

---

# ENSIMAG

## STAGE DE PROGRAMMATION MATLAB

Stefanie Hahmann

---

### MATLAB<sup>TM</sup> : compléments de doc

Le logiciel MATLAB<sup>TM</sup> permet de réaliser les opérations les plus souvent effectuées en calcul scientifique. Matlab signifie *MATrix LABoratory*. Ce logiciel, écrit en C, accepte et interprète des commandes données sous trois formes possibles

- de façon interactive au clavier
- mises dans un fichier de commandes (un *script*)
- écrites en C ou FORTRAN, puis compilées

Seules les deux premières possibilités nous intéresseront. (La troisième nécessite des appels système et une plus grande maîtrise d'Unix).

Le logiciel MATLAB<sup>TM</sup> ainsi que certaines de ses "boîtes à outils" (Toolboxes) sont installés sur **mealy**. Vérifiez que votre **path** (cf fichiers **.profile** ou **.cshrc**) comporte bien le chemin **/usr/local/bin**. Il suffit alors de taper **matlab**.

Une chose importante à connaître en MATLAB<sup>TM</sup> est la commande **help** :

- si on tape seulement **help**, on obtient une liste de sujets pour lesquels on peut avoir une aide en ligne. Cette aide concerne également les fichiers écrits en MATLAB<sup>TM</sup>. Ceci signifie que si vous avez un fichier Matlab **toto.m** et que vous tapez **help toto**, les premières lignes de commentaire de **toto.m** s'afficheront.
- si on tape **help <nom d'un sujet>**, on obtient la syntaxe de la commande pour laquelle on réclame de l'aide, ainsi qu'une brève description de ses fonctionnalités.
- **help help** vous affiche tout sur **help**.

### 1 Avec des nombres

$x=\log(-1)$	Matlab sait aussi calculer avec des complexes
$y=\text{sqrt}(-1)$	encore un exemple
$x*y$	la multiplication s'effectue grâce au signe *
$\text{tanh}(x)$	une fonction trigonométrique hyperbolique
$2*\text{angle}(x)$	cette fonction donne la phase d'un complexe
$\text{abs}(x)$	celle-ci donne le module
$\text{abs}(-2)$	ou la valeur absolue

<code>abs(-2+j)</code>	<code>j</code> remplace <code>i</code> pour les électriciens
<code>conj(x)</code>	conjugaison
<code>format long</code>	changement du format d'affichage (double précision)
<code>abs(-2+j)</code>	effectivement la longueur du nombre affiché a changé
<code>format short</code>	
<code>eps</code>	distance entre 1.0 et le nombre flottant immédiatement supérieur. Attention : c'est un mot réservé, et toute variable <code>epsilon</code> sera refusée.

## 2 Avec des matrices

<code>A = [1 2 3 4 ; 5 6 7 8]</code>	voici une matrice en Matlab
<code>B = [1 0 1 0</code>	en voici le début d'une autre
<code>0 1 0 1</code>	puis la suite
<code>0 0 1 0</code>	encore la suite
<code>0 0 ...</code>	et voici la dernière ligne coupée en deux
<code>0 1]</code>	ouf !
<code>A = zeros(10,20)</code>	création d'une matrice 10 × 20 de zéros
<code>A = ones(10,20)</code>	création d'une matrice 10 × 20 de 1
<code>A = eye(10,20)</code>	création d'une matrice 10 × 20 avec des 1 sur la diagonale
<code>diag(A)</code>	les éléments de la diagonale de $A$
<code>[m,n] = size(A)</code>	$m$ = nombre de lignes de $A$ , $n$ = nombre de colonnes
<code>A*B</code>	le produit de matrices, c'est simple
<code>A\b</code>	résout $Ax = b$
<code>A\B</code>	calcule $A^{-1} * B$ (produit par l'inverse à gauche)
<code>A/b</code>	résout $xA = b$
<code>A/B</code>	calcule $A * B^{-1}$ (produit par l'inverse à droite)
<code>p=poly(B)</code>	le polynôme caractéristique de $B$
<code>roots(p)</code>	les racines du polynôme caractéristique de $B$
<code>roots1(p)</code>	idem, avec une autre méthode
	<code>format long</code>
<code>C = [0.01 0 -1 ; 0 0 -1 ; 0 1 0]</code>	une autre matrice
<code>D = inv(C)</code>	on inverse une matrice très facilement
<code>C(1:1) = 1e-7</code>	on teste la solidité numérique de Matlab
<code>D = inv(C)</code>	ça marche
<code>C(1:1) = 1e-12</code>	on teste la solidité numérique de Matlab
<code>D = inv(C)</code>	ça marche encore
<code>C(1:1) = 1e-16</code>	on teste la solidité numérique de Matlab
<code>D = inv(C)</code>	problème, mais Matlab est lucide et connaît ses limites
<code>exp(B)</code>	exponentielle de chaque coefficient de $B$
<code>expm(B)</code>	exponentielle de la matrice $B$
<code>c = rand(5)</code>	génération aléatoire de matrices: $c$ de taille 5 × 5
<code>c = rand(A)</code>	$c$ de mêmes dimensions que $A$ (i.e. 4 × 2)
<code>rand('uniform')</code>	pour changer la distribution
ou <code>rand('normal')</code>	
<code>rand('dist')</code>	pour connaître la distribution

