



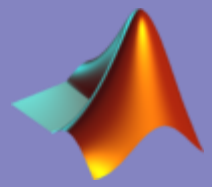
Variables, vecteurs et matrices

Les outils d'indexage

L'opérateur :

Classiquement, l'opérateur **:** (colon en anglais) sert lors de la définition d'un vecteur

```
>> v = 1:5

v =

     1     2     3     4     5

>> v = 7:0.2:8

v =

    7.0000    7.2000    7.4000    7.6000    7.8000    8.0000
```



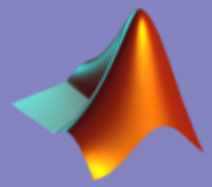
Variables, vecteurs et matrices

Les outils d'indexage

L'opérateur :

Dans le cas de l'indexage, il sert d'abord à spécifier une plage d'indices :

```
>> X = [8 1 ; 2 7 ; 5 9]

X =

     8     1
     2     7
     5     9

>> X(2:3,2) % Eléments des lignes 2 à 3 de la colonne 2

ans =

     7
     9
```



Variables, vecteurs et matrices

Les outils d'indexage

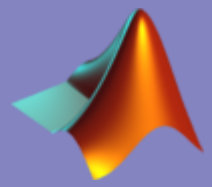
L'opérateur :

Employé seul, il sert aussi à spécifier tous les indices d'une dimension

```
>> X = [8 1 ; 2 7 ; 5 9]

X =

     8     1
     2     7
     5     9
```

```
>> X(:,1) % Tous les éléments de la première colonne

ans =

     8
     2
     5
```

```
>> X(2,:) % Tous les éléments de la seconde ligne

ans =

     2     7
```



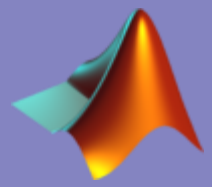
Variables, vecteurs et matrices

Les outils d'indexage

L'opérateur :

```
>> X = [8 1 ; 2 7 ; 5 9]

X =

     8     1
     2     7
     5     9
```

```
>> X(:,:) % Tous les éléments de X (indexage classique)

ans =

     8     1
     2     7
     5     9

>> X(:) % Tous les éléments de X (indexage linéaire)

ans =

     8
     2
     5
     1
     7
     9
```



Variables, vecteurs et matrices

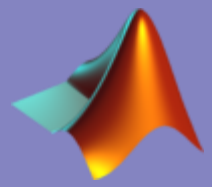
Les outils d'indexage

Les fonction sub2ind et ind2sub

La fonction **sub2ind** sert à passer de l'indexage classique vers l'indexage linéaire

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
>> idx = sub2ind(size(M),4,3)

idx =

    12
```



Variables, vecteurs et matrices

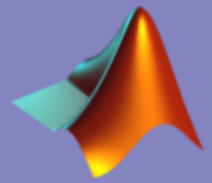
Les outils d'indexage

Les fonction sub2ind et ind2sub

La fonction **ind2sub** sert à passer de l'indexage linéaire vers l'indexage classique.

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
>> [ligne,colonne] = ind2sub(size(M),12)

ligne =

     4


colonne =

     3
```

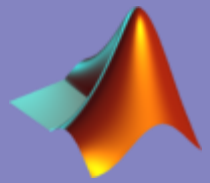


Opération courantes

Concaténation

La concaténation consiste à coller des matrices bout à bout afin d'obtenir une matrice supplémentaire. Cette opération s'effectue entre crochets.

A l'intérieur de ces crochets, les différentes matrices doivent être séparées, soit par des points-virgules pour une concaténation verticale, soit par des virgules ou des espaces pour une concaténation horizontale.



Opération courantes

Concaténation

Concaténation verticale

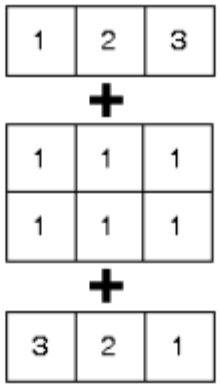
- ✓ La concaténation verticale consiste à mettre des matrices les unes sur les autres.
- ✓ Les différentes matrices doivent impérativement avoir le même nombre de colonnes.

Opération courantes

Concaténation

Concaténation verticale

Par exemple, pour concaténer verticalement les trois matrices suivantes



Concaténation verticale de trois matrices

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

A =

     1     2     3

>> B = ones(2,3)

B =

     1     1     1
     1     1     1

>> C = [3 2 1]

C =

     3     2     1
```

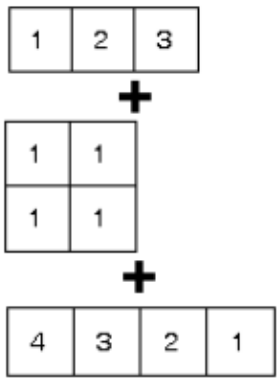
```
>> X = [A ; B ; C]

X =

     1     2     3
     1     1     1
     1     1     1
     3     2     1
```

Opération courantes

Concaténation

Concaténation verticale

Si les matrices n'ont pas le même nombre de colonnes, MATLAB retourne le message d'erreur suivant



Concaténation verticale de matrices impossible

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

A =

     1     2     3

>> B = ones(2)

B =

     1     1
     1     1

>> C = [4 3 2 1]

C =

     4     3     2     1
```

