Introduction à MATLAB. 2008[1]

Georges Rodriguez-Guisantes
Dépt. COMELEC

[1] Dernière mise à jour : Janvier 2008

I Introduction.

MATLAB est une application scientifique interactive orientée au calcul vectoriel et matriciel avec une puissante librairie de visualisation. Vous pouvez résoudre des problèmes de calcul très complexes d'une façon simple et rapide comparée aux langages de programmation traditionnels du type **C** ou **FORTRAN**. Le nom MATLAB est dérivé de l'anglais MATrix LABoratory.

Ce résumé vous présente d'une façon simple, les principales idées pour pouvoir utiliser cet outil de simulation sur les stations de travail disponibles au Centre de Calcul. Il ne remplace pas le guide de l'utilisateur fourni avec l'application (*User's Guide and Reference Guide for MATLAB*). Ce résumé, plus l'aide en ligne, suffit largement pour développer des applications relatives au cours de CAN. Vous êtes invités à vérifier tous les exemples ici présentés directment sur un PC ou station UNIX.

Une fois établie une session MATLAB, la commande `help` *nom_de_fonction*, fournie tous les renseignements relatifs à la fonction *nom_de_fonction* . Par exemple, la commande `help plot` donne toutes les options de la fonction de représentation graphique (*plotting*) en 2D. La commande `help` montre le liste des domaines qui ont une aide en ligne. Ainsi `help` *domaine* affiche toutes les fonctions en relation au domaine. Essayez, par exemple, `help comm`. Ces fonctions sont regroupées à la fin de cet annexe (extrait du guide de l'utilisateur). La commande `demo` présente un tour d'horizon de MATLABsous la forme d'exemples.

A Les commandes de base.

Pour démarrer une session MATLABil suffit d'entrer la commande `matlab` à partir de la ligne de commandes d'une session DOS ou en cliquant sur l'icône MATLAB, à partir de WINDOWS. Sur une station UNIX il suffit de taper `matlab` à partir de la ligne de commande. Une session intéractive est automatiquement ouverte ; MATLABattend des commandes la suite du symbole ». Essayez

```
» help
```

Pour quitter une session MATLABil suffit de taper `exit`.

B Les matrices.

MATLABtravaille essentiellement avec un seul type d'objet mathématique : une *matrice* rectangulaire avec des valeurs entières, réelles ou complexes. *Toutes* les variables sous MATLABsont représentées comme des matrices. Ainsi un scalaire sera pour MATLABune matrice de 1 ligne et 1 colonne. Les matrices avec une seule ligne ou une seule colonne sera un *vecteur*. Comment introduire une matrice sous MATLAB ? Plusieurs formes peuvent être envisagées :

1. comme une liste explicite des éléments de la matrice,
2. grâce à des fonctions internes à MATLABou des fonction externes définies par l'utilisateur,
3. créée dans un fichier externe avec votre éditeur préféré,
4. chargée à partir d'un fichier externe produit par un autre utilisateur ou une autre application.

Les commandes suivantes créent une matrice appelée A (3×3) :

```
A = [1 2 3; 4 5 6; 7 8 9];
```

```
A = [
```

```
1  2  3
4  5  6
7  8  9 ];
```

Essayez ! !

Le ; dans la déclaration de la matrice indique *passage à la ligne suivante.* Le ; peut être supprimé si on passe à la ligne suivante. Essayez ! !

Pour entrer une matrice complexe il suffit d'utiliser l'une des formes ci-dessous :

```
A = [1 2;3 4] + i*[5 6;7 8] ;
```

```
A = [1+5i 2+6i;3+7i 4+8i];
```

Pour visualiser le contenu de A il suffit de faire

A

(attention sans ; à la fin). Le symbole ; à la fin d'une ligne de commande évite l'affichage du contenu de la variable définie.

Essayez de faire :

```
A = [1 2;3 4] + i*[5 6;7 8]
```

Le symbole i ou j peut être utilisé comme unité imaginaire. **ATTENTION !** : si vous utilisez i ou j pour définir une nouvelle variable, vous écrasez l'unité imaginaire. Dans ce cas elle peut être redéfinie avec `ii = sqrt(-1)`.

Si la matrice est de très grandes dimensions, il est plus simple de la créer dans un fichier externe ASCII avec votre éditeur préféré. Ce fichier doit contenir tout simplement les valeurs du tableau rectangulaire souhaité. Soit `A.ext` le nom de ce fichier. La commande `load A.ext`, lit ce fichier et définit la matrice A dans l'environnement de travail MATLAB(le *workspace*).

Il y a des fonctions MATLABpré-définies qui permettent de créer des matrices rapidement. Les commandes `rand`, `magic` ou `hilb`, sont des exemples simples. La commande `rand(n)` définit dans le *workspace* une matrice de *nxn* à valeurs aléatoires uniformément distribués dans l'intervalle $[0, 1)$.

Les valeurs particulières d'un vecteur ou d'une matrice peuvent être lues en indiquant entre parenthèses les indices. Ainsi, la commande `A(2,3)` affiche l'élément de la deuxième ligne et la troisième colonne de A. De façon similaire, $x(3)$ indique le troisième élément du vecteur x. Essayez ! ! Les indices de ligne et colonne doivent être des entiers *positifs.*

C Opérations avec les matrices.

