Introduction (courte) à Matlab®

1 Introduction

Ce document est un guide de MATLAB ('MATrix LABoratory'), simplifié et adapté pour accompagner les exercices du cours 'Introduction à l'analyse numérique' par Prof. J.–P. Berrut. Les notions de base sont présentées de façon simple pour permettre de démarrer rapidement. Plus de détails se trouvent par exemple dans les références [2], [3] (en allemand), [4], [5], [6] (en anglais), [7], [8] (en français), [9] (en italien) et bien d'autres encore (Google). Il existe d'autres programmes (p. ex. Octave [10]) qui ont la même vocation que MATLAB et qui sont gratuits.

1.1 Introduction à Matlab

MATLAB est un logiciel de calcul matriciel à syntaxe simple. Avec ses fonctions spécialisées, MATLAB peut être considéré comme un langage de programmation adapté pour les problèmes scientifiques.

MATLAB est un interpréteur : les instructions sont interprétées et exécutées ligne par ligne. MATLAB fonctionne dans plusieurs environnements tels que Unix, Windows, Macintosh. MATLAB comprend la plupart des commandes de navigation Unix.

Il existe deux modes de fonctionnement :

1. Mode interactif : MATLAB exécute les instructions au fur et à mesure qu'elles sont entrées.
2. Mode exécutif : MATLAB exécute ligne par ligne un 'fichier.m' (programme en langage MATLAB.)

MATLAB s'occupe de déterminer le type et la taille des variables mises en mémoire, ce qui facilite la tâche du programmeur. Tous les types de MATLAB sont basés sur la notion de matrice. Un scalaire est une matrice de taille 1x1, un vecteur est une matrice de taille nx1 ou 1xn, etc.

1.2 Une session de travail Matlab

La configuration par défaut est présentée à la figure 1.

- La sélection de l'onglet 'Workspace' génère l'affichage (dans la fenêtre située en-dessous de l'onglet), des noms, valeurs et classes des variables utilisés dans la session courante.
- L'onglet 'Current Directory', donne, dans cette même fenêtre, la liste des fichiers contenus dans le répertoire courant.
- La fenêtre 'Command history' conserve les commandes écrites dans la fenêtre 'Command Window'.

1.3 Ce qu'il faut garder en mémoire

- Chaque ligne de commande est exécutée immédiatement après la touche 'Return'. Une ligne peut contenir plusieurs instructions séparées par des virgules.
- Tous les indices commencent à 1.
- MATLAB ne calcule *pas* en arithmétique exacte (voir plus bas).
- MATLAB différencie entre caractères majuscules et minuscules.
- A l'entrée du nom d'une variable, MATLAB sort sa valeur actuelle.
- Lorsqu'on ajoute un ' ;' à la fin d'une instruction, celle-ci est exécutée mais le résultat n'est pas affiché.
- Les flèches 'en haut' et 'en bas' du clavier peuvent être utilisées pour retrouver d'anciennes commandes. Une ancienne commande peut également être retrouvée en tapant ses quelques premiers caractères suivis de la flèche 'en haut'.
- La commande `help objet` retourne une courte description de la fonction ou du symbole `objet`. La commande `help` donne une liste des familles de fonctions disponibles.
- On peut quitter MATLAB soit en tapant `exit` ou `quit` soit en utilisant les raccourcis habituels.

