Ingénieurs sup Galilée

*1ère année*

UNIVERSITÉ PARIS 13 NORD

*2012-2013*

# Introduction à Matlab [a]

●●●●●●●

# Analyse Numérique : Support de TPs

*a.* Version du 8 janvier 2013 à 06:04

**par Cuvelier François**

Université Paris Nord - Institut Galilée - LAGA
Av. J.-B. Clément 93430 Villetaneuse
email : cuvelier@math.univ-paris13.fr

# Table des matières


**1 Présentation** **5**

**2 Les matrices** **6**
2.1 Présentation . . . . . 6
2.2 Les éléments d'une matrice . . . . . 6
2.3 Les complexes . . . . . 7
2.4 Format d'affichage . . . . . 8
2.5 Les opérations sur les matrices . . . . . 8
2.5.1 Transposer . . . . . 8
2.5.2 Ajouter et soustraire . . . . . 8
2.5.3 Multiplier . . . . . 9
2.5.4 Diviser . . . . . 9
2.5.5 élever à la puissance . . . . . 9
2.6 Les opérations sur les tableaux . . . . . 9
2.6.1 Ajouter et soustraire . . . . . 9
2.6.2 Multiplier et diviser . . . . . 10
2.6.3 Elever à la puissance . . . . . 10
2.7 Les opérateurs relationnels . . . . . 10
2.8 Les opérateurs logiques . . . . . 11
2.9 Les fonctions mathématiques et les matrices . . . . . 11
2.9.1 fonctions trigonométriques . . . . . 11
2.9.2 Fonctions élémentaires . . . . . 12
2.10 Manipulation des vecteurs et des matrices . . . . . 12
2.10.1 Générer des vecteurs . . . . . 12
2.10.2 Accès aux éléments d'une matrice . . . . . 12
2.10.3 Les matrices spéciales . . . . . 13
2.10.4 Manipulation des matrices . . . . . 13
2.11 Fonctions matricielles . . . . . 13

**3 Notions de base de programmation** **15**
3.1 Les instructions simples et les variables . . . . . 15
3.2 Les instructions composées . . . . . 16
3.2.1 La boucle `for` . . . . . 16
3.2.2 La boucle `while` . . . . . 16
3.2.3 Structure conditionnelle `if` . . . . . 17
3.2.4 Structure conditionnelle `Switch ...case ...end` . . . . 18

**4 M-fichiers : Scripts et Fonctions** **19**
4.1 Les scripts . . . . . 19
4.2 Les fonctions . . . . . 19
4.3 A savoir . . . . . 21
4.3.1 commande inline . . . . . 21
4.3.2 commande @ . . . . . 21

**5 Le graphisme** **23**
5.1 Les Graphiques 2D . . . . . 23
5.1.1 Les fonctions élémentaires . . . . . 23
5.1.2 Générer un graphique . . . . . 23
5.1.3 Styles, couleurs et marques des lignes . . . . . 24
5.1.4 Graphiques plus complexes . . . . . 25
5.2 Les graphiques 3D . . . . . 25

**6 Les fichiers d'entrées/sorties (I/O)** **27**
6.1 Ouverture et fermeture de fichiers . . . . . 27
6.1.1 `fopen` : ouverture d'un fichier. . . . . 27
6.1.2 `fclose` : fermeture d'un fichier. . . . . 27
6.2 Entrées/Sorties non formatées . . . . . 27
6.2.1 `fwrite` : écriture de données binaires dans un fichier. . . . 27
6.2.2 `fread` : lecture de données binaires à partir d'un fichier. . 28
6.3 Entrées/Sorties formatées . . . . . 28
6.3.1 `fprintf` : écriture de données formatées . . . . . 28
6.3.2 `fscanf` : lecture de données formatées . . . . . 29
6.3.3 sprintf : écriture de données formatées dans une chaîne de caractères . . . . . 29
6.3.4 sscanf : lecture d'une chaîne de caractères sous un format de contrôle . . . . . 30

# Table des figures

5.1 premier graphique . . . . . 24
5.2 command hold . . . . . 25
5.3 command subplot . . . . . 26

# Chapitre 1

# Présentation

MatLab [1] interprète et évalue des **expressions**. Les commandes MatLab sont souvent de la forme :

*variable* = *expression* ou plus simplement *expression*

MatLab permet, principalement, de manipuler des **matrices**, qui peuvent être réelles, complexes ou de type chaîne de caractères. On a les correspondances :

- scalaire $\equiv$ matrice $1 \times 1$
- vecteur de $\mathbb{R}^n \equiv$ matrice $n \times 1$
- matrice de $\mathbb{R}^n \times \mathbb{R}^m \equiv$ matrice $n \times m$

Cet interpréteur de commande peut s'utiliser de deux manières différentes. La première est le mode en ligne : l'utilisateur exécute ses commandes une à une. La seconde consiste en l'exécution d'un ensemble de commandes stockées dans un fichier texte d'extension `.m` appelé **script**.

1. le nom MatLab signifie matrix laboratory

# Chapitre 2

# Les matrices

## 2.1 Présentation

Il est possible de définir une matrice de différentes façons :
- en entrant explicitement une liste d'éléments.
- en générant une matrice à l'aide de fonctions prédéfinies.
- en créant une matrice dans un M-fichier.
- en chargeant une matrice à partir d'un fichier externe de données.

