Licence Sciences et Technologies,
Calcul Formel et Numérique

UNIVERSITÉ DE PROVENCE
Aix-Marseille I

Calcul Formel et Numérique, Partie I

N.Vandenberghe
nvdb@irphe.univ-mrs.fr

Table des matières

1 Introduction à Matlab

1.1 Quelques généralités

Matlab est un logiciel commercial de calcul et de développement conçu pour des utilisateurs scientifiques. Il est très largement utilisé dans l'industrie et les institutions de recherche. Il est apprécié pour sa simplicité (apprentissage relativement aisé), pour son interactivité, pour l'étendue des possibilités et pour la rapidité de certains algorithmes. Initialement Matlab était surtout utilisé pour faire des simulations numériques. Il permet maintenant, avec l'ajout de module supplémentaires (toolbox) de faire de l'acquisition et du traitement d'images, de l'acquisition et du traitement de signaux électriques (à l'aide d'une carte d'acquisition) du traitement de données médicales etc...

Matlab interprète les commandes, c'est à dire que lors de l'exécution d'un programme, le code est traduit automatiquement en langage machine. Ce processus est transparent pour l'utilisateur, mais il nécessite un certain temps qui peut nuire à la rapidité d'exécution des programmes.

Matlab traite des données numériques et non pas des équations formelles (Mathematica et Mapple sont des exemples de programme permettant de faire du calcul formel) et est particulièrement adapté au calcul matriciel. Il permet également la visualisation des résultats et dispose de fonctions graphiques relativement puissantes.

Les langages compilés classiques ou orientés objet (C, C++, Objective C, pascal, fortran, java...) permettent en général d'écrire des programmes plus rapides à l'exécution. Le compilateur traduit directement le code en langage machine, et donc l'étape d'interprétation à l'exécution n'a pas lieu. Ces langages sont plus adaptés à l'écriture de programmes complexes et rapides mais en général le temps de développement est supérieur.

Il existe des alternatives open source. Scilab (`http://www.scilab.org`) et Octave (`http://www.octave.org`) peuvent tous les deux être utilisés en remplacement de Matlab et sont partiellement compatibles avec Matlab (un programme écrit pour Matlab peut en général être exécuté avec Scilab ou Octave). Python (`http://www.python.org`) peut également être utilisé pour le calcul numérique (en particulier avec une extension comme numarray), mais n'est pas compatible avec Matlab.

2 Où trouver des informations

L'aide de Matlab est la principale source d'informations (en anglais). On peut y accéder en tapant la commande `helpdesk`. On peut également obtenir une aide en ligne pour une fonction particulière. Par exemple pour obtenir l'aide sur la fonction exponentielle, il suffit d'entrer la commande `help exp`. On peut également utiliser la commande `lookfor` avec un mot clé pour rechercher dans l'aide. (Avec Octave, les commandes sont `help` et `help -i`)

Il existe également un grand nombre de ressources sur internet. Tout d'abord sur le site de *The Mathworks*, `http://www.mathworks.fr`, l'éditeur de Matlab . On y trouve l'aide complète et un grand nombre de liens, d'exemples de programmes... Le guide Getting Started est une bonne introduction. Il est accessible à partir de la page `http://www.mathworks.fr/access/helpdesk/help/techdoc/matlab.html`. Le guide est également téléchargeable en pdf.

On pourra également consulter les sites suivants pour des introductions à Matlab

- `http://maths.insa-lyon.fr/~balac/matlab/matlab.html`
- `http://esm2.imt-mrs.fr/~boiron/home/matlab/matlab.html`
- `http://docs.ufrmd.dauphine.fr/matlab/`

Évidemment un nombre encore plus grand de ressources est disponible en anglais. On trouve des liens à partir du site de The Mathworks ou simplement grâce à Google. Un grand nombre de liens est accessible à partir de ce site `APartialListofOn-LineMatlabTutorialsandMatlabBooks`.

Des ressources à peu près équivalentes (mais plus réduites) existent pour Scilab et Octave. Elles sont accessibles à partir des sites de ces logiciels.

