1 Introduction à MATLAB

1.1 Prise en main de Matlab

MATLAB est à la fois un langage de programmation et un environnement de développement développé et commercialisé par la société américaine MathWorks. MATLAB est utilisé dans les domaines de l'éducation, de la recherche et de l'industrie pour le calcul numérique mais aussi dans les phases de développement de projets.

Histoire

Le nom **MATLAB** est la contraction du terme anglais *matrix laboratory*. Le langage MATLAB a été conçu par Cleve Moler à la fin des années 1970 à partir des bibliothèques Fortran, LINPACK et EISPACK. Alors professeur de mathématiques à l'Université du Nouveau-Mexique, il souhaitait permettre à ses étudiants de pouvoir utiliser ces deux bibliothèques sans connaitre le Fortran.
Cleve Moler l'utilisa ensuite pour des cours donnés à l'université Stanford où il reçut un accueil mitigé de la part des étudiants en mathématiques (habitués au Fortran). Par contre, les étudiants en technologie, en particulier en traitement du signal, furent beaucoup plus intéressés. Un ingénieur, Jack Little en comprend rapidement les capacités et entreprend avec un collègue, Steve Bangert, de le recoder en langage C. Jack Little, Cleve Moler et Steve Bangert créèrent la société The MathWorks en 1984 afin de commercialiser la version 1.0 de MATLAB.
MATLAB a ensuite évolué, en se dotant de nombreuses boites à outils (Toolbox) et en incluant les possibilités données par d'autres langages de programmation comme C++ ou Java.

Outils et modules associés

MATLAB est complété par de multiples boites à outils. Parmi les plus importantes, on trouve :

- Communications Toolbox
- Control System Toolbox
- Excel Link
- MATLAB Compiler
- Neural Network Toolbox
- Optimization Toolbox
- Parallel Computing toolbox
- Real-Time Workshop®, renommé commercialement SimulinkCoder
- Robust Control Toolbox
- SimMechanics
- SimPowerSystems
- Simulink
- Statistics Toolbox
- System Identification Toolbox
- Virtual Reality Toolbox

1-Variables et constantes spéciales

Ans	réponse la plus récente

Pi	nombre pi
Inf	plus l'infini
-inf	moins l'infini
NaN	(Not-a-Number)

2-Opérateurs mathématiques

+	addition
-	soustraction
*	multiplication
/	division
^	puissance

>> (2 + 5.2)*10 / (5^3)
ans =
0.5760
>> -2.52e3
ans =
-2520
>> 2*pi
ans =
6.2832
format long e affiche 16 chiffres :
>> format long e
>> 2*pi
ans =
6.283185307179586e+000
format short (format par défaut) affiche 5 chiffres :
>> format short
>> 2*pi
ans =
6.2832
>> 1 / 0
Warning: Divide by zero
ans =
Inf
>> -1 / 0
Warning: Divide by zero
ans =
-Inf
>> 0 / 0
Warning: Divide by zero
ans =
NaN

3- Fonctions mathématiques

sin(X)	sinus
asin(X)	sinus inverse
cos(X)	cosinus
acos(X)	cosinus inverse
tan(X)	tangente
atan(X)	tangente inverse

avec X : argument en **radians.**

exp(X)	exponentielle
log(X)	logarithme naturel (base e)
log10(X)	logarithme décimal (base 10)
sqrt(X)	racine carrée
abs(X)	valeur absolue

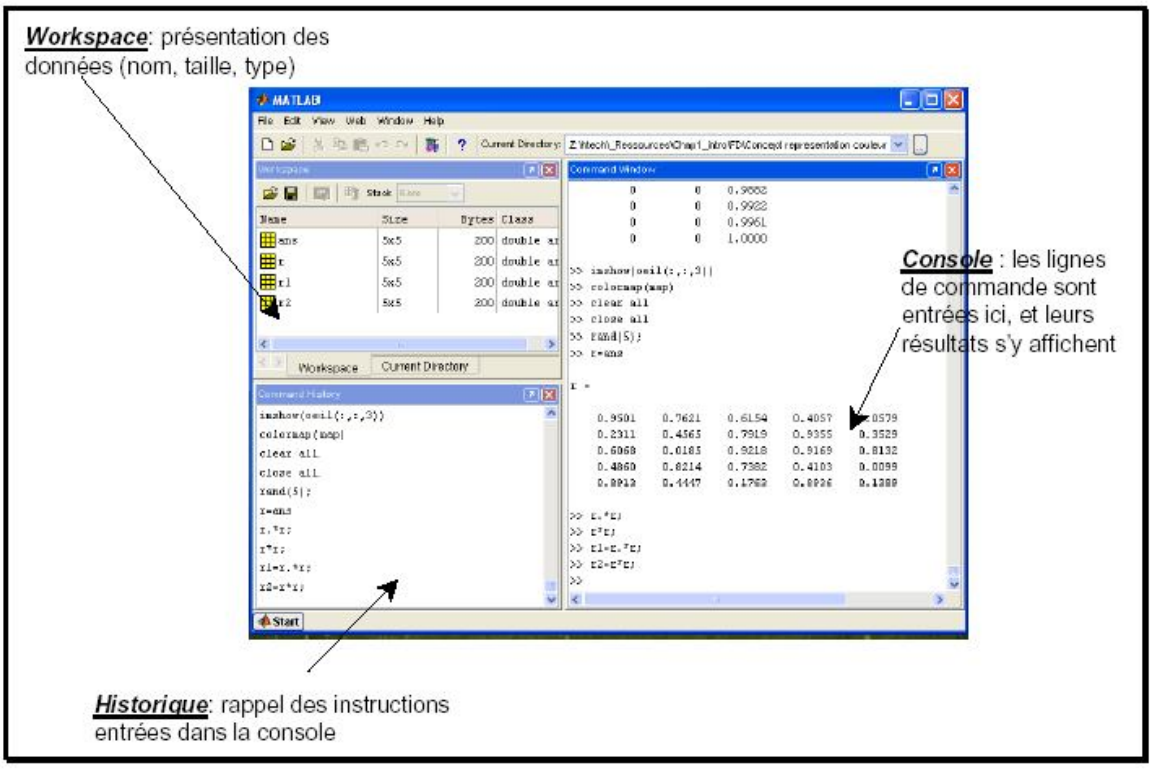
>> sin(2)
ans =
0.9093

sinus (45°) :
>> sin(45*pi/180)
ans =
0.7071
>> 1 + exp(2.5)
ans =
13.1825

4- Utilisation de variables
>> 5*3
ans =
15
>> ans+4
ans =
19
>> a= 2 + log(15)

a =
4.7081
>> b = - 45
b =
-45
>> a * b
ans =
-211.8623
>> c = a - sqrt(abs(b))
c =
-2.0002