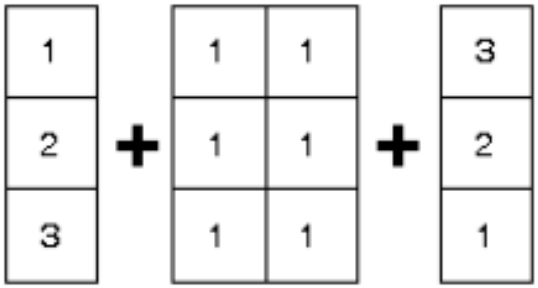
```
>> X = [A ; B ; C]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```

Opération courantes

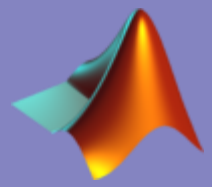
Concaténation

Concaténation horizontale

- ✓ La concaténation horizontale consiste à mettre des matrices les unes à coté des autres.
- ✓ Les différentes matrices doivent impérativement avoir le même nombre de lignes



Concaténation horizontale de trois matrices



Opération courantes

Concaténation

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

A =

     1
     2
     3

>> B = ones(3,2)

B =

     1     1
     1     1
     1     1

>> C = [3;2;1]

C =

     3
     2
     1
```

Concaténation horizontale

```
>> X = [A,B,C]

X =

     1     1     1     3
     2     1     1     2
     3     1     1     1
```

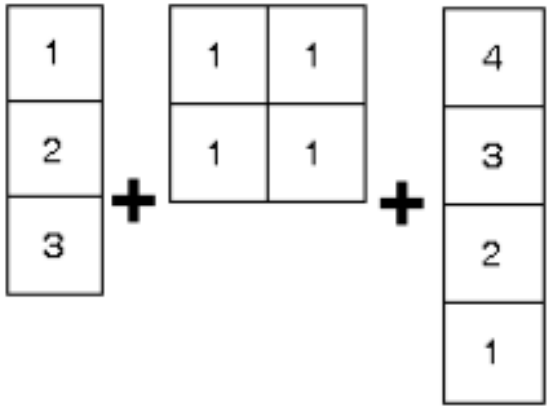
Opération courantes

Concaténation

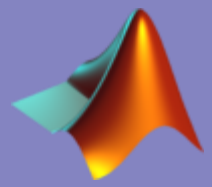
Concaténation horizontale

Si les matrices n'ont pas le même nombre de lignes, MATLAB retourne le message d'erreur suivant :

```
??? Error using ==> horzcat
CAT arguments dimensions are not consistent.
```



Concaténation horizontale de matrices impossible



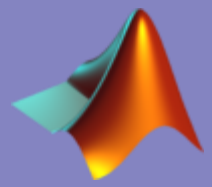
Opération courantes

Concaténation

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

A =

     1
     2
     3

>> B = ones(2)

B =

     1     1
     1     1

>> C = [4;3;2;1]

C =

     4
     3
     2
     1
```

Concaténation horizontale

```
>> C = [4;3;2;1]

C =

     4
     3
     2
     1

>> X = [A,B,C]
??? Error using ==> horzcat
CAT arguments dimensions are not consistent.
```



Les Fonctions Matlab

- Une fonction effectue une opération sur la variable d'entrée que vous lui transmettez
- L'utilisation avec les variables est facile, il suffit de les énumérer entre parenthèses, lorsque vous appelez la fonction

  function_Name(input)

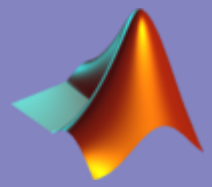
- Vous pouvez aussi utiliser les fonctions sur une partie de la matrices

  >> function_Name(matrix(:, 1))

  or

  >> function_Name(matrix(:, 2:4))

```
>> mean(Results(:, 2:end))

ans =

    0.5900    0.5371    0.5257

>>
```



Les Fonctions Matlab

Le résultat de la fonction peut être mémorisé dans une variable

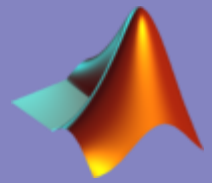
>> output_Variable = function_Name(input)

e.g.

>> mresult = mean(results)

Vous pouvez également indiquer la fonction pour stocker le résultat dans certaines parties d'une matrice

>> matrix(:, 5) = function_Name(matrix(:, 1:4))

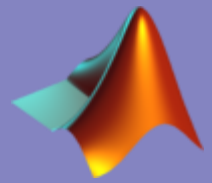


Les Fonctions Matlab

Pour obtenir de l'aide sur une fonction avec le help , entrez

>> help function_Name

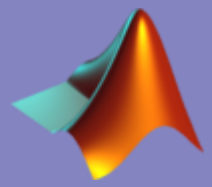
Cela permet d'afficher des informations sur la façon d'utiliser la fonction et ce qu'elle fait



Les Fonctions Matlab

MATLAB a beaucoup de fonctions qui rendent facile l'utilisation d'une série d'opérations statistiques

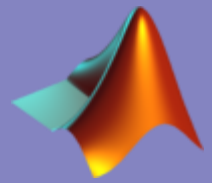
- **sum** – somme le contenu de la variable
- **prod** – Multiplie le contenu de la variable
- **mean** – Calcul la moyenne de la variable
- **median** – Calcul la médiane de la variable
- **mode** – Calcul le Mode de la variable
- **sqrt** – Calcul la racien carré de la variable
- **max** – Détermine la maximum
- **min** – Détermine le minimum
- **std** – Calcul l'écart type de la variable (pour les std(X,0,2) et pour les colone std(X,0,1)
- **size** – Donne la taille de la variable



Les Fonctions Matlab

Il y a un certain nombre de fonctions spéciales qui fournissent des constantes utiles

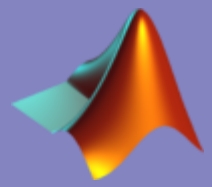
- `pi` = 3.14159265….
- `i` or `j` = square root of -1
- `Inf` = infinity
- `NaN` = not a number



Les Fonctions Matlab

Le passage d'une fonction a un vecteur comme somme, moyenne calculera la propriété dans le vecteur

