Introduction à Matlab

J.-B. Duval
Laboratoire Amiénois
de Mathématique Fondamentale et Appliquée
jean-baptiste.duval@u-picardie.fr

10 janvier 2010

Table des matières

Note

Les signes >> désignent le prompt Matlab. Ainsi toutes les instructions qui suivent ce symbole sont des exemples d'instruction Matlab.

1 Introduction

Matlab est un environnement informatique dédié au calcul scientifique. Ses deux principales caractéristiques sont les suivantes :

- Matlab est un langage de programmation dit de "haut niveau".
- Matlab est un puissant outil de visualisation graphique.

Un langage de "haut niveau" désigne un langage qui est plus proche de la pensée humaine que de celui de la machine. Matlab est en plus un langage interprété, *i.e.* qu'il n'est pas nécessaire de compiler son programme, il suffit de saisir son instruction puis de faire "entree" pour qu'elle soit exécutée.

Nous donnons ici un exemple de ce qu'apporte Matlab par rapport à un langage comme le C. Supposons que l'on veuille additionner deux vecteurs (deux tableaux) `v1` et `v2`. Avec un langage de bas niveau, on devra parcourir chacun des éléments du tableau, puis les additionner, alors qu'avec Matlab une seule instruction suffit :

```
  Langage type langage C                          Matlab

Pour i allant de 1 a n
  v3(i) = v1(i) + v2(i)                         v3 = v1 + v2
FinPour
```

On verra par la suite que beaucoup d'autres fonctions mathématiques sont déjà programmées dans Matlab. Ces fonctions peuvent être élémentaires (multiplication de matrice/vecteur etc...) ou plus complexes (inversion de matrice, calcul du minimum d'une fonction etc ...).

De façon générale comme son nom l'indique (**Mat**rix **Lab**oratory) Matlab est un langage matriciel, *i.e.* qu'il facilite les manipulations de matrices.

A tout moment vous pouvez avoir accès à l'aide en tapant *help commande* où *commande* est le nom d'une fonction dont vous voulez connaître le fonctionnement. Si vous ne connaissez pas la commande, vous pouvez toujours faire *lookfor mot* où *mot* est un mot-clé (écrit en anglais bien sûr) sur le sujet que vous cherchez.

2 Calculs élémentaires avec Matlab

2.1 Variables

a) Types de variables : on définit une variable en lui donnant un nom et une valeur numérique (ou une expression mathématique). Contrairement

à d'autres langages, il n'est pas nécessaire de déclarer les variables, *i.e.* d'indiquer si telle variable est un entier, un réel ou un complexe ou même si c'est un scalaire ou un tableau. En fait toute variable est considérée comme une matrice (un scalaire est une matrice à une ligne, une colonne ; un vecteur est une matrice à une ligne, n colonnes *etc*...). C'est l'utilisation qui détermine le type :

```
>> a = 5.21 ;                  % reel
>> b = 1+i;                    % complexe
>> c = linspace(0,1,100);      % tableau
```

b) Format d'affichage : On peut choisir le format d'affichage de ces variables grâce à l'instruction **format**. Attention cette instruction ne change pas la précision du calcul en mémoire, elle ne modifie que l'affichage.

Ainsi il suffit de taper *format form* où *form* est une des options listées dans le tableau 1, ensuite toutes les instructions seront affichées au format demandé. Le tableau 1 donne le résultat obtenu avec différents formats quand on tape pi.

Format utilisé	Affichage de la valeur π
format short	3.1416
format long	3.14159265358979
format short e	3.1416e+00
format long e	3.141592653589793e+00
format hex	400921fb54442d18

TAB. 1 – Format d'affichage.

c) Variables en mémoire : La valeur des variables reste en mémoire tout au long de la session. Matlab vous permet à tout moment de lister les variables de l'environnement de travail par les deux commandes suivantes :

- **who** : liste le nom des variables courantes.
- **whos** : précise pour chaque variable courante : son nom, sa dimension, sa classe.

