UNIVERSITÉ BADJI MOKHTAR – ANNABA

FACULTÉ DES SCIENCES DE L'INGÉNIORAT

DÉPARTEMENT D'HYDRAULIQUE

Masters

HYDRAULIQUE URBAINE
AMENAGEMENTS ET OUVRAGES HYDRAULIQUES

Cours

Initiation à Matlab

Notes de cours
enrichis avec des applications en hydrologie

Dr. BOUTAGHANE Hammouda

Courriel :

initialtion.matlab@gmail.com

Boutaghane.hamouda@univ-annaba.org

2015

SOMMAIRE

1 PRESENTATION ET GENERALITES

1.1 PRESENTATION

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.)

1.1.1 Lancer/ Arreter Matlab

- Pour Lancer : Sous Windows sélectionner

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

- Pour Arrêter : Sélectionner `File -> Exit MATLAB`
  Ou taper `quit` dans le menu MATLAB command window

1.1.2 Interface Matlab

Matlab est à la fois un langage et un logiciel. Il possède une fenêtre principale ou de commande et un éditeur de programmes (M-files). Dans l'éditeur de programme, plusieurs fenêtres (donc programmes) peuvent être ouvertes à la fois. Un éditeur de figure est aussi disponible pour les graphiques.

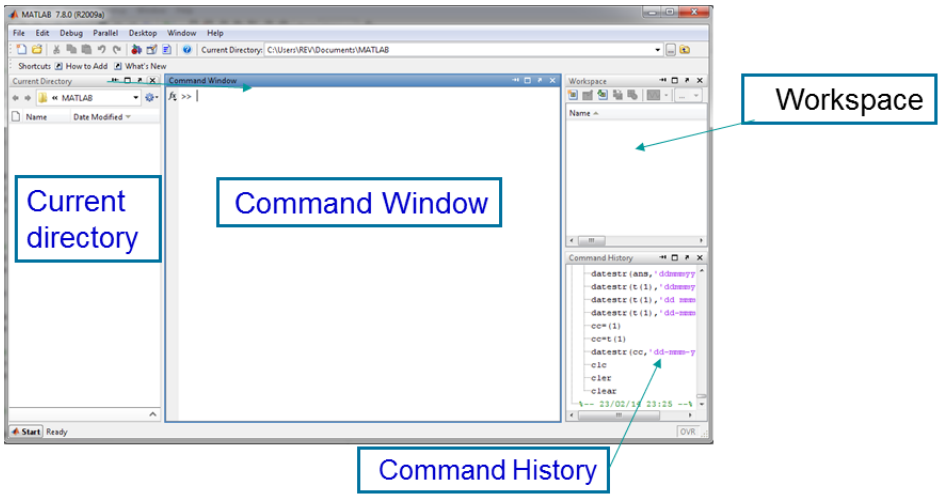


Figure 1 : présentation de l'interface Matlab

1.2 RICHESSE DE MATLAB : SES FONCTIONS PREPROGRAMMEES

Matlab est doté d'une collection de fonctions (m-files) préprogrammées (notamment dans ses Toulboxes) spécifiques à des domaines aussi variés que les statistiques, le traitement du signal et d'image, la logique floue, les réseaux de neurones, les ondelettes, … et qui permettent de résoudre un bon nombre de problèmes relatifs à ces domaines. Pour visualiser ces fonctions, il suffit de taper **`help`** suivi du non de la famille à laquelle appartient la fonction. Pour connaître le nom de ces familles, il suffit juste de taper **`help.`**

Plus de 1500 fonctions préprogrammées :

- Générale (**`help general`**) : help, demo, dir, cd, !, clear, whos, clear, clc,…
- Opérateurs (**`help ops`**) : +, -, *, /, ^, =, ~=, <, >, &, | , ~, .*, ./, .^, …
- Langage (**`help lang`**) : if, else, for, while, case, …
- fonctions élémentaires sur les matrices (**`help elmat`**) : rand, ones, size, diag, ', …
- fonctions mathématiques élémentaires (**`help elfun`**) : cos, tan, sin, sinh, asin, asinh, exp, log, log10, round, …
- fonctions mathématiques spécialisées (**`help specfun`**) : airy, gcd, lcm, factorial, cart2sph, cart2pol,…
- fonctions sur les matrices (**`help matfun`**) : norm, trace, det, inv, eig, …
- analyse de données (**`help datafun`**) : max, min, hist, diff, corrcoef, conv, conv2, …
- polynômes et interpolations (**`help polyfun`**) : interp, interp2, spline, voronoi, polyarea, roots, polyfit, …

Et beaucoup plus dans les toolboxes :

- traitement d'images (`help images`) : imresize, imcontour, edge, histeq, filter2, fft2, bwlabel, colormap, image, …
- statistiques (`help stats`) : betafit, weibfit, betapdf, chi2pdf, poisspdf, betacdf, poisscdf, betarnd, poissrnd, mean, std, kurtosis, skewness, harmmean, geomean, ztest, ttest, polyfit, regress, cluster, dendrogram, princomp,
- réseaux de neurones (`help nnet`) : network, newelm, initlay, trainbfg, …
- logique floue (`help fuzzy`) : mfedit, ruleedit, surview, anfisedit, …
- ondelettes (`help wavelet`): wavemenu, appcoef, dwt, …
- mapping toolbox (`help map`) : antipode, distance, areaint, intrplat, intrplon, distortcalc, mfwdtran, deg2km, …

2 TYPES DE DONNEES ET VARIABLES

2.1 VARIABLES, VECTEURS ET MATRICES:

2.1.1 Création de variables

2.1.1.1 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 celle des commandes Matlab telle que : IF, WHILE, ELSE, END, SUM, etc

2.1.1.2 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.

2.1.2 Création de variables: Variable simple

Pour affecter une valeur à une variable utiliser le symbole égale '**=**'

Exemple : `>> A = 32`

```
>> A = 32

A =

    32

>>
```

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

Exemple : `>> A`

```
>> A

A =

    32

>>
```

La valeur de deux variables peuvent être additionné et le résultat sera visionner sur écran

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

```
>> A = 10

A =

     10

>> A + A

ans =

     20

>> B = A + A

B =

     20

>>
```

où le résultat peut être stocké dans une autre variable

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

2.1.3 Création de Vecteurs

- Le vecteur contient plusieurs nombres Utiliser les crochets **[ ]** pour contenir les nombres
- Pour crier un vecteur ligne utiliser la virgule ' **,** ' 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

>>
```

	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 une vecteur colonne utiliser le point-virgule ' **;** ' to 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

>>
```

	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		

- Un vecteur ligne peut être converti en vecteur colonne en utilisant l'opérateur 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

>>
```

2.1.4 Création de Matrices

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

```
>> 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

>>
```

	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					

- Vous pouvez également utiliser des fonctions intégrées pour créer une matrice

```
>> A = zeros (2, 4)
```

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

```
>> A = zeros(5)
```
ou
```
>> A = zeros(5, 5)
```

Génère une matrice appelée A avec 5 lignes et 5 colonnes

Nous pouvons également utiliser les fonctions intégrées de Matlab pour créer une matrice telque

```
>> ones(lignes, colonnes)
>> rand(lignes, colonnes)
>> magic(n)
```

NB: MATLAB se réfère toujours à la première valeur que le nombre de lignes puis la seconde que le nombre de colonnes. Pour la fonction magic, elle génère une matrice rectangulaire.

2.1.5 Effacer une variable

Vous pouvez utiliser la commande "`clear all`" pour effacer toutes les variables présentes dans le workspace

Vous pouvez effacer une variable en spécifiant son nom:

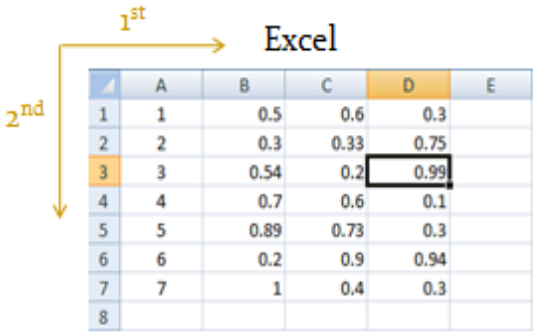
```
>> clear Variable_Name
```

2.1.6 Accès aux éléments des matrices

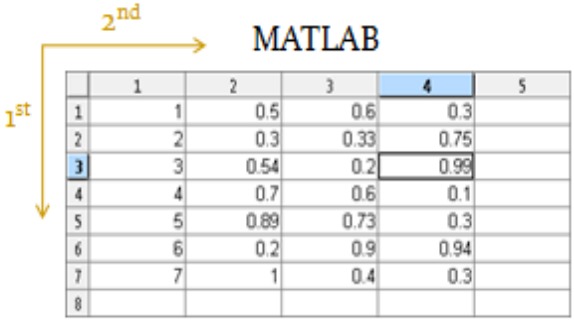
- Un élément est un numéro unique au sein d'une matrice ou d'un vecteur;
- Pour accéder aux éléments d'un type de matrice: Le nom de la matrice suivie par des parenthèses contenant une référence au numéro de ligne et de colonne:
-

```
>> Nom_Variable (Numero_Ligne , Numero_Colonne)
```

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



	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 accéder au résultat de l'exemple

Dans Excel (Colonne, Ligne): `D3`

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

2.1.7 Modification des éléments d'une matrice

- L'élément référencé peut également être modifié

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

ou

```
>> 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

>>
```

2.1.7.1 Accès aux lignes d'une matrice

- Vous pouvez également accéder à plusieurs valeurs à partir d'une matrice en utilisant le symbole colon :
  - 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

>>
```

2.1.7.2 *Accès aux colonnes d'une matrice*

Pour accéder à 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

>>
```

2.1.7.3 *Modification des lignes ou des colonnes d'une matrice*

Ces méthodes de référencement 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(:, 5) = results(:, 3) + results(:, 4)
```

```
>> 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 =

    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

>>
```

❖ 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

>>
```

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

2.1.7.4 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, :)
```

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

ans =

    1.0000    0.5000    0.6000    0.3000
    2.0000    0.3000    0.3300    0.7500

>>
```

✓ Plusieurs Colonnes:

```
>> Nom_Variable(:, start:end)
```

```
>> 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

>>
```

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

✓ Plusieurs Lignes:

```
>> 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

>>
```

✓ Multiples Colonnes:

```
>> Nom_Variable(:, [index1, index2, etc.])
```

```
>> 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

>>
```

2.1.7.5 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

>>
```

2.1.8 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
```

2.1.9 Format de l'Affichage

Tous les calculs dans Matlab sont effectués en double précision. Il existe plusieurs format d'affichage. Nous pouvons switcher entre les différents formats en les activant. Pour activer un format , il faut taper son code dans Matlab. Les formats les plus utilisés sont :

`format short` (active par default) le format d'affichage est fixé à 5 scalaires

`format long` le format d'affichage est fixé à 15 scalaires

Exemple

```
>> pi
ans =
3.1416
>> format long g
>> pi
ans =
3.14159265358979
```

Pour connaitre les autres formats tapez : `help format`

2.1.10 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 …]`

2.1.11 Plus d'operateurs

2.1.11.1 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

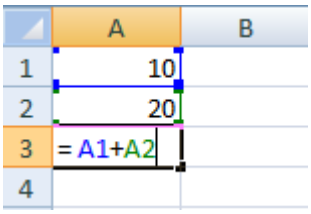
Notez: quand on précède le symbole **/** ou ***** ou **^** ou par un " **.** " cela signifie que l'opérateur est appliqué entre des paires d'éléments de vecteurs de matrices

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
```

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

>>
```

En combinant cela avec les méthodes d'accès aux éléments de matrice , ça 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)
```

Ou

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

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

Exemple:

```
>> 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

>>
```

2.1.11.2 Opérateurs Logiques

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

Les Opérateurs logiques de Matlab sont :

Supérieur à (Greater Than) : **>**

Inférieur à (Less Than) : **<**

Supérieur ou égale à (Greater Than or Equal To) : **>=**

Inférieur ou égale à (Less Than or Equal To): **<=**

Egale à (Is Equal) : **==**

Différent de (Not Equal To) : **~=**

2.1.11.3 Opérateurs Booléens

Les opérateurs Booléens Connecte deux expressions logiques ensemble.

Et (AND): **&**

Ou (OR): **|**

Non (NOT): **~**

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

Exemple

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

3 STOCKAGE DES MATRICES EN MEMOIRE

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

3.1 METHODES D'INDEXAGE

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

3.1.1 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).
```

```
>> 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
```

3.1.2 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.

```
>> 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
```

- 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)

3.1.3 Indexage logique

- ✓ Il est basé 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`**.

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

```
>> 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
```

ou pour obtenir les indices linéaires comme ceci :

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

idx =

     1
     4

>> X(idx)

ans =

     8
     7
```

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
```

3.2 LES OUTILS D'INDEXAGE

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

3.2.1 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.

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

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

M =

     4     7     2

>> M(end)

ans =

     2
```

```
>> 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
```

3.2.2 L'opérateur Colon :

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
```

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
```

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
```

```
>> 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
```

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
```

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
```

3.3 LES OPERATIONS COURANTES

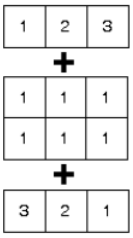
3.3.1 Concaténation

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.

3.3.1.1 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.



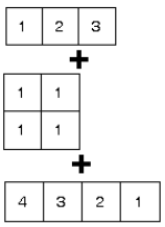
Concaténation verticale de trois matrices

Par exemple, pour concaténer verticalement les trois matrices suivantes

```
>> 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
```

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



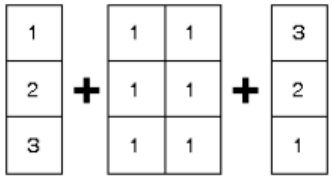
Concaténation verticale de matrices impossible

```
>> 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.
```

3.3.1.2 Concaténation horizontale

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

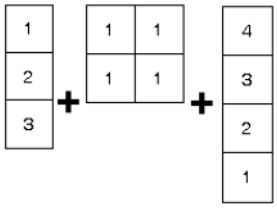


Concaténation horizontale de trois matrices

```
>> 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
```

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

X =

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

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



Concaténation horizontale 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
```

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

C =

     4
     3
     2
     1

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

4 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 matrice

```
>> function_Name(matrix(:, 1))
```

ou

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

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

ans =

    0.5900    0.5371    0.5257

>>
```

Le résultat de la fonction peut être mémorisé dans une variable

```
>> output_Variable = function_Name(input)
```

Exemple

```
>> 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))
```

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
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

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

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
```

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
```

```
>> mean(results)

ans =

    3.5000    4.5000    5.5000    6.5000    2.5000

>> sum(results)

ans =

     7     9    11    13     5

>> std(results)

ans =

    3.5355    3.5355    3.5355    3.5355    3.5355
```

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)
```

Au début

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

Ou

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

Peut maintenant être écrite

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

Ou

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

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
```

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)
```

5 LES INSTRUCTIONS DE CONTROLE : FOR, WHILE ET IF

Les commandes 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.

5.1 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` ».

5.1.1 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.

Exemple :

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

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

Exemple

```
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
```

5.2 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`**.

5.2.1 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

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
```

5.3 L'INSTRUCTION CONDITIONNEE 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.

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

5.3.1 Syntaxe 1:

```
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.

Il existe une séquence conditionnée sous forme d'alternatives :

5.3.2 Syntaxe 2:

```
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.

Exemple

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

5.4 CHOIX VENTILE, L'INSTRUCTION IF

Il est possible d'effectuer un choix en cascade :

5.4.1 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
```

5.5 CHOIX VENTILE, 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.`

5.5.1 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
```

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.

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
```

```
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
```

6 IMPORTATION/EXPORTATION DE DONNEES VERS TABLEUR

Le type fondamental de données de Matlab est le tableau/matrice. Ce type est très proche des feuilles Excel destinées à créer des tables de données. De nombreux autres logiciels utilisent des formats d'échange voisins des concepts utilisés sur Excel.

L'échange de données peut se faire de plusieurs manières:

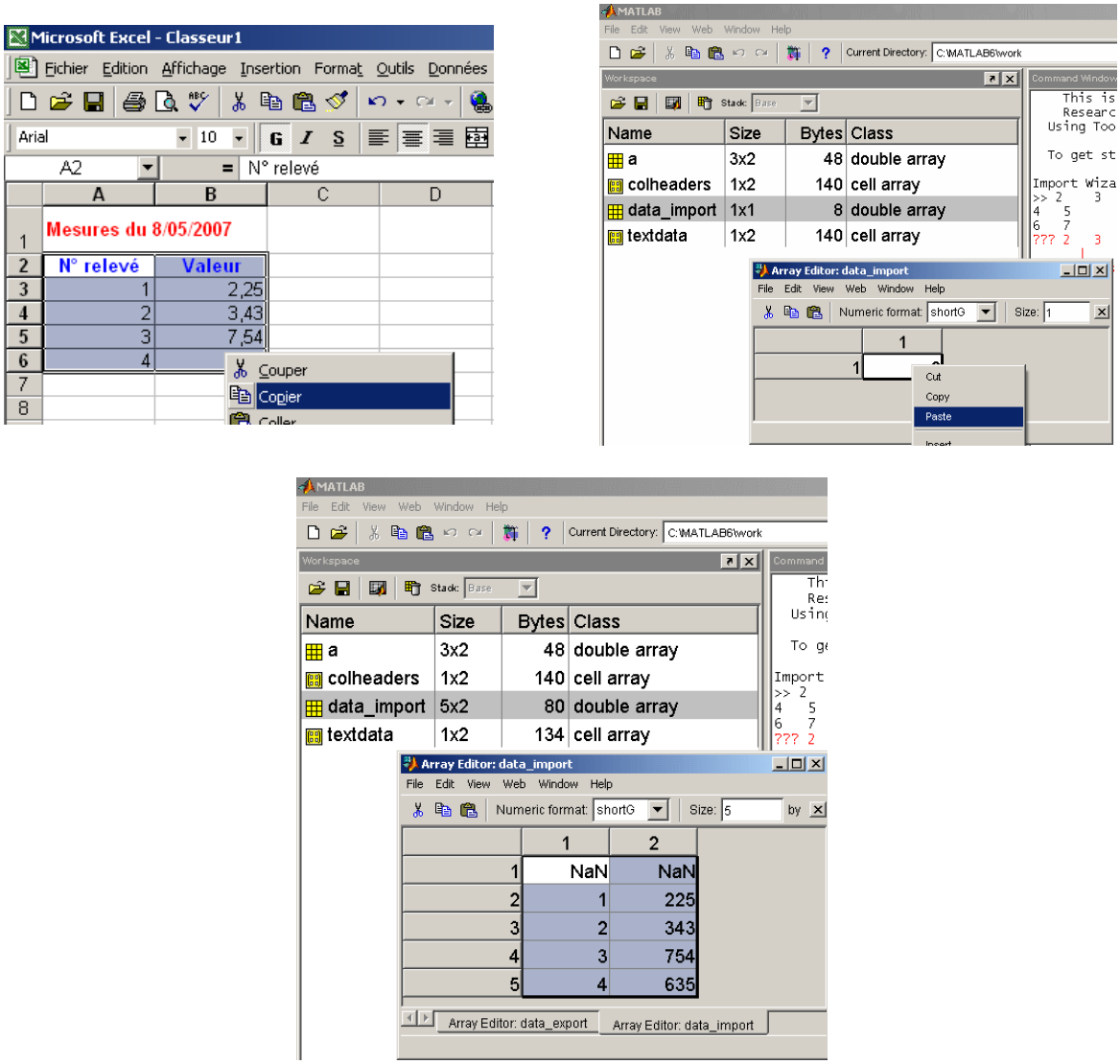
- par la méthode "copier/coller" d'un logiciel vers l'autre (peu conseillé).
- par enregistrement sur disque (fichier) depuis l'application source des données puis relecture sur l'autre application. Matlab met à disposition de l'utilisateur divers outils d'automatisation (wizard, opérations déclenchées par programmation/macro-commande)
- par échange direct de données d'une application à l'autre sous forme échange de type DDE (Direct Dynamic Exchange).