Les opérations matricielles suivantes sont valables sous MATLAB :

$+$	addition
$-$	soustraction
$*$	multiplication
$\hat{}$	puissance
$'$	conjugée et transposée
$\backslash$	division à gauche
$/$	division à droite

Évidemment ses opérations sont aussi valables pour les scalaires (puisqu'il s'agit de matrices de 1×1 !). Si les dimensions des matrices sont incompatibles, un message d'erreur est affiché, sauf pour le cas d'une opération par scalaire ; dans ce cas, l'opération par le scalaire s'effectue sur tous les éléments du tableau.

La *division matricielle* mérite un commentaire à part. Si A est une matrice carrée inversible et b est un vecteur de dimension compatible avec A, alors :

```
x=A\b;
```

est la solution de $A * x = b$. Respectivement

```
x=b/A;
```

est la solution de $x * A = b$.

D Opérations avec les vecteurs.

Les opérations d'addition et soustraction entre matrices se réalisent *élément à élément*. Il en est pas de même pour le produit, la division et la puissance de matrices. Le produit de deux matrices de mêmes dimensions ne correspond pas au produit élément à élément. Il est possible de faire des produits élément à élément grâce à l'opérateur $.*$ (il s'agit bien de *point-étoile* !). Ce genre d'opérateur est très utile puisqu'il permet de réaliser des opérations entre matrices de dimensions incompatibles. Par exemple, supposons vouloir le produit élément à élément des deux vecteurs suivants :

```
x = [1 2 3 4 5];
y = [2 2 2 2 2];
```

Les commandes :

```
z =x.*y;
z
```

affichent

```
z =
     2 4 6 8 10
```

Essayez la commande :

```
z =x*y;
```

La version *élément à élément* existe aussi pour les opérations *puissance* et *division*.

E Expressions et variables.

MATLABest un langage *interprète* ; les expressions que vous entrez dans l'ordinateur sont interprétées et calculées. Une commande MATLABest de la forme

variable = *expression*,

ou bien tout simplement

expression

Les expressions sont composées par des opérateurs, des fonctions et des variables. Le résultat de l'expression est sauvegardé dans la matrice *variable*. Si *variable* n'est pas définie dans la ligne de commande, alors la matrice qui résulte de l'évaluation de l'*expression*, est sauvegardée dans la matrice `ans` (pour *answer*).

ATTENTION ! ! MATLABdistingue mayuscules et minuscules ! Typiquement, `UT` c'est pas la même chose que `ut`.

La commande `who` (ou bien `whos`), affiche toutes les variables définies jusqu'à présent dans le *wokspace*. Une variable peut être supprimée avec la commande `clear` *nom_de_variable*. La commande `clear` supprime **TOUTES** les variables du *workspace*.

Un calcul interminable peut être arrêté avec `CTL c`.

Après une longue journée de travail, toutes les variables du *workspace* sont perdues si on quitte la session MATLAB. La commande `save` permet de sauvegarder toutes les variables du *workspace* dans le fichier `matlab.mat` (`save` *nom_de_fichier* permet de sauvegarder dans le fichier *nom_de_fichier*`.mat`). La commande `load` permet de charger les variables lors de la prochaine session MATLAB.

F Les fonctions pour créer des matrices

Les fonctions suivantes font partie du *noyau* MATLAB :

`eye`	identité
`zeros`	matrice de zéros
`ones`	matrice d'uns
`diag`	création, extraction de la diagonale
`triu`	matrice triangulaire supérieure
`tril`	matrice triangulaire inférieure
`rand`	matrice aléatoire
`hilb`	matrice de Hilbert
`magic`	carré magique
`toeplitz`	voir `help toeplitz`

Par exemple, `zeros(m,n)` produit une matrice de $m \times n$ avec tous les éléments nuls ! Si A est une matrice, alors `zeros(size(A))` produit une matrice de zéros avec les mêmes dimensions que A.

Si x est un vecteur, `diag(x)` est une matrice diagonale avec x dans la diagonale ; si A est une matrice carrée, alors `diag(A)` est un vecteur avec les éléments de la diagonale de A. Qu'est-ce que c'est `diag(diag(A))` ? Essayez !

Les matrices peuvent être construites à partir de sous-matrices. Par exemple si A est une matrice de 3x3, alors `B = [A, zeros(3,2) ; zeros(2,3), eye(2)]`, construit une certaine matrice de 5x5. Essayez-la !

G La gestion des itérations sous MATLAB.

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

- **For.**
  Voici un exemple trivial d'une boucle **for** pour générer un vecteur de dimension `n` :

```
x = [ ];
for i = 1:n
   x = [x,i];
end
```

  Les commandes :

```
x = [ ];
for i = n:-1:1
   x = [x,i];
end
```

  génèrent le même vecteur mais dans l'ordre inverse.
  À noter que `x = [ ]` définit une matrice *VIDE*.
  Les commandes :

```
for i = 1:m
  for j = 1:n
      H(i, j) = 1/(i+j-1);
   end
end
H
```

  génèrent une matrice de Hilbert de $m \times n$ éléments.

- **While.**
  La forme générale d'une boucle **while** est la suivante :
  `while` *relation*
  commandes
  `end`
  La séquence de *commandes* sera répétée jusqu'à ce que la *relation* soit logiquement fausse.
  Voici un exemple :

```
n = 0;
a=10;
while  n < a
   n = n + 1;
end
n
```

  Essayez-le !

