Guide MATLAB®

Court tutoriel pour les étudiants de MTH2210
Édition du 26 août 2009

Steven Dufour
École Polytechnique de Montréal

Introduction

Ce *court* guide d'introduction à MATLAB a pour but de tenter de faciliter vos premiers contacts avec ce logiciel et de vous servir de référence lorsque vous compléterez vos travaux pratiques dans le cadre du cours MTH2210. Le but n'est pas de vous fournir un guide complet des commandes MATLAB, mais plutôt de vous donner les outils nécessaires, **sans plus**, afin de pouvoir fonctionner dans ce cours. En d'autres termes, tout ce dont vous avez besoin est là. Sinon, des informations supplémentaires vous seront données dans l'énoncé des problèmes à résoudre.

Ce guide prend en compte que vous êtes des aspirants ingénieurs, i.e. que vous êtes déjà rompus à l'informatique et que vous êtes suffisamment curieux pour faire un effort personnel afin de devenir comfortable à utiliser cet outil. La présentation est donc la plus directe possible et nous ne ferons aucune introduction sur l'utilisation de l'interface graphique de MATLAB. Explorez!

Plusieurs caractéristiques de MATLAB en font un outil simple et puissant. MATLAB est un langage *interactif*, i.e. dès que vous entrez une commande, vous avez une réponse immédiate du logiciel. MATLAB s'occupe de déterminer le *type* et la *taille* des variables mises en mémoire, ce qui facilite la tâche du programmeur en lui permettant d'expérimenter et de se concentrer sur le problème à résoudre. Tous les types de MATLAB sont basés sur la notion de matrice. Un scalaire est une matrice de taille 1×1, un vecteur est une matrice de taille $n \times 1$ ou $1 \times n$, etc. Un autre concept de base de MATLAB est que si vous vous demandez quel est le calcul qu'une commande exécutera, en général il s'agit de l'opération la plus naturelle. Finalement, l'apprentissage de l'approche *vectorielle* de MATLAB sera utile pour ceux qui auront à utiliser des ordinateurs à architecture vectorielle ou parallèle.

La première section de ce guide est un court tutoriel, qui peut être vu comme votre première session MATLAB. Il couvre les commandes et la philosophie de base de MATLAB. Vous trouverez ensuite un court catalogue de commandes MATLAB, sous la forme d'un tableau. Ce format a pour but de vous permettre de trouver rapidement la commande dont vous avez besoin. Pour chaque commande, un exemple d'utilisation est donné, avec un commentaire ou une illustration du résultat. Une courte bibliographie est donnée en annexe, pour ceux qui veulent aller plus loin dans leur apprentissage de MATLAB. Lorsque ce document est consulté en ligne à l'aide du logiciel *Acrobat*, les hyperliens dirigeront votre fureteur vers les documents pertinents.

Ce guide est dynamique. Je suis ouvert à toute suggestion de correction ou de modification. Des mises-à-jour de ce document seront rendues disponibles sur une base régulière. Consultez la date de l'édition pour voir si le document a été mis à jour. Dans le but de créer des textes académiques, techniques et scientifiques de qualité, ce document est protégé par une license « Creative Commons ».

Steven Dufour
École Polytechnique de Montréal
mathappl.polymtl.ca/steven/
Le 26 août 2009

Tutoriel

Les premiers pas

Une fois MATLAB lancé, nous sommes en présence de l'invite (« *prompt* ») de MATLAB :

```
>>
```

MATLAB est prêt à recevoir nos commandes :

```
>> a=1          % tout ce qui vient après le symbole % est un commentaire
a =
     1
```

Notons qu'il n'est pas nécessaire de déclarer le type ou la taille de `a`. Nous voyons aussi un des avantages de travailler avec un langage interactif : nous avons une réponse immédiate de MATLAB.

```
>> A=2          % un autre scalaire, i.e. une matrice de taille 1x1
A =
     2
>> a            % on peut vérifier ce que contient une variable
a =
     1
>> A            % a et A sont des symboles différents
A =
     2
```