6.1 IMPORTATION ET EXPORTATION DIRECTE PAR COPIER/COLLER

Les données de Matlab étant des tableaux analogues de ceux d'Excel, il semble possible d'utiliser directement "copier/coller". Attention: seules les données numériques sont "collables".

6.1.1 Excel ----------> Matlab

- Sélectionner dans Excel les données à transférer, puis faire "copier"
- Dans Matlab, créer une variable non-vide, par exemple data_import = 0; (les données initiales seront écrasées par le collage).
- Sélectionner cette variable dans le workspace et demander son édition (double clic sur la variable). Par simple "clic droit" dans l'éditeur de la variable d'importation, sélectionner "paste" (collage). Les données sont recopiées dans la variable Matlab



Le collage sera en échec si les données ne sont pas utilisables par Matlab (données textuelles d'en-tête de colonne par exemple). Ces données sont signalées comme "**NaN**" (Not a Number) .

6.1.2 Matlab ----------> Excel

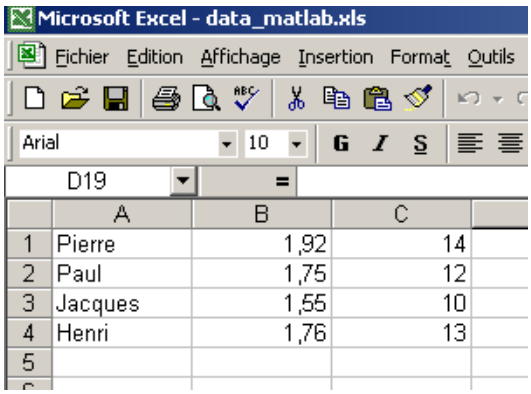
Le "copier/coller" Matlab----> Excel pose encore plus de problème : la virgule est mal interprétée ainsi que les exposants (les nombres sont restitués sous forme chaîne!)

6.2 IMPORTATION PAR FICHIER EXCEL

Matlab peut lire directement un fichier Excel, sous réserve que son contenu représente un tableau de données. La procédure de travail est la suivante:

- mettre les données en tableau dans une feuille Excel (un seul tableau sur une seule feuille dans le classeur), de préférence sans en-tête (ne pas mettre ni titre dans la feuille, ni commentaire ni tout autre information que le tableau des données).
- enregistrer les données Excel avec l'option "Excel97"
- reprendre ces données sous Matlab avec l'instruction:

```
mes_données = xlsread('mon_fichier.xls')
```




```
>> [data , noms] =
xlsread('c:\data_matlab.xls')

data = 1.9200 14.0000
1.7500 12.0000
1.5500 10.0000
1.7600 13.0000

noms = 'Pierre'
'Paul'
'Jacques'
'Henri'
```

Sur version 2007, il est possible de sélectionner la feuille et une partie d'un tableau :

```
donnees = xlsread('c:\mafeuille.xls' , 'feuille2' , 'a2:j5' )
```

La relecture des dates pose aussi problème, étant donné des formats différents. La proposition de Matlab est la suivante :

```
excelDates = xlsread('mon_fichier.xls')
matlabDates = datenum( '30-Dec-1899' ) + excelDates
les_dates = datestr(matlabDates , 2)
les dates 04/12/00
04/13/00
04/14/00
```

6.3 IMPORTATION PAR FICHIER TEXTE

L'importation par fichier texte est une solution simple et éprouvée. Elle permet de transférer de grandes quantités de données *numériques*. La procédure de travail est la suivante:

- mettre les données dans une feuille Excel (une seule feuille dans le classeur), de préférence sans en-tête (ne pas mettre ni titre dans la feuille, ni commentaire ni tout autre information que le tableau des données).
- enregistrer les données Excel avec l'option "Texte" (format txt) ou "Texte avec séparateur" (format csv) ;
- reprendre ces données sous Matlab avec l'instruction :

```
load c:\mes_datas.txt  %la variable créée prend le nom du fichier
```

ou

```
mes_datas = load c:\monfichier.txt
```

Les difficultés sont nombreuses (point décimal...) ; il est conseillé de passer par un éditeur de texte (Notepad, Wordpad ou autre...) pour mettre en forme visuellement les données.

6.4 LECTURE PAR FONCTIONS ENTREE/SORTIE MATLAB

Pour des fichiers particuliers, la lecture de fichiers texte peut se faire ligne à ligne avec analyse de la ligne lue pour en retirer les données utiles.

La lecture ligne à ligne d'un fichier texte se fait par :

```
id_fich = fopen('mon_fichier.txt')
ligne_lue = ' ' ;
while (ligne_lue~=-1)
ligne_lue = fgets(id_fich);
mon_analyse(ligne_lue)
end
fclose(id_fich)
```

L'analyse des lignes lues pourra utiliser la fonction `sscanf` (lecture de données formatées) ou les fonctions `strfind` ou `strread` pour rechercher dans la ligne les "balises" permettant de récupérer les données (convertir les données textuelles numériques en nombre par `str2num`)

6.5 EXPORTATION DE DONNEES

Les principes sont voisins de ceux de l'importation par fichier. Les deux formats sont :

- exportation directe en format Excel d'un tableau Matlab (sous réserve que Excel soit installé):

```
no_erreur = xlswrite('c:\monclasseur.xls',mon_tableau_Matlab,'Feuillle1','A2:C4' )
```

Le tableau peut contenir des nombres ou des chaînes; il est possible d'exporter aussi des structures "cells". Si le bloc de cellule est de taille insuffisante, seule la partie correspondante du tableau est copiée. Le nom de la feuille peut être omis; Matlab crée automatiquement une nouvelle feuille si le fichier Excel existe déjà. En l'absence d'Excel, la sauvegarde est faite en format csv .

- exportation en format texte d'un tableau

```
save nom_du_fichier.txt ma_matrice -ascii
```

6.6 ECHANGE DYNAMIQUE DDE

L'échange dynamique a pour objet l'écrire directe de données dans une feuille Excel depuis Matlab ou inversement la lecture directe depuis une feuille.

La liaison directe s'appelle un lien. C'est une structure d'échange dans laquelle les deux applications Matlab et Excel sont ouvertes simultanément. Matlab peut écrire directement dans les cellules Excel, sans intervention de l'opérateur sur Excel. Les données écrites sur la feuille Excel peuvent être sauvegardées comme pour toute feuille Excel.

6.6.1 - Ecriture dans une feuille Matlab:

```
mon_channel = ddeinit('Excel' , 'mon_classeur.xls')
ok = ddepoke(mon_channel, 'r5c1:r10c5', ma_matrice);
ok = ddeterm(mon_channel)
```

6.6.2 - Lecture depuis une feuille Matlab:

```
mon_channel = ddeinit('Excel' , 'mon_classeur.xls')
ma_matrice = ddereq(mon_channel, 'r1c1:r5c5');
ok = ddeterm(mon_channel)
```

Ces techniques d'échange ont un intérêt pour utiliser Matlab comme outil de calcul sur des données Excel qui sert d'interface utilisateur.

7 GESTION DES DATES ET HEURES

7.1 DIFFERENTS FORMATS DE DATES ET HEURES

Matlab gère la date sous 3 types de format différents :

1. chaînes de caractères,
2. nombres,
3. vecteurs.

La commande **date** retourne la date du jour sous forme d'une chaine de caractères.

```
>> date
ans =
09-Aug-2009
```

Matlab gère aussi les dates sous forme de nombres avec la commande **now**

```
>> now
ans =
  7.3399e+005
```

La commande **now** retourne la date et l'heure actuelle sous la forme d'un seul nombre dont l'entier inférieur correspond à la date du jour et le reste à l'heure actuelle.
Le nombre correspond au nombre de jours passés depuis une date de base. Dans Matlab, cette date de base est le 1er Janvier 2000.

7.1.1 Format string des date

Datestr Transforme une date en nombre sous forme d'une chaîne de caractères.

```
>> datestr(1)
ans =
01-Jan-0000
```

Appliquée directement à la commande **now, datestr** donne l'heure exacte.

```
>> maintenant=datestr(now)
maintenant =
09-Aug-2009 13:45:54
```

On peut spécifier un nombre entre 0 et 14 pour designer la forme de la date.

```
>> maintenant=datestr(now,2)
maintenant =
08/09/09
```

```
>> maintenant=datestr(now,14)
maintenant =
08:56:53 PM
```

7.1.2 Format numérique des dates

Datenum Transforme la date sous forme d'une chaîne en date sous forme de nombre.

```
>> datenum('31-Aug-2009')
ans =
      734016
```

7.1.3 Format vectoriel des dates

`Datevec` Cette commande permet de transformer une date en nombre sous forme d'une date en vecteur.

```
>>  datevec(now)

ans =
  1.0e+003 *
  2.0090    0.0080    0.0090    0.0230    0.0120    0.0401
```

Ce vecteur contient les informations suivantes : [Année mois jour heure minute second].

Tout comme pour le format nombre, la commande `datestr` peut donner le format chaine de caractères

7.1.4 Autres fonctions utiles de mesure du temps

`etime` La commande `etim(t2,t1)` retourne, en secondes, la différence des temps t2 et t1 spécifiés sous le format vecteurs.

```
x = rand(1000000, 1); t = clock;  y=exp(log(x.^2));
TempsMis = etime(clock, t)
TempsMis =
    0.3120
```

`tic` : Initialise le timer.