- **If.**
  La forme générale d'une boucle **if** est la suivante :

`if` *relation*
commandes
`end`

Comme pour les boucles **for**, les commandes sont exécutées si *relation* est vraie. Plusieurs boucles peuvent être concaténées :

```
if n<0
  parite = 0;
elseif  rem(n,2) == 0
  parite = 2;
else
  parite = 1;
end
```

- **Relations.**
  Les relations en MATLABsont :

$<$	inférieur
$>$	supérieur
$<=$	inférieur ou égal
$>=$	supérieur ou égal
$==$	égal
$\sim=$	différent.

ATTENTION ! Comme dans le langage C, ne pas confondre = avec ==.
Les opérateurs de relation peuvent être utilisés conjointement avec les opérateurs logiques :

&	et
|	ou
$\sim$	non.

Appliqués aux scalaires, les opérateurs de relation sont 0 ou 1 selon que la relation soit fausse ou vraie. Essayez
» `3 < 5, 3 > 5, 3 == 5,`
et
» `3 == 3`.
Appliqués aux matrices de mêmes dimensions, la relation est une matrice de O et 1 selon que les éléments correspondants vérifient ou non la relation. Essayez
» `a = rand(5), b = triu(a), a == b`.
Une relation entre matrices est interprétée par **if** ou **while** comme vraie, si chaque élément de la matrice de relation vaut 1.
Les fonctions `any` et `all` peuvent être utilisées pour réduire les relations matricielles sous la forme de relation vectorielle, voire scalaire.
Faire `help any` et `help all` pour plus d'information.

H Fonction Scalaires.

Certaines fonctions de MATLABs'appliquent essentiellement aux scalaires, mais elles agissent élément à élément lorsqu'elles sont appliquées aux matrices. Ces fonctions sont :

sin	*asin*	*exp*	*abs*	*round*
cos	*acos*	*log*(*lognaturel*)	*sqrt*	*floor*
tan	*atan*	*rem*	*sign*	*ceil*

I Fonctions Vectorielles.

Autres fonctions MATLABagissent sur des vecteurs (vecteurs ligne ou colonne), mais elles agissent sur une matrice $m \times n (m \geq 2)$ par colonne pour générer un vecteur ligne. Des opérations ligne par ligne peuvent être effectuées en utilisant la matrice transposée. Voici quelques unes de ces fonctions :

max	*sum*	*median*	*any*
min	*prod*	*mean*	*all*
sort	*std*		

Par exemple, le maximum d'une matrice peut être trouvé grâce à la commande

```
max(max(A));
```

Essayez-la !

J Fonctions matricielles.

La puissance de MATLABse trouve dans ces fonctions matricielles. Voila quelques unes de ces fonctions :

`eig`	valeurs et vecteurs propres,
`chol`	factorisation de Cholesky,
`svd`	décomposition en valeurs singulières,
`inv`	inverse
`lu`	factorisation LU,
`qr`	factorisation QR,
`hess`	forme de Hessenberg
`schur`	décomposition de Schur
`expm`	matrice exponentielle
`sqrtm`	matrice racine carrée
`poly`	polynôme caractéristique
`det`	déterminant
`size`	taille
`norm`	norme
`rank`	rang

Les fonctions MATLABpeuvent avoir une seule variable de retour ou plusieurs. Par exemple `y = eig(A)`, ou simplement `eig(A)` produit un vecteur colonne qui contient les valeurs propres de la

matrice A. Par contre `[U,D] = eig(A)` produit une matrice U avec comme colonnes les vecteurs propres de A et une matrice diagonale D avec les valeurs propres comme diagonale principale. Essayez !

K Sous-matrices.

Les vecteurs et les sous-matrices d'une matrice principale peuvent être utilisés dans des traitements très particuliers. Ce genre de manipulations permet à l'utilisateur de ne pas utiliser des boucles **for** ou **while** qui sont en général très lentes sous MATLAB. *Ça vaut le coup de maîtriser la gestion des sous-matrices et des vecteurs !!.*

Voyons des exemples.

L'expression `x=1 :5 ;` est une façon rapide et simple de définir le vecteur `[1 2 3 4 5]`. On peut aussi construire des vecteurs réels avec `0.2 :0.2 :1.2` pour générer `[0.2, 0.4, 0.6, 0.8, 1.0, 1.2]`, ou bien des listes inverses comme `5 :-1 :1` pour générer `[5 4 3 2 1]`.

Les commandes suivantes sont utilisées pour construire une table de sinus :

```
x = [0.0:0.1:2.0]';
y = sin(x);
[x y]
```

À remarquer que la fonction `sin` opère élément à élément, et produit donc un vecteur y à partir du vecteur x.

La notation : peut être utilisée pour extraire une sous matrice d'une matrice. Par exemple, `A(1 :4,3)` est le vecteur colonne qui contient les 4 premiers éléments de la 3ème colonne de la matrice A.

Les : par eux mêmes indiquent toute la ligne ou colonne d'une matrice : `A( :,3)` est tout simplement la troisième colonne de A, et `A(1 :4, :)` sont les 4 premières lignes de A.

Des vecteurs d'entiers peuvent être utilisés comme indices : `A( :,[2 4])` contient les colonnes 2 et 4 de A.