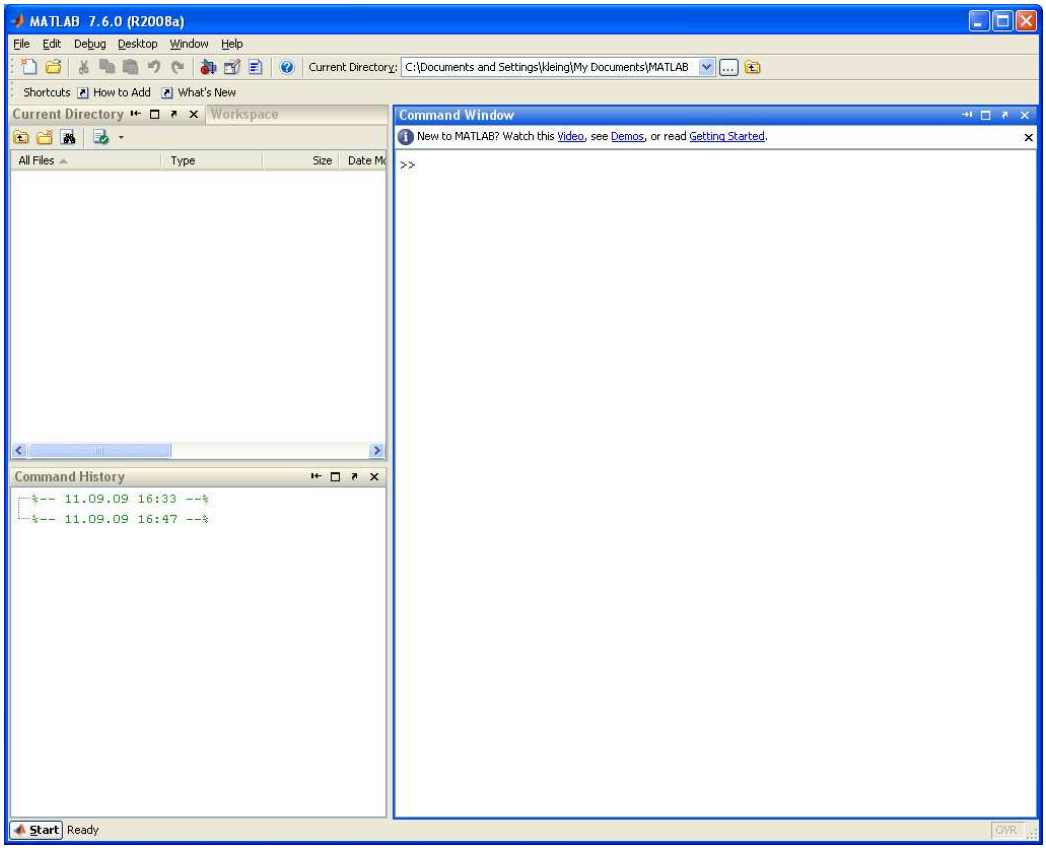


FIGURE 1 – L'espace de travail MATLAB

2 Tutoriel

2.1 Les premiers pas

Une fois MATLAB lancé, nous sommes en présence de l'invite ('prompt') de MATLAB.

```
>>
```

MATLAB est prêt à recevoir des commandes.

```
>> a=1.34722          % tout ce qui vient après le symbole % est un commentaire
a =
    1.3472
>> b=-8.13+15.44i      % un scalaire complexe
b =
   -8.1300 + 15.4400i
>> a+b^2
ans =
   -1.7095e+02 - 2.5105e+02i
>> A=[1.1 2 3; 4 5.5 6; -7 8 9.9]      % une matrice de taille 3x3
A =
    1.1000    2.0000    3.0000
    4.0000    5.5000    6.0000
   -7.0000    8.0000    9.9000
```

```
>> b=[.5  pi  sqrt(3.4)  6]          % un vecteur ligne de dimension 1x4
b =
    0.5000    3.1416    1.8439    6.0000
```

Il est à noter qu'il n'est pas nécessaire de déclarer le type des variables.