3 Opérations simples et graphiques

3.1 Opérations élémentaires

Matlab permet de faire des opérations simples. Par exemple

```
>> 2+2
ans =
     4
```

On peut utiliser les opérateurs `+ - * ? ^` . Les parenthèses s'utilisent de manière usuelle. La racine carrée sera calculée en utilisant

```
>> 2^0.5
ans =
    1.4142
```

On peut également utiliser les nombres complexes. Par exemple

```
>> (1+0.5*i)^2
ans =
   0.7500 + 1.0000i
```

Les nombres complexes sont automatiquement générés par une commande comme

```
>> (-1.5)^0.5
ans =
   0.0000 + 1.2247i
```

On peut définir et utiliser des variables

```
>> x=4;
>> x^2
ans =
    16
```

Le point virgule à la fin d'une commande permet de ne pas afficher le résultat : la commande sans point virgule `x=4` affiche le résultat alors que `x=4;` n'affiche pas le résultat. On peut obtenir de l'information sur les variables à partir des commandes `who` and `whos`. La fenêtre "workspace" permet également d'accéder à la liste des variables. A noter que le nom des variables est sensible à la casse (majuscule/minuscule). La commande `clear` sert à effacer des variables, pour libérer de la mémoire.

Les fonctions mathématiques usuelles `exp, log, sin, ...` sont également définies. Pour obtenir la liste on peut utiliser la commande `help elfun` où `elfun` fait référence aux fonctions élémentaires (elementary math functions). Cette méthode pour demander de l'aide est utilisable avec d'autres thèmes (taper `help` pour avoir la liste des thèmes).

3.2 Vecteurs et matrices

3.2.1 Comment saisir et modifier un vecteur ou une matrice

On peut créer un vecteur en ligne en utilisant les crochets

```
>> v = [1 3.2 4 6 98 10/4]
v =
    1.0000    3.2000    4.0000    6.0000   98.0000    2.5000
```

Notons que le vecteur tel que défini précédemment est en ligne. Il s'agit en fait d'un tableau 1*7 – 1 ligne (row) et 7 colonnes (column). A la place d'espace entre les nombres, on peut utiliser des virgules. On obtient alors le même vecteur. Si on utilise des points virgules, on obtient un vecteur en colonne

```
>> v = [1; 3.2; 4; 6; 98; 10/4]
v =
    1.0000
    3.2000
    4.0000
    6.0000
   98.0000
   2.5000
```

On obtient un tableau 7*1.

Un autre moyen très courant de créer un vecteur est d'utiliser la syntaxe *début :incrément :fin* comme dans l'exemple

```
>> x = 1:10
x =
     1     2     3     4     5     6     7     8     9    10
>> x = 1:2:20
x =
     1     3     5     7     9    11    13    15    17    19
```

Dans la deuxième syntaxe, 2 est le pas entre les éléments consécutifs (qui est de 1 par défaut). La fonction `linspace` permet également de créer un vecteur dont les éléments sont régulièrement espacés.

```
>> y = linspace(0,2*pi,10)
y =
  Columns 1 through 6
         0    0.6981    1.3963    2.0944    2.7925    3.4907
  Columns 7 through 10
    4.1888    4.8869    5.5851    6.2832
```

y est un vecteur formé de dix éléments régulièrement espacé allant de 0 à 2 π inclus.

Une matrice peut être définie de la manière suivante

```
>> m = [1 3.2 4; 6 98 10; 23 42.3 5]
m =
    1.0000    3.2000    4.0000
    6.0000   98.0000   10.0000
   23.0000   42.3000    5.0000
```

Il existe également des fonctions permettant de créer des matrices et vecteurs. Par exemple,

```
m = ones(3,4)
m =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

crée une matrice 3*4 dont tous les éléments sont 1. De même, `zeros(m,n)` crée une matrice m*n dont tous les éléments sont zeros. On pourra également utiliser la fonction `diag` pour créer une matrice diagonale (regarder l'aide pour plus d'informations). La fonction `eye` permet de créer une matrice identité. Une liste plus complète des moyens de créer une matrice est accessible par `help elmat`.

On peut également créer une matrice par blocs

```
>> m = [ones(3,2), eye(3)]
m =
     1     1     1     0     0
     1     1     0     1     0
     1     1     0     0     1
```

3.2.2 Modifier un vecteur ou une matrice

On peut accéder à un élément d'un vecteur en mettant l'indice entre parenthèse. Par exemple +v(3)+ retourne le troisième élément de v.

Par la suite, on utilisera la matrice 6*6 m définie de la manière suivante

```
>> m=magic(6)
m =
    35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
```

On peut accéder aux éléments d'une matrice en utilisant des parenthèses