Pour créer et génèrer la matrice $A$ de dimension $2 \times 3$ donnée par

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

on exécute, par exemple, la commande suivante :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫ `A = [1 2 3;4 5 6]` | `A =`<br>`1 2 3`<br>`4 5 6` |

La partie gauche de ce tableau correspond aux commandes entrées par l'utilisateur. La partie droite correspond à l'affichage lors de l'exécution de ces commandes. Nous utiliserons toujours ce genre de tableau par la suite.

Pour éviter l'*écho* sur l'écran, il suffit de terminer la commande par un point-virgule :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫`A = [1 2 3;4 5 6];` | |

## 2.2 Les éléments d'une matrice

Les éléments d'une matrice peuvent aussi correspondre à toutes expressions MatLab, par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫`x = [-2.5 sqrt(3) 1/3]` | `x =`<br>`-2.5000 1.7321 0.3333` |

Pour accéder individuellement aux éléments d'une matrice, on utilise les parenthèses :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫x(5) = 2.3332 | x = |
| | -2.5000 1.7321 0.3333 0 2.3332 |

On remarque que MatLab redimensionne automatiquement la taille des matrices.

Il est possible de construire des matrices en utilisant d'autres matrices. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫b = [7 8 9]; | |
| ≫A=[A;b] | A =<br>1 2 3<br>4 5 6<br>7 8 9 |

On peut aussi extraire un bloc d'une matrice :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫B=A(1:2,:); | B =<br>1 2 3<br>4 5 6 |

Cette commande extrait les deux premières lignes de $A$ . Des explications et des exemples supplémentaires sont donnés dans *accès aux éléments d'une matrice (paragraphe 2.10.2).*

## 2.3 Les complexes

MatLab autorise les nombres complexes par l'utilisation des fonctions spéciales **i** et **j** dans toutes ses fonctions et opérations. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫ z = 4 + 3*i | z =<br>4.0000 + 3.0000i |
| ≫ z = 4 + 3*j | z =<br>4.0000 + 3.0000i |
| ≫ z = exp(i*pi/2) | z =<br>0.0000 + 1.0000i |

Pour définir une matrice complexe, il y a deux façons simples :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫ A=[1 2;3 4]+i*[5 6;7 8] | A =<br>1.0000 + 5.0000i 2.0000 + 6.0000i<br>3.0000 + 7.0000i 4.0000 + 8.0000i |
| ≫ A=[1+5i 2+6i;3+7i 4+8i] | A =<br>1.0000 + 5.0000i 2.0000 + 6.0000i<br>3.0000 + 7.0000i 4.0000 + 8.0000i |
| ≫ z = exp(i*pi/2) | z =<br>0.0000 + 1.0000i |

## 2.4 Format d'affichage

MatLab affiche à l'écran tous les résultats des commandes d'affectation non suivies d'un point virgule. On peut contrôler le format d'affichage.

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫format short<br>≫ x = [1/3 1.3456e-11] | x =<br>0.3333 0.0000 |
| ≫format short e<br>≫x = [1/3 1.3456e-11] | x =<br>3.3333e-001 1.3456e-011 |
| ≫format long<br>≫x = [1/3 1.3456e-11] | x =<br>0.33333333333333 0.00000000001346 |
| ≫format long e<br>≫x = 1/3 | x =<br>3.333333333333333e-001 |
| ≫format +<br>≫x = [1/3 1.3456e-11] | x =<br>++ |

Ce dernier format + est une manière d'afficher de grandes matrices. En effet, il n'affiche que +,- *ou un blanc* suivant que l'élément est positif, négatif ou nul.

## 2.5 Les opérations sur les matrices

### 2.5.1 Transposer

Le symbole ' (prime ou apostrophe) note la transposé d'une matrice.

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫A = [ 1 2 3; 4 5 6; 7 8 9]; | |
| ≫B=A' | B =<br>1 4 7<br>2 5 8<br>3 6 9 |
| ≫x = [ 1 2 3]' | x =<br>1<br>2<br>3 |

### 2.5.2 Ajouter et soustraire

Les symboles + et - permettent d'ajouter et de soustraire des matrices de même dimension :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫C = A + B | C =<br>2 6 10<br>6 10 14<br>10 14 18 |
| ≫y = x -1 | y =<br>0<br>1<br>2 |

### 2.5.3 Multiplier

Le symbole * note la multiplication de deux matrices :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫A = [ 1 2 ; 3 4 ]; | |
| ≫B = [ 2 1 ; 4 3 ];<br>≫A*B | ans =<br>10 7<br>22 15 |

L'opération A*B n'est définie que si le nombre de colonne de A est égale au nombre de ligne de B ou si A est un scalaire. A l'aide de cette opérateur on peut donc définir le produit scalaire de deux vecteurs

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫x = [ 1 ; 2 ]; | |
| ≫y = [ 2 ; 1 ];<br>≫x'*y | ans =<br>4 |

### 2.5.4 Diviser

Il y a deux symboles de division dans MatLab : \ et /
- X=A \ B est la solution de A*X=B.
- X=B/A est la solution de X*A=B.

### 2.5.5 élever à la puissance

L'expression A^p correspond à A à la puissance p et elle est définie si A est une matrice carrée et p un scalaire.