Expliquer la commande `A( :,[2 4 5]) = B( :,1 :3)`.

Les colonnes 2 et 4 peuvent être multipliés à droite par une matrice 2x2 telle que `[1 2 ;3 4]` avec la commande `A( :,[2,4]) = A( :,[2,4])*[1 2 ;3 4] ;` Essayez ! !

Devinette : si x est un vecteur à n composantes, quel est le résultat des commandes suivantes ?

```
x = x(n:-1:1);
y = fliplr(x);
y = flipud(x');
```

L Fichiers `.m`

MATLABpeut exécuter une séquence de commandes enregistrées dans un fichier. Ces fichiers sont appelés *fichiers .m* parce qu'ils ont une extension *.m*. MATLABles reconnaît automatiquement grâce à cette extension et les exécute immédiatement. Ces fichiers sont des fichiers ASCII qui peuvent être édités avec votre éditeur préféré.

Il y a deux types de fichiers *.m* : les *scripts* et les *fonctions*.

- **Fichiers Script.**

  Un fichier *script* contient une séquence de commandes MATLAB. Si le fichier s'appelle par exemple `test.m`, alors lorsque vous exécuter la commande `test`, toutes les commandes contenues dans le fichier sont exécutées. Les variables d'un fichier *script* sont globales et peuvent écraser une variable de même nom dans le *workspace*.

  Les fichiers *script* servent à définir des matrices de grande taille ou des commandes que l'on répète très fréquemment. Par exemple, on écrit dans le fichier `matA.m`, la commande suivante :

```
A = [
     1 2 3 4
     5 6 7 8
    ];
```

  Si dans une session MATLABon exécute la commande `matA` alors la matrice `A` est définie dans le *workspace*. Observer que l'on peut aussi utiliser la fonction `load` pour réaliser cette tâche.

  Un *script* peut utiliser d'autres *scripts* (lui même pouvant être récursivement exécuté).

  Fichier de Fonctions.

  Un fichier d'une fonction sous MATLABest l'homologue des fonctions en langage **C** ou les sous-routines en **FORTRAN**. Vous pouvez créer vos propres fonctions MATLABparfaitement adaptées à votre problème. Les variables définie dans une fonctions sont des variables locales (variables *automatiques* du **C**) donc elles disparaissent dès que la fonction fini. Elles peuvent aussi être déclarées comme globales (voir `help global`).

  Voici un exemple simple d'une fonction contenue dans le fichier `randint.m` (**ATTENTION !** : le nom du fichier doit être le même que celui de la fonction) :

```
function  a = randint(m,n)
%RANDINT  G{\'e}n{\`e}re une matrice d'entiers al{\'e}atoires entre 0 et 9.
%         La fonction randint(m,n) rend une matrice de mxn.

a = floor(10*rand(m,n));
```

  Voici une version plus générale de cette fonction :

```
function  a = randint(m,n,a,b)
%RANDINT  G{\'e}n{\`e}re une matrice d'entiers al{\'e}atoires.
%         La fonction randint(m,n) rend une matrice de mxn
%         avec des valeurs entre 0 et 9.
%         La fonction randint(m,n,a,b) rend une matrice de mxn
%         avec des valeurs entre a et b.

if nargin < 3, a = 0; b = 9; end
a = floor((b-a+1)*rand(m,n)) + a;
```

La première ligne déclare le nom de la fonction ainsi que les variables d'entrée et de sortie. Sans cette ligne le fichier devient un *script*!! Cette fonction peut être utilisée avec la commande `z = randint(4,5)`. La matrice aléatoire sera une matrice de 4x5 (valeurs des paramètres `m` et `n` respectivement). La fonction rend le résultat dans la matrice `z`. Puisque les variables de la fonction sont locales elles peuvent avoir n'importe quel nom (variables muettes) et ne modifient pas la valeurs d'une variable du *workspace* avec le même nom.
Remarquer que la commande `nargin` (de l'anglais *number of arguments in input*) permet à l'utilisateur de passer un nombre variable de paramètres à la fonction (fonctionnalité semblable aux "..." du langage **C**). Dans cet exemple si on oublie les paramètres `a` et `b`, les valeurs par défaut 0 et 9 sont utilisées.
Une fonction peut avoir plusieurs variable de sortie. Par exemple :

```
function  [mean, stdev] = stat(x)
% STAT  Calcul de la moyenne et de l'{\'e}cart type.
%       Pour un vecteur x, stat(x) rend la moyenne de x;
%       [mean, stdev] = stat(x) rend la moyenne et l'{\'e}cart type.
%       Pour une matrice x, stat(x) agit par colonnes.

[m  n] = size(x);
if m == 1
  m = n;    % tient en compte le cas d'un vecteur ligne
end
mean = sum(x)/m;
stdev = sqrt(sum(x.*x)/m - mean.*mean);
```

Une fois écrite dans un fichier `stat.m`, la commande `MATLAB[xm, xd] = stat(x)`, calcule la moyenne et l'écart type de x et rend ces valeurs dans les variables `xm` et `xd`. Notez que c'est possible de prendre une seule variable de retour, même si la fonction en rend 2. Par exemple la commande `xm = stat(x)` (les crochets [et] ne sont pas nécessaires dans ce cas), calcule seulement la moyenne de `x`.
Le symbole `%` indique que le reste de cette ligne est un commentaire. Les premières lignes de commentaire servent pour décrire la fonction et elles sont affichées par la commande `help` *nom_de_fonction*. Une programmation propre oblige l'inclusion de ces lignes de commentaires pour la description de la fonction.
Quelques fonctions `MATLAB`sont contenues dans le noyau du programme et écrites en code machine pour accélérer leur exécution. C'est le cas, par exemple de la fonction `fft` (Transformée de Fourier Rapide) ou `filt` (fonction de filtrage). D'autres fonction sont contenues dans des fichiers `.m`. Un ensemble de fonctions qui sont en relation avec un sujet spécifique s'appelle une *boî te à outils* (*toolbox*).