Interface de Matlab



Apres exécution du raccourci de Matlab sur le bureau. Vous allez voir apparaître une ou plusieurs nouvelles fenêtres sur votre écran, dont la *fenêtre de commandes*. Les principales caractéristiques de cette fenêtre sont :

Les caractères >> en début de ligne constituent le *prompt* de Matlab. C'est après eux que vous pouvez taper des commandes qui seront exécutées par le logiciel après avoir tapé sur la touche *entrée*.

Le résultat de l'exécution s'inscrit alors dans la fenêtre ou est représenté graphiquement dans une nouvelle fenêtre spécifique (avec possibilité de zoom, d'impression, etc...). Pour rentrer une suite complexe d'instructions (on parle aussi d'un script), on les tape au préalable dans un fichier en utilisant l'éditeur intégré. Une fois le script enregistré, on peut l'exécuter en tapant son nom dans la fenêtre Matlab.

L'historique des instructions entrées depuis le début de la session sur la ligne de commande est accessible par pressions successives de la touche ↑.
Enfin, pour effacer les données en mémoire (par exemple avant d'exécuter un nouveau calcul), il suffit d'utiliser la commande **clear**.

Historique historique des commandes que l'utilisateur a exécutées.

Commencez tout d'abord par vous créer un répertoire de travail via la fenêtre **Current Directory.** Qui permet de naviguer et de visualiser le contenu du répertoire courant de l'utilisateur.

Workspace permet de visualiser, notamment, les variables, leur type et leur état…

Remarque : Matlab distingue toujours les majuscules et les minuscules, que ce soit pour les noms de variables, de fichiers, ou de fonctions. Faites attention.

2 Programmation

2.1 Aspects élémentaires

2.1 Aides

help -> utilisée seule donne une liste et un bref descriptif des sujets contenant une aide. C'est LA commande essentielle dans l'apprentissage de Matlab

La commande ***help nom_fonction*** donne un descriptif de la fonction sur les arguments nécessaires en entrée ainsi que les résultats donnés.

helpdesk : documentation en hypertexte (requiert Netscape ou autre)

helpwin : aide en ligne dans une fenêtre séparée

lookfor : recherche d'un mot clé (lent)

which : localise fonctions et fichiers

what : liste des fichiers matlab dans le répertoire courant

exist : check si une fonction ou une variable existe dans le workspace

who, whos : liste des variables dans le workspace

La commande ***quit*** pour quitter Matlab.

La commande ***clear*** pour effacer toutes les variables existantes.

save sauvegarde une ou plusieurs variables de la session dans un fichier du répertoire courant. **save filename** or **save filename varname**

load retrouve les variables sauvegardées précédemment

load filename or **load filename varname**

clc Efface l'écran (fenêtre) de MATLAB

2.2 Variables scalaires, workspace, opérations élémentaires

>> var=2

var = 2

>> autre=3;

>> who fournit la liste des fonctions et variables définies dans le workspace

Your variables are:

autre var

>>whos donne plus d'informations que who

Sous Windows, vous avez accès au "Workspace browser" dans la barre d'outils.

>> clear autre

>> who

Your variables are:

var

>>clear efface toutes les variables du workspace

Opérations élémentaires:

+ - * / or \ ^

Exemple :

1)

>> 4/2

ans =

2

>> 2\4

ans =

2

2)

Soit à calculer le volume suivant : $V = \frac{4}{3}\pi R^3$ où R=4cm

Pour calculer V, on exécute les commandes suivantes :

>>R=4
R =
4

*>>V=4/3*pi*R^3*
V =
268.0826

3)

x=2, $P(x) = \frac{4x^2 - 2x + 3}{x^3 + 1}$

>>x=2
x =
2

*>>P=(4*x^2-2*x+3)/(x^3+1)*
P =
1.6667

2.3 Commentaires, ponctuation

% Permet de rajouter des commentaires afin d'expliquer les parties de notre programme.

>> s=2+3 % je fais une somme

s = 5

>> cout_moyen = cout **...** % commande sur deux lignes on utilise les 3 points

/ nombre;

2.4 Opérateurs logiques et de relation

< plus petit

> plus grand

<= plus petit ou égal

>= plus grand ou égal

== égal

~= pas égal

& et

| ou

~ not

xor(x,y) ou exclusif

any(x) retourne 1 si un des éléments de x est non nul

all(x) retourne 1 si tous les éléments de x sont nuls

isequal(A,B), ischar etc...

Exemple

1)a=b/c

Si c~=0 alors a=b/c

>> *if c~=0 , a=b/c, end*

Si a>3 et c<0, alors b=15

>>*if a>3 & c<0, b=15, end*

2)>> a='cours'

a =

cours

>> ischar(a)

ans =

1

>> b=6

b =

6

>> ischar(b)

ans =

0

3)>> a=4

a =

4

>> b=5

b =

5

>> isequal(a,b)

```
ans =
    0
>> b=4
b =
    4
>> isequal(a,b)
ans =
    1
```

2.7 Nombres complexes

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

Forme cartésienne: **0.5 + i*2.7** **-1.2 + j*0.789** **2.5 + 9.7i**

Forme polaire: **1.25*exp(j*0.246)**

Utilisation de nombres ou de variables complexes
MATLAB, peut utiliser et gérer les variables ou les nombres complexes. La plupart des fonctions implicites définies pour les réels existent pour les complexes, y compris la puissance.