## 2.6 Les opérations sur les tableaux

### 2.6.1 Ajouter et soustraire

Ces opérations sont identiques à celles sur les matrices

### 2.6.2 Multiplier et diviser

L'opérateur `.*` correspond à la multiplication élément par élément de tableaux. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫x=[1 2 3];y=[4 5 6]; | |
| ≫z=x.*y | z =<br>4 10 18 |

Les expressions `A./B` et `A. \ B` donnent les quotients éléments par éléments.

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫z=y./x | z =<br>4 2.5000 2.0000 |

### 2.6.3 Elever à la puissance

Le symbole `.^`correspond à l'élévation à la puissance élément par élément, l'exposant ou la base pouvant être un scalaire.

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫z=x.^y | z =<br>1 32 729 |
| ≫x.^2 | ans =<br>1 4 9 |
| ≫2.^[x y] | ans =<br>2 4 8 16 32 64 |

## 2.7 Les opérateurs relationnels

Il existe six opérateurs relationnels sous MatLab pour comparer deux matrices de même dimension.

| | |
|---|---|
| < | inférieur à |
| <= | inférieur ou égal à |
| > | supérieur à |
| >= | supérieur ou égal à |
| == | égal à |
| ~= | différent de |

MatLab compare les paires d'éléments correspondant ; le résultat est une matrice ne contenant que des 0 (faux) ou des 1 (vrai). Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫2+2~=4 | ans =<br>0 |
| ≫A=[-1 2 3; 2 4 -2]; | |
| ≫B=(A>0) | B =<br>0 1 1<br>1 1 0 |

## 2.8 Les opérateurs logiques

Les opérateurs logiques sont `&`, `|` et `~`. Ils correspondent respectivement au et , ou et non logique.

- `C=A&B` est la matrice qui a pour éléments 1 si l'élément de `A` et l'élément de `B` sont non nuls, 0 si l'un des deux est nuls. `A` et `B` doivent être de même dimension à moins que l'un des deux soit un scalaire.
- `C=A | B` est la matrice qui a pour éléments 1 si l'un des éléments de `A` ou de `B` est non nuls, 0 si les deux sont nuls. `A` et `B` doivent être de même dimension à moins que l'un des deux soit un scalaire.
- `C=~B` est la matrice qui a pour éléments 1 si l'élément de `B` est nul, 0 si l'élément de `B` est non nul.

## 2.9 Les fonctions mathématiques et les matrices

Un ensemble de fonctions mathématiques s'appliquent élément par élément sur des matrices. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| `≫A=[1 2 3; 4 5 6]; B=cos(pi*A)` | `B =`<br>`-1 1 -1`<br>`1 -1 1` |

### 2.9.1 fonctions trigonométriques

| | |
|---|---|
| `sin` | sinus |
| `cos` | cosinus |
| `tan` | tangente |
| `asin` | arcsinus |
| `acos` | arccosinus |
| `atan` | arctangente |
| `atan2` | |
| `sinh` | sinus hyperboliquefonctionsinus hyperbolique (sinh) |
| `cosh` | cossinus hyperbolique |
| `tanh` | tangente hyperbolique |
| `asinh` | arcsinus hyperbolique |
| `acosh` | arccosinus hyperbolique |
| `atanh` | arctangente hyperbolique |

### 2.9.2 Fonctions élémentaires

| | |
|---|---|
| `abs` | valeur absolue |
| `angle` | |
| `sqrt` | racine carré |
| `real` | partie réelle |
| `imag` | partie imaginairefonctionpartie imaginaire (imag) |
| `conj` | complexe conjugué |
| `round` | arrondi entier |
| `fix` | |
| `floor` | |
| `ceil` | |
| `sign` | |
| `rem` | |
| `gcd` | plus grand commun diviseur |
| `lcm` | plus petit commun multiple |
| `exp` | exponentiel base e |
| `log` | logarithme naturel |
| `log10` | logarithme base 10fonctionlogarithme base 10 (log10) |

## 2.10 Manipulation des vecteurs et des matrices

### 2.10.1 Générer des vecteurs

Le symbole : est très important sous MatLab. Il permet de créer rapidement des vecteurs ou des matrices.

| **commande** (input) | **affichage** (output) |
|---|---|
| `≫x=1:5` | `x = 1 2 3 4 5` |
| `≫y=1:-0.5:0` | `y = 1.0000 0.5000 0` |

### 2.10.2 Accès aux éléments d'une matrice

On accède aux éléments d'une matrice en utilisant des parenthèses.

| **commande** (input) | **affichage** (output) |
|---|---|
| `≫A=[1 2 3;4 5 6;7 8 9];` | |
| `≫A(3,3)=A(1,3)+A(3,1)` | `A =`<br>`1 2 3`<br>`4 5 6`<br>`7 8 10` |

On peut aussi accéder à des sous-matrices. Si `A` et `B` sont deux matrices $10 \times 10$ alors

- `A(1:5,3)` spécifie une matrice $5 \times 1$ contenant les 5 premiers éléments de la troisième colonne de `A`.
- `A(1:5,7:10)` spécifie une matrice $5 \times 4$ contenant les 5 premiers éléments des colonnes 7 à 10 de `A`.

- `A(:,3)` spécifie une matrice $10 \times 1$ correspondant à la troisième colonne de `A`.
- `A(1:5,:)` spécifie une matrice $5 \times 10$ correspondant aux 5 premières lignes de `A`.
- `A(:,[3 5 10])=B(:,1:3)` remplace les colonnes 3, 5 et 10 de `A` par les colonnes 1,2 et 3 de `B`.