```
>> sum([1,2,3,4,5])
= 15

>> mean([1,2,3,4,5])
= 3
```

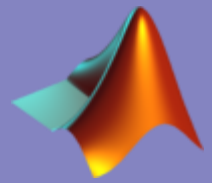


Les Fonctions Matlab

Lors du passage de la fonction aux matrices, la propriété, par défaut, sera calculé sur les colonnes

```
>> results = [1,2,3,4,5;6,7,8,9,0]

results =

     1     2     3     4     5
     6     7     8     9     0

>> sum(results)

ans =

     7     9    11    13     5
```

```
>> mean(results)

ans =

    3.5000    4.5000    5.5000    6.5000    2.5000

>> std(results)

ans =

    3.5355    3.5355    3.5355    3.5355    3.5355
```



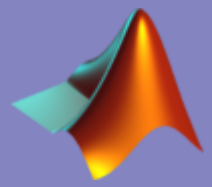
Les Fonctions Matlab

Pour changer la direction du calcul dans l'autre dimension (colonnes) utiliser:

>> function_Name(input, 2)

Quand vous utiliser std, max and min vous avez besoin d'écrire :

>> function_Name(input, [ ], 2)



Les Fonctions Matlab

Au début

```
>> results(:, 5) = results(:, 1) + results(:, 2) + results(:, 3) +
   results(:, 4)
```

or

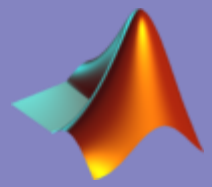
```
>> results(:, 5) = results(:, 1) .* results(:, 2) .* results(:, 3) .*
   results(:, 4)
```

Peut maintenant être écrite

```
>> results(:, 5) = sum(results(:, 1:4), 2)
```

or

```
>> results(:, 5) = prod(results(:, 1:4), 2)
```



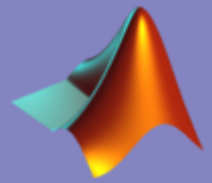
Les Fonctions Matlab

Plus utile, vous pouvez maintenant prendre la moyenne et l'écart type des données, et de les ajouter à la matrice

```
Results =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500
    3.0000    0.5400    0.2000    0.9900
    4.0000    0.7000    0.6000    0.1000
    5.0000    0.8900    0.7300    0.3000
    6.0000    0.2000    0.9000    0.9400
    7.0000    1.0000    0.4000    0.3000
```

```
>> Results(:, 5) = mean(Results(:, 2:4), 2)

Results =

    1.0000    0.5000    0.6000    0.3000    0.4667
    2.0000    0.3000    0.3300    0.7500    0.4600
    3.0000    0.5400    0.2000    0.9900    0.5767
    4.0000    0.7000    0.6000    0.1000    0.4667
    5.0000    0.8900    0.7300    0.3000    0.6400
    6.0000    0.2000    0.9000    0.9400    0.6800
    7.0000    1.0000    0.4000    0.3000    0.5667
```

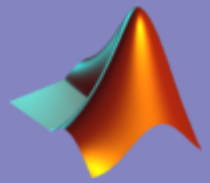
```
>> Results(:, 6) = 2 * std(Results(:, 2:4), [], 2)