### 3 Les fonctions qui rendent plusieurs résultats

- valeurs propres de  $B$  :  $lambda = eig(B)$   
et les vecteurs propres ?  
 $[v, lambda] = eig(B)$  ;  $v$  est une matrice dont les colonnes sont les vecteurs propres de  $B$ ,  $lambda$  est une matrice diagonale, et les éléments de  $lambda$  sont les valeurs propres correspondantes.
- décomposition LU :  
 $u = lu(B)$   
 $[l, u, p] = lu(B)$  :  $p*B = l*u$ ,  $p$  est une matrice de permutation,  $l$  et  $u$  sont les matrices de la décomposition LU.  
 $[l1, u] = lu(B)$  :  $B = l1*u$  avec  $l1 = p^{-1}*l$ .
- décomposition QR :  
 $[q,r] = qr(B)$

### 4 Les fichiers

Il est possible d'écrire des instructions Matlab dans un fichier suffixé par ".m". Ces fichiers sont appelés des M-fichiers dans la documentation Matlab. Ces fichiers sont de deux sortes : les *scripts* et les *functions*.

- fichier *script* : exemple : fibo.m  

```
% Ce signe permet d'introduire un commentaire
% Ce fichier calcule les 100 premiers termes de la suite de Fibonacci
f=[1 1];
% le caractere ";" permet d'eviter l'affichage du resultat
i = 1;
while i < 100
    f(i+2) = f(i+1) + f(i);
    % au passage, remarquez la programmation dynamique
    % Matlab agrandit le vecteur f au fur et a mesure des besoins
    i = i+1;
end
plot(f)
```

Pour appeler cette suite de commandes, il suffit que ce fichier se trouve dans un des répertoires spécifiés dans la variable d'environnement *MATLABPATH* et de taper *fibo* dans la session Matlab.

- fichier *function* : exemple : fibo1.m  
On va pouvoir choisir à quel terme on s'arrête et récupérer fibo(i) et fibo(i+1).  

```
function [fi, fii] = fibo1(ind)
for i=1:ind+1
    f(i+2) = f(i+1) + f(i);
    i = i+1;
```

```

end
plot(f);
fi = f(ind);
fii = f(ind + 1);

```

Pour utiliser cette fonction, il suffit d'appeler *fib1(20)*.

Remarque : ceci implique que l'on ne peut mettre qu'une fonction par fichier, puisque la fonction et le fichier doivent avoir le même nom.

- *save*

cette commande sauve des variables matlab dans un fichier <file>.mat (par défaut matlab.mat), sous une forme non ascii, sauf si on utilise l'option *-ascii*.

- *load*

cette commande charge des variables sauvées par *save*.

- *diary*

cette commande permet de sauvegarder une session sous forme ascii :

*diary toto* sauve toute la session qui va venir dans le fichier *toto*

*diary off* cesse de sauvegarder la session

- *fscanf*

comme en C on peut lire dans un fichier ascii.

p.ex.:

```

% lecture d'un fichier de reels
fid = fopen('fname', 'r')
y = fscanf(fid, '%f', n)
% lit les n premier reels
% si n n'est pas precise, tout est lu jusqu'a EOF.
fclose(fid)

```

- *fprintf*

comme en C on peut écrire dans un fichier ascii.

p.ex.:

```

% ecrire des donnees reelles dans le fichier fname:
fid = fopen('fname', 'w')
fprintf(fid, '%f %f \n', y) % ecrit 2 elements par ligne
fclose(fid)

```

Pour plus de renseignements utilisez la commande *help*.

## 5 Autres fonctions utiles

- *input*

cette commande permet de lire des données au clavier.

*val = input('Entrez une valeur')*

- *disp*  
cette commande permet d'écrire des données à l'écran.  
*disp('Voici la valeur')*  
*disp(val)*
- *error*  
cette commande permet d'afficher un message d'erreur.
- Pour connaître la durée d'un calcul  
*t = clock;* on prend le temps au début  
*calculs*  
*duree = etime(clock, t);* on a la durée des calculs.
- *who, whos*  
*who* permet de savoir quelles sont les variables utilisées  
*whos* permet en plus de connaître leur place mémoire.
- *clear*  
permet d'effacer toutes les variable de la session.
- *sort*  
trie dans l'ordre croissant.
- les possibilités graphiques :  
*hist(vecteur)* dessine l'histogramme du vecteur  
*plot(vecteur)* dessine une courbe (*i, vecteur(i)*)  
*mesh(matrice)* donne un beau dessin en 3-D  
*subplot(n,m,p)* partage la fenêtre graphique en une (*n, m*)-matrice. Le graphique actuel sort dans la *p*-ème case.