```
>> m(2,3)
ans =
     7
```

Ici, on a affiché l'élément de la deuxième ligne et de la troisième colonne. On peut extraire une sous matrice

```
>> subm = m(2:4,3:5)
subm =
     7    21    23
     2    22    27
    33    17    10
```

De même, on peut accéder à une colonne entière

```
>> v = m(:,3)
v =
     6
     7
     2
    33
    34
    29
```

v est alors le vecteur colonne formé des éléments de toutes les lignes (c'est le role de / :|) de la troisième colonne.

On peut également utiliser ce type de syntaxe pour supprimer des parties d'une matrice. Par exemple

```
>> m0=m;
>> m0(:,3)=[]
```

```
m0 =
    35     1    26    19    24
     3    32    21    23    25
    31     9    22    27    20
     8    28    17    10    15
    30     5    12    14    16
     4    36    13    18    11
```

supprime la 3e colonne de *m0*.

3.2.3 Opérations élémentaires sur les matrices et vecteurs

Les opérations élémentaires (`+, -, *`) s'utilisent sur les matrices comme sur les nombres. Il faut bien sûr que les tailles des matrices et vecteurs soient compatibles. L'apostrophe `'` permet d'obtenir le vecteur ou la matrice transposé. Ainsi le produit scalaire de deux vecteurs en ligne *v1* et *v2* s'obtient à partir de

```
>> v1=1:3;
>> v2=2:4;
>> v1*v2'
ans =
    20
```

On peut également effectuer des opérations élément par élément. Par exemple la multiplication de deux vecteurs élément par élément s'obtient par l'opérateur `.*`. Cela fonctionne également pour la division élément par élément `./` et l'élévation à la puissance élément par élément `.^`

```
>> v1 = 1:8;
>> v2=10.^(-1:6)
v2 =
   1.0e+06 *
  Columns 1 through 6
    0.0000    0.0000    0.0000    0.0001    0.0010    0.0100
  Columns 7 through 8
    0.1000    1.0000
>> v = v1./v2
v =
  Columns 1 through 6
   10.0000    2.0000    0.3000    0.0400    0.0050    0.0006
  Columns 7 through 8
    0.0001    0.0000
```

On notera que l'affichage de Matlab peut être trompeur : les éléments 7 et 8 du vecteur valent respectivement 7×10^{-5} et 8×10^{-6}.

Les fonctions mathématiques usuelles peuvent être appliquées à un vecteur ou une matrice. Dans ce cas la fonction est appliquée à chaque élément.

3.3 Graphiques simples

Matlab peut effectuer des représentations graphiques de vecteurs. Ainsi pour représenter la fonction sinus entre -2π et 2π, il faut calculer la valeur des points $y = f(x)$ en différents points on utilisera par exemple la suite d'instruction suivante

```
>> x = linspace(-2*pi, 2*pi, 400);
>> s = sin(x);
>> plot(x,s)
```

Les échelles sont calculées automatiquement. On pourra alors ajouter un titre et les axes

```
>> title('Fonction sinus')
>> xlabel('x')
>> ylabel('sin(x)')
```

La notation entre apostrophes est utilisée pour les chaînes de caractères.

On peut zoomer directement dans la figure (à l'aide de la souris) après avoir entré la commande `zoom on` (`zoom off` pour sortir de ce mode) ou afficher une grille avec la commande `grid on` (`grid off` pour ne pas afficher la grille).

Si on souhaite tracer une autre courbe, on peut définir un autre vecteur

```
>> c = cos(x);
>>  plot(x,c)
```

La première courbe a été effacée. Pour tracer deux courbes sur le même graphe, il faut utiliser la commande `hold on` avant le second tracé. Cette commande permet de passer d'un mode de remplacement des courbes à un mode d'ajout (`hold off` effectue l'opération inverse).

4 Introduction a la programmation

4.1 Environnement de travail et programme simple

Ecrire des programmes suppose la sauvegarde de fichier. L'étape préliminaire est de se placer dans un répertoire de travail, au choix de l'utilisateur, soit en utilisant la commande `cd` (comme sous unix) soit en utilisant l'interface graphique (qui dépend de la platforme et de la version de Matlab). Pour connaître le répertoire où l'on se trouve, on peut utiliser la commande `pwd`.

En utilisant la commande `new`, on ouvre une fenêtre d'éditeur[1]. On peut alors entrer une suite d'instruction. Par exemple, la suite d'instruction ci dessous permet de tracer les fonctions sinus, cosinus et tangente. Les lignes commençant par `%` sont des commentaires.

[1] Avec Octave, l'éditeur n'est pas integré et on travaille dans un éditeur externe