M Texte, messages d'erreur et entrée sous MATLAB.

Une chaî ne de texte peut être définie en enfermant la chaî ne par des guillemets simples. Voici un exemple :

```
s = 'Message syst{\`e}me';
```

La variable `s` contient maintenant la chaî ne de caractères `Message système`. Les chaî nes de caractères peuvent être affichées avec la fonction `disp` au lieu d'être définies par une variable. Par exemple :

```
disp('Patientez SVP......');
```

affiche le message sur l'écran.

Les messages d'erreur sont mieux affichés avec la fonction `error`. Dans un fichier `.m` cette fonction affiche le message et arrête l'exécution du programme.

Dans une fonction, l'utilisateur peut entrer une valeur d'un paramètre grâce à la fonction `input`. Par exemple :

```
iter = input('Entrez le nombre d'it{\'e}rations :  ');
```

affiche le message et attend une valeur à partir du clavier. La touche `Entrée` associe la valeur entrée à la variable `iter` et continue l'exécution du programme.

N Session MATLAB.

Toutes les commandes d'une session `MATLAB`peuvent être enregistrées dans un fichier. pour cela il suffit de faire `diary` au début d'une session `MATLAB`. Toutes les commandes entrées dans cette session seront automatiquement sauvegardées dans le fichier `diary`. Ce fichier peut être utilisé comme *script* dans une prochaine session `MATLAB`.

O Présentation des résultats sous MATLAB.

`MATLAB`peut représenter des graphiques d'une façon simple et puissante. Les fonctions de représentation plus employées sont `plot, plot3, mesh,` et `surf`. Voici une introduction rapide aux commandes de représentation graphique.

- **Graphiques 2D**
  La commande `plot` génère des graphiques $x - y$ linéaires ; si x et y sont des vecteurs de même dimension alors `plot(x,y)` ouvre une fenêtre graphique et dessine les valeurs de y en ordonnées et celles de x en abscisses. Voici un exemple simple :

  ```
  x = -4:.01:4;
  y = sin(x);
  plot(x,y)
  ```

  On peut avoir plusieurs figures au même temps. Par exemple, si l'on a déjà une figure dans une fenêtre, on peut ouvrir une deuxième fenêtre pour une nouvelle figure avec la commande `figure`. Ainsi le figures sont numérotées en ordre croissant. Pour revenir sur une figure quelconque il suffit de faire `figure(`*numérodefigure*`)`. Le commande `gcf` rend le numéro de la figure active.

Les figures peuvent avoir des titres, ainsi de même pour les axes coordonnés. On peut également placé un texte dans n'importe quelle position d'une figure. Voici la liste des commandes MATLABles plus utilisées pour la gestion des figures :

title	Titre de la figure,
xlabel	titre de l'axe des abscisses
ylabel	titre de l'axe des ordonnées
gtext	insère du texte grâce à la souris
text	insère du texte à certaines coordonnées

La commande `grid` dessine une grille dans la figure actuelle.
Les axes sont calculés automatiquement. La commande `axis` permet d'éviter cette configuration automatique. Voir `help axis`.
Il y a deux façons de représenter deux courbes sur la même figure. Voici un exemple :

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,x,y2,x,y3)
```

La deuxième façon consiste à utiliser la commande `hold`. La commande `hold on`, retient le contenu de la fenêtre graphique de façon à pouvoir superposer une nouvelle courbe sur la même fenêtre. La commande `hold off` relâche la fenêtre.
On peut changer aussi les types des lignes et des points par défaut. En voici un exemple :

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
```

Essayez cet exemple pour pouvoir apprécier les différences. Les types disponibles de marqueurs pour les figures sont les suivants :
Types de lignes :

(a) marqueur solide (`-`)
(b) tiret `-`
(c) point `:`
(d) pointillé `-.`

Types de marqueurs :

(a) point `.`
(b) plus `+`
(c) étoile `*`
(d) cercle `o`

(e) x `x`

On peut aussi changer les couleurs des lignes et des marqueurs. Les couleurs disponibles sont :

- jaune `y`
- magenta `m`
- ciel `c`
- rouge `r`
- vert `g`
- bleu `b`
- blanc `w`
- noir `k`

Par exemple, `plot(x,y,'r-')` dessine des lignes rouges avec des tirets.

La commande `subplot` peut être utilisée pour diviser l'écran en plusieurs morceaux. Voir `help subplot`.

Autres fonctions pour la représentation des figures sont `polar, bar, hist, quiver, compass, feather, rose, stairs, fill`. Utilisez la fonction `help` pour obtenir une description de ces commandes.

- **Impression.**

  La figure affichée peut être imprimée grâce à la commande `print`. La commande `printopt` affiche la configuration par défaut de l'imprimante à laquelle est ratachée votre ordinateur. Voir `help printopt`.