`toc`:Stoppe le timer et affiche le temps mis depuis le temps passé en argument.

```
t1= tic ;
x=randn(1000);
TempsMis=toc(t1)
TempsMis =
    0.0680
```

`Datetick :` Permet d'avoir dans les graphes des abscisses sous forme de dates (années ou mois).

```
t = (1954:2:1962)';     y = [9 8.8 8.2 7.8 7.5];
plot(datenum(t,1,1),y) % Conversion des années en format nombre
datetick('x','yyyy')   % Remplace les ticks de l'axe x par des années
                       % à 4 digits
```

8 PLOTTING (TRACE)

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

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

8.1 PRINCIPALES FONCTIONS GRAPHIQUES 2D

`plot`	Graphe en 2D avec une échelle linéaire
`plotyy`	Graphe avec deux axes y différents à gauche et à droite
`loglog`	Graphe en 2D avec une échelle logarithmique pour les deux axes
`semilogx`	Graphe en 2D avec une échelle logarithmique pour l'axe des x
`semilogy`	Graphe en 2D avec une échelle logarithmique pour l'axe des y
`figure, close`	Permet d'ouvrir ou de fermer une figure
`subplot`	Tracer plusieurs graphes alignés sur une même figure

Tapez '`help plot`' pour une liste complète des options

8.1.1 Spécification des types de ligne avec plot

Style des lignes/couleur	**Option dans plot**
Ligne continue rouge	`-r`
Traits longs noirs	`--k`
Pointillés mauves	`:m`
Traits longs + pointillés bleus	`-.b`
Cercles verts	`og`
Ligne continue + cercles jaunes	`-yo`
Carrés bleu foncé	`sb`
Croix bleu clair	`xc`
Losanges + pointillés	`.d`
Étoiles	`h`
Astérisques	`*`

8.1.2 Options du graphe : titre, labels, axes

Quelques autres fonctions qui sont utiles pour créer des tracés:

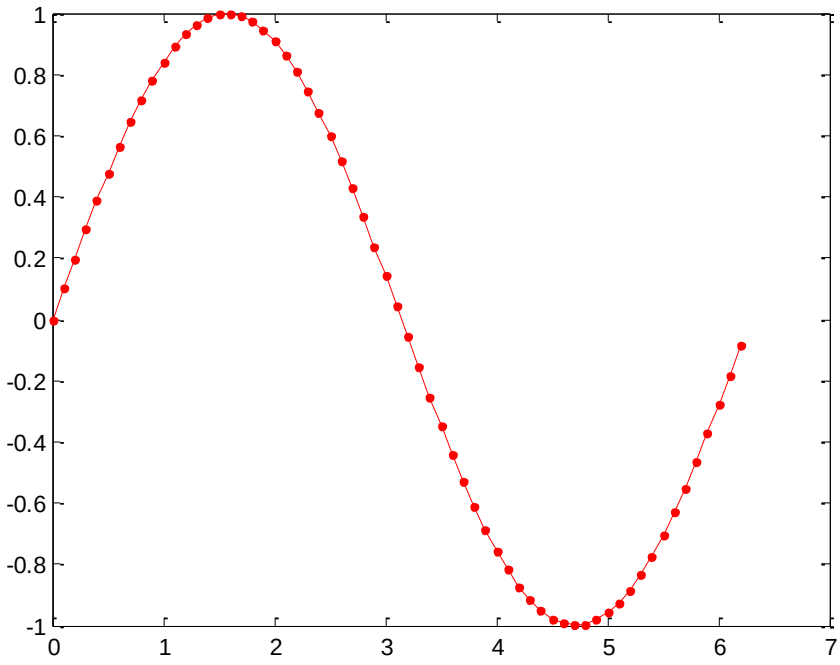
Mots-clés	Fonction
`title`	Définir le titre du graphe
`xlabel`	Label de l'axe des x
`ylabel`	Label de l'axe des y
`zlabel`	Label de l'axe des z
`legend`	Ajouter une légende sur le graphe
`text`	Permet d'ajouter du texte sur le graphe
`axis`	Définir xmin, xmax, ymin et ymax du graphe. voir aussi axis equal
`hold on/hold off`	Active/désactive le mode superposer

8.2 TRACE DE BASE

La fonction *plot* permet de tracer des courbes en MATLAB. Les arguments de cette fonction sont les vecteurs des variables indépendantes et dépendantes (en alternance), comme dans l'exemple qui suit :

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

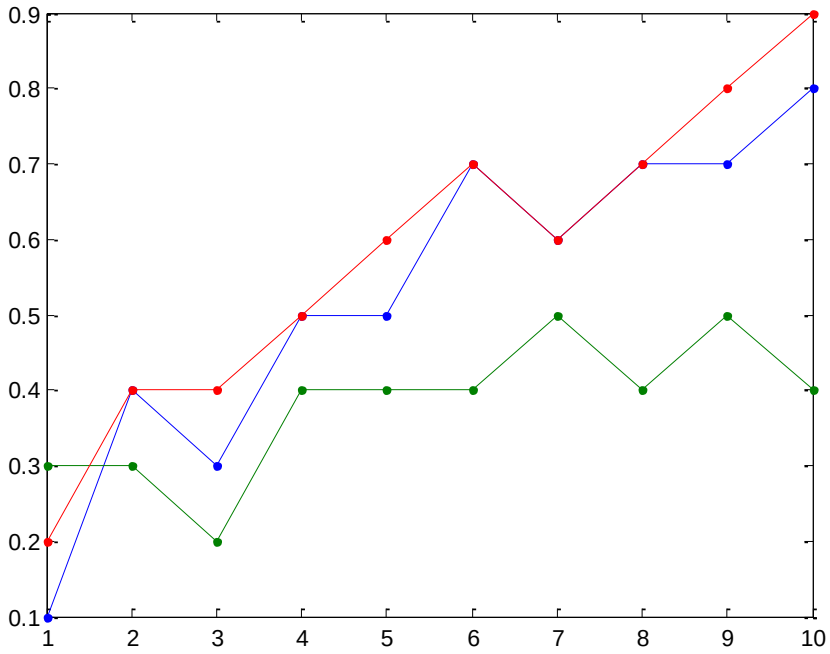
qui produit la sortie graphique



8.3 TRACE D'UNE MATRICE

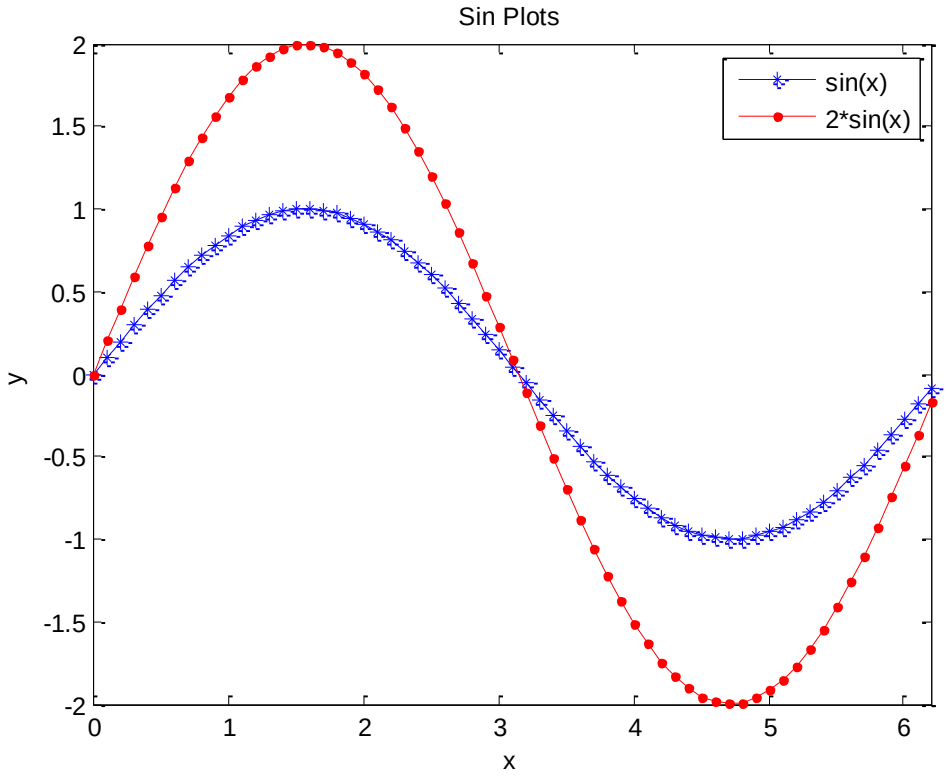
MATLAB permettra de traiter chaque colonne comme un ensemble différent de données

```
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, '.-')
```



Exemple

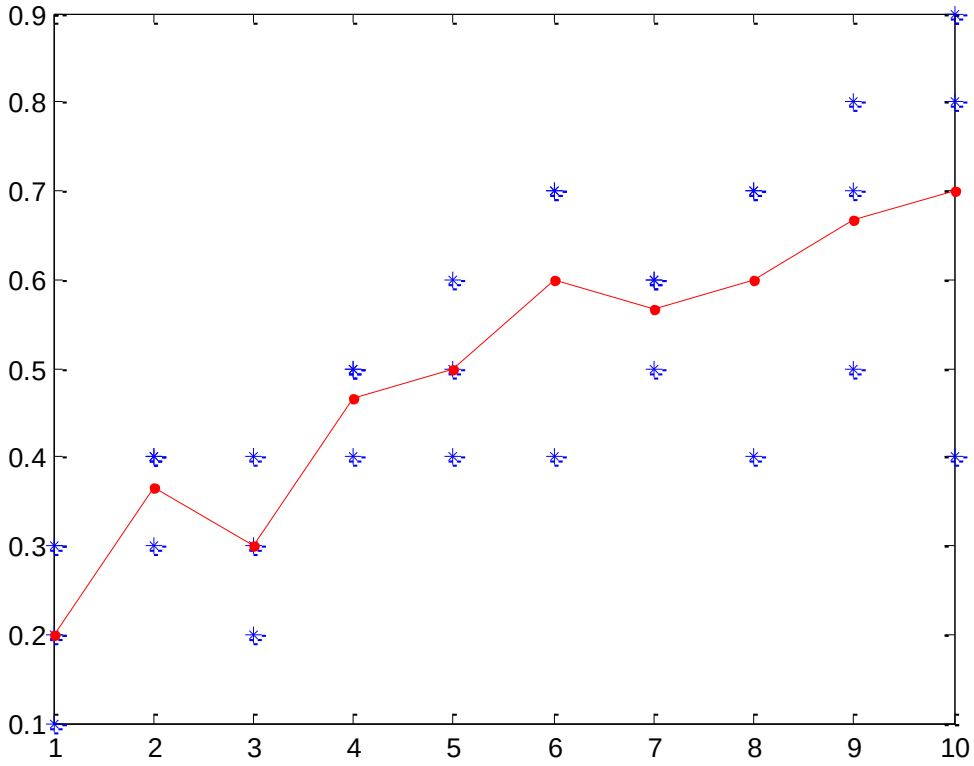
```
>> x = [0:0.1:2*pi];
>> y = sin(x);
>> plot(x, y, 'b*-')
>> hold on
>> plot(x, y*2, 'r.-')
>> grid on

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

>> xlabel('x');
>> ylabel('y');
>> hold off
```