Voici comment déclarer un vecteur et une matrice :

```
>> b=[1 2]      % un vecteur ligne de dimension 1x2
b =
     1     2
>> b=[1,2]      % la même chose
b =
     1     2
>> c=[3;4;5]    % un vecteur colonne de dimension 3x1
c =
     3
     4
     5
>> d=[0.1 pi sqrt(2) 2+2 eps]  % laissez aller votre imagination
d =
    0.1000    3.1416    1.4142    4.0000    0.0000
>> A=[1 2;3 4]  % une matrice de dimension 2x2 (on écrase A=2)
A =
     1     2
     3     4
```

On aura une meilleure idée de ce que contient `d` en changeant le format d'impression :

```
>> format short e    % on aurait aussi pu utiliser <<format long e>>
>> d
d =
   1.0000e-01   3.1416e+00   1.4142e+00   4.0000e+00   2.2204e-16
>> format            % on revient au format initial
```

On peut facilement faire la concaténation de tableaux :

```
>> B=[A;b]            % on peut joindre une matrice et un vecteur ligne
B =
     1     2
     3     4
     1     2
>> C=[c B]            % ou une matrice et un vecteur colonne
C =
     3     1     2
     4     3     4
     5     1     2
```

On va chercher les composantes des tableaux de la même façon qu'avec les autres langages :

```
>> c(2)               % la deuxième composante de c
ans =
     4
>> A(1,2)             % la composante de A en première ligne, deuxième colonne
ans =
     2
>> A(3)               % ou, puisque les entrées sont stockées par colonne...
ans =
     2
```

Si nous n'assignons pas le résultat d'un calcul à une variable, MATLAB met le résultat dans la variable `ans` (« answer »), que nous pouvons utiliser au même titre que les autres variables :

```
>> ans(1)+sin(1)+pi  % MATLAB peut être utilisé comme une calculatrice
ans =
     5.9831
```

MATLAB nous donne plus de liberté que les autres langages pour accéder aux composantes des tableaux :

```
>> c([2,3])           % les deuxième et troisième composantes de c
ans =
     4
     5
>> A([2,3,4])         % les composantes (1,2), (2,1) et (2,2) de A
ans =
     3     2     4
```

L'opération précédente peut vous sembler plus intuitive si nous utilisons la commande `sub2ind` :

```
>> indices=sub2ind(size(A),[2,1,2],[1,2,2]);
>> A(indices)
ans =
     3     2     4
```

Une condition booléenne peut aussi être utilisée pour accéder aux composantes voulues :

```
>> A(A>3)             % les composantes de A supérieures à 3
ans =
     4
```

```
>> A(mod(A,2)~=0)     % on recherche les entrées impaires
ans =
     1
     3
```

Dans l'environnement MATLAB, une expression booléenne vraie est égale à 1 et une expression fausse est égale à 0. Les opérateurs booléens sont similaires à ceux rencontrés en C, soient « == », « ~ », « & » et « | » pour n'en nommer que quelques-uns.

Il est facile de modifier les matrices :

```
>> A(1,1)=0           % n'oubliez pas que les indices commencent à 1
A =
     0     2
     3     4
>> A(3,3)=9           % la matrice est redimensionnée automatiquement
A =
     0     2     0
     3     4     0
     0     0     9
```

Nous sommes aussi assurés que les autres nouvelles composantes seront initialisées à « 0 ». MATLAB nous donne aussi des outils puissants pour modifier les tableaux. Voici deux exemples :

```
>> A([6,8])=[7,5]     % puisque les entrées sont stockées par colonne
A =
     0     2     0
     3     4     5
     0     7     9
>> A(sub2ind(size(A),[1,3,1],[1,1,3]))=1     % quel est le résultat?
```

Nous verrons d'autres exemples à la section suivante. Si nous ne nous souvenons plus des variables que nous avons utilisées :

```
>> whos
  Name          Size            Bytes  Class

  A             3x3                72  double array
  B             3x2                48  double array
  C             3x3                72  double array
  a             1x1                 8  double array
  ans           2x1                16  double array
  b             1x2                16  double array
  c             3x1                24  double array
  d             1x5                40  double array
  indices       1x3                24  double array

Grand total is 40 elements using 320 bytes
```

Nous voyons que tous les objets créés sont des matrices avec des composantes stockées comme des réels en format double précision IEEE, même si nous avons défini ces tableaux à l'aide d'entiers.