```
% trace les fonctions sin, cos, tan entre -2 pi et 2 pi
x = linspace(-2*pi, 2*pi, 1000);
s = sin(x);
c = cos(x);
t = tan(x);
figure
plot(x,s,x,c,x,t)
title('fonctions trigonometriques')
legend('sin','cos','tan')
axes([-2*pi, 2*pi, -3, 3])
```

Il faut alors sauver le programme à partir de l'éditeur par exemple sous le nom *plottrig.m*. Ensuite en se plaçant dans le bon répertoire (celui où *plottrig.m* a été enregistré), le programme peut s'exécuter à partir de la commande

```
>> plottrig
```

Les programmes de ce type s'appellent des scripts. Il est équivalent d'exécuter un script et de taper les commandes dans la fenêtre de commandes. Ainsi après l'exécution de *plottrig*, les variables x, s, c, t sont accessibles à l'utilisateur.

Notons qu'il est important de placer des commentaires dans les programmes. Les premières lignes en particulier doivent décrire ce que fait le programme.

4.2 Fonctions

Matlab utilise les fonctions. Une fonction dans Matlab peut être telle qu'on l'entend au sens mathématique. Par exemple, dans un nouveau fichier, on peut définir la fonction

```
function f = myfunc(x)
% f = myfunc(x)
% Calcule x - exp(x)
f = x - exp(x);
end
```

On l'enregistre sous le nom *myfunc.m* dans le répertoire de travail. Notons que cette fonction peut retourner un scalaire, un vecteur ou une matrice, selon la nature de x.

On peut alors effectuer une représentation graphique de la fonction

```
>> x = linspace{-5, 5, 200);
>> y = myfunc(x);
>> plot(x,y)
```

4.3 Fonctions et variables

De manière plus générale, une fonction est une suite d'instruction qu'on enregistre dans un fichier. A la différence d'un script, une fonction peut effectuer des opérations en interne qui ne modifient pas l'environnement de travail. Par exemple, on considère le script suivant qu'on enregistre sous le nom trigscript.m

```
% calcule y = sin(x^2) + cos(x^2)
x = x^2;
y = sin(x)+cos(x);
```

Lorsqu'on appelle ce script, la valeur de x est modifiée comme dans l'exemple ci dessous

```
>> x = linspace(-pi,pi,200);
>> trigscript
```

x est alors un vecteur qui varie entre 0 et π^2.

Si on utilise une fonction, le comportement est différent. Soit la fonction

```
function y = trigfunc(x)
% y = trigfunc(x)
% calcule y = sin(x^2) + cos(x^2)
x = x^2;
y = sin(x)+cos(x);
```

On calcule alors y de la manière suivante

```
>> x = linspace(-pi,pi,200);
>> y=trigfunc(x);
```

La valeur de x à l'extérieur de la fonction n'est pas modifiée et après exécution de ce script, x reste égal à *linspace(-pi,pi,200)*. Ainsi, une fonction ne peut agir que sur les variables de sortie, celles qui apparaissent à gauche du signe =.

Une fonction commence toujours par *function*. Parfois une fonction ne retourne pas de variable de sortie (mais c'est rare). Par exemple, on peut écrire la fonction qui trace un diagramme log-log

```
function myloglogplot(x,y)
% function myloglogplot(x,y)
% trace un diagramme log-log
% on suppose que les valeurs dans x et y sont positives
x = log(x);
y = log(y);
plot(x,y)
xtitle('log(x)')
ytitle('log(y)')
```

Cette fonction trace le diagramme log log, mais il serait préférable de tester que x et y sont positifs.

Une fonction peut avoit plusieurs variables d'entrées

```
function f = myfunc(x1,x2, mu)
% f = myfunc(x1,x2, mu)
% Calcule x1 * (mu - x2)
f = mu*x1 - x1.*x2;
end
```

A noter que pour que la fonction retourne un vecteur (si *x1* et *x2* sont des vecteurs), nous avons utilisé l'opérateur `.*` Si *x1* et *x2* sont de taille différentes, la fonction produira une erreur.

Une fonction peut également retourner plusieurs variables