Exemple

```
>>z=3.5-1.25i ;

>>log(z)
ans =
1.3128 - 0.3430i

>>cos(2-i)
ans =
-0.6421 + 1.0686i
```

L'imaginaire pur est noté par *i* ou *j*. Ainsi, i^2 ou j^2 donne :

```
>>i^2
ans =
-1.0000 + 0.0000i
```

Soit *z* un nombre complexe. Son conjugué est donné par la fonction '*conj(z)*'.

>>z=3.5-1.25i
z =
3.5000 - 1.2500i

>>conj(z)
ans =
3.5000 + 1.2500i

Opérations sur les nombres complexes
1. Addition de nombres complexes :
>>z1
z1 =
3.5000 - 1.2500i

>>z2
z2 =
1.3140 - 0.0948i

>>z1+z2
ans =
4.8140 - 1.3448i

2. Soustraction de nombres complexes :
>>z1-z2
ans =
2.186 - 1.1552i

3. Multiplication de nombres complexes :
*>>z1*z2*
ans =
4.4805 - 1.9743i

4. Division de nombres complexes :
>>z1/z2
ans =
2.7181 - 0.7551i

5. Opération de puissance :
>>z1^z2
ans =
4.5587 - 2.9557i

Produit factoriel
La fonction '***gamma***' donne le produit factoriel d'un nombre *n*.
Exemple : pour *n=6*, on a : *6!=6*5*4*3*2*1=720*

>>factorielle=gamma(6+1)
factorielle =
720

La fonction '***gamma***' peut calculer la factorielle des nombres entiers et même des nombres réels.

Aspect des nombres dans MATLAB
Dans MATLAB, il n'existe aucune distinction entre les nombres entiers, les nombres réels ou les nombres complexes. Les nombres sont donc traités automatiquement. La précision des calculs est affectée par le type des variables traitées.

Les paramètres essentiels qui déterminent cette précision dans un langage de programmation sont :

- le plus petit nombre positif : realmin
- le plus grand nombre positif : realmax
- l'erreur de la machine (epsilon) : eps
- *précision relative des nombres réels (distance entre 1.0 et le nombre réel flottant le plus proche)*

Le tableau ci-dessous donne une comparaison des différents paramètres obtenus par le langage MATLAB :

Précision du langage	*MATLAB*
realmin	2.2251e-308
realmax	1.7977e+308
Eps	2.2204e-016

2.8 Syntaxe du langage
Si une instruction MATLAB est suivie d'un point virgule, le résultat de cette instruction n'est pas affiché. Pour ré-afficher un résultat contenu dans une variable, il suffit de taper le nom de la variable. Le résultat de la dernière instruction exécutée peut être rappelé par la commande **ans**:

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

```
ans =
   91   67   67
   67   91   67
   67   67   91
>>
```

Plusieurs instructions MATLAB peuvent figurer sur une même ligne. Il faut alors les séparer par une virgule ou par un point virgule. D'autre part, si une commande est trop longue pour tenir sur une ligne, il est possible de poursuivre sur la ligne suivante en terminant la ligne par 3 points (**...**).

```
>> B = [ 1 3; 4 2 ]; B*B
ans =
   13    9
   12   16
>> x = 1 + 2 + 3 + 4 + 5 + 6 ...
+7 + 8 + 9 + 10
x =
   55
>>
```

Si la syntaxe de l'instruction soumise est erronée ou si vous demandez à MATLAB d'exécuter une instruction illégale (qui n'a pas de sens mathématique par exemple), vous obtiendrez un message d'erreur. Ce message vous indique les sources d'erreurs possibles et doit-vous permettre de corriger rapidement votre erreur.

```
>> A + B
??? Error using ==> +
Matrix dimensions must agree.
>> C = [ 1 2 3; 4 5]
??? Number of elements in each row must be the same.
>> whose
??? Undefined function or variable 'whose'.
>>
```

Dans la première instruction, on tente d'effectuer la somme de 2 matrices aux dimensions incompatibles. Dans le second exemple on tente de définir une matrice dont le nombre d'éléments dans chaque ligne diffère. Enfin la troisième instruction est inconnue de MATLAB: il ne s'agit n'y d'une fonction ni d'une variable incorporée ou utilisateur.

2.9 M-files ou scripts

Jusqu'à présent, nous avons travaillé en ligne sur la fenêtre Matlab. Cette façon de travailler n'est pas optimale. Si on commet une erreur, on est obligé de retaper toute la ligne ! Pour palier à cet inconvénient, on peut utiliser un fichier d'instructions. Matlab peut en effet exécuter une suite d'instructions stockées dans un fichier. On appellera ces fichiers des M-fichiers, et leur nom doit être suivi du suffixe .m.

Un script (ou M-file) est un fichier (message.m par exemple) contenant des instructions Matlab.

Voici un exemple de script:

```
% message.m affiche un message
% ce script affiche le message que s'il fait beau
beau_temps=1;
if beau_temps~=0
disp('Hello, il fait beau')
end
return % (pas nécessaire à la fin d'un M-file)
```

Matlab vous offre un éditeur pour écrire et mettre au point vos M-files:

>> edit % lance l'éditeur de MatLab. Voir aussi la barre d'outils

Tout autre éditeur de texte convient aussi.

Les M-files sont exécutés séquentiellement dans le "workspace", c'est à dire qu'ils peuvent accéder aux variables qui s'y trouvent déjà, les modifier, en créer d'autres etc.

