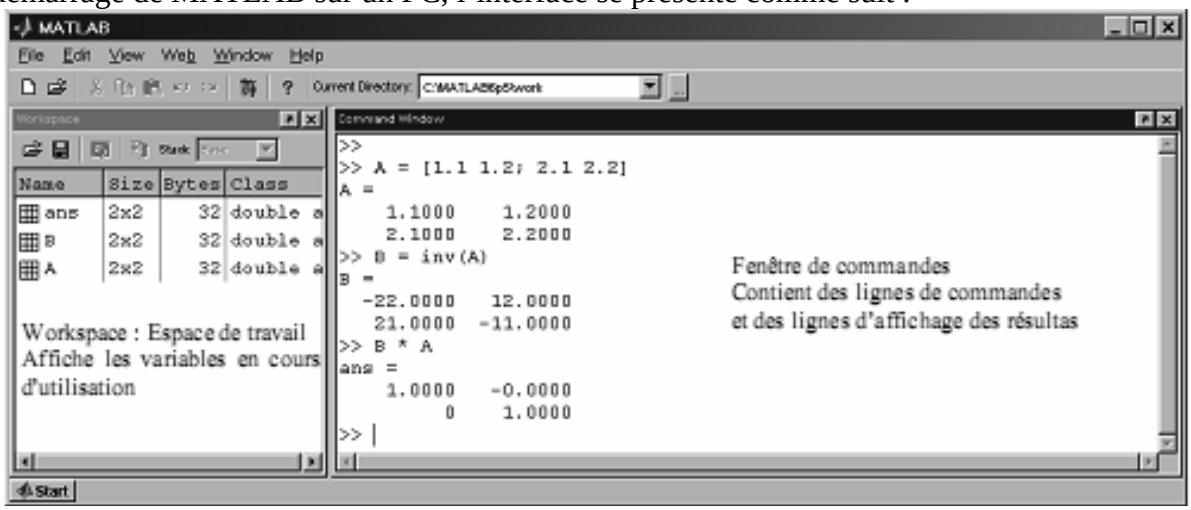
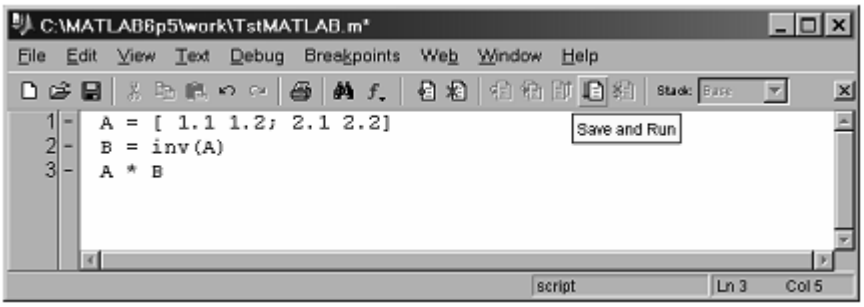
1. Aide mémoire matlab

MATLAB est un logiciel interactif basé sur le calcul matriciel (MATrix LABoratory). Il est utilisé dans les calculs scientifiques et les problèmes d'ingénierie parce qu'il permet de résoudre des problèmes numériques complexes en moins de temps requis par les langages de programmation, et ce grâce à une multitude de fonctions intégrées et à plusieurs programmes outils testés et regroupés selon usage (boites à outils ou Toolbox). Au démarrage de MATLAB sur un PC, l'interface se présente comme suit :



Les déclarations et les commandes peuvent être introduites ligne par ligne dans la fenêtre des commandes ou sous forme d'un script dans un éditeur qui peut être évoqué par la commande edit.



L'exécution du script (les commandes une après une) se fait à l'aide du bouton Save and Run ou avec le menu debug/Save and Run ou bien, simplement, en appuyant sur la touche fonction F5. Les résultats d'exécution sont affichés dans la fenêtre des commandes de la même manière que si les commandes sont entrées dans cette fenêtre.

Remarques :

Une ligne de commande peut contenir plusieurs instructions séparées par des virgules ou par des points-virgules. Le résultat d'une instruction suivie par un point-virgule ne sera pas affiché.

1.1. INTRODUCTION DES MATRICES

MATLAB fonctionne essentiellement avec un seul type d'objets : des matrices rectangulaires avec des éléments pouvant être réels, complexes ou symboliques. Les scalaires sont interprétés comme des matrices 1×1 !! Il existe plusieurs façons pour introduire une matrice dans MATLAB :

1-Entrée explicitement par liste des éléments

```
>>A = [ 1.1 1.2
       2.1 2.2 ]
>>A = [ 1.1, 1.2 ; 2.1, 2.2 ]
```

Les colonnes de la matrice sont séparés par des virgules (,) ou des espaces et les lignes par des points-virgules (;) ou des sauts de ligne (RETURN).