L'opérateur « : »

Un outil puissant de MATLAB est l'opérateur « : » :

```
>> x=1:10       % création d'un vecteur ligne formé de 10 entiers
x =
     1     2     3     4     5     6     7     8     9    10
>> x=10:-1:1    % à rebours
x =
    10     9     8     7     6     5     4     3     2     1
>> x=0:0.1:0.5  % avec un incrément de 0.1 plutôt que 1
x =
         0    0.1000    0.2000    0.3000    0.4000    0.5000
```

On peut utiliser l'opérateur « : » pour extraire plusieurs composantes d'un vecteur ou d'une matrice :

```
>> d=x(2:4)     % on va chercher les composantes 2, 3 et 4 du vecteur x
d =
    0.1000    0.2000    0.3000
>> B=A(2:3,2:3) % soutirons une sous-matrice 2x2 de la matrice A
B =
     4     5
     7     9
>> e=A(2,:)     % ou si on veut extraire la deuxième ligne de A
e =
     3     4     5
>> e=A(2:end,1) % utilisons le dernier indice de la ligne (end=3 ici)
e =
     3
     1
```

Un outil utile pour modifier les matrices est le tableau vide « [] » :

```
>> C(2,:)=[]    % on peut effacer la deuxième ligne de C
C =
     3     1     2
     5     1     2
```

qui est plus pratique que la commande équivalente `C=C([1 3],:)` pour de grands tableaux. L'opérateur « : » peut aussi être utilisé pour intervertir les lignes et les colonnes des tableaux :

```
>> C(:,[3 2 1])
ans =
     2     1     3
     2     1     5
```

Les opérations mathématiques

Les opérations de MATLAB entre vecteurs et matrices suivent les conventions utilisées en mathématiques :

```
>> x=(1:3);
>> y=(3:-1:1);
>> x-y
ans =
    -2     0     2
>> x*y
??? Error using ==> *
Inner matrix dimensions must agree.
```

Du point de vue de l'algèbre linéaire, nous ne pouvons pas multiplier deux vecteurs lignes. Utilisons l'opérateur de transposition « ' » :

```
>> x*y'
ans =
    10
>> dot(x,y);          % on obtient le même résultat
>> B*b                % même problème entre la matrice B et b=[1 2]
??? Error using ==> *
Inner matrix dimensions must agree.
>> B*e                % pas de problème avec e=[3;1]
ans =
    17
    30
```

Pour ce qui est des opérations matrice-matrice :

```
>> B*C                % produit matrice-matrice
ans =
    37     9    18
    66    16    32
>> B^2                % l'opération B*B
ans =
    51    65
    91   116
>> B\C                % les divisions matricielles
ans =
     2     4     8
    -1    -3    -6
>> inv(B)*C           % qui est équivalent à B\C
ans =
    2.0000    4.0000    8.0000
   -1.0000   -3.0000   -6.0000
>> b/B                % qui est équivalent à b*inv(B)
ans =
    -5     3
```

Ce qui nous mène naturellement vers la résolution d'un système d'équations linéaires $A\vec{x} = \vec{c}$:

```
>> x=A\c              % résolution par factorisation LU, et non par inv(A)*c
x =
    0.4231
    1.6923
   -0.8077
```

Une autre caractéristique de MATLAB qui en fait un outil puissant est sa série d'opérations sur les composantes des tableaux :

```
>> B/2                % chaque composante est divisée par 2
ans =
    2.0000    2.5000
    3.5000    4.5000
>> B+1                % on ajoute 1 à chaque composante de B (et non B+I)
ans =
     5     6
     8    10
```

qui donne le même résultat que

```
>> B+ones(2,2)        % ones(2,2) est une matrice de <<1>> de dimension 2x2
ans =
     5     6
     8    10
```