### 2.10.3 Les matrices spéciales

MatLab possède un ensemble de fonctions générant des matrices utilisées en algèbre linéaire et en traitement du signal.

| | |
|---|---|
| `compan` | |
| `diag` | diagonal |
| `gallery` | matrice de test |
| `hadamard` | Hadamard |
| `hankel` | Hankel |
| `hilb` | Hilbert |
| `invhilb` | Hilbert Inverse |
| `kron` | |
| `magic` | |
| `pascal` | |
| `toeplitz` | Toeplitz |
| `vander` | Vandermonde |
| `zeros` | zeros |
| `ones` | constantes |
| `rand` | matrice aléatoire uniforme |
| `randn` | matrice aléatoire normale |
| `eye` | Identité |
| `linspace` | |
| `logspace` | |
| `meshgrid` | utilisée avec des fonctions de deux variables |

### 2.10.4 Manipulation des matrices

Des fonctions ou opérateurs permettent de manipuler des matrices :

| | |
|---|---|
| `rot90` | rotation |
| `fliplr` | |
| `flipud` | |
| `diag` | extrait ou créée une diagonale |
| `tril` | partie triangulaire inférieure (lower) |
| `triu` | partie triangulaire supérieure (upper) |
| `reshape` | redimensionne |
| `'` | transposition |
| `:` | rearrangement |

## 2.11 Fonctions matricielles

- `[L,U]=lu(A)` correspond à une factorisation LU.

- `[Q,R]=qr(A)` correspond à la factorisation orthogonale QR.
- `[U,S,V]=svd(A)` correspond à une décomposition en valeur singulière. En fait `A=U*S*V'` où `U` et `V` sont orthogonales et `S` diagonale.
- `[X,D]=eig(A)` correspond à une décomposition en vecteurs et valeurs propres.

# Chapitre 3

# Notions de base de programmation

## 3.1 Les instructions simples et les variables

MatLab est un langage basé sur les expressions : il les interprète et les évalue. Une expression est composée à partir d'opérateurs, de caractères spéciaux, de fonctions et de noms de variables. L'évaluation d'une expression produit une matrice. Celle-ci peut-être affichée à l'écran ou assignée à une variable pour un usage futur. La syntaxe d'une instruction MatLab est la suivante :

| variable = expression<br><br>**ou**<br><br>expression |
|---|

Lorsque l'on omet d'assigner une expression à une variable, MatLab créé automatiquement une variable `ans` auquel est assignée l'expression. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫`1/3` | `ans =`<br>`0.3333` |

Lorsque l'expression est suivie d'un point-virgule, elle est bien évaluée mais elle n'est pas affichée.

Lorsque une expression est trop grande pour tenir sur une ligne, on peut passer à la ligne en utilisant un blanc et trois points consécutifs ( `...`) suivi d'un retour chariot. Par exemple :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫`s=1-2+3-4 ...`<br>`+5-6+7-8` | `s =`<br>`-4` |

Les noms de variables et de fonctions sont composés d'une lettre suivi par n'importe quel nombre de lettres, chiffres ou underscores (_). MatLab ne prends en compte que les 19 premiers caractères du nom.

MatLab fait la distinction entre les minuscules et les majuscules : la variable `a` est différente de la variable `A`.

## 3.2 Les instructions composées

### 3.2.1 La boucle `for`

Cette instruction permet de répéter un nombre de fois fixé un ensemble d'instructions. Sa syntaxe est la suivante :

**for** v=*liste*
    *instruction*$_1$;
    ...
    *instruction*$_n$;
**end**

ou, en condensé (moins lisible),

**for** v=*liste* , *instruction*$_1$; ...*instruction*$_n$; **end**

La variable v va parcourir les éléments de *liste* du premier au dernier et pour chacun d'entre eux les instructions *instruction*$_1$ à *instruction*$_n$ sont exécutées.

Par exemple , **for** i=1:n,x(i)=0;**end** permet d'initialiser les `n` premiers éléments de `x` à `0`. On peut aussi écrire de façon plus lisible :

Listing 3.1 – Boucles pour

```
for i=1:m
  for j=1:n
    z(i,j)=x(i)*y(j);
  end
end
```

### 3.2.2 La boucle `while`

Cette instruction permet de répéter un nombre indéfini de fois un ensemble d'instructions. Il faut toutefois s'assurer que la boucle n'est pas infinie. Sa syntaxe est la suivante :

**while** *condition*
    *instruction*$_1$;
    ...
    *instruction*$_n$;
**end**

ou, en condensé (moins lisible),

**while** *condition* , *instruction*$_1$;...*instruction*$_n$; **end**

Les instructions entre la condition et le end de la boucle sont effectuées tant que la condition est vérifiée. Par exemple ,

Listing 3.2 – Boucles while (v1)

```
n = 1;
while prod(1:n) < 1e10 ,  n = n+1; end
n
```

permet de déterminer le premier entier $n$ tel que $n!$ ai plus de 10 chiffres.