On exécute un M-file en utilisant le nom du script comme commande :

```
>> message
Hello, il fait beau
```

Exemple 2
Etant donné le script suivant :
y=x^2
Nommé square.m
Si l'on veut faire appel à ce script il faut donner une valeur à la variable x au préalable :

```
>> x=4 ;
Square
Y =
    4
```

3 Vecteurs
On définit un vecteur ligne en donnant la liste de ses éléments entre crochets [ ]. Les éléments sont séparés au choix par des espaces ou par des virgules. On définit un vecteur colonne en donnant la liste de ses éléments séparés au choix par des points virgules (;) ou par des retours chariots (touche Entrée/Enter). On peut transformer un vecteur ligne x en un vecteur colonne et réciproquement en tapant x' (**'** est le symbole de transposition). Il est inutile de définir la longueur d'un vecteur au préalable. Cette longueur sera établie automatiquement à partir de l'expression mathématique définissant le vecteur ou à partir des données. On peut obtenir la longueur d'un vecteur donné grâce à la commande **length(vecteur)**. Un vecteur peut également être défini << par blocs >> selon la même syntaxe. Si par exemple x_1, x_2 et x_3 sont trois vecteurs (on note x1, x2 et x3 les variables

MATLAB correspondantes), on définit le vecteur bloc (x_1 | x_2 | x_3) par l'instruction X = [x1 x2 x3].

3.1 Création de vecteurs

Par défaut, le vecteur est une ligne à plusieurs colonnes

a) vecteur par énumération des composantes

```
>> x1 = [1 2 3], x2 = [4,5,6,7], x3 = [8; 9; 10]
x1 =
    1    2    3
x2 =
    4    5    6    7
x3 =
    8
    9
   10
```

Pour définir le vecteur colonne x = $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$

```
>>x=[1;-1]

x =
    1
   -1
>> length(x2), length(x3)
ans =
    4
ans =
    3
>> whos
  Name      Size         Bytes  Class

  x1        1x3             24  double array
  x2        1x4             32  double array
  x3        3x1             24  double array
```

b) vecteur ligne par description

```
>> x = [0 : pi/10 : pi] % [valeur-initiale : incrément : valeur-finale]
x =
0   0.3142   0.6283   0.9425   1.2566   1.5708   1.8850   2.1991   2.5133   2.8274   3.1416
```

c) vecteur colonne

L'opérateur apostrophe utilisé pour créer un vecteur colonne est en fait l'opérateur transposition

```
>> xcol = x'

xcol =

   0
   0.3142
   0.6283
   0.9425
   1.2566
   1.5708
   1.8850
   2.1991
   2.5133
   2.8274
   3.1416
```

d) génération de vecteurs métriques

La commande **linspace** permet de définir un vecteur x de longueur *N* dont les composantes forment une suite arithmétique de premier terme *a* et de dernier terme *b* (donc de raison (*b-a*)/*N-1*). Les composantes du vecteur sont donc *lin*éairement e*spac*ées. La syntaxe est

x = linspace(a,b,N).

```
>> x = linspace(0, pi, 11) % génère le même x que ci-dessus (11 valeurs. réparties de 0 à pi)
x =
0   0.3142   0.6283   0.9425   1.2566   1.5708   1.8850   2.1991   2.5133   2.8274   3.1416
```

3.2 Adressages et indexages

```
>> x(3) % 3ème élément du vecteur x
ans =
   0.6283
>> x(2 : 4) % un bloc de composantes
ans =
   0.3142    0.6283    0.9425
>> x([8 3 9 1]) % une sélection de composantes (on les désigne avec un autre vecteur!)
ans =
2.1991    0.6283    2.5133         0
```

3.3 Combinaison de vecteurs

a) Accolage de deux vecteurs

```
>> a = [1:3]
a =
    1     2     3
```

```
>> b=[10:10:30]
b =
   10   20   30
>> c = [a b]
c =
    1    2    3   10   20   30
```
On peut faire plus compliqué
```
>> d=[a(2:-1:1) b] % on accole b avec une portion de a dans l'ordre renversé
d =
    2    1   10   20   30
```

3.4 Vecteurs spéciaux

Les commandes ones, zeros et rand permettent de définir des vecteurs dont les éléments ont respectivement pour valeurs 0, 1 et des nombres générés de manière aléatoire.

ones(1,n) : vecteur ligne de longueur n dont tous les éléments valent 1

ones(m,1) : vecteur colonne de longueur m dont tous les éléments valent 1

zeros(1,n) : vecteur ligne de longueur n dont tous les éléments valent 0

zeros(m,1) : vecteur colonne de longueur m dont tous les éléments valent 0

rand(1,n) : vecteur ligne de longueur n dont les éléments sont générés de manière aléatoire entre 0 et 1

rand(m,1) : vecteur colonne de longueur m dont les éléments sont générés de manière aléatoire entre 0 et 1

sum(x) : somme des éléments du vecteur x

prod(x) : produit des éléments du vecteur x

max(x) : plus grand élément du vecteur x

min(x) : plus petit élément du vecteur x

mean(x) : moyenne des éléments du vecteur x

sort(x) : ordonne les éléments du vecteur x par ordre croissant

fliplr(x) : échange la position des éléments du vecteur x

4. Matrices

4.1 Création de matrices

Une matrice est un ensemble de lignes comportant toutes le même nombre de colonnes

a) Par énumération des éléments

```
>> m1 = [ 1 2 3 ; 4 5 6 ; 7 8 9]  % on sépare les lignes par des point-virgules
m1 =
    1    2    3
    4    5    6
    7    8    9
```
On peut étendre aux matrices les autres manières de définir des vecteurs.
Par exemple:

```
>> m2 = [1:1:3 ; 11:1:13]
m2 =
    1     2     3
   11    12    13
```

4.2 Transposition

L'opérateur apostrophe utilisé pour créer un vecteur colonne est en fait l'opérateur transposition :

```
>> m2'
ans =
    1    11
    2    12
    3    13
```

4.3 Opérations scalaires-matrices

Une telle opération agit sur chaque élément de la matrice :

```
>> m2' * 10        % de même:  4*m2     m2-10     m2/4
ans =
   10   110
   20   120
   30   130
```

Une exception:

```
>> m2^2
??? Error using ==> ^
Matrix must be square.
```

Dans ce cas, Matlab veut calculer le produit matriciel m2 * m2

La solution est l'usage du point qui force l'opération sur chaque élément:

```
>> m2  .^ 2
ans =
    1     4     9
  121   144   169
```