Results =

    1.0000    0.5000    0.6000    0.3000    0.4667    0.3055
    2.0000    0.3000    0.3300    0.7500    0.4600    0.5032
    3.0000    0.5400    0.2000    0.9900    0.5767    0.7925
    4.0000    0.7000    0.6000    0.1000    0.4667    0.6429
    5.0000    0.8900    0.7300    0.3000    0.6400    0.6102
    6.0000    0.2000    0.9000    0.9400    0.6800    0.8323
    7.0000    1.0000    0.4000    0.3000    0.5667    0.7572
```



Les Fonctions Matlab

Nous pouvons utiliser les fonctions et l'indexation logique pour extraire tous les résultats pour un sujet qui se situent entre deux écarts types de la moyenne

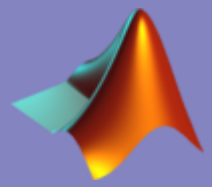
```
>> r = results(:,1)
>> ind = (r > mean(r) – 2*std(r)) & (r < mean(r) + 2*std(r))
>> r(ind)
```



Les Instructions de Contrôle :FOR, WHILE et IF

Les commande de contrôle des boucles d'itération, suivent une syntaxe assez semblable aux langages de programmation classique du style **C** ou **FORTRAN**.

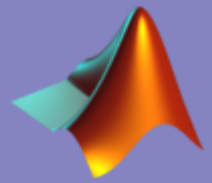
Ce sont les boucles et les branchements. Les boucles permettent de répéter commodément une suite d'instructions ; le nombre de fois est soit connu d'avance (boucles inconditionnelles), soit déterminé au cours de l'exécution (boucles conditionnelles). Les branchements conditionnels (ou tests) permettent de choisir un traitement parmi plusieurs possibles en fonction de critères évalués lors de l'exécution.



Les Instructions de Contrôle :FOR, WHILE et IF

Boucle FOR : parcours d'un intervalle

Une première possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle pour les valeurs d'un indice, incrémenté à chaque itération, variant entre deux bornes données. Ce processus est mis en œuvre par la « boucle for ».



Les Instructions de Contrôle :FOR, WHILE et IF

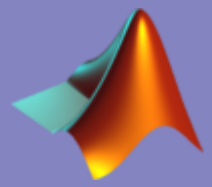
Boucle FOR : parcours d'un intervalle

Syntaxe :

```
for indice = borne_inf : borne_sup
        séquence d'instructions
end
```

Où

- indice est une variable appelée l'indice de la boucle ;
- borne_inf et borne_sup sont deux constantes réelles (appelées paramètres de la boucle) ;
- séquence d'instructions est le traitement à effectuer pour les valeurs d'indices variant entre borne_inf et borne_sup avec un incrément de On parle du corps de la boucle.



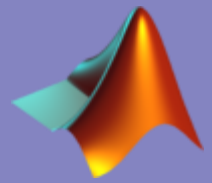
Les Instructions de Contrôle :FOR, WHILE et IF

Boucle FOR : parcours d'un intervalle

Exemple :

```
x=1;
for k=1:4
k
x=x*k
end
```

```
>> x=1; for k=1:4,x=x*k, end

x =

     1


x =

     2


x =

     6


x =

    24
```

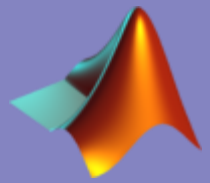


Les Instructions de Contrôle :FOR, WHILE et IF

Boucle FOR : parcours d'un intervalle

Boucles imbriquées : ces boucles peuvent être imbriquées comme dans l'exemple ci-dessous (calcul des 6 premières lignes du triangle de Pascal)

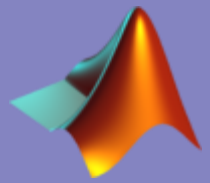
```
A=zeros(6);
        for i=1:6
                A(i,1)=1;
                for j=2:i
                A(i,j)=A(i-1,j-1)+A(i-1,j);
                end
        end
```



Les Instructions de Contrôle :FOR, WHILE et IF

Boucle WHILE : tant que . . . faire

Une seconde possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle tant qu'une condition reste vérifiée. On arrête de boucler dès que cette condition n'est plus satisfaite. Ce processus est mis en œuvre par la « boucle while ».



Les Instructions de Contrôle :FOR, WHILE et IF

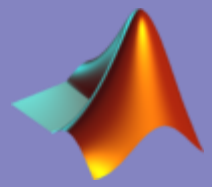
Boucle WHILE : tant que . . . faire

Syntaxe :

```
while expression logique
séquence d'instructions
end
```

où

- expression logique est une expression dont le résultat peut être vrai ou faux ;
- séquence d'instructions est le traitement à effectuer tant que expression logique est vraie.



Les Instructions de Contrôle :FOR, WHILE et IF

Boucle WHILE : tant que . . . faire

Exemple

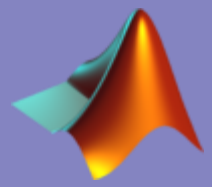
```
x=1;
while x<14,
x=x+5,
end
```

```
>> x=1;
    while x<14
    x=x+5,
    end

x =

     6

x =

    11

x =

    16
```

Les Instructions de Contrôle :FOR, WHILE et IF

L'instruction conditionnée IF

On a parfois besoin d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée au préalable. Différentes formes d'instruction conditionnée existent sous matlab.



Les Instructions de Contrôle :FOR, WHILE et IF

L'instruction conditionnée IF

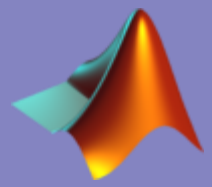
L'instruction conditionnée la plus simple a la forme suivante:

Syntaxe :

```
if expression logique
   séquence d'instructions
end
```

Où:

- expression logique est une expression dont le résultat peut être vrai ou faux ;
- séquence d'instructions est le traitement à effectuer si expression logique est vraie.



Les Instructions de Contrôle :FOR, WHILE et IF

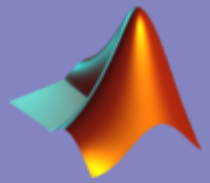
L'instruction conditionnée IF

Il existe une séquence conditionnée sous forme d'alternatives :

Syntaxe :