On peut aussi écrire de manière plus lisible :

Listing 3.3 – Boucles while (v2)

```
A=rand(5,5);
n=1;
while norm(A,1)>1e-3
  A=A/2;
  n=n+1;
end
n
```

### 3.2.3 Structure conditionnelle `if`

Cette instruction permet d'effectuer différentes instructions suivant qu'une ou plusieurs conditions sont vérifiées. Sa syntaxe est la suivante

**if** *condition*
  *instructions*;
**end**

**ou**

**if** *condition*
  *instructions*;
**else**
  *instructions*;
**end**

**ou**

**if** *condition*$_1$
  *instructions*;
**elseif** *condition*$_2$
  *instructions*;
**elseif** *condition*$_3$
  *instructions*;
⋮
**else**
  *instructions*;
**end**

Par exemple

Listing 3.4 – Instruction if

```
if n<0
  A = negative(n);
elseif rem(n,2) == 0
  A = even(n);
else
  A = odd(n);
end
```

### 3.2.4 Structure conditionnelle `Switch ...case ...end`

```
switch expression
    case valeur_1
        instructions_1;
    case valeur_2
        instructions_2;
    ⋮
    otherwise
        instructions;
end
```

Ici *expression* est de type scalaire ou chaîne de caractères. Si *expression* == $valeur_i$ alors $instructions_i$ est effectuées. Si toutes les conditions *expression* == $valeur_i$ sont fausses alors ce sont les *instructions* du branchement **otherwise** qui sont effectuées.

Par exemple

Listing 3.5 – Structure conditionnelle `Switch ...case ...end`

```
switch i
  case 1
    j=j+1;
  case 2
    j=j-1;
  case 3
    j=j*2;
  otherwise
    j=1;
end
```

# Chapitre 4

# M-fichiers : Scripts et Fonctions

Un **M-fichier** est un fichier texte (ASCII) ordinaire d'extension ".m". Il contient un ensemble de commande MatLab pouvant faire référence à d'autres M-fichiers. Un M-fichier peut s'appeler récursivement.

Deux types de M-fichier peuvent être créé : les **scripts** et les **fonctions**.

## 4.1 Les scripts

Les scripts permettent d'exécuter un nombre important de commandes. Etant stockés dans des fichiers, ceux-ci peuvent être aisément modifiés en utilisant un éditeur de texte. Par exemple, voici le contenu du fichier `fibno.m` :

Listing 4.1 – Script : suite de Fibonacci (`fibno.m`)

```
% M-Fichier permettant de calculer les 16 premiers nombres
% de Fibonacci et de les afficher.
f = [1  1]; i =1;
while f(i) + f(i+1) < 20
  f(i+2) = f(i) + f(i+1);    % redimensionnement automatique de f
                             % -> à éviter.
  i = i+1;
end
f
```

Ici le symbole `%` indique que le reste de la ligne est un commentaire. On utilisera ce M-fichier de la manière suivante

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫`fibno` | `f =`<br>`1 1 2 3 5 8 13` |

## 4.2 Les fonctions

Les M-fichiers correspondant à la définition d'une fonction commence toujours par le mot clé **function**. Ils permettent d'étendre les possibilités de MatLab

en autorisant l'utilisateur à créer ses propres fonctions à l'aide du langage MatLab. Le nom du M-fichier correspondant est le nom de la fonction suivi de l'extension `.m`. La syntaxe de la **déclaration d'une fonction** est la suivante :

```
function [args1,args2,...] = nomfonction(arge1,arge2,...)
   instructions
```

- arge1,arge2,... sont les paramètres d'entrée de la fonction (type quelconque)
- args1,args2,... sont les paramètres de sortie de la fonction (type quelconque)
- instructions sont les instructions permettant d'affecter les arguments de sortie.
- nomfonction nom de la fonction (sans caractères blanc,...)

L'**appel d'une fonction** s'effectue de la manière suivante :

```
[vars1,vars2,...] = nomfonction(vare1,vare2,...)
```

Pour une fonction avec un seul argument de sortie, on peut simplifier la syntaxe de sa **déclaration** :

```
function args = nomfonction(arge1,arge2,...)
   instructions
```

On donne ici un exemple simple de fonction. Le fichier `mean.m` contient les instructions suivantes :

Listing 4.2 – Fonction : moyenne (`mean.m`)

```
function y = mean(x)
  %MEAN retourne la valeur moyenne.
  %Pour un vecteur, MEAN(x) retourne la valeur moyenne.
  %Pour une matrice, MEAN(x) retourne un vecteur colonne contenant
  %la valeur moyenne de chaque colonne.
  [m,n] = size(x);
  if m == 1
    m = n;
  end
  y = sum(x)/m;
```

L'existence d'un tel fichier défini une nouvelle fonction appelée **mean**. Cette fonction s'utilise alors comme n'importe qu'elle autre fonction MatLab à la condition toutefois que MatLab puisse la "trouver" dans le répertoire courant ou dans les répertoires de recherche de Matlab (help **path**).

Voici un exemple d'utilisation de la fonction **mean** :

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫y = 1:99;mean(y) | ans = 50 |

Voici quelques commentaires sur le fichier `mean.m` :

- La première ligne déclare un nom de fonction, les paramètres d'entrées et de sortie. Sans cette ligne, ce fichier serait un script.
- Le symbole `%`
- Les premières lignes en commentaire sous la déclaration de la fonction permettent d'obtenir l'aide en ligne lorsque l'on entre la commande `help mean`.
- Les variables `m`, `n` et `y` sont des variables locales à la fonction : Elles n'ont d'existence qu'à l'intérieur de celle-ci.
- Il n'est pas nécessaire d'appeler la fonction en utilisant la variable `x`. En fait, seul le contenu des paramètres d'entrée est passé et permet d'initialiser les variables locales à la fonction.

## 4.3 A savoir

Il est possible de déclarer de manière différentes les fonctions simples à l'aide des commandes inline, @.

