Une brève introduction à Matlab

Paul ARMAND[†]

3 décembre 2004

Ce document s'adresse aux étudiants de la licence de mathématiques. Le but est d'acquérir une connaissance suffisante de Matlab, pour résoudre les exercices d'application du cours d'analyse numérique qui seront proposés en travaux pratiques.

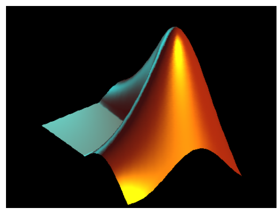

Table des matières

1 Introduction **3**

2 Début **3**
- 2.1 Démarrer . . . 3
- 2.2 Quitter . . . 4

3 Créer des matrices **4**
- 3.1 Matrice, vecteur, scalaire . . . 4
- 3.2 Vectoriser . . . 5
- 3.3 Créer . . . 6
- 3.4 Accéder . . . 7
- 3.5 Modifier . . . 8
- 3.6 Aide . . . 9
- 3.7 Exercices . . . 9

4 Opérations sur les matrices **9**
- 4.1 Opérations algébriques . . . 9
- 4.2 La division . . . 10
- 4.3 Opérations éléments par éléments . . . 11
- 4.4 Opérateurs relationnels . . . 11
- 4.5 Opérateurs logiques . . . 12
- 4.6 Le type logical . . . 12
- 4.7 Exercices . . . 13

[†]LACO - CNRS UMR 6090, Université de Limoges, Faculté des Sciences, 123, avenue Albert Thomas, 87060 Limoges (France) ; e-mail : `armand@unilim.fr`.

5 Fonctions usuelles **14**
5.1 Fonctions scalaires 14
5.2 Fonctions vectorielles 15
5.3 Fonctions matricielles 15
5.4 Exercices 16

6 Aide **16**
6.1 Aide en ligne et recherche par mot clé 16
6.2 Fonctions prédéfinies 17
6.3 Contenu de l'espace de travail 17
6.4 Contenu d'un répertoire 17
6.5 Exercices 17

7 Textes et chaînes de caractères **17**
7.1 Chaine = vecteur de caractères 17
7.2 Affichage, lecture et évaluation 18

8 Fichiers script **18**
8.1 Exemple 18
8.2 Chemin d'accés 19
8.3 Exercices 20

9 Boucles et contrôles **20**
9.1 Branchement conditionnel (`If...then...else`) 20
9.2 Branchement multiple (`Switch`) 21
9.3 Boucle finie (`For`) 22
9.4 Boucle infinie (`While`) 22
9.5 Exercices 23

10 Les fonctions **24**
10.1 Déclaration, arguments 24
10.2 Exemple : résolution d'une équation non linéaire 25
10.3 Exercices 28

11 Graphiques **28**
11.1 Graphiques 2d 29
11.2 Graphiques 3d 30
11.3 Exercices 31

12 Références **33**

1 Introduction

Matlab signifie **Ma**trix **lab**oratory. C'est un logiciel de calcul numérique. Il est destiné à traiter des applications à partir des outils de l'analyse numérique matricielle. Matlab possède aussi tout un ensemble de fonctionnalités graphiques permettant de visualiser les résultats numériques. Il possède des boîtes à outils, c'est à dire des fonctionnalités supplémentaires, dédiées à des domaines particuliers du calcul scientifique, comme la résolution d'équations aux dérivées partielles, l'optimisation, l'analyse de données, etc. Matlab est aussi un langage de programmation avec des possibilités d'interfaces vers des programmes écrits en C ou en Fortran.

En Matlab les calculs sont effectués avec une arithmétique à précision finie. Ceci le différencie des logiciels de calcul symbolique tel que Maple, mais la comparaison n'a pas lieu d'être. Calcul numérique et calcul symbolique sont des outils complémentaires du calcul scientifique.

Matlab a initialement été développé en Fortran par Cleve Moler. Aujourd'hui Matlab est écrit en C et utilise les bibliothèques LINPACK et ARPACK. Il est distribué par la société The MathWorks (www.mathworks.com). Il existe des logiciels gratuits de calcul numérique similaires à Matlab. Le logiciel Octave est un premier exemple. La syntaxe d'Octave est très semblable à celle de Matlab. Il est distribué gratuitement sous licence GNU (www.octave.org) et est inclus dans certaines distributions de Linux comme Mandrake. Scilab est un autre exemple de logiciel de calcul numérique dans l'esprit de Matlab. C'est un logiciel assez complet, avec une syntaxe un peu différente de celle de Matlab. Il est distribué gratuitement par la société Saphir Control (www.saphir-control.fr) et peut être installé sous Windows ou Linux.