Vous pouvez également effacer de la mémoire Matlab une ou plusieurs variables grâce à l'instruction **clear** :

```
>> clear all;  %apres 'clear all' aucune variable n'est en memoire.
>> k = 1;
>> i = 5;
>> clear k;    %apres 'clear k' seule k a été effacée de la mémoire.
```

2.2 Variables spéciales

Des constantes sont prédéfinies :
- **pi** représente π.
- **eps** vaut environ 2.2204e-16. C'est la distance entre deux réels informatiques consécutifs. Cette valeur est utilisée au niveau des tests d'arrêt des algorithmes.
- **inf** est le nombre infini, par exemple le résultat de $1/0$.
- **nan** est une abréviation de "Not a Number". C'est par exemple le résultat de l'opération $0/0$.
- **realmin** et **realmax** désignent respectivement le plus petit et plus grand nombre flottant.
- **i**, **j** : nombre complexe égal à $\sqrt{-1}$.

2.3 Opérateurs

Sont définies en Matlab, les opérations classiques : addition, soustraction, multiplication, division, et opérateur puissance donnés respectivement par :

$$+, -, *, \backslash, \wedge,$$

et l'opérateur d'affectation donnée par

$$=$$

Exemple :

```
>> x = 2 + 2;
>> y = 2*x;
>> z = y^5;
```

Après que ces trois instructions aient été exécutées, la variable x vaudra 4, y vaudra 8 et z vaudra 32768. Si une instruction se termine par un point virgule, le résultat de l'instruction n'est pas affiché à l'écran. Ainsi dans l'exemple précédent, si l'on veut que la valeur de z soit affichée, on écrira plutôt :

```
>> x = 2 + 2;
>> y = 2*x;
>> z = y^5
```

Attention aux utilisateurs de Maple pour qui le point virgule produit l'affichage du résultat et les :, le non affichage.

2.4 Fonctions mathématiques

Evidemment toutes les fonctions mathématiques classiques sont définies par Matlab et notamment :

- sin, cos, tan, cosh, sinh, tanh...
- asin, acos, atan, acosh, asinh, atanh ...
- exp, log, log10, sqrt, ..

3 Vecteurs

L'objet mathématique qu'est le vecteur est représenté par un tableau monodimensionnel.

3.1 Définition

a) Plusieurs façons de définir un vecteur :

- Si l'on souhaite définir chacun de ses éléments :

```
>> x1 = [1 5 1 89 2];
>> x2 = [1, 5 ,1, 89, 2];
>> y = [1; 5 ;1; 89; 2];
```

Les deux premières instructions produisent des vecteurs lignes, la troisième produit un vecteur colonne.

- Si les composantes de x varient d'un pas constant, on peut définir un vecteur par l'instruction $a : pas : b$ qui désigne le vecteur :
$(a, a + pas, a + 2 * pas, \cdots, a + n * pas)$ avec $a + n * pas \leq b$,
ou bien par la commande $linspace(a, b, n)$ qui désigne le vecteur à n composantes :
$(a, a + (b - a)/(n - 1), a + 2 * (b - a)/(n - 1), \cdots b)$.
Exemple :

```
>> x3 = 0:10;                  % le pas vaut par defaut 1
>> x4 = 10:-1:0;               % le pas peut etre negatif
>> x5 = linspace(0,10,101);
```

La variable `x3` contient le vecteur $(0, 1, 2, \cdots, 9, 10)$, `x4` contient $(10, 9, \cdots, 0)$ et le vecteur `x5` est le vecteur $(0, 1/100, 2/100, \cdots, 1)$.

b) Transposition : Les instructions précédentes produisent des vecteurs lignes. Si l'on veut des vecteurs colonnes, on peut les transposer :

```
>> y3 = x3'; y4 = x4'; y5 = x5';
```

c) Accès aux éléments : Il y a plusieurs façons d'accéder aux éléments d'un vecteur :

```
>> x1  = [0:2:25];
>> x1(5)
>> y = x1(5:12);
```

`x1(5)` désigne le 5eme élément de `x1`, *i.e.* ici 8. `y` est un vecteur qui contient les éléments 5, 6, ..., 12 du vecteur `x1`, *i.e.* ici le vecteur $(8\,,10\,,12\,,14\,,16\,,18\,,20\,,22)$.