### 4.3.1 commande inline

```
args = inline(expression,arge1,arge2,...);
```

- expression est une chaine de caractères contenant une expression matlab avec comme paramètres le contenu des chaines de caractères arge1,arge2 ,... .
- args est une fonction Matlab.

Listing 4.3 – Fonction : méthode `inline`

```
%On défini la fonction f(x)=x.*x :
f=inline('x.*x','x');
%On défini la fonction g(R,theta)=R*cos(theta) :
g=inline('R*cos(theta)','R','theta');
```

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫f(1:3) | **ans** = 1 4 9 |
| ≫g(1,[**pi**/4 3∗**pi**/4]) | **ans** = 0.7071 -0.7071 |

### 4.3.2 commande @

```
args = @(arge1,arge2,...) expression
```

- expression est expression matlab avec comme paramètres arge1,arge2 ,... .
- args est une fonction Matlab.

Listing 4.4 – Fonction : méthode @

```
% On défini la fonction f(x)=x.*x :
f=@(x)    x.*x;
% On défini la fonction g(R,theta)=R*cos(theta) :
g=@(R,theta)  R*cos(theta);
```

| **commande** (input) | **affichage** (output) |
|---|---|
| ≫f(1:3) | **ans** = 1 4 9 |
| ≫g(1,[**pi**/4 3∗**pi**/4]) | **ans** = 0.7071 -0.7071 |

# Chapitre 5

# Le graphisme

## 5.1 Les Graphiques 2D

### 5.1.1 Les fonctions élémentaires

On donne ici une liste de fonctions qui diffèrent uniquement dans le choix du système de coordonnées. Chacune d'entre elles accepte comme paramètres d'entrée des vecteurs ou des matrices.

| | |
|---|---|
| `plot` | créé un tracé à partir d'un vecteur ou des colonnes d'une matrice |
| `loglog` | crée un tracé en coordonnées logarithmiques |
| `semilogx` | crée un tracé en coordonnée logarithmique pour l'axe des x |
| `semilogy` | crée un tracé en coordonnée logarithmique pour l'axe des y |

On peut aussi ajouter des titres, libeller les axes, mettre une grille et positionner du texte en utilisant :

| | |
|---|---|
| `title` | ajoute un titre au tracé |
| `xlabel` | ajoute un libellé à l'axe des x |
| `ylabel` | ajoute un libellé à l'axe des y |
| `text` | place un texte à une position donnée |
| `gtext` | place un texte à l'aide de la souris |
| `grid` | affiche une grille |

### 5.1.2 Générer un graphique

Si $y$ est un vecteur `plot(y)` génère le graphe $(i, y(i))$. Si l'on spécifie deux vecteurs comme arguments `plot(x,y)` génère le graphe $(x(i), y(i))$. Il est possible d'appeler plusieurs vecteurs pour des tracés multiples : `plot(x,y1,y2)` génère sur le même graphe les deux tracés $(x(i), y1(i))$ et $(x(i), y2(i))$. Voici un exemple de tracé :

```
t = 0:pi/100:2*pi;
x = sin(t);
y1 = sin(t+.25);
y2 = sin(t+.5);
```

```
plot(x,y1,'r-',x,y2,'g--')
title('Premier graphique');
xlabel('x=sin(t)');
ylabel('y=sin(t+)');
```

le résultat est représenté en figure 5.1.

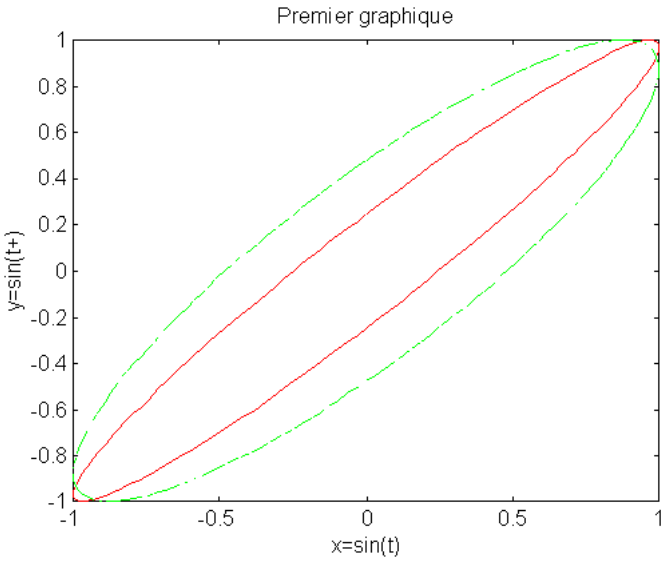


FIGURE 5.1 – premier graphique

### 5.1.3 Styles, couleurs et marques des lignes

Comme nous l'avons vu dans l'exemple précédent, il est possible de passer une chaîne de caractères comme argument pour spécifier le style et la couleur. L'appel s'effectue de la manière suivante : `plot(x,y,s)`

La chaîne `s` possède au maximum 3 caractères construits comme combinaison des caractères suivant :

| **Symbole** | **Couleur** | **Symbole** | **Style** |
|---|---|---|---|
| `y` | jaune (yellow) | `.` | point |
| `m` | magenta | `o` | cercle |
| `c` | cyan | `x` | marque x |
| `r` | rouge | `+` | plus |
| `g` | vert (green) | `*` | étoile |
| `b` | bleu | `-` | trait continu |
| `w` | blanc (white) | `:` | points |
| `k` | noir (black) | `-.` | trait et point |
| | | `--` | pointillé |

### 5.1.4 Graphiques plus complexes

La commande `hold` permet d'ajouter des lignes à un graphique :

```
plot(x)
hold on
plot(y1,'--')
plot(y2,'-.')
hold off
```

pour obtenir la fenêtre graphique 5.2.