Les fonctions mathématiques élémentaires sont aussi définies sur les composantes des tableaux :

```
>> sqrt(B)
ans =
    2.0000    2.2361
    2.6458    3.0000
```

Qu'en est-il des opérations «⋆», «/» et «^» sur les composantes des tableaux? C'est ici que l'opérateur «.» vient simplifier notre tâche :

```
>> B.*B               % B(i,j)*B(i,j) plutôt que B*B
ans =
    16    25
    49    81
>> C.^2               % C(i,j)^2 plutôt que C^2=C*C, qui n'a pas de sens ici
ans =
     9     1     4
    25     1     4
```

Par exemple, ces opérations sont utiles pour simplifier l'impression de tableaux de résultats numériques :

```
>> n=1:5;
>> [n ; n.^2 ; 2.^n]'
ans =
     1     1     2
     2     4     4
     3     9     8
     4    16    16
     5    25    32
```

Nous verrons plus loin que cette notation est souvent utilisée pour définir des fonctions.

Les programmes MATLAB (fichiers-M)

Afin d'éviter de devoir retaper une série de commandes, il est possible de créer un programme MATLAB, connu sous le nom de « fichier-M » (« M-file »), le nom provenant de la terminaison « `.m` » de ces fichiers. Il s'agit, à l'aide de l'éditeur de MATLAB (Menu « File → New → M-File ») ou d'un éditeur de texte, de créer un fichier en format texte qui contient une série de commandes MATLAB. Une fois le fichier sauvegardé (sous le nom `nomdefichier.m` par exemple), il s'agit de l'appeler dans MATLAB à l'aide de la commande :

```
>> nomdefichier
```

Les commandes qui y sont stockées seront alors exécutées. Si vous devez apporter une modification à votre série de commandes, vous n'avez qu'à modifier la ligne du fichier-M en question et à réexécuter le fichier-M en entrant le nom du fichier dans MATLAB à nouveau (essayez la touche ↑). Cette procédure vous évite de retaper une série de commandes à répétition. C'est la procédure recommandée pour vos travaux pratiques.

Il est possible de programmer des boucles et des branchements dans les fichiers-M :

```
for i=1:10              % boucle for
  if i<5                % branchement
    x(i)=i;             % n'oubliez pas votre point-virgule, sinon...
  else                  % on peut utiliser le else
    x(i)=0;
  end                   % on n'oublie pas de terminer le branchement
end                     % et la boucle
```

Le point-virgule à la fin d'une ligne signale à MATLAB de ne pas afficher le résultat de l'opération à l'écran. Une pratique courante est de mettre des « ; » à la fin de toutes les lignes et d'enlever certains de ceux-ci lorsque quelque chose ne tourne pas rond dans un programme, afin de voir ce qui se passe.

En général, on essaie d'éviter un tel programme. Puisque l'on utilise un langage interactif, ce bout de code fait un appel au logiciel à chaque itération de la boucle et à chaque évaluation de la condition. De plus, le vecteur change de taille à chaque itération. MATLAB doit donc faire une demande au système d'exploitation pour avoir plus de mémoire à chaque itération. Ce n'est pas très important pour un programme de petite taille, mais si on a un programme qui fait une quantité importante de calculs, un programme optimisé peut être accéléré par un facteur 1000!

Il est possible d'éviter ces goulots d'étranglement en demandant initialement l'espace mémoire nécessaire à vos calculs, puis en *vectorisant* la structure du programme :

```
x=zeros(1,10);          % un vecteur de dimension 1x10 est initialisé à zéro
x(1:4)=1:4;             % une boucle à l'aide de l'opérateur <<:>>
```

Dans cet exemple, l'espace mémoire n'est alloué qu'une seule fois et il n'y a que deux appels faits à MATLAB. Notons que le concept de vectorisation n'est pas particulier à MATLAB. Il est aussi présent dans les langages de programmation d'ordinateurs vectoriels et parallèles. Ceci étant dit, précisons que la vectorisation d'algorithmes est un « art » difficile à maîtriser et qui demande du travail.