2.2 Informations sur l'espace de travail

Si on n'assigne pas le résultat d'un calcul à une variable, MATLAB le met dans la variable `ans`, que l'on peut utiliser au même titre que les autres variables.

```
>> ans+cos(2)
ans =
   -1.7137e+02 - 2.5105e+02i
```

Pour obtenir une liste des variables contenues dans l'espace de travail en cours, on peut utiliser les instructions `who` (affichage des variables dans l'espace de travail) ou `whos` (affichage détaillé). On peut également consulter la fenêtre 'Workspace'.
La commande `clear` est utilisée pour effacer des variables de l'espace de travail.

```
>> clear a b      % efface les variables a et b
>> clear all      % efface toutes les variables de l'espace de travail actuel
>> clc            % clear command window, efface toutes les commandes du 'Command Window'
```

2.3 Enregistrement des variables de travail ou de commandes dans un fichier

Pour enregistrer les variables de l'espace de travail dans un fichier, on utilise les instructions suivantes :

```
>> save       % enregistre toutes les variables dans un fichier matlab.mat, ou
>> save fichier1.mat a b A B
```

où la dernière commande enregistre les variables a, b, A, B sous fichier1.mat. Pour ramener l'espace de travail enregistré dans une session ultérieure, on utilise

```
>> load                    % respectivement
>> load fichier1
```

Si on veut enregistrer toutes les commandes que l'on entre ainsi que les résultats lors d'une session MATLAB, la commande

```
>> diary nomdufichier      % enregistre tout sous forme de texte dans le fichier nomdufichier.
```

3 Opérations mathématiques

3.1 Nombres et opérations arithmétiques

Les nombres réels peuvent être écrits sous différents formats.

```
5   1.42443   0.5243e-12   12.76e+06   0.0023323   -231.088
```

Les nombres complexes peuvent être écrits sous forme cartésienne ou polaire.

```
>> 0.5 + i*2.7          % forme cartésienne
>> -1.2 + j*0.443       % idem
>> 2.5 + 9.7i           % idem
>> 1.24*exp(0.288i)     % forme polaire
```

Pour choisir le format d'affichage des nombres, on utilise les commandes suivantes :

```
>> format short, exp(1)
ans =
    2.7183
>> format long, exp(1)
ans =
    2.718281828459046
>> format short e, exp(1)^10
ans =
    2.2026e+04
>> format long e, exp(1)^10
ans =
    2.20264657948067e+04
>> format hex, exp(1)^10
ans =
    40d5829dcf95055d
>> format rational, exp(1)
ans =
    1457/536
>> format          % on revient au format initial
>> help format     % pour d'autres formats
```

Les différents opérateurs arithmétiques à disposition sont les suivants (attention à la priorité des opérations) :

```
>> a+b     % addidtion
>> a-b     % soustraction
>> a*b     % multiplication
>> a/b     % division à droite
>> a\b     % division à gauche
>> a^b     % puissance
```

3.2 Vecteurs et matrices

3.2.1 Vecteurs

On peut définir un vecteur en donnant la liste de ses éléments,

```
>> x=[0.4 3 -12 44 4.9]
x =
    0.4000    3.0000   -12.0000    44.0000     4.9000
```

ou en donnant la suite qui forme le vecteur :

```
>> 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
>> x1=2:0.6:5              % avec un incrément de 0.6 au lieu de 1
x1 =
    2.0000    2.6000    3.2000    3.8000    4.4000    5.0000
>> y=linspace(1,10,6)      % 6 nombres entre 1 et 10 à distance égale
y =
```

```
    1.0000    2.8000    4.6000    6.4000    8.2000   10.0000
>> y=logspace(1,3,7)      % 7 nombres entre 10^1 et 10^3 à distance logarithmique égale
y =
   10.000    21.544    46.416   100.000   215.443   464.159  1000.000
```

La commande `length` donne la taille d'un vecteur.

```
>> length(x1)
ans =
     6
```

Remarque : La réutilisation du nom d'une variable déjà existante en écrase l'ancienne définition.