8.4 TRACE DE DONNEES

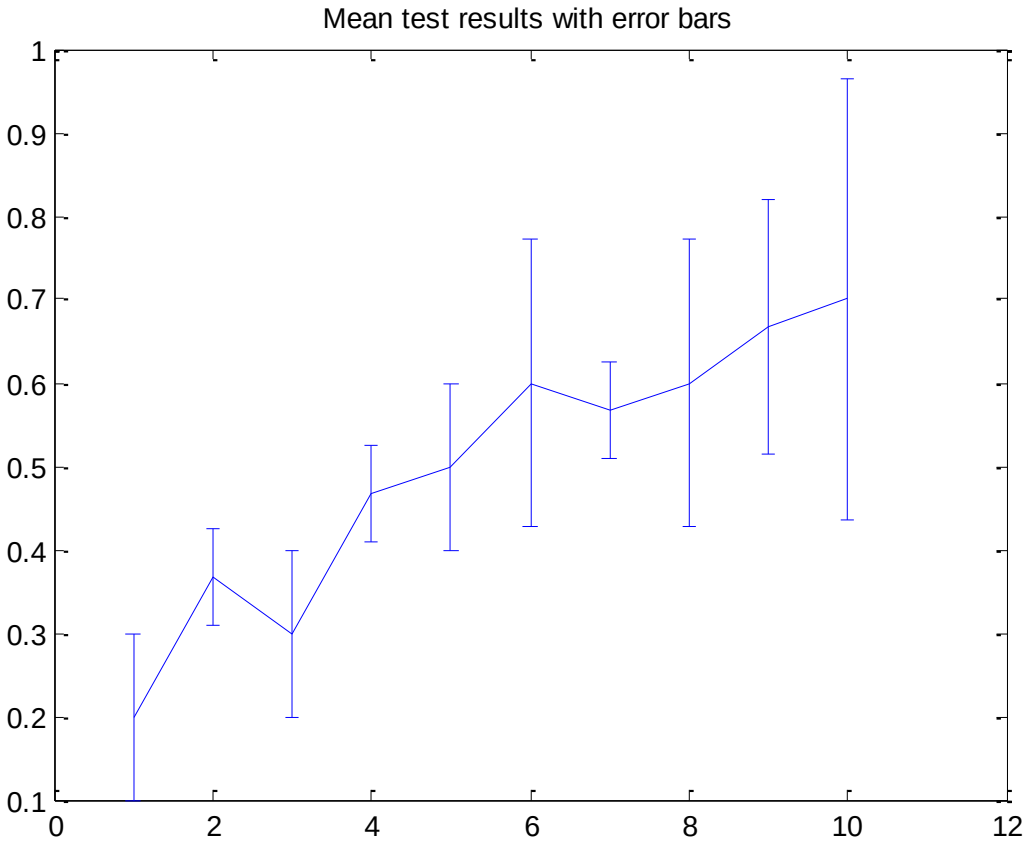
```
results =

    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.-')
```



8.5 TRACÉ ERROR BAR

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



Vous pouvez fermer toutes les graphes en cours en utilisant '`close all`'

8.6 LE TRACE AVEC SUBPLOT

- De multiples tracés (plots) peuvent être placés dans la même fenêtre
- Cette tache peut être effectuée avec la commande subplot
- La fenêtre est divisée en une grille dont la taille est spécifiée lors de la saisie de la commande

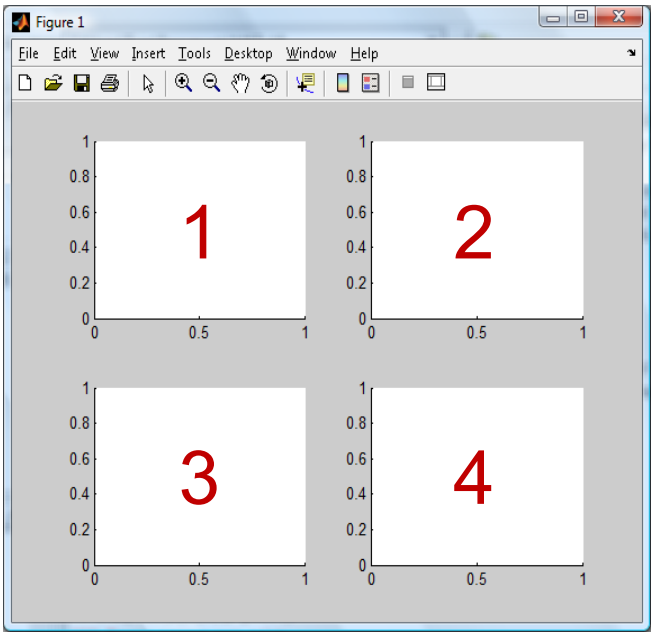
8.6.1 Forme générale de la fonction

```
subplot(i,j,k)
```

- i : nombre de lignes,
- j : nombre de colonnes,
- k : numéro du graphe actuel.

```
>> 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.



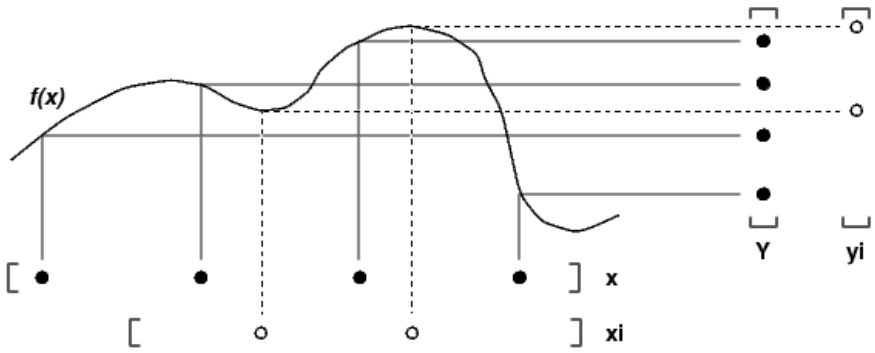
8.7 INTERPOLATION LINEAIRE

L'interpolation est une opération mathématique permettant le calcul des points intermédiaires entre deux points donnés. L'interpolation la plus simple entre deux points est la ligne droite ou interpolation linéaire.

8.7.1 Syntaxe

```
yi = interp1(x,Y,xi)
```

8.7.2 Description

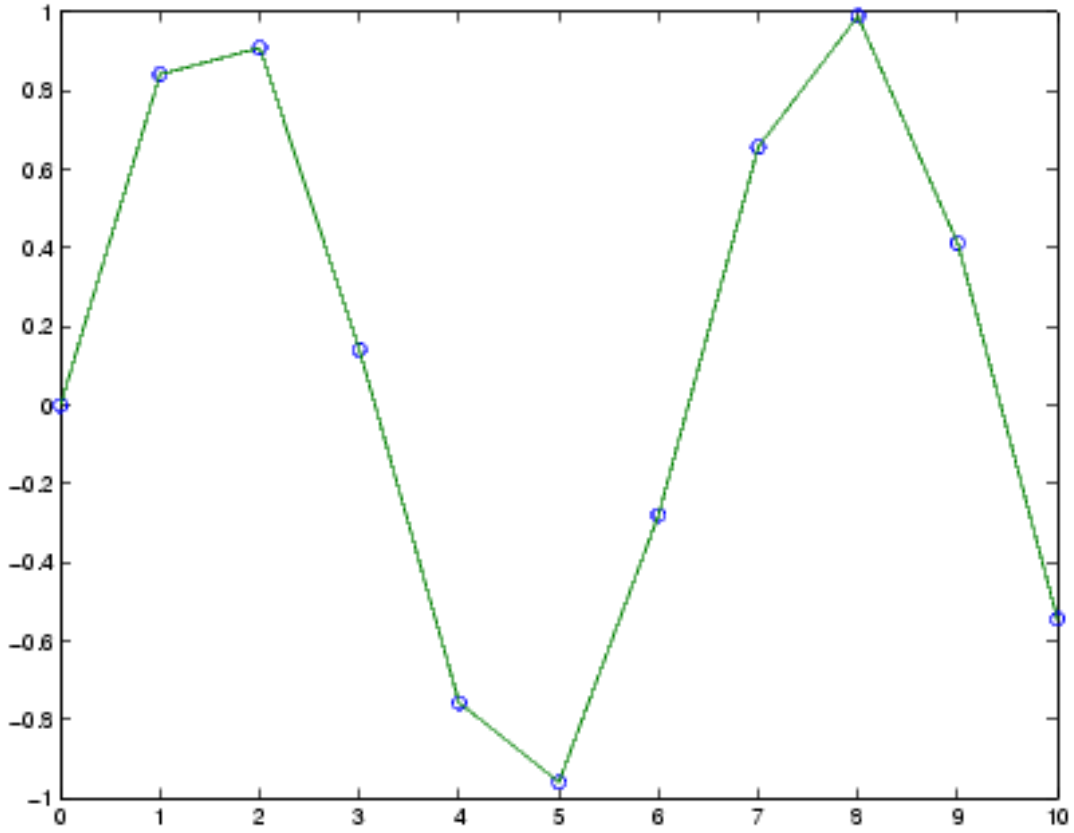


La commande retourne un vecteur de mêmes dimensions que x_i et dont les valeurs correspondent aux images des éléments de x_i déterminées par interpolation sur *x* et *y*.

Exemple

Nous allons générer une courbe sinusoïdale grossière (x de 0 à 10 avec un pas de 1) et l'interpoler sur une abscisse plus fine (X de 0 à 10 &vec un pas de 0 ,25).