Les programmeurs expérimentés prennent l'habitude de débuter leurs programmes MATLAB avec une série de leurs commandes préférées qui ont pour but d'éviter les mauvaises surprises. Par exemple :

```
>> clear all;           % on efface toutes les variables, fonctions, ...
>> format compact;      % supression des sauts de lignes superflus
>> format short e;      % éviter qu'une petite quantité s'affiche 0.0000
>> dbstop if error      % facilite l'impression d'une variable en cas d'erreur
```

Les fonctions MATLAB

En plus des fonctionnalités de base de MATLAB, une vaste bibliothèque de fonctions (les « toolbox » en langage MATLAB) sont à votre disposition. Pour avoir une liste des familles de fonctions qui sont disponibles, entrez la commande `help`. Pour voir la liste des fonctions d'une famille de fonctions, on peut entrer `help matfun` par exemple, afin de voir la liste des fonctions matricielles. Pour obtenir de l'information sur une fonction en particulier, il s'agit d'utiliser la commande `help` avec le nom de la fonction, soit `help cond` pour avoir de l'information sur la fonction `cond`. Si la fonction n'a pas été compilée afin de gagner de la vitesse d'exécution, il est possible de voir le code source en entrant `type cond`, par exemple.

Il est aussi possible de créer ses propres fonctions MATLAB. Le concept de fonction en MATLAB est similaire aux fonctions avec d'autres langages de programmation, i.e. une fonction prend un/des argument(s) en entrée et produit un/des argument(s) en sortie. La procédure est simple. Il s'agit de créer un fichier-M, nommons-le `mafonction.m`. Ce qui différentie un fichier-M d'une fonction est que la première ligne de la fonction contient le mot clef `function`, suivi de la définition des arguments en entrée et en sortie. Par exemple, voici une fonction qui interverti l'ordre de deux nombres :

```
function [y1,y2]=mafonction(x1,x2)
%
% Définition de la fonction "mafonction":
%   arguments en entrée: x1 et x2
%   arguments en sortie: y1 et y2
%
  y1 = x2;
  y2 = x1;
```

Il s'agit ensuite d'appeler votre fonction à l'invite MATLAB :

```
>> [a,b]=mafonction(1,2)
a =
     2
b =
     1
```

Si on veut définir une fonction pour calculer x^2, on écrira

```
function y=carre(x)
%
% Définition de ma fonction x^2
%
  y = x*x;
```

En entrant `help carre`, vous verrez les lignes de commentaires qui suivent immédiatement le mot clef `function` :

```
>> help carre


  Definition de ma fonction x^2
```

Utilisez cette fonctionnalité pour vos fonctions.

Les communications entre les fonctions et les programmes se font donc à l'aide des arguments en entrée et en sortie. La portée des variables définies à l'intérieur d'une fonction est donc limitée à cette fonction.

Cependant, dans certaines situations, il peut être pratique d'avoir des variables globales, qui sont définies à l'aide de la commande `global` :

```
function y=Ccarre(x)
%
% Définition de la fonction (C^3)*x^2
%
  global C          % variable globale, initialisée à l'extérieur de Ccarre
  e = 3             % variable locale a Ccarre, donc invisible ailleurs
  y = (C^e)*(x*x);
  C = 1             % C est modifiée partout ou elle est définie globalement
```

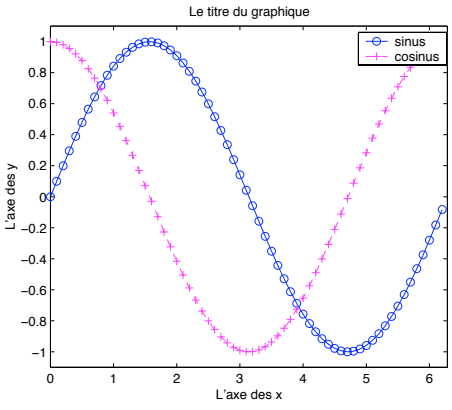
où la variable `C` serait aussi définie comme globale dans les fichiers-M où on veut y avoir accès. Par convention, les variables globales sont écrites en majuscules.