Note : les indices des tableaux commencent à 1.

3.2 Opérations sur les vecteurs

3.2.1 Opérations élémentaires

Les opérations élémentaires comme l'addition, la soustraction et la multiplication sont applicables à des vecteurs. Evidemment les dimensions des deux vecteurs doivent concorder. Attention plus particulièrement il n'y a pas de sens à multiplier avec le signe $*$ deux vecteurs de même taille, la multiplication se fait au sens matriciel du terme.

```
>> x1 = [1 2 3 4 5];
>> y = [6;7;8;9;10];
>> x1 + 2*y';
>> x1*x1';
>> x1 + 5;
```

Cette dernière instruction qui ne parait mathématiquement pas très correcte (addition d'un vecteur et d'un scalaire ?), est acceptée par Matlab, il ajoute 5 à chacun des éléments de $x1$.

3.2.2 Autres opérations

Des fonctions prédéfinies sont à votre disposition pour manipuler les vecteurs :

length(v)	calcule la taille du vecteur v.
sum(v)	additionne tous les éléments du vecteur v.
prod(v)	multiplie tous les éléments du vecteur v.
max(v)	calcule le plus grand élément du vecteur v.
min(v)	calcule le plus petit élément du vecteur v.

4 Matrices

4.1 Définitions

4.1.1 Définition terme à terme

De façon similaire aux vecteurs (un vecteur n'est en fait qu'une matrice à 1 colonne ou 1 ligne) les matrices se définissent comme suit :

```
>> A = [1,2,3 ; 4,5,6 ; 7,8,9];
>> B = [1,2,3 ; 4:2:8 ; 7,8,9];
```

et même à partir de vecteurs :

```
>> v1 = [1 2 3 4 5 6];
>> v2 = [7 8 9 1 2 3];
>> A = [v1;v2]

A =

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

4.1.2 Quelques matrices particulières

Quelques matrices sont déjà prédéfinies, il suffit d'indiquer leur taille.

eye(n)	matrice identité de taille n.
zeros(n) ou zeros(n,m)	matrices de taille n ou n×m remplie de zeros.
ones(n), ones(n,m)	matrices de taille n ou n×m remplie de uns.
rand(n), rand(n,m)	matrices de taille n ou n×m remplie de nombres aléatoires.

4.1.3 Accès aux éléments

On peut accéder à un élément ou à une partie des éléments d'une matrice :
- On accède à l'élément qui se trouve sur la i$^{\text{ème}}$ ligne et la j$^{\text{ème}}$ colonne par l'instruction $A(i, j)$
- On extrait la n$^{\text{ème}}$ ligne de A grâce à l'instruction : $A(n, :)$. Le résultat est donc un vecteur ligne.

- On extrait la n$^{\text{ème}}$ colonne grâce à l'instruction $A(:, n)$. Le résultat est donc un vecteur colonne.
  Exemple :

```
   >> A = [0:2:10 ; 1:2:11 ; 20:2:30];
   >> A(2,:)
ans =

     1     3     5     7     9    11

   >> A(1,2:5)
ans =

     2
     4
     6
     8
```

4.2 Opérations sur les matrices

4.2.1 Opérations élémentaires

Les opérations élémentaires comme l'addition, la soustraction et la multiplication sont applicables à des matrices sous condition de compatibilité des dimensions.

A'	Transposée de A
inv(A)	Inverse de A
A + B	Addition de A et B
A - B	Soustraction de A et B
X*Y	Multiplication
A\ B	Equivalent à inv(A)*B
B/A	Equivalent à B*inv(A)

Evidemment on peut multiplier une matrice et un vecteur si les dimensions concordent :

```
>> A = ones(5,2);
>> b = ones(2,1)*4;
>> A*b
```

```
ans =

     8
     8
     8
     8
     8
```

4.2.2 Suppression/Ajout de colonne et de ligne

La suppression d'une colonne (resp. ligne) se fait par substitution avec une colonne (resp. ligne) vide.