```
x = 0:10
y = sin(x)
xi = 0:.25:10
yi = interp1(x,y,xi)
plot(x,y,'o',xi,yi)
```



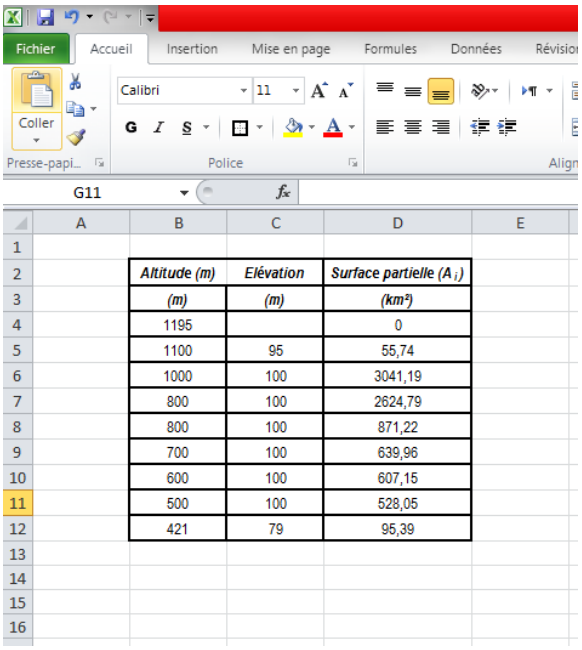
9 APPLICATIONS EN HYDROLOGIE

9.1 TP1 : CONSTRUCTION DE LA COURBE HYPSOMETRIQUE ET DETERMINATION DE CES CARACTERISTIQUES

9.1.1 Enoncé du problème

L'objectif est de rédiger un script Matlab qui construit la courbe hypsométrique et détermine ses caractéristiques. La lecture des données se fait à partir d'un fichier Excel (voir figure).

Le script doit permettre le calcul a partir du graphe des Altitude à 95% , altitude à 5% et de l'altitude médiane à 50% .



	A	B	C	D	E
1					
2		*Altitude (m)*	*Elévation*	*Surface partielle (A i)*	
3		*(m)*	*(m)*	*(km²)*	
4		1195		0	
5		1100	95	55,74	
6		1000	100	3041,19	
7		800	100	2624,79	
8		800	100	871,22	
9		700	100	639,96	
10		600	100	607,15	
11		500	100	528,05	
12		421	79	95,39	
13					
14					
15					
16					

Model de fichier de données

Le script doit permettre les calculs présentés dans le tableau suivant et leur écriture dans fichier résultat type Excel

Tableau : Données hypsométriques du bassin versant de Mazzer

Altitude	*Elévation*	*Surface partielle (A_i)*	*Surface partielle (A_i)*	*Surface cumulée (A)*	*Surface cumulée*
(m)	*(m)*	*(km²)*	*(%)*	*(km²)*	*(%)*
1195	-	0	0	0	0
1100	95	55,74	0,66	55,74	0,66
1000	100	3041,19	35,93	3096,94	36,59
800	100	2624,79	31,01	5721,73	67,60
800	100	871,22	10,29	6592,95	77,90
700	100	639,96	7,56	7232,91	85,46
600	100	607,15	7,17	7840,06	92,63
500	100	528,05	6,24	8368,11	98,87
421	79	95,39	1,13	8463,49	100,00
Total		**8463,49**	**100,00**		

Altitude à 95% $H_{95\%}$ =560 m
Altitude à 5% $H_{5\%}$ =1090 m
Altitude médiane à 50% $H_{50\%}$ =690 m

9.1.2 Solution du TP1 :

NB : toute instructions dans le script précédée d'un % n'es pas exécuté par Matlab. Ce symbole est utilisé pour inscrire des commentaires dans les scripts.

```
%++++++++++++++++++++++++++++++++++
% programme courbe hypsométrique     +
% ++++++++++++++++++++++++++++++++++

clear all
clc

% liste des variables
% ALinf : altitute inferieur
% ALsup : altitude superieur
% Elevation : élévation
% C: les cotes en mètre
% Sp: surface partielle entre deux côtes en km²*
% Spp: surface partielle en %
% Sc: surface cumulée km²
% Scp: surface cumulée en %
%

% lecture des données ;
[ALinf,ALsup,Sp]= textread('Data_hypso.txt')    ;
n=length (ALinf);  % nombre de ligne ;

for i=1:n;
   Elevation(i)=ALsup(i)-ALinf(i) ;
end

% Surface Cumulée
Sc(1)=Sp(1);

for i=2:n;
   Sc(i)=Sc(i-1)+ Sp(i);
end

% Surface partielle et cumulée en %

for i=1:n;
   Spp(i)=(Sp(i)/Sc(n) )*100;
   Scp(i)=(Sc(i)/Sc(n) )*100;
end

% tableau de calcul global

tab_final=[ALinf, ALsup, Elevation', Sp, Spp', Sc', Scp'];

% dessin des courbes

plot(Scp, ALsup)
hold on
grid on

title('Courbe Hypsométrique du bassin versant de Mazzer');
legend('Surface cummulée en %', 'Altitude en m');
axis([0 100 400 1200])
```

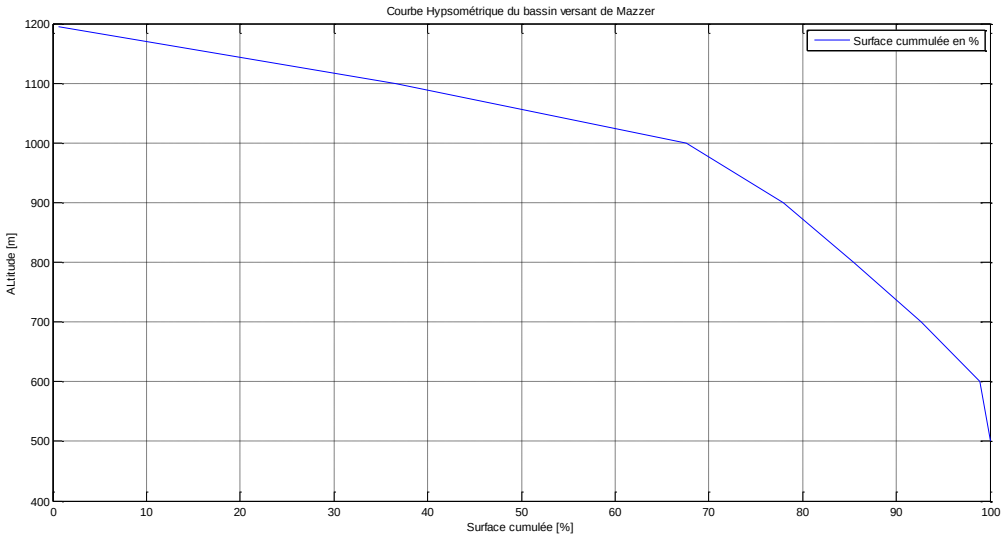
```
xlabel('Surface cumulée [%]');
ylabel('ALtitude [m]');
hold off