3.2.2 Matrices

On définit une matrice en donnant ses éléments.

```
>> A=[.5 2.7 3.8;4.5 -2.15 7;3.89 -4 1.1]
A =
    0.5000    2.7000    3.8000
    4.5000   -2.1500    7.0000
    3.8900   -4.0000    1.1000
```

Certaines matrices particulières sont prédéfinies dans MATLAB.

```
>> B=eye(4)          % matrice identité
B =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> C=ones(4,3)       % matrice de '1' de taille 4x3
C =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
>> D=zeros(3)        % matrice nulle de taille 3x3
D =
     0     0     0
     0     0     0
     0     0     0
```

La commande `size` donne la taille d'une matrice.

```
>> [n,m]=size(C)
n =
     4
m =
     3
```

Les éléments d'un vecteur ou d'une matrice peuvent être atteints en utilisant les indices sous la forme suivante :

```
>> x(8)          % 8e élément du vecteur x
ans =
```

```
     3
>> x(6:end)       % du 6e au dernier élément (end=10 ici)
ans =
    5    4    3    2    1
>> A(3,1)         % élément se trouvant à la 3e ligne, 1re colonne
ans =
     3.8900
>> B(:,3)         % la 3e colonne de la matrice B
ans =
     0
     0
     1
     0
>> C(3,:)         % 3e ligne de la matrice C
ans =
     1    1    1
>> D(2:3,:)       % les lignes 2 à 3 de la matrice D
ans =
     0    0    0
     0    0    0
>> A(8)           % 8e élément de la matrice A, les éléments sont stockés par colonnes
ans =
     7
>> A(A>3)         % les éléments de la matrice A supérieurs à 3
ans =
    4.5000
    3.8900
    3.8000
    7.0000
>> A(mod(A,2)~=0)       % les éléments impairs (condition booléenne)
ans =
    0.5000
    4.5000
    3.8900
    2.7000
   -2.1500
    3.8000
    7.0000
    1.1000
>> diag(A)        % vecteur contenant les éléments de la diagonale de A
ans =
    0.5000
   -2.1500
    1.1000
```

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 au langage C, soient '==', '~', '&' et '|' pour n'en nommer que quelques-uns.
Il est facile de modifier les matrices et les vecteurs.

```
>> D(1,1)=2
D =
    2    0    0
    0    0    0
```

```
    0     0     0
>> D([6   8])=[7   3]     % on profite du stockage par colonnes des éléments d'une matrice
D =
    2     0     0
    0     0     3
    0     7     0
>> D(4,4)=9               % la matrice est redimensionnée automatiquement
D =
    2     0     0     0
    0     0     3     0
    0     7     0     0
    0     0     0     9
```

Les autres nouvelles composantes introduites au dernier exemple sont toutes initialisées automatiquement à 0.
Un outil de modification utile est le tableau vide [ ].

```
>> D(2,:)=[]              % efface la 2e ligne de la matrice D
D =
    2     0     0     0
    0     7     0     0
    0     0     0     9
```

Pour intervertir les colonnes on procède comme suit :

```
>> D(:,[4   3   2   1])          % analogue pour les colonnes
ans =
    0     0     0     2
    0     0     7     0
    9     0     0     0
```

3.2.3 Opérations

Les opérations de MATLAB entre vecteurs et matrices suivent les conventions utilisées en mathématiques (attention aux tailles des matrices!).

```
>> A'           % matrice transposée
ans =
    0.5000    4.5000    3.8900
    2.7000   -2.1500   -4.0000
    3.8000    7.0000    1.1000
>> x'           % vecteur transposé
>> sum(x)       % somme des composantes de x
>> max(x), min(x)     % composante maximale resp. minimale de x
>> inv(A)       % matrice inverse de A
>> eig(A)       % valeurs propres de A
>> det(A)       % déterminant de A
>> norm(A), norm(A,'inf'), norm(A,1)       % différentes normes
>> A1+A2        % addition
>> A1-A2        % soustraction
>> A1*A2        % multiplication matricielle
>> x1*x2'       % produit scalaire entre x1 et x2, équivaut à dot(x1,x2)
>> A*x          % multiplication matrice-vecteur (vecteur-colonne!)
>> A^4          % puissance matricielle, équivaut à A*A*A*A
```

```
>> sqrt(A)   % racine carrée par composantes
>> A1/A2     % équivaut à A1*inv(A2)
>> A1\A2     % équivaut à inv(A1)*A2
>> A1/3      % division par composantes par un scalaire
>> A1+1      % addition par composantes par un scalaire
```