- **Figures à 3-D.**

  La commande `plot` en deux dimensions a une version 3D appelée `plot3` qui produit des courbes en 3 dimensions. Si x, y, et z sont trois vecteurs de la même taille, alors la commande `plot3(x,y,z)` produit une figure en 3D en perspective avec une courbe qui passe par les points x, y, et z. Voici un exemple :

```
t=.01:.01:20*pi;
x=cos(t);
y=sin(t);
z=t.*t.*t;
plot3(x,y,z)
```

  Essayez cet exemple.

P Référence des commandes MATLAB

Managing Commands and Functions	
help	help facility
what	list M-files on disk
type	list named M-file
lookfor	keywork search through the help entries
which	locate functions and files
demo	run demonstrations
path	control MATLAB's search path
cedit	set parameters for command line editing and recall
version	display MATLAB version you are running
whatsnew	display toolbox README files
info	info about MATLAB and The MathWorks
why	receive flippant answer

Managing Variables and the Workspace	
who	list current variables
whos	list current variables, long form
save	save workspace variables to disk
load	retrieve variables from disk
clear	clear variables and functions from memory
pack	consolidate workspace memory
size	size of matrix
length	length of vector
disp	display matrix or text

Working with Files and the Operating System	
cd	change current working directory
pwd	show current working directory
dir, ls	directory listing
delete	delete file
getenv	get environment variable
!	execute operating system command
unix	execute operating system command ; return result
diary	save text of MATLAB session

Controlling the Command Window	
clc	clear command window
home	send cursor home—to top of screen
format	set output format
echo	echo commands inside script commands
more	control paged output in command window

Starting and Quitting from MATLAB	
quit	terminate MATLAB
startup	M-file executed when MATLAB is started
matlabrc	master startup M-file

Matrix Operators - Array Operators			
+	addition	+	addition
−	subtraction	−	subtraction
*	multiplication	.*	multiplication
^	power	.^	power
/	right division	./	right division
\	left division	.\	left division
'	conjugate transpose		
.'	transpose		
kron	Kronecker tensor product		

Relational and Logical Operators			
<	less than	&	and
<=	less than or equal	|	or
>	greater than	∼	not
>=	greater than or equal	xor	exclusive or
==	equal		
∼=	not equal		

Special Characters	
=	assignment statement
[ ]	used to form vectors and matrices ; enclose multiple function output variables
( )	arithmetic expression precedence ; enclose function input variables
.	decimal point
..	parent directory
...	continue statement to next line
,	separate subscripts, function arguments, statements
;	end rows, suppress printing
%	comments
:	subscripting, vector generation
!	execute operating system command

Special Variables and Constraints	
ans	answer when expression not assigned
eps	floating point precision
realmax	largest floating point number
reammin	smallest positive floating point number
pi	π
i, j	imaginary unit
inf	infinity
NaN	Not-a-Number
flops	floating point operation count
nargin	number of function input arguments
nargout	number of function output arguments
computer	computer type

Time and Date	
date	current date
clock	wall clock
etime	elapsed time function
tic, toc	stopwatch timer functions
cputime	elapsed CPU time

Special Matrices	
zeros	matrix of zeros
ones	matrix of ones
eye	identity
diag	diagonal
toeplitz	Toeplitz
magic	magic square
compan	companion
linspace	linearly spaced vectors
logspace	logarithmically spaced vectors
meshgrid	array for 3-D plots
rand	uniformly distributed random numbers
randn	normally distributed randon numbers
hilb	Hilbert
invhilb	inverse Hilbert (exact)
vander	Vandermonde
pascal	Pascal
hadamard	Hadamard
hankel	Hankel
rosser	symmetric eigenvalue test matrix
wilkinson	Wilkinson's eigenvalue test matrix
gallery	two small test matrices

Matrix Manipulation	
diag	create or extract diagonals
rot90	rotate matrix 90 degrees
fliplr	flip matrix left-to-right
flipud	flip matrix up-to-down
reshape	change size
tril	lower triangular part
triu	upper triangular part
.'	transpose
:	convert matrix to single column ; `A( :)`

Logical Functions	
exist	check if variables or functions exist
any	true if any element of vector is true
all	true if all elements of vector are true
find	find indices of non-zero elements
isnan	true for NaNs
isinf	true for infinite elements
finite	true for finite elements
isieee	true for IEEE floating point arithmetic
isempty	true for empty matrix
issparse	true for sparse matrix
isstr	true for text string
strcmp	compare string variables

Control Flow	
if	conditionally execute statements
else	used with **if**
elseif	used with **if**
end	terminate **if, for, while**
for	repeat statements for a specific number of times
while	repeat statments while condition is true
break	terminate execution of **for** or **while** loops
return	return to invoking function
error	display message and abort function

Programming	
input	prompt for user input
keyboard	invoke keyboard as if it were a script file
menu	generate menu of choices for user input
pause	wait for user response
function	define function
eval	execute string with MATLAB expression
feval	evaluate function specified by string
global	define global variables
nargchk	validate number of input arguments