% calcul à partir du plot de l'altitude Altitude à 95%
H95% =560 m
AL95= interp1 ( Scp, ALsup, 95)
% calcul à partir du plot Altitude à 5%                          H5% =1090 m
AL5= interp1 ( Scp, ALsup, 5)
% calcul à partir du plot Altitude médiane à 50%          H50% =690 m
AL50= interp1 ( Scp, ALsup, 50)
```

9.1.3 Les résultats du script Matlab

En exécutant le script nous obtenons la matrice et le graphe suivants

tab_final <8x7 double>

	1	2	3	4	5	6	7	8
1	1100	1195	95	55.7400	0.6586	55.7400	0.6586	
2	1000	1100	100	3.0412e+03	35.9330	3.0969e+03	36.5916	
3	900	1000	100	2.6248e+03	31.0131	5.7217e+03	67.6047	
4	800	900	100	871.2200	10.2939	6.5929e+03	77.8986	
5	700	800	100	639.9600	7.5614	7.2329e+03	85.4600	
6	600	700	100	607.1500	7.1738	7.8400e+03	92.6338	
7	500	600	100	528.0500	6.2392	8.3681e+03	98.8729	
8	421	500	79	95.3900	1.1271	8.4635e+03	100	
9								
10								



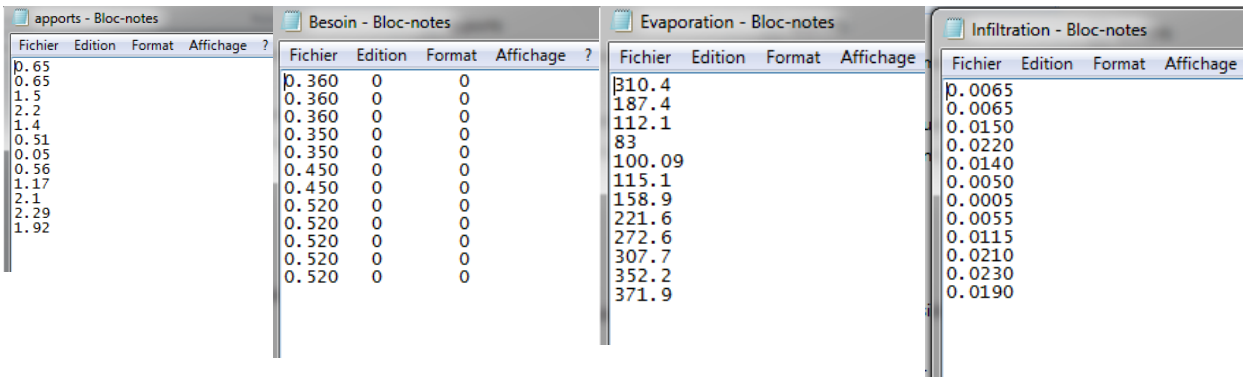
9.2 TP2 : REGULARISATION D'UNE RETENUE ET DETERMINATION DU NIVEAU NORMALE DE RETENUE (NNR)

9.2.1 Enoncé du problème

L'objectif est de rédiger un script Matlab qui permet d'étudié la régularisation d'une retenue et de déterminer son Niveau Normale de Retenue (NNR)

Les données de base du problème sont fournies dans des fichiers texte (.txt)

Nous avons quatre fichiers : apports, besoins, évaporation et infiltration



9.2.2 Eléments de réponses

Le travail doit être effectué en deux étapes :

1. La première étape consiste à construire les courbes caractéristiques de la retenue : courbe Hauteurs-surfaces et courbe Hauteurs-volumes
2. La deuxième étape consiste à programmer la régularisation et à utiliser les courbes Hauteurs-Surfaces et les courbes Hauteurs-Volumes numériquement (pas graphiquement) à l'aide de la fonction Interp (interpolation) de Matlab.

9.2.3 Présentation de la régularisation

Régularisation des écoulements

La régularisation des apports est la distribution dans le temps par rapport à la modulation, du régime de la demande par rapports aux pertes par infiltration et évaporation. La régularisation de l'écoulement est l'accumulation de l'eau dans les périodes d'excès pour l'utiliser dans les périodes de déficit.

La régularisation nous permet de définir la variation du niveau du plan d'eau en fonction des volumes d'eau entrant et sortant, et cela pour déterminer la hauteur économique du barrage.

Données de la régularisation

Les données disponibles sont :

- ✓ ***Un apport solide :*** est donnée par l'étude hydrologique de 2 hm^3 ;
- ✓ ***Un apport liquide :*** est donnée par l'étude hydrologique de 15 hm^3 ;

- ***Répartition de l'écoulement***

En réalité, le régime pluviométrique est très irrégulier, la quantité annuelle tombe souvent en une seule fois, causant des crues et beaucoup de dégâts. La répartition de l'écoulement est proportionnelle aux mesures des précipitations. A partir des mesures des précipitations, on estime la répartition mensuelle des écoulements. Les valeurs sont mentionnées dans le tableau

Tableau : Répartition mensuelles des écoulements

Mois	IX	X	XI	XII	I	II	III	IV	V	VI	VII	VIII	Total
Apport (%)	4,36	4,36	9,97	14,64	9,35	3,43	0,31	3,74	7,79	14,02	15,26	12,77	100
Apport (hm^3)	0,65	0,65	1,50	2,20	1,40	0,51	0,05	0,56	1,17	2,10	2,29	1,92	15

- ***Répartition de l'évaporation***

L'évaporation est un élément climatique important à prendre en considération compte tenu de son rôle, ainsi que son influence sur les précipitations. Elle est exprimée en millimètre d'eau. La station la plus proche à Mazzer est celle de Béni Abbès. Les mesures des moyennes mensuelles pour la période 1973-2008, sont disponibles

Tableau : Evaporation moyenne mensuelle station de Béni Abbès (1973-2008)

Mois	IX	X	XI	XII	I	II	III	IV	V	VI	VII	VIII	Total
Evap (mm)	310,4	187,4	112,1	83	100,09	115,1	158,9	221,6	272,1	307,7	352,2	371,9	2592,5

Les données de la station de Béni Abbès montrent que l'évaporation augmente progressivement pour atteindre le maximum aux mois de Juillet et Août, (371.94 mm) puis diminue jusqu'aux valeurs de 83 à 100 mm, durant les mois de Décembre et Janvier.

- ***Perte par infiltration***

Nous proposons de considéré les pertes d'infiltration égale à 1% de l'apport moyen annuels.

Tableau : Répartition mensuelles de l'infiltration

Mois	IX	X	XI	XII	I	II	III	IV	V	VI	VII	VIII	Total
Infiltration (10^3 m^3)	6.5	6.5	15	22	14	5	0.5	5.5	11.5	21	23	19	150

- ***Besoin en irrigation***

La superficie des terres agricoles à Mazzer englobe quelques 210 hectares (source PDAU de IGLI, 2008), essentiellement des Oasis (90%). Ces terres sont irriguées à partir de puits (120 puits) et sources (6 sources). Le débit de ces puits et sources sera soutenu par la construction de la digue de Mazzer, qui va assurer une alimentation des puits et sources de la localité.

Nous considérons dans notre étude une surface irrigable de 210 hectares. Il est recommandé d'utiliser des volumes supérieurs à 25000 m^3 par hectares et par ans (source : Cultiver le Palmier-Dattier de Gilles Peyron, 2000).

La distribution typique de l'irrigation pour les palmier-Dattier selon le guide INRA (2003) est donnée dans le tableau suivant :

Tableau 1 : Distribution typique généralisée de la demande pour l'irrigation

Mois	IX	X	XI	XII	I	II	III	IV	V	VI	VII	VIII	Total
Taux (%)	7	7	7	6	6	8.5	8.5	10	10	10	10	10	100
Besoins (hm^3)	0.368	0.368	0.368	0.351	0.351	0.45	0.45	0.525	0.525	0.525	0.525	0.525	5.25

Calcul de la régularisation

La régularisation des écoulements se fait selon la formule suivante :

$$V_{f_i} = Y_i + V_m - V_{évap} - V_{inf} - V_{irrig} - V_{AEP} - V_{ind}$$

Où :
Y_i : apport liquide du premier mois i
V_m : volume mort= 1.95 hm^3
$V_{évap}$: volume évaporé
V_{inf} : volume infiltré
V_{irri} : volume irrigation

Tableau 2 : Calcul d la régularisation pour la retenue de Mazzer

mois	apport	volume stocké	surface inondée	évaporation	Volume évaporé	pertes infiltration	besoin irrigation	volume fin du mois	cote	Hauteur Plan d'eau
	hm^3	hm^3	km^2	mm	hm^3	hm^3	hm^3	hm^3	m	m
IX	0,65	2,600	2.92	310,4		0,0065	0,360			
X	0,65	1,952		187,4		0,0065	0,360			
XI	1,5	2,674		112,1		0,0150	0,360			
XII	2,2	4,162		83		0,0220	0,350			
I	**1,4**	**4,825**		**100,09**		**0,0140**	**0,350**			
II	0,51	4,451		115,1		0,0050	0,450			
III	0,05	3,493		158,9		0,0005	0,450			
IV	0,56	2,999		221,6		0,0055	0,520			
V	1,17	2,890		272,6		0,0115	0,520			
VI	2,1	3,586		307,7		0,0210	0,520			
VII	2,29	4,166		352,2		0,0230	0,520			
VIII	1,92	4,063		371,9		0,0190	0,520			

9.2.4 Solution du TP 2

```
++++++++++++++++++++++++++++++++++++
% programme courbe hauteur surface    +
%                  hauteur volume     +
%                                     +
% ++++++++++++++++++++++++++++++++++++

clear all
clc

% liste des variables
% C: les cotes en mètre
% Sp: surface partielle entre deux côtes en km²*
% Sc: surface cumulée km²
% Hp: hauteur partielle en mètre
% Hc: hauteur cumulée en mètre
% Sm: surface moyenne en km²
% Vp : volume partielle en HM3
% Vt: volume totale en HM3
%

% lecture des données cote et Sp ;
[C,Sp]=textread('Data.txt')   ;
i=1:17 ;  % nombre de ligne ;

% calcul de la surface cuùulée

% pour la première ligne initialisation
Sc(1)= 0;

% pour les lignes de 2 à 17

for i=2:17;
   Sc(i)=Sc(i-1)+Sp(i) ;
end

% calcul de la hauteur partielle

% pour la première ligne initialisation
Hp(1)= 0;

% pour les lignes de 2 à 17

for i=2:17;
   Hp(i)=C(i)-C(i-1) ;
end

% calcul de la hauteur cumulée

% pour la première ligne initialisation
Hc(1)= 0;

% pour les lignes de 2 à 17

for i=2:17;
   Hc(i)=Hc(i-1)+Hp(i) ;
end
```

```
% calcul de la surface moyenne
% pour la première ligne initialisation
Sm(1)= 0;
% pour les lignes de 2 à 17

for i=2:17;
   Sm(i)= (Sp(i-1)+Sp(i))/2 ;
end

% calcul du volume partiel

% pour la première ligne initialisation
Vp(1)= 0;

% pour les lignes de 2 à 17

for i=2:17;
   Vp(i)= Hp(i)* Sm(i) ;
end

% calcul du volume totale

% pour la première ligne initialisation
Vt(1)= 0;

% pour les lignes de 2 à 17

for i=2:17;
    Vt(i)=Vt(i-1)+Vp(i) ;
end

% tableau de calcul global

tab_final=[C, Sp, Sc', Hp', Hc', Sm', Vp', Vt'];

% dessin des courbes

plot(Hc, Sc)
hold on
plot(Hc, Vt,'k')
grid on
title('Courbe Hauteur-Surface-Volume');
legend('Hauteur-Surface', 'Hauteur-Volume');
axis([0 18 0 14])
xlabel('Hc');
ylabel('Sc/Vt');
hold off
```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Calcul de la régularisation      %
%                                     %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Mois
Mois =1:12;
% lecture des  apports
apport= textread('apports.txt');
% lecture des  infiltrations
Inf= textread('Infiltration.txt');
% lecture des  besoins
[Irr, AEP, Ind]= textread('besoin.txt');
% lecture de l'évaporation
Evaporation= textread('Evaporation.txt');
Vm= 1.95;
Volume_Stock (1)= apport(1)+ Vm;
Hauteur_Plan_Eau(1)= interp1 (Vt,Hc, Volume_Stock (1));
% surface inondée par le barrage
Surface_Inondee (1) = interp1 (Hc,Sc,Hauteur_Plan_Eau(1) ) ;
% Volume pertte évaporation
Volume_evapore(1)= (Surface_Inondee (1)* Evaporation(1))/ 1000 ;
% volume qui reste a la fin du mois
Volume_fin_mois(1)= Volume_Stock (1)- Volume_evapore(1)- Inf(1)-Irr(1)-
AEP(1)-Ind(1);
% hauteur d'eau dans le barrage à la fin du mois
Hauteur_Plan_Eau_2(1)= interp1 (Vt,Hc, Volume_fin_mois(1));
for i=2:12
Volume_Stock (i)= apport(i)+ Volume_fin_mois(i-1);
Hauteur_Plan_Eau(i)= interp1 (Vt,Hc, Volume_Stock (i));
% surface inondée par le barrage
Surface_Inondee (i) = interp1 (Hc,Sc,Hauteur_Plan_Eau(i) ) ;
% Volume pertte évaporation
Volume_evapore(i)= (Surface_Inondee (i)* Evaporation(i))/ 1000 ;
% volume qui reste a la fin du mois
Volume_fin_mois(i)= Volume_Stock (i)- Volume_evapore(i)- Inf(i)-Irr(i)-
AEP(i)-Ind(i);
% hauteur d'eau dans le barrage à la fin du mois
Hauteur_Plan_Eau_2(i)= interp1 (Vt,Hc, Volume_fin_mois(i));
end
NNR=max (Hauteur_Plan_Eau_2)
```