```
if expression logique
    séquence d'instructions 1
else
    séquence d'instructions 2
end
```

- expression logique est une expression dont le résultat peut être vrai ou faux ;
- séquence d'instructions 1 est la séquence d'instructions à exécuter dans le cas où expression logique est vraie et séquence d'instructions 2 est la séquence d'instructions à exécuter dans le cas où expression logique est faux.

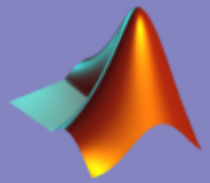


Les Instructions de Contrôle :FOR, WHILE et IF

L'instruction conditionnée IF

Exemple

```
x=16
        if x>0,
           y=-x,
        else
           y=x,
        end
```



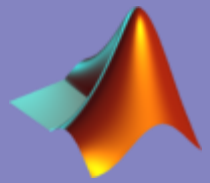
Les Instructions de Contrôle :FOR, WHILE et IF

Choix ventilé, l'instruction IF

Il est possible d'effectuer un choix en cascade :

Syntaxe :

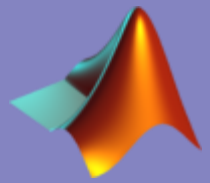
```
if expression logique 1
        séquence d'instructions 1
elseif expression logique 2
        séquence d'instructions 2
...
elseif expression logique N
        séquence d'instructions N
else
        séquence d'instructions par défaut
end
```



Les Instructions de Contrôle :FOR, WHILE et IF

Choix ventilé, l'instruction switch

Une alternative à l'utilisation d'une séquence d'instructions conditionnées pour effectuer un choix en cascade. Il s'agit de l'instruction switch.

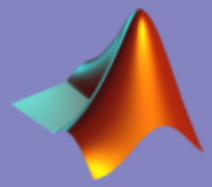


Les Instructions de Contrôle :FOR, WHILE et IF

Choix ventilé, l'instruction switch

Syntaxe :

```
switch var
        case cst_1 ,
        séquence d'instructions 1
        case cst_2 ,
        séquence d'instructions 2
        ...
        case cst_N ,
        séquence d'instructions N
otherwise
séquence d'instructions par défaut
end
```

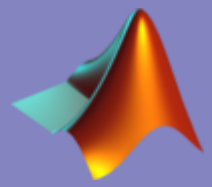


Les Instructions de Contrôle :FOR, WHILE et IF

Choix ventilé, l'instruction switch

Où

- var est une variable numérique ou une variable chaîne de caractères ;
- cst_1, . . ., cst_N, sont des constantes numérique ou des constantes chaîne de caractères;
- séquence d'instructions i est la séquence d'instructions à exécuter si var==cst_i.

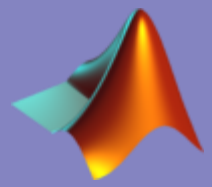


Les Instructions de Contrôle :FOR, WHILE et IF

Choix ventilé, l'instruction switch

Exemple

```
n=round(10*rand(1,1))
switch n
case 0
   fprintf('cas numero 0')
case 1
          fprintf('cas numero 1')
case 2
         fprintf('cas numero 2')
otherwise
          fprintf('autre cas')
end
```



Plotting (Tracé)

La fonction plot (tracé) peut être utilisé de différentes façons:

```
>> plot(data)
>> plot(x, y)
>> plot(data, 'r.-')
```

Dans le dernier exemple, le style de trait est définie par:

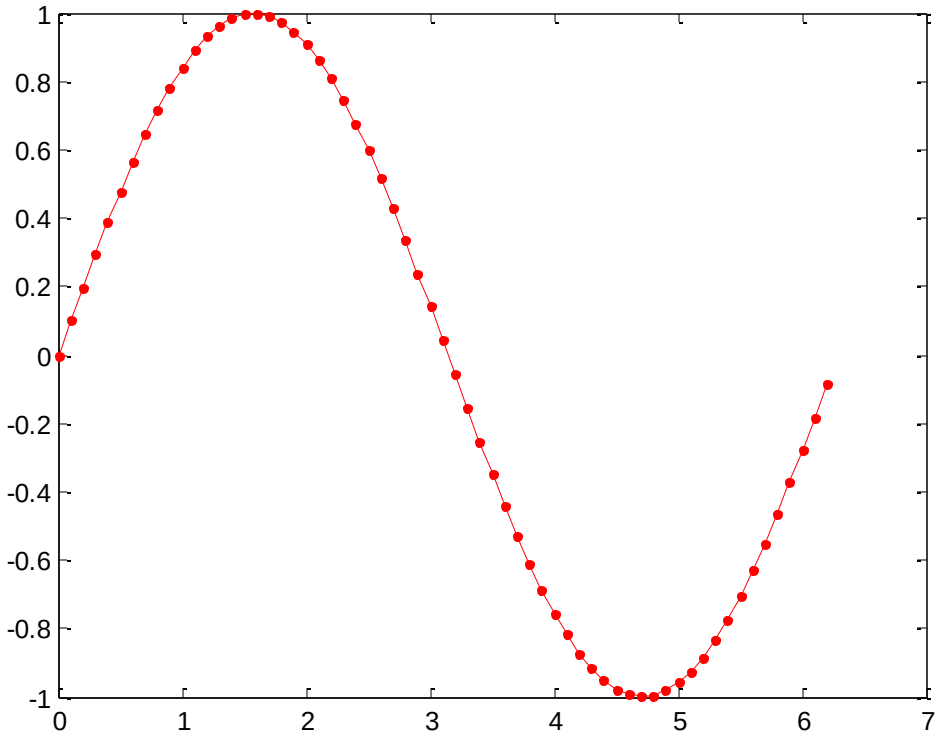
- Colour: r, b, g, c, k, y etc.
- Point style: . + * x o > etc.
- Line style: - -- : .-

Tapez 'help plot' pour une liste complète des options

Plotting (Tracé)

Tracé de base