Les graphiques

La fonction de base pour tracer un graphique avec MATLAB est la commande `plot` qui prend comme arguments une série de points donnés sous la forme de 2 vecteurs, qu'elle relie de segments de droites. C'est la responsabilité de l'utilisateur de générer assez de points pour qu'une courbe régulière paraisse régulière à l'écran. De plus, il est possible de donner des arguments additionnels, ou d'utiliser d'autres fonctions, pour contrôler l'apparence d'un graphique :

```
>> x=0:0.1:2*pi;                          % l'ordonnée
>> plot(x,sin(x),'b-o',x,cos(x),'m--+');  % le graphe du sinus et du cosinus
>> axis([0 2*pi -1.1 1.1]);               % définition des axes
>> title('Le titre du graphique');
>> xlabel('L''axe des x');
>> ylabel('L''axe des y');
>> legend('sinus','cosinus');
```

Ces commandes donnent comme résultat :



On aurait aussi pu tracer ce graphique avec les commandes :

```
>> clf reset                  % on réinitialise l'environnement graphique
>> hold on
>> plot(x,sin(x),'b-o');      % un premier graphique
>> plot(x,cos(x),'m--+');     % on superpose un deuxième graphique
>> hold off
```

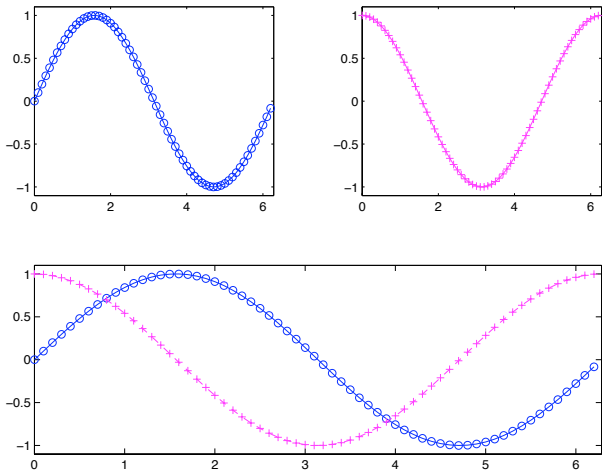
La commande `hold` évite que le premier graphique soit écrasé par le deuxième.

L'opérateur « `:` » n'est pas idéal pour créer le vecteur `x` dans ce contexte. Comme vous pouvez le voir sur la figure précédente, `x(end)` n'est pas égal à 2π. Dans ce cas, on préfère utiliser la commande `linspace` pour générer le nombre de points voulu dans un intervalle donné, alors que l'opérateur « `:` » nous donne plutôt le contrôle sur la distance entre les points.

```
>> x1=0:1:2*pi                % combien a-t-on de points entre 0 et 2*pi?
x1 =
     0     1     2     3     4     5     6
>> x2=linspace(0,2*pi,7)      % ici on sait qu'on en a 7, allant de 0 à 2*pi
x2 =
         0    1.0472    2.0944    3.1416    4.1888    5.2360    6.2832
```

On peut aussi comparer des résultats sous forme graphique à l'aide de la commande `subplot` :

```
>> subplot(2,2,1);   plot(x,sin(x),'b-o'); axis([0 2*pi -1.1 1.1]);
>> subplot(2,2,2);   plot(x,cos(x),'m-+'); axis([0 2*pi -1.1 1.1]);
>> subplot(2,2,3:4); plot(x,sin(x),'b-o',x,cos(x),'m--+');
>> axis([0 2*pi -1.1 1.1]);
```