Text and Strings	
string	about character strings in MATLAB
abs	convert string to numeric values
blanks	a string of blanks
eval	evaluate string with MATLAB expression
num2str	convert number to string
int2str	convert integer to string
str2num	convert string to number
isstr	true for string variables
strcmp	compare string variables
upper	convert string to uppercase
lower	convert string to lowercase
hex2num	convert hex string to floating point number
hex2dec	convert hex string to decimal integer
dec2hex	convert decimal integer to hex string

Debugging	
dbstop	set breakpoint
dbclear	remove breakpoint
dbcont	remove execution
dbdown	change local workspace context
dbstack	list who called whom
dbstatus	list all breakpoints
dbstep	execute one or more lines
dbtype	list M-file with line numbers
dbup	change local workspace context
dbdown	opposite of **dbup**
dbquit	quit debug mode

Sound Processing Functions	
saxis	sound axis scaling
sound	convert vector to sound
auread	Read Sun audio file
auwrite	Write Sun audio file
lin2mu	linear to mu-law conversion
mu2lin	mu-law to linear conversion

Elementary Math Functions	
abs	absolute value or complex magnitude
angle	phase angle
sqrt	square root
real	real part
imag	imaginary part
conj	complex conjugate
gcd	greatest common divisor
lcm	least common multiple
round	round to nearest integer
fix	round toward zero
floor	round toward $-\infty$
ceil	round toward ∞
sign	signum function
rem	remainder
exp	exponential base e
log	natural logarithm
log10	log base 10

Trigonometric Functions	
sin, asin, sinh, asinh	sine, arcsine, hyp sine, hyp arcsine
cos, acos, cosh, acosh	cosine, arccosine, hyp cosine, hyp arccosine
tan, atan, tanh, atanh	tangent, arctangent, hyp tangent, hyp arctangent
cot, acot, coth, acoth	cotangent, arccotangent, hyp cotan., hyp arccotan.
sec, asec, sech, asech	secant, arcsecant, hyp secant, hyp arcsecant
csc, acsc, csch, acsch	cosecant, arccosecant, hyp cosecant, hyp arccosecant

Special Functions	
bessel	bessel function
beta	beta function
gamma	gamma function
rat	rational approximation
rats	rational output
erf	error function
erf / inv	inverse error function
ellipke	complete elliptic integral
ellipj	Jacobian elliptic integral
expint	exponential integral
log2	dissect floating point numbers
pow2	scale floating point numbers

Matrix Decompositions and Factorizations	
inv	inverse
lu	factors from Gaussian elimination
rref	reduced row echelon form
chol	Cholesky factorization
qr	orthogonal-triangular decomposition
nnls	nonnegative least squares
lscov	least squares in presence of know covariance
null	null space
orth	orthogonalization
eig	eigenvalues and eigenvectors
hess	Hessenberg form
schur	Schur decomposition
cdf2rdf	complex diagonal form to real block diagonal form
rsf2csf	real block diagonal form to complex diagonal form
balance	diagonal scaling for eigenvalue accuracy
qz	generalized eigenvalues
polyeig	polynomial eigenvalue solver
svd	singular value decomposition
pinv	pseudoinverse

Matrix Conditioning	
cond	condition number in 2-norm
rcond	LINPACK reciprocal condition number estimator
condest	Hager/Higham condition number estimator
norm	1-norm,2-norm,F-norm,∞-norm
normest	2-norm estimator
rank	rank

Elementary Matrix Functions	
expm	matrix exponential
expm1	M-file implementation of **expm**
expm2	matrix exponential via Taylor series
expm3	matrix exponential via eigenvalues and eigenvectors
logm	matrix logarithm
sqrtm	matrix square root
funm	evaluate general matrix function
poly	characteristic polynomial
det	determinant
trace	trace

Polynomials	
poly	construct polynomial with specified roots
roots	polynomial roots—companion matrix method
roots1	polynomial roots—Laguerre's method
polyval	evaluate polynomial
polyvalm	evaluate polynomial with matrix argument
conv	multiply polynomials
deconv	divide polynomials
residue	partial-fraction expansion (residues)
polyfit	fit polynomial to data
polyder	differentiate polynomial

Column-wise Data Analysis	
max	largest component
min	smallest component
mean	average or mean value
median	median value
std	standard deviation
sort	sort in ascending order
sum	sum of elements
prod	product of elements
cumsum	cumulative sum of elements
cumprod	cumulative product of elements
hist	histogram

Signal Processing	
abs	complex magnitude
angle	phase angle
conv	convolution and polynomial multiplication
deconv	deconvolution and polynomial division
corrcoef	correlation coefficients
cov	covariance matrix
filter	one-dimensional digital filter
filter2	two-dimensional digital filter
cplxpair	sort numbers into complex pairs
unwrap	remove phase angle jumps across 360° boundaries
nextpow2	next higher power of 2
fft	radix-2 fast Fourier transform
fft2	two-dimensional FFT
ifft	inverse fast Fourier transform
ifft2	inverse 2-D FFT
fftshift	zero-th lag to center of spectrum

Finite Differences and Data Interpolation	
diff	approximate derivatives
gradient	approximate gradient
del2	five point discrete Laplacian
subspace	angle between two subspaces
spline	cubic spline interpolation
interp1	1-D data interpolation
interp2	2-D data interpolation
interpft	1-D data interpolation via FFT method
griddata	data gridding

Numerical Integration	
quad	adaptive 2-panel Simpson's Rule
quad8	adaptive 8-panel Newton-Cotes Rule
trapz	trapezoidal method