Les opérations élément par élément des vecteurs et des matrices sont effectuées en ajoutant un point '.' devant les opérateurs '*', '\', '/' et '^'.

```
>> A.*A          % différent de A*A, équivaut à A.^2
ans =
    0.2500    7.2900   14.4400
   20.2500    4.6225   49.0000
   15.1321   16.0000    1.2100
>> x.*x
ans =
   100    81    64    49    36    25    16     9     4     1
>> A1.*A2        % multiplication par composantes (attention aux dimensions)
>> A.^(1/2)      % équivaut à sqrt(A)
>> y./(1:7)
ans =
   10.000    10.772    15.472    25.000    43.089    77.360   142.857
```

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
```

3.3 Les polynômes

Les polynômes sont définis comme des vecteurs contenant les coefficients dans l'ordre décroissant (ne pas oublier les coefficients 0!). On considère le polynôme $x^3 - 5x + 8$.

```
>> pol=[1 0 -5 8]
pol =
   1   0  -5   8
>> r=roots(pol)     % détermine les zéros du polynôme
r =
  -2.8026
   1.4013 + 0.9439i
   1.4013 - 0.9439i
>> poly([2 1])      % génère un polynôme dont les zéros sont 2 et 1
ans =
   1  -3   2
```

3.4 L'évaluation de fonctions

Il existe plusieurs manières d'évaluer une fonction. La première consiste à définir d'abord la fonction comme une chaîne de caractères et de l'évaluer pour un vecteur donné.

```
>> f='sin(x)';
>> x=[0 pi/2 pi 3*pi/2 2*pi];
>> eval(f)
   0    1.0000    0.0000   -1.0000   -0.0000
```

On peut également définir et évaluer une fonction de manière plus intuitive, soit avec `inline`, soit avec `@` (les deux sont équivalents).

```
>> g=inline('t.^2+2');
>> g(3)
   11
>> h=@(x) x.^2+5*x-2;
>> h([1 2])
   4    12
```

4 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 d'un vecteur des abscisses suivi d'un vecteur des ordonnées. De plus, il est possible de donner des arguments additionnels (options) ou d'utiliser d'autres fonctions pour contrôler l'apparence d'un graphique.

```
>> x=linspace(0,2*pi,60);
>> plot(x,sin(x),'b-o'), hold on, plot(x,cos(x),'m--+');
>> plot(x,sin(x),'b-o',x,cos(x),'m--+');      % fait la même chose
>> axis([0 2*pi -1.1 1.1]);                   % x dans [0,2pi], y dans [-1.1, 1.1]
>> title('Le titre du graphique','Fontsize',14);
>> xlabel('L''axe des x','Fontsize',14);
>> ylabel('L''axe des y','Fontsize',14);
>> h=legend('sinus','cosinus','Location','SouthWest');
>> set(h,'Fontsize',14)
>> grid on                                    % ajouter une grille
```

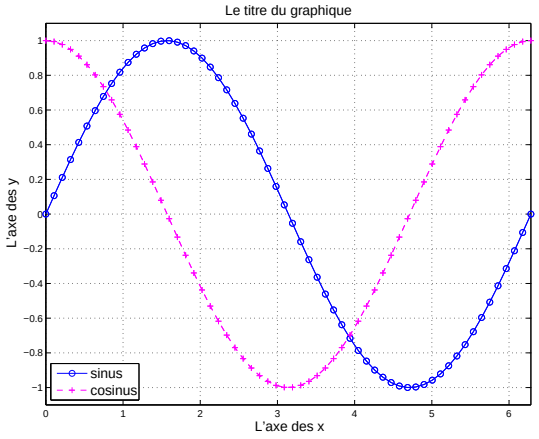
La commande `hold on` évite que le premier graphique ne soit écrasé par le deuxième.
Ces commandes donnent comme résultat :

On peut choisir le format du graphique