4.4 Opérations entre matrices

a) Multiplications

```
>> m1 % rappelons la définition de m1
m1 =
    1     2     3
    4     5     6
    7     8     9
>> m2 % rappelons la définition de m2
m2 =
    1     2     3
   11    12    13
m3 =
   1.0000    2.0000    3.0000
   1.0000    3.1623   10.0000
```

```
>> m1 * m2' % le produit matriciel n'est possible que lorsque les dimensions sont cohérentes
ans =
   14   74
   32  182
   50  290
>> m1 * m2
??? Error using ==> *
Inner matrix dimensions must agree.
```
Multiplication élément par élément:
```
>> m2 .*m3 % (m2 et m3 ont les mêmes dimensions)
ans =
   1.0000    4.0000    9.0000
  11.0000   37.9473  130.0000
```

b) Divisions
```
>> m2/m3  % division matricielle à droite
ans =
   1.0000   -0.0000
   9.5406   -1.5960
>> m2\m3 % division matricielle à gauche
ans =
  -0.5000   -0.8257   -0.4500
        0         0         0
   0.5000    0.9419    1.1500
```
Division élément par élément:
```
>> m2./m3 % chaque élément de m2 est divisé par l'élément équivalent de m3
ans =
   1.0000    1.0000    1.0000
  11.0000    3.7947    1.3000
>> m2.\m3 % chaque élément de m3 est divisé par l'élément équivalent m2
ans =
   1.0000    1.0000    1.0000
   0.0909    0.2635    0.7692

>> m3./m2 % chaque élément de m3 est divisé par l'élément équivalent m2
ans =
   1.0000    1.0000    1.0000
   0.0909    0.2635    0.7692
```

4.5 Matrices particulières
```
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
```

```
>> zeros(2,5)
ans =
   0   0   0   0   0
   0   0   0   0   0
>> eye(4)
ans =
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
>> diag([1 : 4])
ans =
   1   0   0   0
   0   2   0   0
   0   0   3   0
   0   0   0   4
>> rand(1,7) % nombres aléatoires entre 0 et 1
ans =
  0.9355   0.9169   0.4103   0.8936   0.0579   0.3529   0.8132
```

4.6 Caractéristiques des matrices

```
>> size(m3) % dimensions
ans =
   2   3
>> length(m3) % equivalent à max(size(m3)) : dimension maximum
ans =
   3
```

4.7 Manipulations de matrices et sous-matrices

a) Exercices de manipulations avec les notions vues jusqu'ici

- Définissez A une matrice 3x3
- Mettez à zéro l'élément (3,3)
- Changez la valeur de l'élément dans la 2ème ligne, 6ème colonne, que se passe-t-il?
- Mettez tous les éléments de la 4ème colonne à 4
- Créez B en prenant les lignes de A en sens inverse
- Créer C en accolant toutes les lignes de la première et troisième colonne de B à la droite de A
- Créer D sous-matrice de A faite des deux premières lignes et les deux dernières colonnes de A. Trouvez aussi une manière de faire qui ne dépende pas de la taille de A.

Note: chacun de ces exercices se fait en une seule instruction, sans boucles ittératives.

b) Fonctions de manipulation des matrices:

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

```
>> flipud(A)    % flip up-down
ans =
    7    8    9
    4    5    6
    1    2    3
>> fliplr(A)    % flip left-right
ans =
    3    2    1
    6    5    4
    9    8    7
>> rot90(A,2)     %2 rotations de 90 degres (sens trigo)
ans =
    9    8    7
    6    5    4
    3    2    1

>> reshape(A,1,9)    % change la forme de la matrice
ans =
    1    4    7    2    5    8    3    6    9
>> diag(A)      % extrait la diagonale de A
ans =
    1
    5
    9
>> diag (ans)    % diag travaille dans les 2 sens !
ans =
    1    0    0
    0    5    0
    0    0    9
>> triu(A)    % extrait le triangle supérieur de A
ans =
    1    2    3
    0    5    6
    0    0    9
>>magic(N) %crée une matrice N*N d'éléments allant de 1 jusqu'à N^2
```

5 L'instruction conditionnée

On a parfois besoin d'exécuter une séquence d'instructions seulement dans le cas où une condition donnée est vérifiée au préalable. Différentes formes d'instruction conditionnée existent sous MATLAB.

5.1 L'instruction IF

a) L'instruction conditionnée la plus simple a la forme suivante:

Syntaxe :

If *expression logique*

séquence d'instructions

end

où

- *expression logique* est une expression dont le résultat peut être vrai ou faux;
- *séquence d'instructions* est le traitement à effectuer si *expression logique* est vraie.

Interprétation

la *séquence d'instructions* n'est exécutée que si le résultat de l'évaluation de l'*expression logique* est vraie (c'est-à-dire vaut 1). Dans le cas contraire on exécute l'instruction qui suit le mot clé end. Dans le cas où l'*expression logique* est vraie, après exécution de la *séquence d'instructions* on reprend le programme à l'instruction qui suit le mot clé end.

b) Il existe une séquence conditionnée sous forme d'**alternatives**:

Syntaxe

If *expression logique*

séquence d'instructions 1

else

séquence d'instructions 2

end

où

- *expression logique* est une expression dont le résultat peut être vrai ou faux;

- *séquence d'instructions 1* est la séquence d'instructions à exécuter dans le cas où *expression logique* est vraie et *séquence d'instructions 2* est la séquence d'instructions à exécuter dans le cas où *expression logique* est faux.

Interprétation
Si *expression logique* est vraie la *séquence d'instructions 1* est exécutée, sinon c'est la *séquence d'instructions 2* qui est exécutée. Le déroulement du programme reprend ensuite à la première instruction suivant le mot clé end.

Il est bien entendu possible d'imbriquer des séquences d'instructions conditionnées (au sens où la séquence d'instruction conditionnée contient des séquences d'instructions conditionnée). Pour une meilleure lisibilité, il est recommandé d'utiliser des indentations afin de mettre en évidence l'imbrication des séquences d'instructions conditionnées.

c) Il est possible d'effectuer un **choix en cascade**

Syntaxe

if*expression logique 1*

séquence d'instructions 1

elseif *expression logique 2*

séquence d'instructions 2

...

elseif *expression logique N*

séquence d'instructions N

else

séquence d'instructions par défaut

end

Interprétation
Si *expression logique 1* est vraie la *séquence d'instructions 1* est exécutée et le programme reprend ensuite à la première instruction suivant le mot clé end, sinon si *expression logique 2* est vraie la *séquence d'instructions 2* est exécutée et le programme reprend ensuite à la première instruction suivant le mot clé end, *etc*. Si aucune des expressions logiques 1 à N

n'est vraie alors *séquence d'instructions par défaut* est exécutée.