La fonction `fplot` facilite le tracé de graphes de fonctions, en automatisant le choix des points où les fonctions sont évaluées :

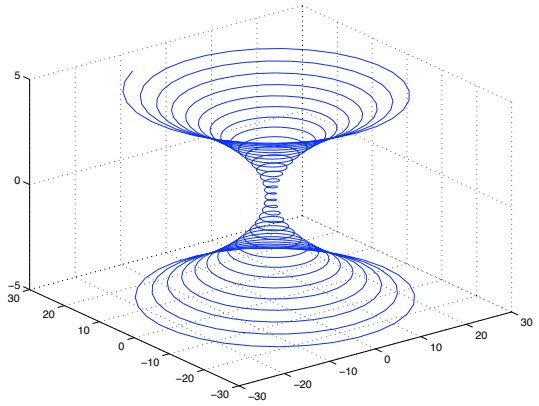
```
>> fplot('[sin(x),cos(x)]',[0 2*pi],'b-+')
```

On perd cependant de la latitude dans le choix de certaines composantes du graphique.

Finalement, les graphiques tridimensionnels, de type paramétrique, sont tracés à l'aide d'une généralisation de la commande `plot` :

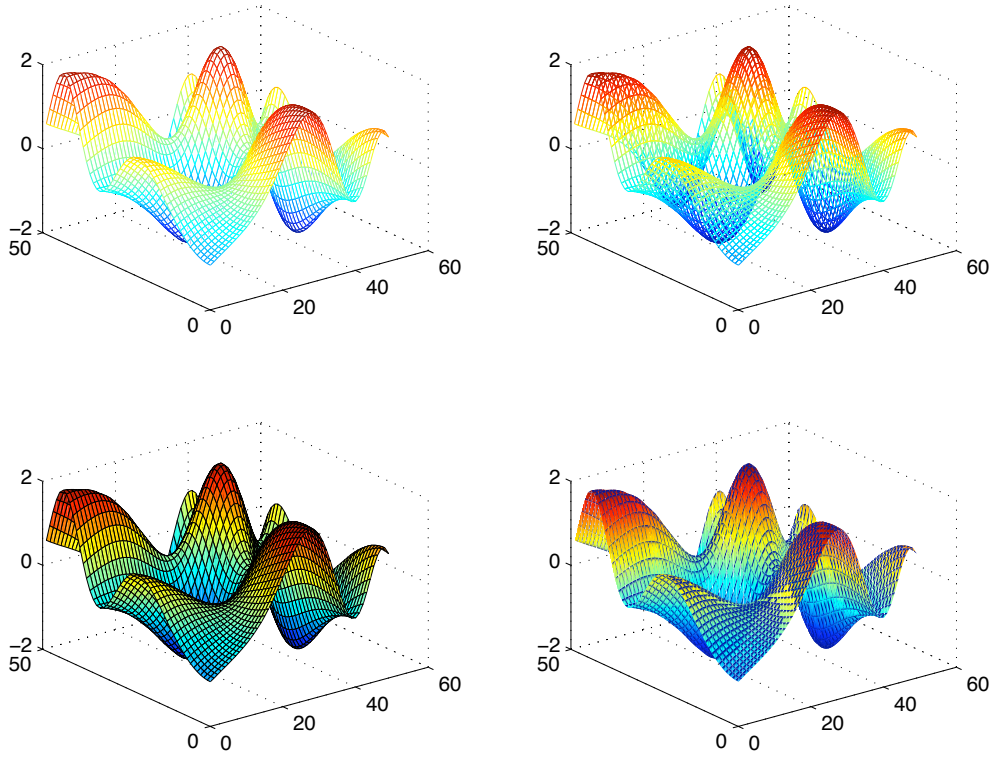
```
>> t=linspace(-5,5,1000);
>> x=(1+t.^2).*sin(20*t);     % utilisation de l'opérateur <<.^>>
>> y=(1+t.^2).*cos(20*t);
>> z=t;
>> plot3(x,y,z);
>> grid on;
```

qui donne comme résultat :