```
   >>  A=[1 1 3; 4 0 6; 2,5,-1]

A =

     1     1     3
     4     0     6
     2     5    -1

   >>  A(2,:)=[] %suppression de la 2eme ligne

A =

     1     1     3
     2     5    -1

   >>  A(:,3)=[] %suppression de la 3eme colonne

A =

     1     1
     2     5
```

Les instructions suivantes permettent d'ajouter une ligne et une colonne

```
   >>  A=[A;3 4] %ajout d'une 3eme ligne

A =
```

```
     1     1
     2     5
     3     4

   >>  A(:,3)=[7 9 12]' %ajout d'une 3eme colonne

A =

     1     1     7
     2     5     9
     3     4    12
```

4.2.3 Manipulation des colonnes et des lignes

La commande **fliplr** (respectivement **flipud**) a pour effet de retourner les colonnes (respectivement les lignes) d'une matrice, par exemple si la i$^{\text{ème}}$ ligne de A est [a,b,c], alors la i$^{\text{ème}}$ ligne de $fliplr(A)$ sera [c,b,a].

```
   >> A=[1 1 3; 4 0 6; 2,5,-1]

A =

     1     1     3
     4     0     6
     2     5    -1
   >> fliplr(A)

ans =

     3     1     1
     6     0     4
    -1     5     2
```

Les deux lettres *lr* signifient *left/right*. De la même façon les deux lettres *ud* signifieront *up/down*. La fonction **flipud** fonctionne alors ainsi :

```
   >> flipud(A)

ans =

     2     5    -1
```

```
     4     0     6
     1     1     3
```

La commande **rot90** effectue une rotation de 90° d'une matrice :

```
   >> A = [1 1  7 ; 2 5 9 ; 3 4 12]

A =

     1     1     7
     2     5     9
     3     4    12

   >> rot90(A)

ans =

     7     9    12
     1     5     4
     1     2     3
```

La commande **reshape**(x, m, n) construit une matrice de taille m×n à partir du vecteur x (x doit avoir m*n éléments) :

```
   >> x=[1 2 3 4 5 6]; reshape(x,2,3)

ans =

     1     3     5
     2     4     6
```

Toutes les opérations s'effectuant sur des matrices ne sont évidemment pas citées ici. Nous n'en avons indiqué que les principales pour que le lecteur comprenne le mode de fonctionnement de Matlab.

4.3 Autres fonctions

Matlab possède plusieurs fonctions qui agissent soit sur les vecteurs soit sur les colonnes de matrices :

min, max	valeurs minimale et maximale des éléments
sum	somme des éléments
prod	produit des éléments
cumsum	somme cumulée des éléments
cumprod	produit cumulé des éléments
mean	valeur moyenne des éléments
sort	tri des éléments par ordre croissant

Voici un exemple d'utilisation de la fonction **max**. Cette fonction renvoie le plus grand élément de chaque colonne de la matrice :

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

A =

     1     2     3
     4     5     6
     7     8     9

   >> max(A)

ans =

     7     8     9
```

Une autre fonction utile est la fonction **repmat** qui (comme son nom l'indique) permet de construire une matrice à partir de la répétition en ligne et en colonne d'une autre matrice. Exemple :

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

C =

     1     2
     3     4

   >> repmat(C,2,3)
```

```
ans =

     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

4.4 Opérations de l'algèbre linéaire

L'ensemble des fonctions de l'algèbre linéaire élémentaires peuvent être effectuées par Matlab :

rank(A)	rang de A
det(A)	déterminant de A
trace(A)	trace de A
expm(A), logm(A), sqrtm(A)	exponentielle, logarithme et racine carrée de A.

Des fonctions permettent d'extraire les informations propres aux éléments nuls et non nuls d'une matrice creuse :

[i, j, s] = find(A)	i et j = indices des éléments non nuls , s= valeur de ces éléments.
nnz(A)	donne le nombre d'éléments non nuls de A.
nonzeros(A)	donne le vecteur des éléments non nuls de A.
spy(A)	affiche sur un graphique 2D la répartition des éléments non nuls

```
>> A = [3 2 0; -5 0 7; 0 0 1]

A =

     3     2     0
    -5     0     7
     0     0     1

>> [i,j,s]=find(A);
>> i'

ans =

     1     2     1     2     3
```

```
>> j'

ans =

     1     1     2     3     3

>> s'

ans =

     3    -5     2     7     1

>> nnz(A)

 ans =

     5

>> nonzeros(A)

 ans =

     3
    -5
     2
     7
     1

>> spy(A)
```

4.5 Opérations terme à terme

Les opérations terme à terme constituent une particularité de Matlab bien utile à certains calculs. Comme on va le voir, il faut toutefois être très prudent avec ces opérations.