Remarque :
Attention à ne **pas** laisser d'espace entre else et if; le mot clé est elseif.

On utilise fréquemment un choix en cascade lors d'initialisation de données. Par exemple, on initialise une matrice A en fonction de la valeur d'une variable num (numéro d'exemple) de la manière suivante:

```
if num == 1
 A = ones(n);
elseif num == 2
 A = magic(n);
elseif num == 3 | num == 4
 A = rand(n);
else
error('numero d''exemple non prevu ...');
end
```

L'instruction switch

Une alternative à l'utilisation d'une séquence d'instructions conditionnées pour effectuer un choix en cascade existe. Il s'agit de l'instruction **switch**.

Syntaxe

switch var
case cst_1,
séquence d'instructions 1
case cst_2,
séquence d'instructions 2
...
Case cst_N,
séquence d'instructions N
otherwise
séquence d'instructions par défaut
end

où

- var est une variable numérique ou une variable chaîne de caractères;
- cst_1, ..., cst_N, sont des constantes numérique ou des constantes chaîne de caractères;
- *séquence d'instructions i* est la séquence d'instructions à exécuter si le contenu de la variable var est égal à la constante cst_i (var==cst_i).

Interprétation
Si la variable var est égale à l'une des constantes cst_1, ..., cst_N, (par exemple cst_i) alors la séquence d'instructions correspondante (ici *séquence d'instructions i*) est exécutée. Le

programme reprend ensuite à la première instruction suivant le mot-clé end. Si la variable var n'est égale à aucune des constantes la *séquence d'instructions par défaut* est exécutée.

Remarque
La variable var doit bien entendu être du même type que les constantes cst_1, ..., cst_N.
Il n'est pas nécessaire de prévoir un cas par défaut (bien que cela soit préférable). S'il n'y a pas de cas par défaut et si la variable var n'est égale à aucune des constantes, alors le programme continue à la première instruction suivant le mot-clé end.
Il est possible de regrouper plusieurs << cas >> si la séquence d'instructions à exécuter est la même pour ces différents cas. La syntaxe est alors,

case{ cst_k , cst_l , ...**}**

Séquence d'instructions commune

Reprenons l'exemple où l'on souhaite initialiser une matrice A en fonction de la valeur prise par une variable numérique num (numéro d'exemple). On obtient:

```
switch num
  case 1,
    A = ones(n)
  case 2,
    A = magic(n);
  case {3,4},
    A = rand(n);
  otherwise
    error('numero d''exemple non prevu ...');
end
```

Voici un exemple de choix portant sur une variable de type chaîne de caractères.

```
Disp('voulez vous continuer ? o/n')
switch rep
  case {'oui','o'},
    disp('bravo ...');
  case {'non','n'}
    disp('perdu ...');
end
```

6 Instructions de contrôle

Les instructions de contrôle sous MATLAB sont très proches de celles existant dans d'autres langages de programmation.

6.1 Boucle FOR : parcours d'un intervalle

Une première possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle pour les valeurs d'un indice, incrémenté à chaque itération, variant entre deux bornes données. Ce processus est mis en œuvre par la **boucle for**.

Syntaxe

For indice**=***borne_inf***:***borne_sup*
séquence d'instructions
end

où

indice est une variable appelée l'*indice de la boucle*;

borne_inf et ***borne_sup*** sont deux constantes réelles (appelées *paramètres de la boucle*);

Séquence d'instructions est le traitement à effectuer pour les valeurs d'indices variant entre *borne_inf* et *borne_sup* avec un incrément de 1. On parle du c*orps de la boucle*.

Interprétation

Si *borne_inf* est plus petit ou égal à *borne_sup*, le traitement *séquence d'instructions* est exécuté *borne_sup - borne_inf +1* fois, pour les valeurs de la variable indice égales à *borne_inf, borne_inf*+1, ..., *borne_sup*. Si *borne_inf* est strictement plus grand que *borne_sup*, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (end).

Remarque :

L'indice de boucle ne prend pas nécessairement des valeurs entières. D'autre part il n'est pas nécessaire que l'indice de la boucle apparaisse dans le corps de la boucle; par contre il est interdit de modifier sa valeur s'il apparaît. Il est possible d'imbriquer des boucles mais elles ne doivent pas se recouvrir. On peut utiliser un incrément (*pas*) autre que 1 (valeur par défaut). La syntaxe est alors ***borne_inf: pas : borne_sup***. Le pas peut être négatif. Attention à bien gérer la borne supérieure! Voici un exemple venant illustrer les possibilités de variations de l'indice de la boucle

```
>> for r=1.1:-0.1:0.75
    disp(['r = ', num2str(r)]);
  end
r = 1.1
r = 1
r = 0.9
r = 0.8
>>
```

Voici un exemple d'utilisation d'une boucle pour calculer *n*! (le lecteur attentif sait calculer *n*! plus simplement.

```
>> n = 4;
>> nfac = 1;
>> for k = 1:n
    nfac = nfac*k;
  end
>> nfac
nfac =
   24
>>
```

6.2 Boucle WHILE : tant que . . . faire

Une seconde possibilité pour exécuter une séquence d'instructions de manière répétée consiste à effectuer une boucle tant qu'une condition reste vérifiée. On arrête de boucler dès que cette condition n'est plus satisfaite. Ce processus est mis en œuvre par la boucle while.

Syntaxe

While *expression logique*
séquence d'instructions
end

où

- *expression logique* est une expression dont le résultat peut être vrai ou faux;
- *séquence d'instructions* est le traitement à effectuer tant que *expression logique* est vraie.