2 Début

2.1 Démarrer

Sous Unix, on peut saisir la commande `matlab` dans une fenêtre de terminal ou bien cliquer sur l'icône associé à Matlab. Le logo apparaît pendant quelques secondes dans une fenêtre séparée et un bureau est créé (figure 1).

Il contient plusieurs fenêtres :
- fenêtre des commandes (Command Window),
- historique des commandes (Command History),
- répertoire courant (Current Directory),
- espace de travail (Workspace),
- rampe de lancement (Launch Pad).

Seule la première nous intéresse pour l'instant. Pour que cette fenêtre soit la seule visible, il suffit de sélectionner **Desktop Layout/Command Window Only** dans le menu **View** (voir figure 2). À ce stade le prompt `>>` est visible dans la fenêtre des commandes. Il indique que Matlab attend une commande.

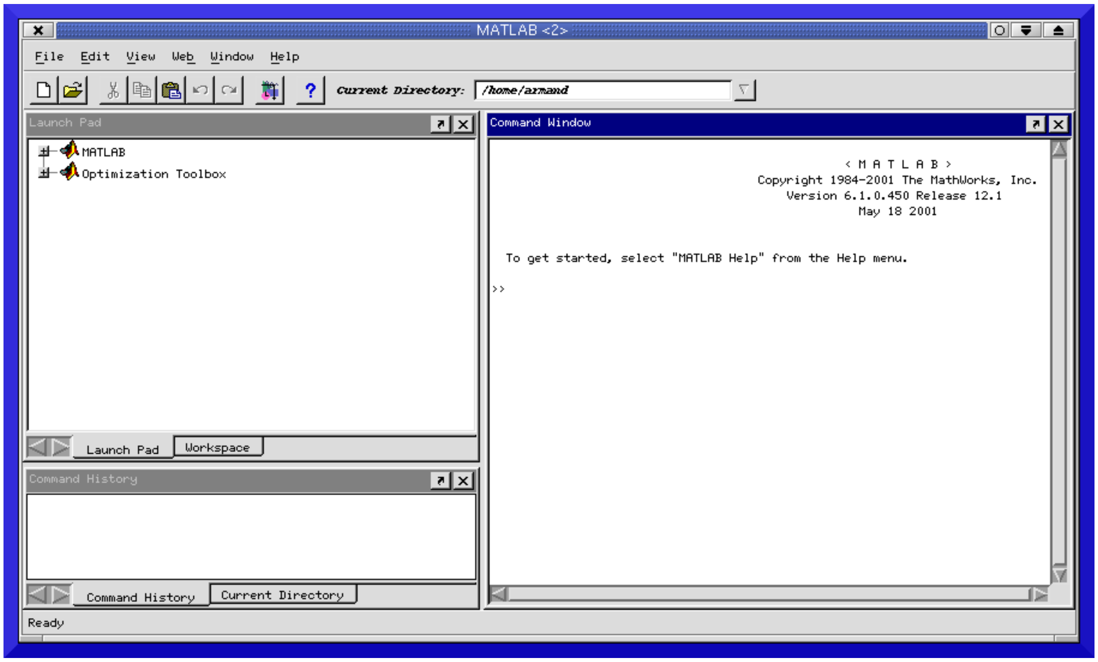

FIG. 1 – Bureau de démarrage.

2.2 Quitter

Pour quitter Matlab, sélectionner **Exit MATLAB** dans le menu **File** (figure 3) ou bien saisir la commande `exit` dans la fenêtre des commandes. On peut aussi utiliser la commande `quit`, elle permet d'exécuter un certain nombre de taches avant de quitter.

Important : sur la machine limrec, le nombre d'utilisateurs simultanés du logiciel est limité à 16. Il est donc important de quitter correctement le programme avant de terminer une session Unix, sinon on prend le risque de laisser Matlab en attente sur la machine et ainsi d'empêcher d'autres utilisateurs d'y accéder. Si, pour une raison quelconque, il n'est pas possible de quitter Matlab normalement, vous devez avertir un responsable.

3 Créer des matrices

3.1 Matrice, vecteur, scalaire

Avec Matlab on travaille essentiellement avec un seul type d'objet : une matrice. En Matlab une matrice est un tableau rectangulaire de nombres représentés en virgule flottante avec une double précision, c'est à dire 8 octets pour un nombre réel et 16 octets pour un nombre complexe. Une matrice 1×1 est interprétée comme un scalaire, celle ayant une seule ligne ou une seule colonne comme un vecteur.

```
>> mat = rand(2,3), scal = pi, vlig = [1 2 3 4], vcol = zeros(3,1)
```

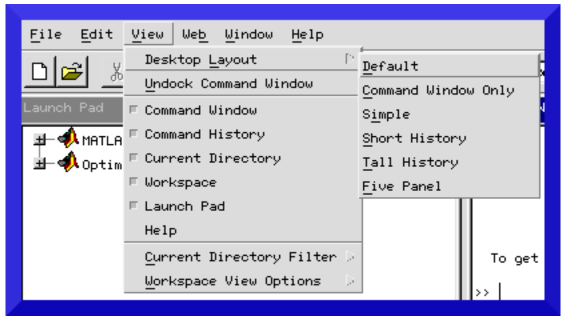

FIG. 2 – Choix de la disposition du bureau.