• On peut appliquer des fonctions à chacun des termes d'une matrice :

```
   >> A=[pi/2 pi pi/3 ; pi 0 pi/2 ; pi 0 pi/4]
A =

   1.57079632679490   3.14159265358979   1.04719755119660
```

```
   3.14159265358979                   0   1.57079632679490
   3.14159265358979                   0   0.78539816339745

   >> sin(A)

ans =

   1.00000000000000   0.00000000000000   0.86602540378444
   0.00000000000000                   0   1.00000000000000
   0.00000000000000                   0   0.70710678118655
```

On peut ainsi appliquer n'importe quelle fonction à chacun des termes de A. Attention notamment aux fonctions **exp**, **log sqrt** par exemple qu'il ne faut pas confondre aux fonctions **expm**, **logm** et **sqrtm** (voir section (4.4)).

• On peut également multiplier terme à terme deux matrices (ces matrices doivent avoir la même taille) :

```
   >> A = [1 2 3 ; 4 5 6 ; 7 8 9];
   >> B = [2 2 2 ; 2 2 2 ; 2 2 2];
   >> A.*B

ans =

     2     4     6
     8    10    12
    14    16    18
```

Il existe de façon similaire l'opérateur puissance terme à terme et la division terme à terme.

Encore une fois ces opérateurs ne sont que des outils proposés par Matlab et ne correspondent pas à des opérateurs mathématiques. Il faudra donc bien réfléchir avant de les utiliser.

5 Nombres complexes

Les nombres complexes sont gérés par Matlab sous la forme algébrique $a+i*b$ ou sous la forme géométrique $rho*exp(j*theta)$. L'imaginaire pur tel que son carré vaut 1 est représenté par i ou j. Attention si vous utilisez par ailleurs i ou j comme variable réelle entière (par exemple par l'instruction $i = 1$), ces variables ne désigneront plus le nombre complexe...

Pour générer un nombre complexe, il y a deux possibilités :

```
>> z1 = 1 + 2*i;
>> z2 = complex(1,2);
>> z3 = sqrt(2)*exp(i*pi/4);
```

Voici la liste des fonctions relatives aux nombres complexes :

real	partie réelle
imag	partie imaginaire
abs	module
angle	argument
conj	conjugué
'	conjugué d'une matrice

6 Graphiques

Matlab est également un puissant outil de visualisation en 2D et en 3D.

6.1 Graphiques 2D

Pour tracer "à la main" la représentation graphique d'une fonction d'une variable $y = f(x)$, on choisit quelques points d'abscisses x_0, x_1, x_2, puis on calcule les ordonnées correspondantes : y_0, y_1, y_2. Ensuite, on joint ces points avec habileté et quelques connaissances sur la fonction (notamment sur la dérivée). Plus le nombre de point est élevé meilleure est l'allure de la courbe. Matlab fait le même raisonnement avec la commande *plot* mais peut utiliser un nombre très élevé de points.

6.1.1 L'instruction *plot*

L'instruction graphique la plus simple est :

```
plot(x,y)
```

où x et y sont des vecteurs contenant respectivement les abscisses et les ordonnées des différents points considérés.