Interprétation

Tant que *expression logique* est vraie le traitement *séquence d'**instructions*** est exécuté sous forme d'une **boucle**. Lorsque *expression logique* devient faux, on passe à l'instruction qui suit immédiatement l'instruction de fin de boucle (end).

Remarque

expression logique est en général le résultat d'un test (par exemple i < Imax) ou le résultat d'une fonction logique (par exemple all(x)). Il est impératif que le traitement de la *séquence d'instructions* agisse sur le résultat de *expression logique* sans quoi on boucle indéfiniment

Voici comment calculer *n*! avec une boucle while:

```
>> n = 4;
>> k = 1; nfac = 1;
>> while k <= n
    nfac = nfac*k;
    k = k+1;
  end
>> nfac
nfac =
   24
>>
```

7 Les entrées-sorties

7.1 Les formats d'affichage des réels

MATLAB dispose de plusieurs formats d'affichage des réels. Par défaut le format est le format court à 5 chiffres. Les autres principaux formats sont:

format long : format long à 15 chiffres.

format short e : format court à 5 chiffres avec notation en virgule flottante.

format long e : format long à 15 chiffres avec notation en virgule flottante.

MATLAB dispose également des formats **format short g** et **format long g** qui utilise la << meilleure >> des deux écritures à virgule fixe ou à virgule flottante. On obtiendra tous les formats d'affichage possibles en tapant help format. On impose un format d'affichage en tapant l'instruction de format correspondante dans la fenêtre de contrôle, par exemple format long. Pour revenir au format par défaut on utilise la commande format.

```
>> pi
ans =
   3.1416
>> format long
>> pi
ans =
  3.14159265358979
>>  format short e
>> pi^3
ans =
  3.1006e+01
>> format short g
>> pi^3
ans =
     31.006
>>
```

7.2 Affichage simple, la commande disp

La commande **disp** permet d'afficher un tableau de valeurs numériques ou de caractères. L'autre façon d'afficher un tableau est de taper son nom. La commande disp se contente d'afficher le tableau sans écrire le nom de la variable ce qui peut améliorer certaines présentations.

```
>> A = magic(4);
>>  disp(A)
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
>> A
A =
```

```
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
>>
```

On utilise fréquemment la commande disp avec un tableau qui est une chaîne de caractères pour afficher un message. Par exemple disp('Calcul du déterminant de la matrice A'). On utilise également la commande disp pour afficher un résultat. Par exemple disp(['Le déterminant de la matrice A vaut ', num2str(det(A))]). On remarque que l'usage de la commande disp est alors un peu particulier. En effet un tableau doit être d'un type donné, les éléments d'un même tableau ne peuvent donc être des chaînes de caractères et des valeurs numériques.

On a donc recours à la commande **num2str** (<< number to string >>) pour convertir une valeur numérique en une chaîne de caractères.

Attention, si la chaîne de caractères contient une apostrophe il est impératif de doubler l'apostrophe.

7.3 Lecture

La commande **input** permet de demander à l'utilisateur d'un programme de fournir des données.

La syntaxe

```
var = input(' demande ').
```

La phrase demande est affichée et MATLAB attend que l'utilisateur saisisse une donnée au clavier. Cette donnée peut être une valeur numérique ou une instruction MATLAB. Un retour chariot provoque la fin de la saisie. Une valeur numérique est directement affectée à la variable var tandis qu'une instruction MATLAB est évaluée et le résultat est affecté à la variable var.

Il est possible de provoquer des sauts de ligne pour aérer le présentation en utilisant le symbole **\n** de la manière suivante:

```
var = input('\n une phrase : \n ').
```

Pensez à mettre un point virgule (;) à la fin de l'instruction.

Sous cette forme il est impossible d'avoir une donnée de type chaîne de caractères dans la mesure où MATLAB essaie d'interpréter cette chaîne de caractères comme une instruction. Si l'on souhaite saisir une réponse de type chaîne de caractères on utilise

La syntaxe

```
var = input(' demande ','s').
```

Signalons qu'un retour chariot (sans autre chose) initialise la variable var au tableau vide **[]**.
Voici un exemple d'utilisation de la commande input (on suppose que la variable res contient une valeur numérique).

```
rep = input(' Affichage du resultat ? o/n [o] ','s');
if isempty(rep), rep = 'o'; end
if rep == 'o' |  rep == 'y'
  disp(['Le resultat vaut ', num2str(res)])
end
```

8 Scripts et fonctions

Il est possible d'enregistrer une séquence d'instructions dans un fichier (appelé un *M-file*) et de les faire exécuter par MATLAB. Un tel fichier doit **obligatoirement** avoir une extension de la forme .m (d'où le nom M-file) pour être considéré par MATLAB comme un fichier d'instructions. On distingue 2 types de M-file, les fichiers de scripts et les fichiers de fonctions. Un script est un ensemble d'instructions MATLAB qui joue le rôle de programme principal. Si le script est écrit dans le fichier de nom nom.m on l'exécute dans la fenêtre MATLAB en tapant nom. Même si l'on ne souhaite pas à proprement parler écrire de programme, utiliser un script est très utile. Il est en effet beaucoup plus simple de modifier des instructions dans un fichier à l'aide d'un éditeur de texte que de retaper un ensemble d'instructions MATLAB dans la fenêtre de commande.
Les fichiers de fonctions ont deux rôles. Ils permettent à l'utilisateur de définir des fonctions qui ne figurent pas parmi les fonctions incorporées de MATLAB (<< built-in functions >>) et de les utiliser de la même manière que ces dernières (ces fonctions sont nommées fonctions utilisateur). Ils sont également un élément important dans la programmation d'applications où les fonctions jouent le rôle des fonctions et procédures des langages de programmation usuels.

On définit la fonction fonc de la manière suivante:

function [*$vars_1$, ..., $vars_m$***] = fonc(***$vare\_1$, ..., $vare_n$**)**

séquence d'instructions

où

- $vars_1$, ..., $vars_m$ sont les variables de sortie de la fonction;
- $vare_1$, ..., $vare_n$ sont les variables d'entrée de la fonction;
- *séquence d'instructions* est le corps de la fonction.