9.2.5 Les résultats du script Matlab

Courbe Hauteurs-Surfaces et courbe Hauteur-Volumes

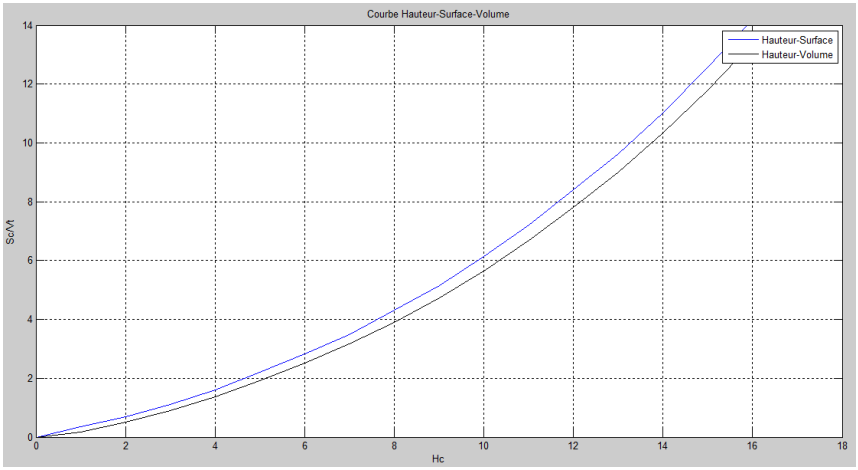
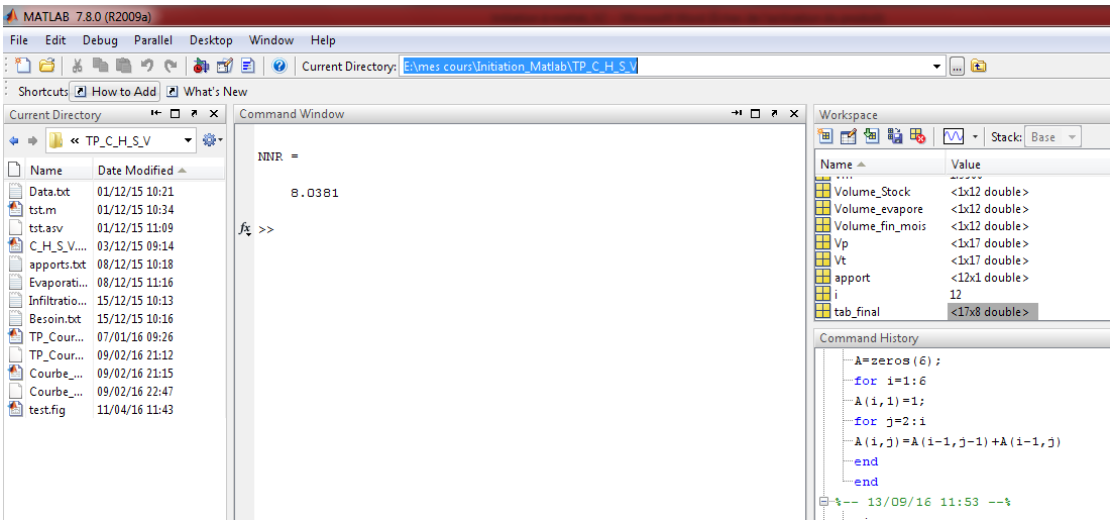


Tableau des résultats



tab_final <17x8 double>

	1	2	3	4	5	6	7	8
1	469	0	0	0	0	0	0	0
2	470	0.3400	0.3400	1	1	0.1700	0.1700	0.1700
3	471	0.3500	0.6900	1	2	0.3450	0.3450	0.5150
4	472	0.4200	1.1100	1	3	0.3850	0.3850	0.9000
5	473	0.5000	1.6100	1	4	0.4600	0.4600	1.3600
6	474	0.5900	2.2000	1	5	0.5450	0.5450	1.9050
7	475	0.6200	2.8200	1	6	0.6050	0.6050	2.5100
8	476	0.6700	3.4900	1	7	0.6450	0.6450	3.1550
9	477	0.8200	4.3100	1	8	0.7450	0.7450	3.9000
10	478	0.8500	5.1600	1	9	0.8350	0.8350	4.7350
11	479	0.9800	6.1400	1	10	0.9150	0.9150	5.6500
12	480	1.0700	7.2100	1	11	1.0250	1.0250	6.6750
13	481	1.1900	8.4000	1	12	1.1300	1.1300	7.8050
14	482	1.2400	9.6400	1	13	1.2150	1.2150	9.0200
15	483	1.3800	11.0200	1	14	1.3100	1.3100	10.3300
16	484	1.5500	12.5700	1	15	1.4650	1.4650	11.7950
17	485	1.6400	14.2100	1	16	1.5950	1.5950	13.3900
18								

9.3 TP3 : LECTURE DES DONNEES PLUVIOMETRIQUES ANRH

9.3.1 Enoncé du problème

Écrire un script Matlab qui permet à partir du fichier de données pluviométriques ANRH (voir modèle de fichier) de :

1. Lire les données pluviométriques journalières à partir d'un fichier Excel. (ANRH)
2. Déterminer la pluie journalière maximale pour chaque mois (**Pjmax**)
3. Déterminer le jour du Pjmax (**DayPjmax**).
4. Déterminer le nombre de jours de pluie (**Npluie**)
5. Calculer la pluie totale mensuelle (**Ptotmens**) et la pluie totale annuel (**Ptotan**).
6. Écrire ces résultats dans le même fichier Excel

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		sept	oct	nov	dec	janv	fev	mars	avril	mai	juin	juil	aout
2	Code station : 140606												
3	Nom station : AIN BERDA												
4	X : 937,288												
5	Y : 387,737												
6	Z :												
7													
8	1970												
9													
10	1	0	0	0	0	0	1.4	5.6	0	0	0	0	0
11	2	0	11.5	0	0	20.3	0.4	11.5	14.4	0	0	0	0
12	3	0	0	0	0	0.5	0	0	0	0	0	0	0
13	4	0	0	0	0	1.4	0	2.1	0.3	3	1	0	0
14	5	0	0	0	0	7.3	18.1	1.7	1	7.8	0	0	0
15	6	0	0	0	0	25	0	1.6	0	0	1.3	0	0
16	7	0	0	0	0	9.2	0	1.6	0	0	0	0	0
17	8	0	0	0	0	7.4	0	0.8	0	0	0	0	0
18	9	0	0	0	0	0	4.4	0.8	19.1	0	0	0	0
19	10	0	0	0	0	0	7.7	0	26.9	0	0	0	0
20	11	0	0	0	0	0	20.2	0	0	15.5	0	0	0
21	12	0	0.4	0	0	0	0	0	0	11.6	0	0	0
22	13	0	13.4	0	0	1	0	1	0	0	0	0	0
23	14	0	2.8	1.2	0	0	0	1.1	0	0	0	1.4	0
24	15	0	6	0.8	10	1.8	0	5.5	0	0	0	0	0
25	16	0	19.5	3.5	13.5	13.1	2.1	3.5	0	0	0	0	0
26	17	0	7.4	1.3	1.2	8.3	1.8	0.5	0	0	0	0	0
27	18	0	2	0	0	6.1	0	0	3.4	0	0	0	0
28	19	3.5	0	0	0	3.4	0	0	0	0.8	0	0	0
29	20	0	0	0	2.2	1.4	14	0	0	0	0	0	0
30	21	0	2.4	0	13.1	0	3	11.7	0	0	0	0	0
31	22	0	7.9	0.4	7.8	2.2	0.2	3.5	0	0	0	0	0
32	23	0	3.4	0	3.4	2.9	5.8	7.6	0	1.9	0	0	0

Modèle fichier ANRH

9.3.2 Solution du TP 3

```
%================================================================
% programme lecture et traitement des données journalières      =
%
%================================================================
%
%   Pjmax: la pluie journalière maximale pour chaque mois
%   DayPjmax:le jour du Pjmax (DayPjmax).
%   Npluie  :le nombre de jours de pluie
%   Ptotmens: la pluie totale mensuelle
%   Ptotan: la pluie totale annuel
%   LE 18/04/2016
%   Master HU
clc
clear all
A = xlsread('ain berda.xls','B10:M40'); % Lire les données pluviométriques
journalières à partir d'un fichier Excel
[i,j] = max(A);
Pjmax=i ; % pluie journalier max
DayPjmax=j ;% jour du max
Npluie=sum(A>0) ;% nbr des jour pluvieuse par mois
NbreJP=sum(Npluie); %Nombre pluvieuse de jour par ans

Moyenne = mean(A( find(A>0 ) ) );
Ecartype = std(A( find(A>0 ) ) );
cv=Ecartype/Moyenne;
Ptotmens=NaNsum(A);
Ptotan=sum(Ptotmens);
Pjmaxannuel=max (Pjmax)

% écriture des résultats dans le fichier excel

xlswrite('ain berda.xls',Pjmax,'B42:M42')
xlswrite('ain berda.xls',DayPjmax,'B43:M43')
xlswrite('ain berda.xls',Npluie,'B44:M44')
xlswrite('ain berda.xls',Ptotmens,'B45:M45')
xlswrite('ain berda.xls',Ptotan,'tab_pv','N45')
xlswrite('ain berda.xls',Pjmaxannuel,'tab_pv','N42')
xlswrite('ain berda.xls',NbreJP,'tab_pv','N44')
```

10 REFERENCES BIBLIOGRAPHIQUES

1. Adrian Biran et Moshe Breiner, 2004, « MATLAB pour l'ingénieur », livre, Édition : Pearson Education - 504 pages.
2. Jean-Thierry Lapresté ; 2009, « Introduction à MATLAB »livre, Édition : Ellipses - 239 pages, 3eédition.
3. Vincent Isoz, Video de formation « Découverte de Matlab 2014a », 2015.
4. Site web : http://www.developpez.net/forums/f148/environnements-developpement/matlab/