```
>> x = [0:0.1:2*pi]
>> y = sin(x)
>> plot(x, y, 'r.-')
```

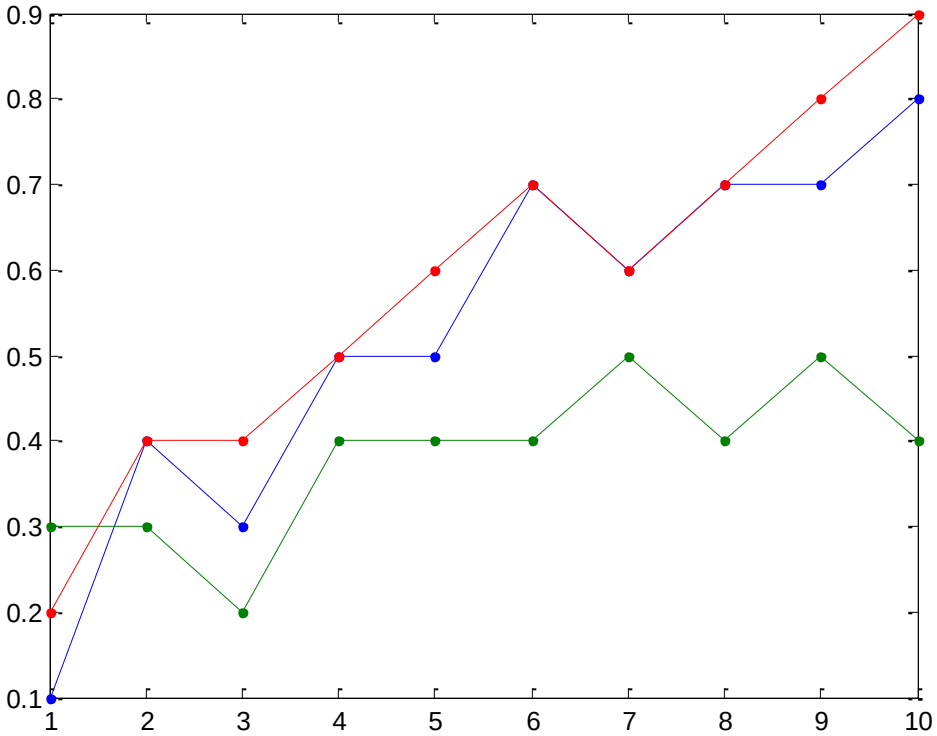


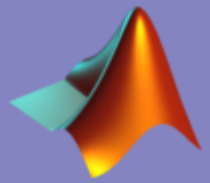
Plotting (Tracé)

Tracé d'une matrice

MATLAB permettra de traiter chaque colonne comme un ensemble différent de données

```
results =

    0.1000    0.3000    0.2000
    0.4000    0.3000    0.4000
    0.3000    0.2000    0.4000
    0.5000    0.4000    0.5000
    0.5000    0.4000    0.6000
    0.7000    0.4000    0.7000
    0.6000    0.5000    0.6000
    0.7000    0.4000    0.7000
    0.7000    0.5000    0.8000
    0.8000    0.4000    0.9000