2- Générée par une fonction interne (built-in function)

```
>>A = eye(3)                    % matrice identité
A =     1     0     0
    0     1     0
    0     0     1
>>B = ones(3)                   % matrice carrée contenant des 1
B =     1     1     1
    1     1     1
    1     1     1
>>C = zeros(2,3)                % matrice rectangulaire nulle
C =     0     0     0
    0     0     0
>>D = diag([1:3])               % matrice dont la diagonale varie de 1 à 3
D =     1     0     0
    0     2     0
    0     0     3
>> A = rand(2,3)                % matrice aléatoire, un 2eme appel de rand
A =                             % renvois une autre matrice 2×3
    0.7271    0.8385    0.3704
    0.3093    0.5681    0.7027
```

3- Entrée à partir d'un fichier script avec l'éditeur

4- Chargée à partir d'un fichier de données ou importée d'une application.

```
>>Load matrice.txt
>>A = load('matrice.txt')
```

Le fichier matrice.txt contient des valeurs numériques disposées en tableau.

La première commande charge les valeurs de la matrice et affecte le nom matrice au résultat par contre la seconde permet de choisir un nom pour la matrice chargée.

La structure du fichier matrice.txt est :

```
1 5 7
5 4 2
4 2 1
```

Les espacements entre les composantes peuvent être irréguliers. Des virgules ou points-virgules peuvent être utilisés pour séparer les colonnes et les lignes. Une ligne incomplète provoque une erreur d'exécution.

1.2. OPERATIONS SUR LES MATRICES

Les opérations matricielles suivantes sont disponibles sous MATLAB

+	Addition	C = A + B
-	Soustraction	C = A - B
*	Multiplication	C = A * B
^	Puissance	C = A^2 ou bien C = A * A
'	Transposée	C = A' ou bien C = transpose(A)
\	division gauche	x = A\b

/ division droite x = b/A

Ces opérations matricielles s'appliquent aussi sur des scalaires. Une erreur est provoquée dans le cas où les rangs des matrices sont incompatibles.

Remarque 1
La division matricielle implique la résolution de systèmes d'équations linéaires. Si A est une matrice carrée inversible :
x = A\b est la solution du système A*x = b, avec x et b sont des vecteurs colonne
x = b/A est la solution du système x*A = b, avec x et b sont maintenant des vecteurs lignes

```
>> A = [3 2; 4 7]
A =
     3     2
     4     7
>> b = [2;3]
b =
     2
     3
>> x = A\b
x =
    0.6154
    0.0769
>> A * x
ans =
     2
     3
y = b'
y =
     2     3
>> z = y/A
z =
    0.1538     0.3846
>>z * A
ans =
     2     3
```

Remarque 2
Si un point est ajouté à gauche des opérateurs * ^ \ et / ils seront appliqués sur les éléments des matrices avec correspondance d'indices.

```
>> A * A% équivalent à A ^ 2
ans =
    17    20
    40    57
A .*A                    % équivalent à A .^ 2
ans =
     9     4
    16    49
```

1.2.1. OPERATIONS SUR LES VECTEURS

Les vecteurs sont considérés comme des matrices rectangulaires d'une ligne ou d'une colonne. Toutes les opérations matricielles s'appliquent sur les vecteurs mais il faut faire attention aux correspondances des tailles. Les opérateurs précédés d'un point sont particulièrement utiles dans les calculs de vecteurs et dans les représentations graphiques. Exemple de manipulation de vecteurs :

```
>>a = [2 1 3]                % vecteur ligne
a =          2     1     3
>> b = [1;3;2]               % vecteur colonne
b =
     1
```

```
     3
     2
>> a * b     % produit scalaire
ans =          11       >> a .* b'                    % produit de composantes (remarquer b')
ans =           2      3     6
>> x = 0:0.2:1                    % vecteur à pas fixe
x =
         0    0.2000       0.4000      0.6000      0.8000       1.0000
>> y = x .^2 .* cos(x)              % y(x) = x cos(x)
y =
         0    0.0392       0.1474      0.2971      0.4459       0.5403
>> plot(x,y)                      % affiche une fenêtre du graphe y(x)
```

1.3. FONCTIONS SUR LES MATRICES

MATLAB contient plusieurs fonctions qui s'appliquent sur les matrices. Celles qui peuvent nous intéresser sont les suivantes. (Utiliser help ou demo pour en savoir plus)

eig	valeurs et vecteurs propres	[Vects,Vals] = eig(A)	
chol	décomposition de cholesky	C = chol(A)	A = C'*C
lu	décomposition LU	[L,U] = lu(A)	A = L *U
qr	décomposition QR	[Q,R] = qr(A)	A = Q *R
svd	décomp. en val. Singulières	[U,S,V] = svd(A)	A = U*S*V'
inv	inverse	B = inv(A)	I = B * A
det	déterminant	d = det(A)	
norm	norme plusieurs définitions voir help norm		