Voici une façon de tracer la fonction $x \to \sin(x)$ sur l'intervalle $[-2\pi, 2\pi]$:

```
>> x = -2*pi:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y);
```

6.1.2 L'instruction *plot* et ses options

Couleur et Style : On peut spécifier la couleur et le style de la courbe par l'instruction :

```
>> plot(x,y,S)
```

où S est un des symboles donné dans le tableau suivant :

Couleur		Symbole		style de la ligne	
b	bleu	.	point	-	solid
g	vert	o	cercle	:	dotted
r	rouge	x	x-mark	-.	dashdot
c	cyan	+	plus	–	dashed
m	magenta	*	étoile	(none)	no line
y	jaune	s	carré		
k	noir	d	diamand		
		v	triangle (down)		
		∧	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagone		
		h	hexagone		

Si on veut par exemple tracer la fonction $x \longrightarrow \sin(x)$ en pointillés rouges :

```
>> x = -2*pi:pi/100:2*pi;
>> plot(x,sin(x),'r:');
```

Epaisseur traits/symboles : On peut choisir l'épaisseur respectivement du "trait" et des symboles par les options *'LineWidth'* et *'Markersize'* :

```
>> x = -2*pi:pi/100:2*pi;
>> plot(x,sin(x),'LineWidth',5);
>> hold on
>> plot(x,cos(x),'r+-','Markersize',5);
```

Habiller son graphique : Pour "habiller" son graphique

- **axis([Xmin Xmax Ymin Ymax])** fixe les bornes des axes.
- **grid on** /**grid off** ajoute ou enlève des grilles dans les plans des axes.
- **xlabel('texte')** / **ylabel('texte')** / **zlabel('texte')** ajoute une légende à un axe.
- **title('texte')** ajoute un texte en haut du graphique.
- **text(x,y,'texte')** positionne un texte de documentation d'une courbe

au point de coordonnées (x,y).
- **gtext('texte') :** analogue à la fonction text sauf que le point est choisi à la souris.
- **legend('leg1', 'leg2', ...)** : crée une boîte de légende pour les courbes tracées.

Echelles logarithmiques : On peut également observer ses données dans une échelle logarithmique :
- **loglog** permet de tracer log(y) en fonction de log(x).
- **semilogx** permet de tracer y en fonction de log(x).
- **semilogy** permet de tracer log(y) en fonction de x.

Plusieurs graphiques : Si on veut observer plusieurs courbes en même temps, trois possibilités s'offrent à nous :
- on peut tracer toutes les courbes sur un même graphique et donc sur une même figure.

Dans ce cas on trace le premier graphique puis l'on tape **hold on** qui donne l'instruction à Matlab de ne pas effacer le premier graphique et de superposer tous les graphiques suivants au premier.

```
>>     x = [-pi:0.01:pi];
>>     y = sin(x);
>>     z = cos(x);
>>     plot(x,y,'r*-');
>>     hold on
>>     plot(x,z,'b--');
```

- on peut tracer chaque courbe sur une figure différente. Dans ce cas l'instruction **figure** permet d'ouvrir une nouvelle fenêtre graphique.

```
>>     x = [-pi:0.01:pi];
>>     y = sin(x);
>>     z = cos(x);
>>     plot(x,y,'r*-');
>>     figure
>>     plot(x,z,'b--');
```

- on peut subdiviser une figure en une matrice de sous-fenêtres en utilisant la commande **subplot**. Elle s'utilise de la manière suivante :

```
subplot(m,n,p)
```

Cette commande symbolise le découpage de la figure en une matrice m lignes et n colonne. Le nombre p est compris entre 1 et $n * m$ il désigne le numéro de la sous-fenêtre où l'on veut tracer son graphe. Par exemple, si on veut tracer sin, cos, et tan sur une même figure mais sur des graphiques différents :

```
>>     x = linspace(0,2*pi,50);
```

inclure le résultat

```
>>      y = sin(x);
>>      z = cos(x);
>>      w = tan(x);

>>      subplot(2,2,1)
>>      plot(x,y)
>>      subplot(2,2,2)
>>      plot(x,z)
>>      subplot(2,2,3)
>>      plot(x,w)
```

Il ne faut pas non plus hésiter à utiliser l'interface graphique des nouvelles versions de Matlab.