>> plot(results, '.-')
```

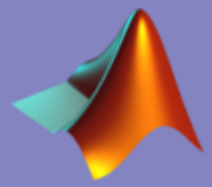




Plotting (Tracé)

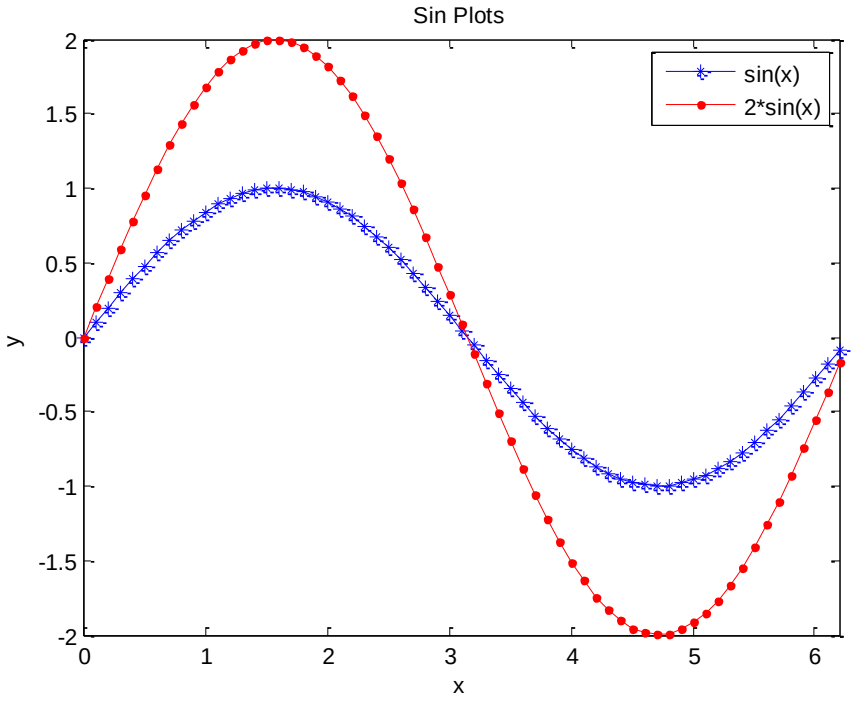
Quelques autres fonctions qui sont utiles pour créer des tracés:

- **hold on and hold off**
- **title**
- **legend**
- **axis**
- **xlabel**
- **ylabel**



Plotting (Tracé)

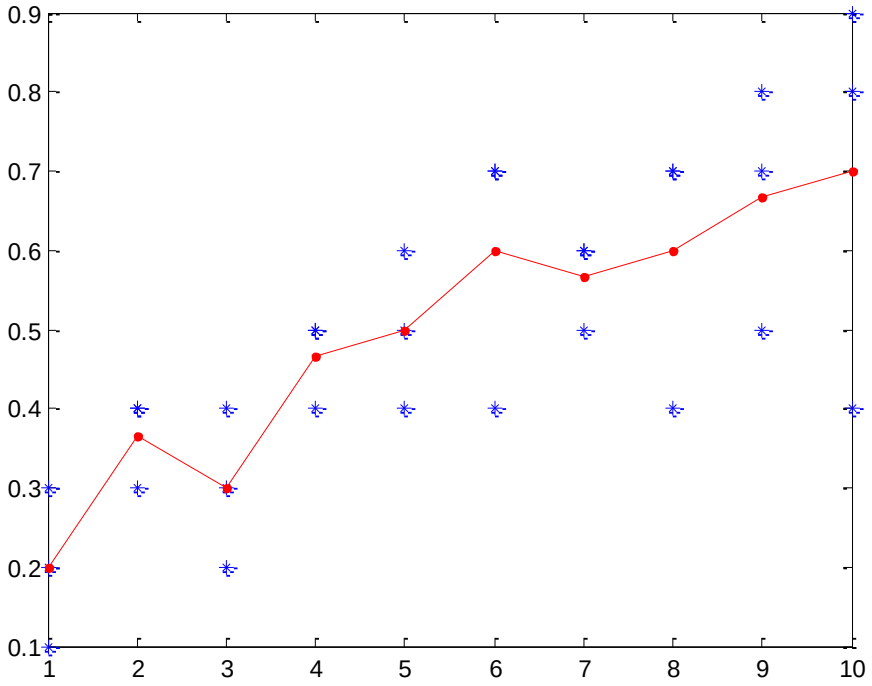
```
>> x = [0:0.1:2*pi];
>> y = sin(x);
>> plot(x, y, 'b*-')
>> hold on
>> plot(x, y*2, ‘r.-')
>> grid on

>> title('Sin Plots');
>> legend('sin(x)', '2*sin(x)');
>> axis([0 6.2 -2 2])

>> xlabel(‘x’);
>> ylabel(‘y’);
>> hold off
```



Plotting (Tracé)

Tracé de données

```
results =

    0.1000    0.3000    0.2000
    0.4000    0.3000    0.4000
    0.3000    0.2000    0.4000
    0.5000    0.4000    0.5000
    0.5000    0.4000    0.6000
    0.7000    0.4000    0.7000
    0.6000    0.5000    0.6000
    0.7000    0.4000    0.7000
    0.7000    0.5000    0.8000
    0.8000    0.4000    0.9000