```
>> plot(x,y)          % tracer y(x) avec échelle linéaire
>> semilogx(x1,y1)    % tracer y1(x1) avec une échelle logarithmique selon x
>> semilogy(x2,y2)    % logarithmique selon l'axe des y
>> loglog(x2,y2)      % logarithmique selon les deux axes
>> polar(theta,r)     % tracer r(theta) en coordonnées polaires
>> bar(x3,y3)         % tracer y3(x3) sous forme de barres
>> help plot          % toujours utile pour plus d'informations
```

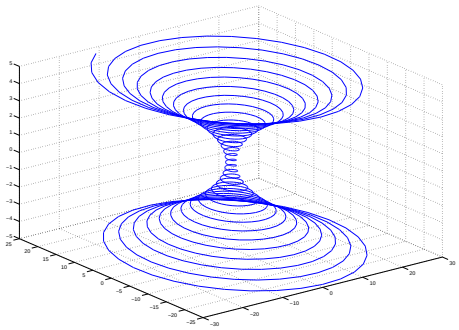
La fonction `fplot` facilite le tracé de graphes de fonctions en automatisant le choix des points où les fonctions sont évaluées. Cependant on perd de la latitude dans le choix de certaines composantes du graphique.

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



Les graphiques de type paramétrique sont tracés à l'aide d'une généralisation de la commande `plot`.

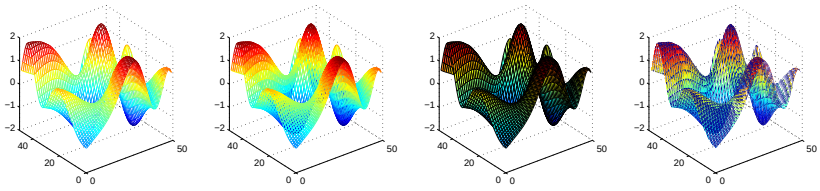
```
>> t=linspace(-5,5,1000);
>> x=(1+t.^2).*sin(20*t);
>> plot(x,t);
>> y=(1+t.^2).*cos(20*t);      % pour un plot tridimensionel
>> z=t;
>> plot3(x,y,z);               % plot en 3D
>> grid on;
```



Pour les graphiques tridimensionnels, une étape intermédiaire est nécessaire avant d'utiliser les diverses commandes à disposition.

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

La commande `subplot` permet d'afficher plusieurs graphiques selon une distribution matricielle.



La commande `clf` 'clear current figure' efface la figure ouverte.

Finalement la commande **ezplot** (dit 'easy plot') est utile pour illustrer sur $[-\pi, \pi]$ une fonction qui est définie partout.

```
>> ezplot('x^2 - 2*x + 1')
```

Les commandes **ezpolar**, **ezplot3**, **ezcontour**, **ezsurf** et **ezmesh** fonctionnent de manière similaire.

5 Programmation avec Matlab

5.1 Communication avec l'usager

On peut afficher un message ou une valeur à l'écran avec la commande **disp**.

```
>> disp('Ceci est un test')
Ceci est un test
```

On peut aussi entrer une valeur avec la commande **input**.

```
>> x=input('Valeur de x= ')
Valeur de x=
```

MATLAB attend qu'un nombre soit tapé sur le clavier (qui sera récupéré dans la variable x).

5.2 Boucle `for`

On peut créer une ou plusieurs boucle(s) en utilisant **for ... end**.

- Boucle **for** simple :

```
for k=1:100
  x(k)=2*k;
end
```

- Deux boucles **for** imbriquées :

```
for i=1:50
  for j=1:60
    A(i,j)=i+j;
  end
end
```

5.3 Boucle `while`

On peut créer une boucle en utilisant **while ... end**.