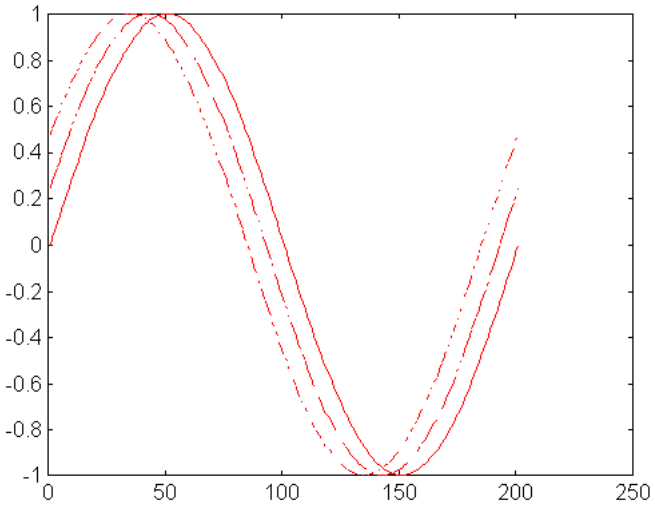


FIGURE 5.2 – command hold

La commande `subplot` permet de visualiser plusieurs graphiques simultanément.

```
subplot(2,1,1)
plot(y1)
subplot(2,1,2)
plot(y2)
```

Le résultat de ces commandes est représentée en figure 5.3.

## 5.2 Les graphiques 3D

MatLab possède de nombreuses fonctions pour représenter des données en 3D. Nous en donnons un résumé :

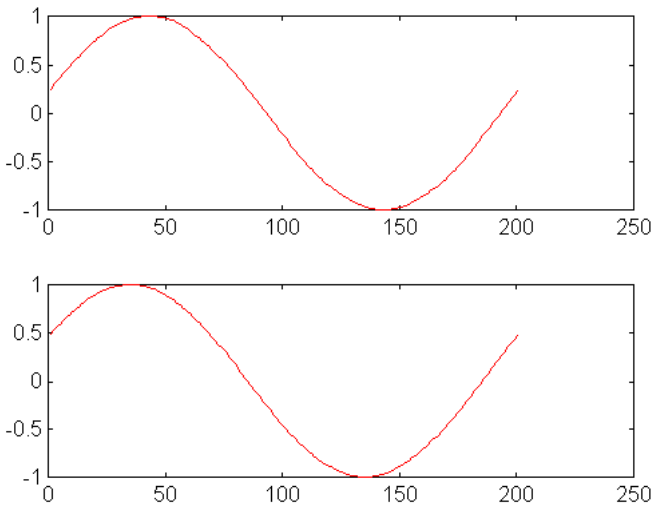


FIGURE 5.3 – command subplot

| | |
|---|---|
| `plot3` | trace des lignes et des points en 3D |
| `contour, contour3` | trace des contours (lignes de niveaux) |
| `pcolor` | |
| `image` | |
| `mesh,meshc,meshz` | |
| `surf,surfc,surfl` | |
| `fill3` | |

# Chapitre 6

# Les fichiers d'entrées/sorties (I/O)

On traite ici des fonctions de bas niveaux pour les fichiers d'entrées/sorties, c'est à dire de l'écriture et de la lecture de données contenues dans des fichiers.

## 6.1 Ouverture et fermeture de fichiers

### 6.1.1 `fopen` : ouverture d'un fichier.

```
fid = fopen('NomFichier',permission)
```

Ouvre le fichier de nom 'NomFichier' avec la permission spécifiée. Une permission est l'une des chaînes suivantes :

| | |
|---|---|
| `'r'` | lecture |
| `'w'` | écriture avec création si nécessaire |
| `'a'` | ajouter avec création si nécessaire |
| `'r+'` | lecture et écriture sans création |
| `'w+'` | |
| `'a+'` | lecture et ajout avec création si nécessaire |

Par défaut, les fichiers sont ouverts en mode binaire. Pour ouvrir un fichier texte, ajouter `'t'` à la chaîne de permission, par exemple `'rt'` et `'wt+'`.

### 6.1.2 `fclose` : fermeture d'un fichier.

`fclose(fid)` ferme le fichier d'identificateur `fid` qui est un entier obtenu par l'utilisation de `fopen`.
`fclose('all')` ferme tous les fichiers ouverts.
`fclose` retourne 0 s'il n'y a pas eu de problèmes, -1 sinon.

## 6.2 Entrées/Sorties non formatées

### 6.2.1 `fwrite` : écriture de données binaires dans un fichier.

```
count = fwrite(fid,A,precision)
```

Ecrit les éléments de la matrice A dans le fichier d'identificateur fid en convertissant les valeurs à la précision spécifiée. Les données sont écritent dans l'ordre des colonnes. count est le nombre d'éléments correctement écrits.