>>
```

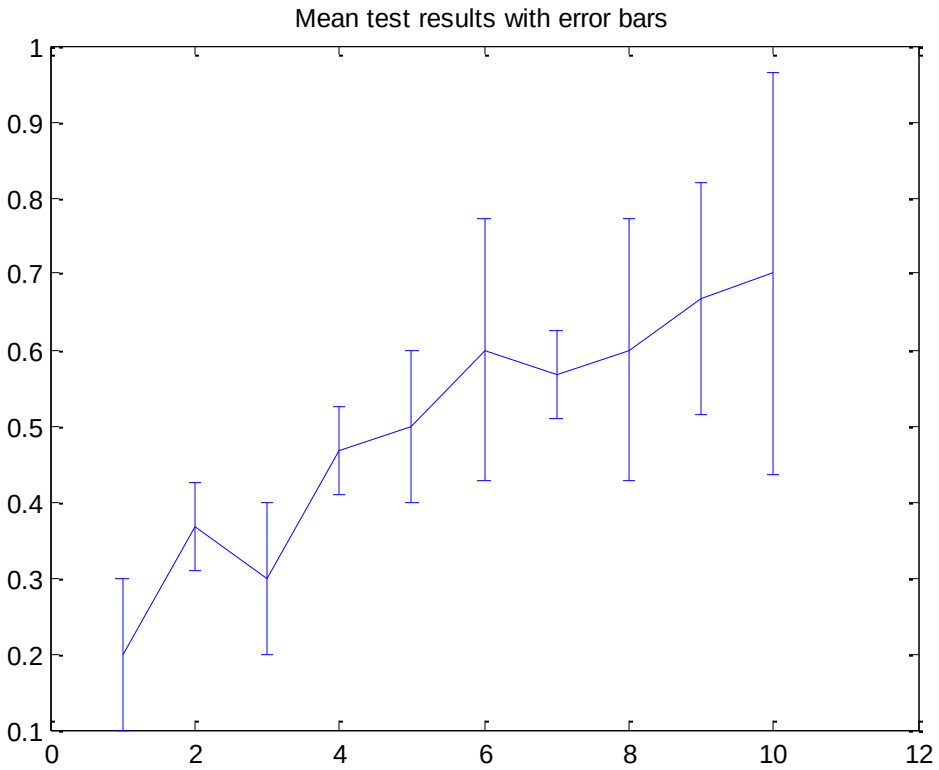



```
>> results = rand(10, 3)
>> plot(results, 'b*')
>> hold on
>> plot(mean(results, 2), 'r.-')
```

Plotting (Tracé)

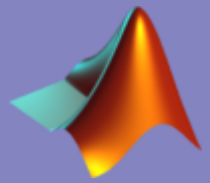
Tracé error bar

>> errorbar(mean(data, 2), std(data, [], 2))



Plotting (Tracé)

Vous pouvez fermer toutes les graphes en cours en utilisant ‘close all’



Plotting (Tracé)

Subplot

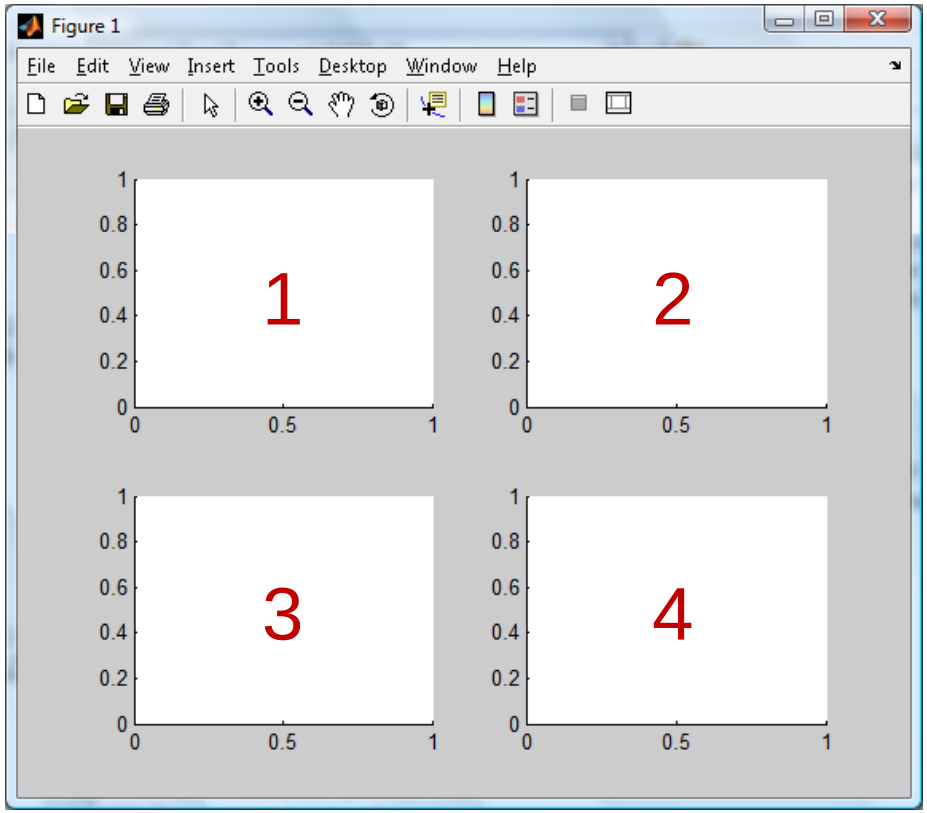
- De multiple tracés (plots) peuvent être placé dans la même fenêtre
- Cette tache peut etre effectuer avec la commande **subplot**
- La fenêtre est divisée en une grille dont la taille est spécifiée lors de la saisie de la commande

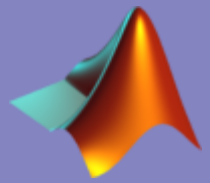
```
>> subplot(2,2,1)
```

Cela crée une grille de 2 x 2 de taille (4 tracés) et définit le tracé actuel pour le premier de ceux-ci.

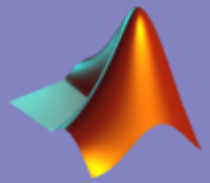
Plotting (Tracé)

```
>> subplot(2,2,1)
```





Plotting (Tracé)



Plotting (Tracé)

Merci de votre attention !