Le fichier doit impérativement commencer par le mot-clé **function**. Suit entre crochets les variables de sortie de la fonction, le symbole **=**, le nom de la fonction et enfin les variables d'entrée entre parenthèses. Si la fonction ne possède qu'une seule variable de sortie, les crochets sont inutiles. Il est impératif que la fonction ayant pour nom **fonc** soit enregistrée dans un fichier de nom **fonc.m** sans quoi cette fonction ne sera pas << visible >> par MATLAB.

Dans l'exemple qui suit, on définit une fonction modulo qui calcule la valeur de a modulo n en prenant pour système de résidus *{1, 2, ..., n}* au lieu de *{0, 1, ..., n-1}* (système de résidus considéré par la fonction incorporée **mod**). Les lignes qui suivent doivent être enregistrées dans un fichier de nom modulo.m.

```
function [r,q] = modulo(a,n)

% Calcule la valeur de a modulo n en prenant pour systeme de residus
% 1, ... , n au lieu de 0, ... , n-1.
%
```

```
% appel : [r,q] = modulo(a,n)
%
% Arguments de sortie :
%  r : le residu
%  q : le quotient

q = floor(a./n);
r = a - n*q;
```

Les lignes précédées du symbole **%** sont des lignes de commentaire. Les lignes de commentaire situées entre la ligne function ... et la 1-ere ligne d'instructions sont affichées si l'on demande de l'aide sur la fonction modulo.

```
>> help modulo

 Calcule la valeur de a modulo n en prenant pour systeme de residus
 1, ... , n au lieu de 0, ... , n-1.

 appel : [r,q] = modulo(a,n)

 Arguments de sortie :
  r : le residu
  q : le quotient

>>
```

L'appel d'une fonction utilisateur s'effectue de la même façon que l'appel de n'importe quelle fonction MATLAB:

```
>> b = 10 ; m = 4;
>> [r,q] = modulo(b,m)
r =
    2
q =
    2
>> modulo(5,10)
ans =
    5
>>
```

Remarques

1. Il n'y a pas de mot-clé (par exemple end) pour indiquer la fin de la fonction. La fonction est supposée se terminer à la fin du fichier. Il est toutefois possible de provoquer un retour au programme appelant dans le corps de la fonction grâce à la commande **return**.
2. On ne peut écrire qu'une seule fonction par fichier (qui doit porter le nom de cette fonction). Toutefois dans la version 5 de MATLAB existe la notion de << sous-fonction >>. Une sous-fonction est une fonction écrite dans le même fichier qu'une autre

fonction (dite principale) et qui ne sera utilisable que par cette fonction principale (une sous-fonction ne peut pas être appelée par un autre sous-programme que la fonction principale).

3. Si le fichier ne commence pas par le mot-clé function on a tout simplement écrit un script!

La gestion des variables d'entrée et de sortie est très souple sous MATLAB. Si l'on n'est intéressé que par le résidu et pas par le quotient, on peut se contenter de ne mettre qu'une seule variable de sortie, v = modulo(10,4). Dans cet appel la variable v contiendra le résidu (la première variable de sortie). Par contre, même si l'on ne souhaite recueillir que le quotient, on est obligé d'effectuer un appel de la forme [r,q] = modulo(10,4) et donc de définir une variable inutile. Aussi, d'une manière générale, il est bon de ranger les variables de sortie par ordre << d'importance >>. Il est également possible d'appeler une fonction donnée avec moins de variables d'entrée que le nombre indiqué pour la définition de la fonction (il faut bien entendu que le corps de la fonction soit programmé de sorte de prévoir cette éventualité). Il existe deux fonctions MATLAB utiles pour gérer cette situation: **nargin** qui retourne le nombre de variables d'entrée utilisés lors de l'appel et **nargout** qui retourne le nombre de variables de sortie prévues lors de l'appel. Voici un petit exemple venant illustrer ces possibilités.

```
function [A,rang] = matale(T,m,n)

% Construit une matrice A de m lignes et n colonnes ayant des elements
% entiers generes de maniere aleatoire entre 0 et T.
% Calcule le rang de la matrice si l'appel est effectue avec 2 arguments
% de sortie.
% Si la matrice est carree, le parametre n peut etre omis.
%
% Appels:
%      [A,r] = Matale(T,m,n)
%      [A,r] = Matale(T,m)
%         A = Matale(T,m,n)
%         A = Matale(T,m)

if nargin == 2
  A = fix(T*rand(m));
else
  A = fix(T*rand(m,n));
end

if nargout == 2
 rang = rank(A);
end
```

Dans cet exemple, on gère les variables d'entrée de la fonction de sorte de ne pas avoir besoin de donner lors de l'appel le nombre de lignes et de colonnes si la matrice est carrée.

On gère aussi les variables de sortie afin de ne pas calculer le rang de la matrice si aucune variable de sortie pour le résultat n'est prévue lors de l'appel.

```
>> [A,r] = matale(20,3,4)
A =
   16   13   13   10
   10   16    7   14
    4    0   16    8
r =
    3
>> [A,r] = matale(20,3)
A =
   12    0   18
    5   14    9
    3    8    8
r =
    3
>> A = matale(20,3)
A =
    8    7    2
   17   16    4
    1    0    3
>>
```

Un point important concerne la gestion des variables entre le programme principal (ou le workspace) et les fonctions de l'utilisateur. Toutes les variables définies à l'intérieur d'une fonction sont des variables locales à cette fonction. La communication avec des variables du programme principal (ou du workspace) ou avec des variables d'autres fonctions se fait uniquement par les variables d'entrée et sortie de la fonction. Une alternative existe toutefois: il est possible de déclarer certaines variables comme des *variables globales* . Une variable globale peut être partagée entre un programme principal et plusieurs fonctions sans qu'il soit besoin de la spécifier parmi les variables d'entrée-sortie des différentes fonctions. On déclare une variable globale grâce au mot clé **global**. Par exemple pour déclarer la variable numex globale on écrit global numex. Attention, la déclaration global numex doit être reprise dans chaque fonction utilisant numex comme variable.