```
n=1;
while n<100
  x=n*0.05;
  y(n)=5.75*cos(x);
  z(n)=-3.4*sin(x);
  n=n+1;
end
```

5.4 Instruction `if ... elseif ... else`

L'instruction `if` ... `elseif` ... `else` ... `end` permet de choisir entre plusieurs options.

```
n=input('Entrer un nombre positif ');
if rem(n,3)==0
  disp('Ce nombre est divisible par 3.')
elseif rem(n,5)==0
  disp('Ce nombre est divisible par 5.')
else
  disp('Ce nombre n'est divisible ni par 3 ni par 5.')
end
```

6 Les programmes Matlab

6.1 Les 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' ; le nom provenant de la terminaison '.m' de ces fichiers. A l'aide de l'éditeur de MATLAB (Menu File → New → M-File), ou d'un éditeur de texte, on crée un fichier texte qui contient une série de commandes MATLAB. Une fois le fichier sauvegardé sous le nom de `nomdefichier.m` par exemple, il suffit de naviguer dans le bon répertoire (en utilisant la commande `cd` ou en inscrivant le chemin d'accès dans l'onglet 'Current Directory') et de l'appeler dans MATLAB à l'aide de la commande suivante :

```
>> nomdefichier
```

ou à l'aide du bouton 'Save and run' dans l'éditeur. Les commandes qui y sont stockées seront alors exécutées. Pour apporter des modifications à la série de commandes, il suffit de modifier le fichier.m et de le compiler.

6.2 Les fonctions Matlab

MATLAB dispose d'une vaste bibliothèque de fonctions appelée 'toolbox'. Il est également possible de créer de nouvelles fonctions MATLAB. Le concept de fonctions en MATLAB est similaire aux fonctions dans d'autres langues de programmation, i.e. une fonction prend des arguments en entrée et produit des arguments en sortie. La procédure est simple. Il s'agit de créer un fichier.m, par exemple `mafonction.m`. Ce qui différencie un fichier.m d'une fonction est que la première ligne de la fonction contient le mot clé `function`, suivi de la définition des arguments en entrée et en sortie. Par exemple, voici une fonction qui intervertit l'ordre de deux nombres :

```
function [y1,y2]=mafonction(x1,x2)
%MAFONCTION intervertit l'ordre de deux nombres
% input : x1 et x2
% output : y1 et y2
y1=x2;
```

```
y2=x1;
```

Il s'agit ensuite d'appeler la fonction dans MATLAB.

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

En entrant `help mafonction`, MATLAB affiche les premières lignes de commentaire qui suivent le mot clé `function` et qui commencent par un symbole %.

```
>> help mafonction
MAFONCTION intervertit l'ordre de deux nombres
 input: x1 et x2
 output: y1 et y2
```

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 limitée à cette fonction. Cependant, dans certaines situations, il peut être pratique d'avoir des variables globales, qui sont définis à l'aide de la commande `global` à l'intérieur du programme, par exemple,

```
global C; C=...
```

où la variable `C` serait aussi définie comme globale dans les fichiers.m.

7 Quelques applications

7.1 Matlab ne calcule pas en arithmétique exacte

Voici un programme qui approche le plus petit nombre machine non-nul (epsilon0.m).

```
epsilon=1;
iter=0;
while epsilon
    epsilon=epsilon/10;
    iter=iter+1;
end
epsilon=10^(-iter+1)
iter
```

```
>> epsilon0
epsilon = 9.8813e-324
iter =  324
```

Un programme qui cherche le plus grand nombre qui, ajouté à 1, ne change pas la valeur 1 est le suivant (epsilon1.m) :

```
epsilon=0.1;
iter=0;
while 1+epsilon~=1
    epsilon=epsilon/10;
    iter=iter+1;
end
```