```
function [mi,ma,avg] = vectstat(x)
% [mi,ma,avg] = vectstat(x).
% Calcule le min, max et la moyenne d'un vecteur
l = length(x);
mi = min(x);
ma = max(x);
avg = sum(x) / l;
end
```

Après avoir sauvé cette fonction (sous le nom *vectstat.m*) on peut l'utiliser de la manière suivante

```
>> v = rand(1000,1);
>> [valmin,valmax,valmoy]=vectstat(v)
valmin =
   2.2817e-04
valmax =
    0.9995
valmoy =
    0.5021
```

La variable l est une variable interne à la fonction. Elle est effacée dès que l'exécution de la fonction prend fin et est donc inconnue de Matlab en dehors de la fonction.

Dans une fonction, comme pour la ligne de commande, les lignes d'instruction peuvent se terminer par un point virgule (affiche le résultat de l'instruction) ou pas (n'affiche rien). Si on a besoin d'entrer une instruction sur plus d'une ligne, il faut terminer les lignes par trois points `...` comme dans l'exemple suivant

```
dat = [0.012, 0.0022, 0,0051, 0.0014, ...
        0.017, 0.0034, 0.087];
```

4.4 Sauvegarder une session

La commande `save` permet de sauvegarder l'ensemble des variables. Elle s'utilise par exemple de la manière suivante

```
save env
```

Un fichier *env.mat* est alors créer dans le répertoire courant. Ce fichier est dans un format spécifique à Matlab est n'est donc pas lisible par un autre logiciel. On peut ensuite utiliser `load env.mat` pour charger les variables en mémoire.

5 Structure de programmations

5.1 Conditions

Des instructions peuvent être utilisées de manière conditionnelle. Dans la syntaxe,

```
if condition
        suite d'instructions 1
else
        suite d'instructions 2
end
```

si la `condition` est vraie alors la suite d'instructions 1 est exécutée, sinon la suite d'instructions 2 est exécutée. Il peut y avoir une ou plusieurs instructions dans les suites d'instructions .

Par exemple dans la fonction suivante, on distingue les cas $x > 0$ et $x < 0$

```
function f = expinv(x)
% f = expinv(x).
% Calcule f telle que
% f(x) = 0 si x<0 et f(x) = exp(-1/x) si x>0
if (x>0)
        f = exp(-1/x);
else
        f = 0;
end
```

On pourra alors vérifier que cette fonction n'accepte pas de prendre un vecteur en entrée. x doit être un nombre. Le problème est dans l'évaluation de la condition $x > 0$.

La syntaxe `elseif` est également définie

```
function f = expinvsym(x)
% f = expinvsym(x).
% Calcule f tel que
```

```
%         f(x) = -exp(1/(x+1)) si x < -1
%         f(x) = 0 si 1 <= x <= 1
%         f(x) = exp(-1/(x-1)) si x > 1
if (x < -1)
        f = -exp(1.0/(x+1));
elseif ( (x >= -1) && (x <= 1) )
        f = 0;
else
        f = exp(-1.0/(x-1));
end
```

On peut utiliser la condition *if* pour rendre les programmes plus robustes. Par exemple si l'on reprend l'exemple de la fonction *expinv* ci dessus, on peut tester que x est un nombre et non un vecteur ou une matrice.

```
function f = expinv(x)
% f = expinv(x).
% Calcule f telle que
% f(x) = 0 si x<0 et f(x) = exp(-1/x) si x>0

s = size(x);
if ( (s(1) == 1)&&(s(2)==1) )
        if (x>0)
                f = exp(-1/x);
        else
                f = 0;
        end
else
        r1='Erreur -- expinv --'
        r2='x n'est pas un nombre'
        f = 0;
end
```

La fonction *size* retourne un vecteur contenant le nombre de lignes et le nombre de colonnes et il faut alors tester que les deux dimensions sont égales à 1. Il est important de différentier la condition `a==b` et l'affectation `a=b`.

Lorsqu'on évalue une condition comme `a==b` le résultat est dit booléen : il est soit vrai soit faux. Pour Matlab si la condition est vraie elle vaut 1, si elle est fausse, elle vaut zéro :

```
>> a = 1;
>> b = 3;
>> x = (a==b)
x = 0
>> y = (a<b)
y = 1
```

les variables x et y sont dites variables booléennes. On peut les utiliser dans un tableau. Par exemple

```
>>a = rand(1,5)
a =
  0.74026  0.65677  0.13654  0.12424  0.43435
>> x = a>0.5
x =
  1  1  0  0  0
```

5.2 Boucles

Très souvent, il est utile de répéter des opérations un certain nombre de fois. On utilise alors une boucle *for*.