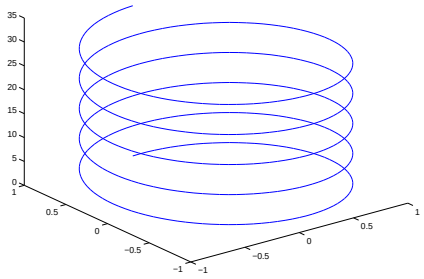
6.2 Graphiques en 3D

6.2.1 Courbes 3D

Le tracé de courbes en 3D se fait à l'aide de la commande **plot3** qui possède une syntaxe semblable à **plot**.

Exemple pour tracer une ellipse :

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t);
```



6.2.2 Surfaces

Pour la représentation de la surface $z = f(x, y)$, on a besoin de connaître les triplets de coordonnées $(X_i, Y_i, Z_{i,j} = f(X_i, Y_j))$ pour un certain nombre de points (X_i, Y_i) $i = 1..N, j = 1..M$. On voit que Z a la structure d'une matrice. Matlab demande également à ce que X et Y soient également des matrices. On construit ces dernières avec l'instruction **meshgrid**.

Si $x = (x_0, x_1, \cdots, x_n)$ et $y = (y_0, y_1, \cdots, y_m)$, l'instruction $[X, Y] =$

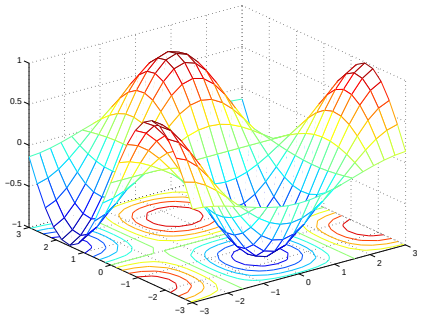


FIG. 1 – **meshc.**

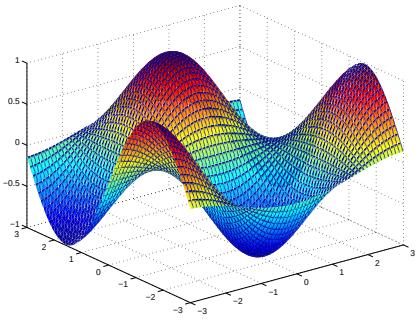


FIG. 2 – **surf.**

$meshgrid(x, y)$; produit les matrices X et Y à m lignes, n colonnes suivantes :

$$X = \begin{pmatrix} x_0 & x_1 & \cdots & x_n \\ x_0 & x_1 & \cdots & x_n \\ \cdots & \vdots & \vdots & \vdots \\ x_0 & x_1 & \cdots & x_n \end{pmatrix}, \qquad Y = \begin{pmatrix} y_0 & y_0 & \cdots & y_0 \\ y_1 & y_1 & \cdots & y_1 \\ \cdots & \vdots & \vdots & \vdots \\ y_m & y_m & \cdots & y_m \end{pmatrix}.$$

Grillage en perspective : mesh

```
>>% on commence par construire deux vecteurs qui définissent le maillage
>>% sur chacun des axes
>> x=[-3:0.3:3];
>> y=[-3:0.3:3];
>>% meshgrid fournit les matrices X et Y a partir de x et y
>> [X,Y] = meshgrid(x,y);
>> Z = cos(X).*sin(Y);
>> mesh(X,Y,Z)              % tracé de la surface
>> meshc(X,Y,Z)             % idem mais avec le contour sur le plan de base
```

L'instruction **mesh** donne un aspect "grillagé" à la surface. L'instruction **meshc** trace les lignes de niveau de la surface.

Surfaces colorées : surf

```
>>% on commence par construire deux vecteurs qui définissent le maillage
>>% sur chacun des axes
>> x=[-3:0.1:3];
>> y=[-3:0.1:3];
>>% meshgrid fournit les matrices X et Y a partir de x et y
>> [X,Y] = meshgrid(x,y);
>> Z = cos(X).*sin(Y);
>> surf(X,Y,Z)              % tracé de la surface
>> shading interp           % meilleure interpolation
```

La différence entre **surf** et **mesh** est que l'instruction **surf** colore chaque élément du grillage. Pour avoir un aspect plus uniforme, on tape **shading interp**.

Courbes de niveau : contour L'instruction :