```
epsilon
iter
```

```
>> epsilon1
epsilon =  1.0000e-16
iter =  15
```

7.2 Approximation de l'ensemble de Mandelbrot

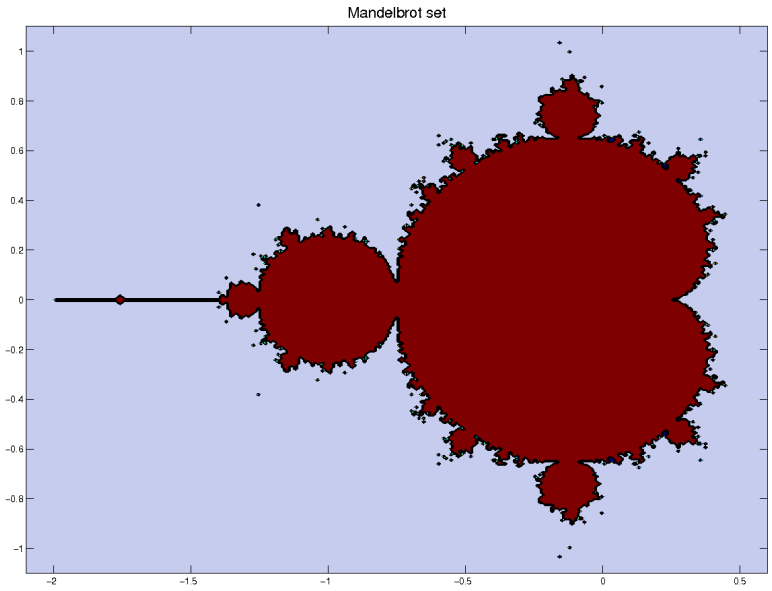
Voici le code mandel.m :

```
function mandel
%MANDEL dessine une approx. de l'ensemble de Mandelbrot

x=linspace(-2.1,0.6,301);
y=linspace(-1.1,1.1,301);
[X,Y]=meshgrid(x,y);
C=complex(X,Y);

Z_max=1e6; it_max=50;
Z=C;
for k=1:it_max
   Z=Z.^2 +C;
end

contourf(x,y,double(abs(Z)<Z_max))
title('Mandelbrot set','Fontsize',16)
```

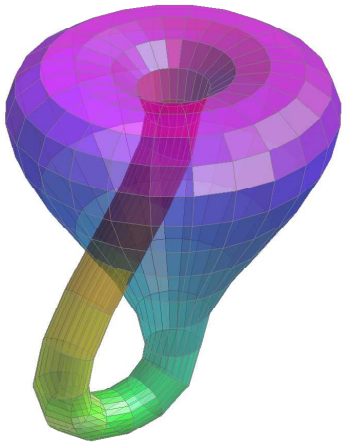


7.3 Démos et exemples

MATLAB vient avec une panoplie de démonstrations et d'exemples dans la toolbox 'demos' qui se trouve dans l'arborescence de l'installation. Par exemple la commande

```
>> xpklein
```

retourne une belle image de la bouteille de Klein.



Références

[1] Guide MATLAB, Steven Dufour, École Polytechnique de Montréal

[2] http ://people.inf.ethz.ch/arbenz/MatlabKurs

[3] http ://ipa.iwr.uni-heidelberg.de/dokuwiki/lib/exe/fetch.php ?media=teaching:st08:mtp_ matlab.pdf

[4] http://www.maths.dundee.ac.uk/~ftp/na-reports/MatlabNotes.pdf

[5] D. J. Higham and N. J. Higham, MATLAB Guide, 2nd edition, SIAM, 2005

[6] T. A. Driscoll, Learning MATLAB, SIAM, 2009

[7] perso.mines-albi.fr/~louisnar/matlab/polymatlab.pdf

[8] http://enacit1.epfl.ch/cours_matlab/

[9] http://linux3.dti.supsi.ch/~bucher/wp-content/uploads/2011/03/matlab.pdf

[10] http://www.gnu.org/software/octave

Notes