Les fonctions scalaires telles que cos, sin, exp, log … etc. s'appliquent sur les éléments des matrices Exemple de calcul de valeurs et vecteurs propres :

```
>> A = [5 2 1; 2 7 3; 1 3 8] ;
>> [vects,vals] = eig(A)
vects =
    0.7040     0.6349      0.3182
   -0.6521     0.4005      0.6437
    0.2812    -0.6607      0.6959
vals =
    3.5470          0          0
         0     5.2209          0
         0          0    11.2322
>> vects' * A * vects
ans =
    3.5470    -0.0000    -0.0000
   -0.0000     5.2209    -0.0000
   -0.0000    -0.0000    11.2322
```

2. ACCES AUX ELEMENTS DES MATRICES

L'accès aux éléments d'une matrice se fait à l'aide de leurs indices ligne et colonne mis, après le nom de la matrice, entre parenthèses : A(i,j) est l'élément de ligne i et de colonnes j. Il est possible d'obtenir une sous matrice en spécifiant une liste d'indices. Deux points aux lieu d'un indice ou d'une liste d'indice est interprété comme ligne ou colonne entière.

```
>> A = [1.1 1.2 1.3 1.4
       2.1 2.2 2.3 2.4
       3.1 3.2 3.3 3.4
       4.1 4.2 4.3 4.4 ]
```

```
>> A(2,2:4)
 ans =
    2.2000    2.3000    2.4000

 >> A(3,:)
 ans =
    3.1000    3.2000    3.3000    3.4000

 >> A(:,3)
 ans =
    1.3000         2.3000         3.3000         4.3000

 >> t = [1,4]
 t =          1     4

 >> A(t,t)
 ans =
    1.1000    1.4000         4.1000    4.4000
```

2.1. LES BOUCLES DANS MATLAB

La structure des boucles dans MATLAB ne diffère pas de celle d'autres langages de programmation La boucle FOR a la structure suivante :

```
for valeur = valeur_initiale : pas : valeur_finale
   instruction 1
   instruction 2
   . . . . . .
End
```

La boucle while a la structure suivante :

```
Initialisations
 while expression logique
        instruction 1
        instruction 2
        . . . . . .
End
```

Les deux boucles suivantes donne un même vecteur :

```
f = w = 2 4 6 8 10
for i = 1:5                % boucle for à pas unité
    f(i) = 2*i;            % l'incrémentation de i est automatique
end;
i = 1;                     % initialisation de i
while i <= 5               % boucle tant que i est inférieur ou égal à 5
    w(i) = 2*i;
    i = i+1;               % l'incrémentation est importante sinon la
end;                       % boucle est infinie
```

2.2. LES TESTS LOGIQUES

Les testes logiques dans MATLAB se font à l'aide des mots clés if, else et end. Les opérateurs relationnels utilisés dans les instructions logiques sont :

```
< inférieur      <=  inférieur ou égal
> supérieur      >=  supérieur ou égal
== égal          ~=    non égal (différent)
```

et les opérateurs logiques sont :

```
 & et  exemple : c = a & b  ou bien c = and(a,b)
| ou  exemple : c = a | b  ou bien c = or(a,b)
~ non exemple : c =~a          ou bien c = not(a)
```

Exemple de test if

```
if a < b
```

```
    m = a
else
    m = b
end
```

2.3. LES FICHIERS FONCTIONS

Les fichiers scripts permettent d'écrire et de sauvegarder les commandes dans un fichier disque. Les fichiers fonctions ont plus d'importance, ils offrent une extensibilité à MATLAB. On peut créer une fonction spécifique à un problème donné et l'utiliser de la même manière que les autres fonctions offertes par MATLAB. Un fichier fonction possède la structure suivante :

```
function [res1, res2, ...] = nomfonction(var1, var2, ...)
% mettre ici les lignes commentaire à afficher
% dans la fenêtre de commandes suite à une demande d'aide
% avec help nomfonction
instruction
instruction
```

Le mot clé *function* est indispensable. Les variables d'entrée sont *var1, var2* ... et les résultats sont res1, res2, ... Autant de variables et de résultas que nécessaire peut être utilisé. Le fichier doit être sauvegarder sous le nom *nomfonction.m* et peut contenir plus d'une fonction.

Un mot clé return peut être ajouté en fin de chaque fonction.

```
function R = rot3dz(theta)
% R = rot3dz(theta)
% retourne une matrice de rotation 3d
% d'un angle autour theta de l'axe z
% theta en degrés mesuré à partir de l'axe x
theta = theta * pi/180;              % transformation en radian
c = cos(theta);                      % cos et sin pour plus de rapidité
s = sin(theta);
R = [ c  s  0
     -s  c  0
      0  0  1
     ];
return
```

Pour avoir une rotation de 30° autour de z, on peut appeler la fonction comme suit :

```
>> r  = rot3dz(30)
r =
   0.8660    0.5000         0
  -0.5000    0.8660         0
   0         0    1.0000
```