```
>> contour(X,Y,Z,20);
```

construit 20 lignes de niveau.

7 Programmation

Matlab est un langage de programmation dont la syntaxe est proche de celle du C, de fortran ...

7.1 Opérateurs logiques et de relation

Les opérateurs suivants servent à comparer différentes variables :

$<$	inférieur,
$>$	supérieur,
$<=$	inférieur ou égal,
$<=$	supérieur ou égal,
$==$	egal,
$\tilde{=}$	non égal.

Ces opérateurs sont les opérateurs logiques :

&	et,
|	ou,
$\tilde{}$	négatif.

7.2 Structures de contrôle

– L'instruction **for**

```
for var=val_ini:pas:val_fin
      instructions
end
```

– L'instruction **while**

```
while expression_logique
      instructions
```

```
end
```

– L'instruction **if**

```
if ex_log_1
      instructions a executer si ex_log_1 est vraie
elseif exp_log_2
      instructions a executer si ex_log_1 est fausse et exp_log_2 vraie
else
      instructions a executer si ex_log_1 et  exp_log_2 sont fausses.
end
```

7.3 Scripts

Les instructions Matlab peuvent être tapées directement dans la fenêtre graphique. Mais si on a beaucoup d'instructions à taper il peut être intéressant de les écrire dans un fichier que l'on sauvera sur le disque dur.

Un script Matlab est un fichier dont l'extension est .m et qui contient une suite d'instructions.

Voici l'exemple d'un script Matlab **prem.m :**

```
% tout ce qui commence par le pourcent n'est lu par Matlab
clear all        % tout bon script devrait commencer par
close all        %   un clear et un close
disp('Mon premier script');
```

Ensuite dans la fenêtre Matlab, on s'assure d'être dans le répertoire où l'on a stocké son fichier, puis on exécute le script en tapant son nom :

```
>> pwd       % rend le nom du repertoire ou l'on se trouve
>> cd /home/users/nom/RepMatlab/TP1/   % cd permet d'aller dans le repertoire souha
>> ls        % ls permet de lister tous les fichiers presents dans le repertoire
>> prem      % appel au script prem.m (attention il n'y a pas de .m pour l'appel)
```

7.4 Fonctions

Une fonction est une script mais que l'on appelle en précisant la valeur d'un ou plusieurs paramètres. Exemple

```
function [s]=somme(a,b)
 temp = a+b;
 s = temp/2;
```

Attention cette fonction doit être stockée dans un fichier somme.m. Ensuite on peut l'utiliser soit dans un script, soit dans une autre fonction, soit directement dans la fenêtre de commande :

```
>> u = 5;
>> v = 10;
>> res = somme(u,v);
```

Il faut noter que les variables locales à la fonction (ici temp) n'existent pas en dehors de cette fonction, *i.e.* temp n'a aucune valeur dans la fenêtre de commande ou dans aucun script.

8 Foire aux messages d'erreur

Dans ce paragraphe nous répertorions les messages d'erreurs qui apparaissent très régulièrement lors des premières séances de TP. Matlab donne beaucoup d'indications sur l'erreur commise. Il suffit en général de lire le message d'erreur du début à la fin pour comprendre ce que Matlab veut nous dire.

```
   >> k
??? Undefined function or variable 'k'.
```

Vous n'avez pas donné de valeur numérique à k. Il faut qu'il y ait quelque part une instruction du type : *k=0*.

```
   >> x+y
??? Error using ==> plus
Matrix dimensions must agree.
```

Vous essayez d'additionner des vecteurs qui n'ont pas la même taille. Vérifiez tout de suite la taille de x et y en tapant *size(x)*, *size(y)*.

```
   >> x(700)
??? Index exceeds matrix dimensions.
```

Vous souhaitez accéder au $700^{\text{ème}}$ élément de x mais celui-ci ne possède pas autant d'éléments. Vérifiez tout de suite la taille de x en tapant *size(x)*.

```
   >> x(0)
??? Subscript indices must either be real positive integers or logicals.
```

L'élément 0 d'un tableau n'existe pas. Le premier élément du tableau x est $x(1)$. Attention également si vous écrivez $x(i)$, i doit être un entier strictement positif.

Index