Pour ce qui est des graphes tridimensionnels, une étape intermédiaire est nécessaire avant d'utiliser les diverses commandes à notre disposition :

```
>> x=linspace(0,pi,50);
>> y=linspace(0,pi,50);
>> [X,Y]=meshgrid(x,y);          % on génère une grille sur [0,pi]x[0,pi]
>> Z=sin(Y.^2+X)-cos(Y-X.^2);    % encore l'opérateur <<.^>>
>> subplot(2,2,1); mesh(Z);
>> subplot(2,2,2); mesh(Z); hidden off;
>> subplot(2,2,3); surf(Z);
>> subplot(2,2,4); surf(Z); shading interp
```



Les sorties formattées

Nous aurons souvent à présenter des résultats numériques sous la forme de tableaux de nombres. Les programmeurs C ne seront pas dépaysés par les commandes MATLAB qui nous permettent de formatter l'écriture dans des fichiers :

```
>> uns=(1:5)' ones(5,2)                              % création d'un tableau
uns =
     1     1     1
     2     1     1
     3     1     1
     4     1     1
     5     1     1
>> sortie=fopen('sortie.txt','w');                   % ouverture du fichier
>> fprintf(sortie,'\t%s\n','Une serie de uns:');     % écriture de caractères,
>> fprintf(sortie,'\t%s\n','-----------------');     %  d'entiers et de réels
>> fprintf(sortie,'\t%2d %4.2f %5.2e\n',uns');       % attention à la transposée
>> fclose(sortie);                                   % fermeture du fichier
```

où la sortie dans le fichier `sortie.txt` sera :

```
>> type sortie.txt

        Une serie de uns:
        -----------------
         1 1.00 1.00e+00
         2 1.00 1.00e+00
         3 1.00 1.00e+00
         4 1.00 1.00e+00
         5 1.00 1.00e+00
```

Les caractères de contrôle pour le format de l'impression sont identiques à ceux que l'on retrouve dans le langage C. Dans l'exemple ci-dessus, « `\t` » représente une tabulation, « `\n` » un saut de ligne, « `%s` » une chaîne de caractères, « `%`n`d` » imprime un entier sur n caractères, « `%`m`.`n`f` » un réel sur m caractères avec n caractères après le point et « `%`m`.`n`e` » fait la même chose en notation scientifique.

Comme la plupart des commandes MATLAB, `fprintf` peut être utilisée sous forme vectorielle. Vous êtes donc encouragés à utiliser cette fonctionnalité dans vos programmes pour qu'ils s'exécutent plus rapidement.

Dernières remarques

Nous avons vu plus haut que la commande `type` nous permet d'afficher le contenu d'un fichier. Les commandes `dir` et `ls` affichent le contenu d'un répertoire et `cd` nous permet de changer de répertoire. Nous pouvons aussi passer des commandes directement au système d'exploitation en les précédant du caractère « ! » à l'invite MATLAB. Finalement, la commande `path` nous permet d'ajouter un chemin vers les fichiers-M que nous voulons utiliser. Par exemple, si nous avons des fichiers MATLAB sur une disquette :

```
>> path(path,'A:');           % sur DOS
>> path(path,'/mnt/floppy');  % sur RedHat Linux, par exemple
```

Ces chemins seront alors ajoutés à la variable `path` qui contient tous les chemins.

Catalogue de commandes

Disponible bientôt.

Bibliographie

Les livres

- MATLAB Guide, par Higham et Higham
- Mastering MATLAB 7, par Hanselman et Littlefield

Les guides

- MATLAB : Guide d'apprentissage, par Cloutier, Gourdeau et Leblanc (disponible à COOPOLY)
- An Introduction to MATLAB, par David F. Griffiths
- Introduction to MATLAB, par Graeme Chandler
- A Practical Introduction to MATLAB, par Mark S. Gockenbach
- Introduction to MATLAB, par Jeffery Cooper
- MATLAB Array Manipulation Tips and Tricks, par Peter J. Acklam