Differential Equation Solution	
ode23	2nd/3rd order Runge-Kutta method
ode23p	solve via **ode23**, displaying plot
ode45	4th/5th order Runge-Kutta-Fehlberg method

Nonlinear Equations and Optimization	
fmin	minimize function of one variable
fmins	minimize function of several variables
fsolve	solution to a system of nonlinear equations
	(find zeros of a function of several variables)
fzero	find zero of function of one variable
fplot	plot graph of a function

Two Dimensional Graphs	
plot	linear plot
loglog	log-log scale plot
semilogx	semilog scale plot
semilogy	semilog scale plot
fill	draw filled 2-D polygons
polar	polar coordinate plot
bar	bar graph
stairs	stairstep plot
errorbar	error bar plot
hist	histogram plot
rose	angle histogram plot
compass	compass plot
feather	feather plot
fplot	plot function

Graph Annotation	
title	graph title
xlabel	x-axis label
ylabel	y-axis label
zlabel	z-axis label for 3-D plots
grid	grid lines
text	text annotation
gtext	mouse placement of text
ginput	graphical input from mouse

Figure Window/Axis Creation and Control	
figure	create figure (graph window)
gcf	get handle to current figure
clf	clear current figure
close	close figure
hold	hold current graph
ishold	return hold status
subplot	create axes in tiled positions
axes	create axes in arbitrary positions
gca	get handle to to current axes
axis	control axis scaling and appearance
caxis	control pseudocolor axis scaling

Graph Hardcopy and Storage	
print	print graph or save graph to file
printopt	configure local printer defaults
orient	set paper orientation

Three Dimensional Graphs	
mesh	3-D mesh surface
meshc	combination mesh/contour plot
meshz	3-D mesh with zero plane
surf	3-D shaded surface
surfc	combination surface/contour plot
surf / l	3-D shaded surface with lighting
plot3	plot lines and points in 3-D space
fill3	draw filled 3-D polygons in 3-D space
contour	contour plot
contour3	3-D contour plot
clabel	contour plot elevation labels
contourc	contour plot computation (used by `contour`)
pcolor	pseudocolor (checkerboard) plot
quiver	quiver plot
image	display image
waterfall	waterfall plot
slice	volumetric visualization plot

3-D Graph Appearance	
view	3-D graph viewpoint specification
viewmtx	view transformation matrices
hidden	mesh hidden line removal mode
shading	color shading mode
axis	axis scaling and apearance
caxis	pseudocolor axis scaling
specular	specular reflectance
diffuse	diffuse reflectance
surfnorm	surface normals
colormap	color lookup table (see below)
brighten	brighten or darken color map
spinmap	spin color map
rgbplot	plot colormap
hsv2rgb	hsv to rgb color map conversion
rgb2hsv	rgb to hsv color map conversion

Color Maps	
hsv	hue-saturation-value (default)
jet	variant of `hsv`
gray	linear gray-scale
hot	black-red-yellow-white
cool	shades of cyan and magenta
bone	gray-scale with tinge of blue
copper	linear copper tone
pink	pastel shades of pink
flag	alternating red, white, blue, and black

3-D Objects	
sphere	generate sphere
cylinder	generate cylinder
peaks	generate demo surface

Movies and Animation	
moviein	initialize movie frame memory
getframe	get movie frame
movie	play recorded movie frames

Handle Graphics Objects	
figure	create figure window
axes	create axes
line	create line
text	create text
patch	create patch
surface	create surface
image	create image
uicontrol	create user interface control
uimenu	create user interface menu

Handle Graphics Operations	
set	set object properties
get	get object properties
reset	reset object properties
delete	delete object
drawnow	flush pending graphics events

Sparse Matrix Functions	
spdiags	sparse matrix formed from diagonals
speye	sparse identity matrix
sprandn	sparse random matrix
spones	replace nonzero entries with ones
sprandsym	sparse symmetric random matrix
spfun	apply function to nonzero entries
sparse	create sparse matrix ; convert full matrix to sparse
full	convert sparse matrix to full matrix
find	find indices of nonzero entries
spconvert	convert from sparse matrix external format
issparse	true if matrix is sparse
nnz	number of nonzero entries
nonzeros	nonzero entries
nzmax	amount of storage allocated for nonzero entries
spalloc	allocate memory for nonzero entries
spy	visualize sparsity structure
gplot	plot graph, as in "graph theory"
colmmd	column minimum degree
colperm	order columns based on nonzero count
dmperm	Dulmage-Mendelsohn decomposition
randperm	random permutation vector
symmmd	symmetric minimum degree
symrcm	reverse Cuthill-McKee ordering
condest	estimate 1-norm condition
normest	estimate 2-norm
sprank	structural rank
spaugment	form least squares augmented system
spparms	set parameters for sparse matrix routines
symbfact	symbolic factorization analysis
sparsefun	sparse auxillary functions and parameters

Low-level I/O Functions	
fclose	close file
fopen	open file
fread	read binary data from file
fwrite	write binary data to file
fgetl	readline from file, discard newline character
fgets	readline from file, keep newline character
fprintf	write formatted data to file
fscanf	read formatted data from file
sprintf	write formatted data to string
sscanf	read string under format control
ferror	inquire file I/O error status
frewind	rewind file
fseek	set file position indicator
ftell	get file position indicator