Par exemple,

```
fid = fopen('rand5.bin','wb') ;
fwrite(fid,rand(5),'real*4');
```

crée un fichier binaire de 25x32-bits, contenant les 25 éléments de la matrice aléatoire 5x5 stockés comme des flottants 32-bits.

### 6.2.2 `fread` : lecture de données binaires à partir d'un fichier.

```
[A,count] = fread(fid,size,precision)
```

Lit les données binaires à partir du fichier d'identificateur `fid` et les écrit dans la matrice `A`. La valeur optionnel `count` retournée est le nombre d'éléments correctement lu. L'argument size est optionnel ; s'il n'est pas spécifié, le fichier entier est lu ; s'il est spécifié, les valeurs correctes sont :

| | |
|---|---|
| `N` | lit `N` éléments dans un vecteur colonne |
| `inf` | lit jusqu'à la fin du fichier |
| `[M,N]` | lit les éléments pour remplir une matrice `M` par `N`, dans l'ordre des colonnes. `N` peut être `inf` mais pas `M`. |

L'argument precision contrôle le nombre de bits lu pour chaque valeur et interprète ces bits comme des valeurs de type caractère, entier ou flottant.

Par exemple :

```
fid=fopen('rand5.bin','r');
A=fread(fid,[5,5],'real*4');
```

va permettre de lire la matrice 5x5 stockée dans le fichier 'rand5.bin' (voir exemple précédent).

## 6.3 Entrées/Sorties formatées

### 6.3.1 `fprintf` : écriture de données formatées

```
count = fprintf(fid,format,A,...)
```

formate les données des matrices `A`, ..., avec le contrôle spécifié par la chaîne `format`, et les écrits dans le fichier associé à l'identificateur `fid`. la valeur optionnelle `count` est le nombre d'éléments correctement écrits.

`format` est une chaîne de caractères correspondant au spécificateur du langage C.

Pour les nombres entiers :

| | |
|---|---|
| `%d` | `d` comme décimal |
| `%o` | `o` comme octal |
| `%x` | `x` comme héxadécimal |

Pour les nombres réels :

| | |
|---|---|
| `%f` | `f` comme "floating point", en notation $\pm ddd.dddddd$, avec 6 chiffres après le point décimal. |
| `%e` | `e` comme exposant, en notation $\pm d.dddddde \pm ddd$, avec 6 chiffres après le point décimal, et un seul avant qui n'est jamais 0. |
| `%g` | g comme général, équivalent en gros à `%f` si le nombre n'est ni trop grand ni trop petit, à `%e` sinon, et fourni avec exactement 6 chiffres significatifs. |

Dans le cas où il est important d'afficher un réel avec un nombre précis de chiffres, on pourra spécifier les formats comme suit :

| | |
|---|---|
| `%x.yf` | `y` chiffres après le point décimal, et `x` avant. Exemple : `%4.2f` . |
| `%.ye` | `y` chiffres après le point décimal, et un seul avant qui n'est jamais 0. Exemple : `%.16e` . |

Pour les chaînes de caractères :

| | |
|---|---|
| `%s` | `s` comme string (chaîne de caractères) |

`fprintf` diffère du langage C du fait de sa vectorisation.
Par exemple :

```
x = 0:pi/20:2*pi; y = [x; cos(x)];
fid = fopen('cos.txt','w');
fprintf(fid,'%6.2f  %12.8f\n',y);
```

crée un fichier texte contenant une table de la fonction cosinus

### 6.3.2 `fscanf` : lecture de données formatées

```
[A,count] = fscanf(fid,format,size)
```

lit des données contenues dans le fichier spécifié par l'identificateur `fid`, à l'aide du `format` spécifié (voir `fprintf`), et le retourne dans la matrice `A`. `count` est un argument de sortie optionnel qui retourne le nombre d'éléments correctement lus. Pour `size` voir `fprintf.`
`fscanf` diffère du langage C du fait de sa vectorisation.
Par exemples :

`S = fscanf(fid,'%s')` lit une chaîne de caractères.
`A = fscanf(fid,'%5d')` lit un entier décimal stocké sur 5 caractères.

### 6.3.3 sprintf : écriture de données formatées dans une chaîne de caractères

```
[S,errmsg] = sprintf(format,A,...)
```

formate les valeurs des matrices `A, ...`, à l'aide du `format` spécifié, et les retourne dans la variable chaîne de caractères `S`. `errmsg` est un argument de sortie optionnel qui retourne un message d'erreur sous forme de chaîne de caractères si une erreur est survenue ou une matrice vide sinon. Pour le `format` voir `fprintf`.

`sprintf` diffère du langage C du fait de sa vectorisation.

Par exemple :

```
S = sprintf('rho = %5.3f',(1+sqrt(5))/2)
```

crée la chaîne de caractères `S` ayant pour valeur `'rho = 1.618'`.

### 6.3.4 sscanf : lecture d'une chaîne de caractères sous un format de contrôle

```
[A,count,errmsg,nextindex] = sscanf(S,format,size)
```

lit les données à partir de la variable `S` de type chaîne de caractères à l'aide du `format` spécifié, et les retourne dans la matrice `A`. `count` est un argument de sortie optionnel qui retourne le nombre d'éléments correctement lus. Pour `size` et `format` voir `fprintf`.

`sscanf` diffère du langage C du fait de sa vectorisation.

Par exemple :

```
S = '2.7183  3.1416';
A = sscanf(S,'%f')
```

crée le vecteur à 2 colonnes `A` contenant les approximations de `e` et `pi`.