```
for k = kmin:kmax
        suite d'instructions
end
```

La variable k est un compteur. Elle prend successivement les valeurs $kmin$, $kmin+1$, ... jusqu'à $kmax$. Ainsi, la suite d'instructions est exécutée $kmax - kmin + 1$ fois. A l'intérieur de la suite d'instructions, on peut faire appel au compteur k. De manière plus générale, on peut utiliser la forme `for k =kvect`. Avec cette forme, k prendra successivement les valeurs $kvect(1)$, $kvect(2)$, ... Les instructions dans la boucle seront effectué n fois où n est la longueur du vecteur kvect.

Par exemple la fonction suivante retourne la somme de deux vecteurs

```
function c=addvect(a,b)
% c=addvect(a,b).
% Calcule la somme des vecteurs a and b supposés être en colonnes.
% Realise la meme operation que la commande a + b
n=length(a);
m=length(b);
if (n ~= m)
        r1 = 'Erreur -- addvect -- '
        r2 = 'La taille des vecteurs n'est pas compatible'
        return
end
c=zeros(size(a));
for k=1:n
        c(k) = a(k)+b(k);
end
```

L'opérateur `||` est l'opérateur "ou". Ainsi, si m est différent de n, la fonction affiche un message d'erreur et n'effectue pas le calcul. Dans la boucle *for*, l'indice est incrémenté automatiquement. Ainsi, les instructions dans la boucle sont effectuées

une première fois avec $k = 1$ et $c(1)$ est calculé, puis avec $k = 2$ et $c(2)$ est calculé et ainsi de suite jusqu'à $k = n$ où $c(n)$ est calculé. On voit qu'à l'intérieur de la boucle, on peut utiliser la variable k.

La structure `while` (tant que), permet de répéter une suite d'instruction jusqu'à ce qu'une condition soit fausse.

```
while condition
        suite d'instruction
end
```

La suite d'instruction est exécutée tant que la condition est vraie. Si la condition est tout le temps fausse, la suite d'instruction n'est jamais exécutée. Si la condition est tout le temps vraie, la suite d'instruction est exécutée indéfiniment. Ainsi dans la suite d'instruction, les variables doivent évoluer de manière à ce que la condition puisse changer et devenir fausse.

La fonction ci dessous retourne la partie entière du logarithme de base 2 d'un nombre.

```
function l=twolog(x)
% l=twolog(x).
% Calcule la partie entiere du logarithme de base 2 de x
% c'est a dire l'entier de type 2^k le plus proche de x.
l=0;
m=2;
while m<=n
        l=l+1;
        m=2*m;
end
```

Ici, la variable m est changée à l'intérieur de la boucle, et donc la condition peut devenir fausse ce qui permettra l'arrêt de l'exécution.

5.3 Récursivité

On peut également utiliser des fonctions qui s'appelent elles-même. On parle de récursivité. Par exemple, la fonction suivante permet d'élever un nombre à la puissance n (entière).

```
function p=pow(x,n)
% p=pow(x,n).
%Calcule
%         x^k ou k = floor(n) si n>=1
%         1 sinon
if (n>=1)
        p=x*pow(x,n-1);
else
```

```
        p=1;
end
```

A titre d'exercice, on pourra modifier cette fonction pour qu'elle calcule également lorsque l'exposant est négatif. On pourra également lui faire retourner un avertissement si n est non entier.

5.4 Quelques conseils

Lorsqu'on écrit des programmes, on commet toujours des erreurs. Une partie importante du temps de développement est donc du temps de "déboguage". Lorsqu'une fonction retourne une erreur, Matlab sort de cette fonction et donc l'état des variables est perdu. Il est possible de figer l'état de l'exécution d'un programme en utilisant préalablement la commande `keyboard` dans un programme. Il est également possible d'utiliser la commande `dbstop if error` qui fige l'état des variables à chaque fois qu'une erreur se produit. Pour quitter ce mode utiliser la commande `dbclear if error`.

6 Pour en savoir plus

Le but de cette brève introduction est d'aider un utilisateur pendant les premières heures sur Matlab . Très vite le contenu de cette introduction se trouvera dépassé. L'aide de Matlab doit alors prendre le relais. Un grand nombre de fonctions seront vues ultérieurement dans le cadre du cours.