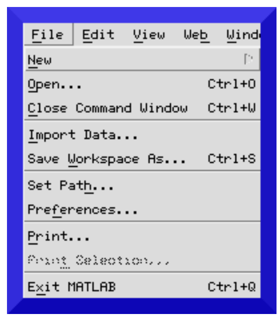

FIG. 3 – Menu File.

```
mat =
    0.4103    0.0579    0.8132
    0.8936    0.3529    0.0099
scal =
    3.1416
vlig =
     1     2     3     4
vcol =
     0
     0
     0
```

3.2 Vectoriser

La plupart des opérations et des fonctions agissent directement sur une matrice toute entière ou bien colonne par colonne. Il est important de bien comprendre cela dès le début. Ceci permet de simplifier considérablement l'écriture des programmes et aussi de réduire les temps d'exécution. Le langage de Matlab est interprété, à la différence des langages compilés comme Fortran ou C, d'où une relative lenteur d'exécution qui doit être compensée par une écriture “optimisée” des programmes. Par exemple, créons une matrice de taille 1000×1000.

```
>> A = rand(1000,1000);
```

Comptons le temps nécessaire pour diviser tous les éléments de `A` par 2.

```
>> tic, A=A/2; toc
elapsed_time = 0.1097
```

Pour diviser 1 million de nombres par 2, Matlab a besoin d’environ 1 dixième de seconde. Maintenant divisons chaque élément par 2, comme on le ferait dans un programme écrit en Pascal, c’est à dire avec deux boucles imbriquées.

```
>> tic, for i=1:1000, for j=1:1000, A(i,j)=A(i,j)/2; end, end, toc
elapsed_time =
   12.9849
```

Matlab a besoin de 100 fois plus de temps[1].

3.3 Créer

Pour créer une matrice il suffit d’ouvrir un `[`, énumérer les éléments ligne par ligne, puis fermer le `]`. Sur une même ligne les éléments sont séparés par une virgule ou un espace, deux lignes successives sont séparées par un point virgule ou un retour chariot. Par exemple, l’instruction

```
>> A = [ 1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

crée une matrice 4×4 appelée `A`. La réponse de Matlab est

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

L’instruction suivante crée la même matrice, mais le résultat n’est pas affiché car un point-virgule termine l’instruction :

```
>> A = [1, 2, 3, 4
5, 6, 7, 8
9 10 11 12; 13, 14, 15, 16
];
```

On peut aussi créer la même matrice `A` avec l’instruction suivante :

```
>> A = [1:4; 5:8; 9:12; 13:16]
```

[1]Tests réalisés avec un Pentium III à 650MHz et Matlab 6.0.0.88.

Nous avons utilisé ici l'opérateur deux-points. Cet opérateur est très important, on l'utilise très souvent. Il permet de discrétiser un intervalle avec un pas constant. L'instruction

```
>> u = 0:9
```

crée un vecteur `u` contenant les entiers de 0 à 9. Par défaut le pas est égal à 1, mais on peut spécifier une autre longueur de pas (essayez !) :

```
>> 0:2:9, 9:-3:0, 0:pi/3:2*pi
```

Noter que dans la ligne ci-dessus, nous avons utilisé une virgule pour séparer deux instructions sur une même ligne. On voit donc que virgule et point-virgule permettent de séparer des éléments d'une matrice et que ce sont aussi des séparateurs d'instructions.

,	sépare 2 éléments sur une ligne	termine une instruction avec affichage
;	sépare 2 lignes	termine une instruction sans affichage

Certaines fonctions permettent de créer des matrices particulières :
- `eye(n)` crée une matrice identité de dimension n ;
- `ones` et `zeros` créent des matrices de 1 et de 0 ;
- `rand` et `randn` des matrices dont les éléments sont choisis au hasard selon une loi uniforme sur $]0,1[$ et une loi normale centrée réduite ;
- `magic(n)` crée un carré magique de dimension n ($n \geq 3$) ;
- `diag` permet d'extraire des diagonales ou bien de créer des matrices diagonales ;
- `triu` et `tril` permettent d'extraire les parties triangulaire supérieure et triangulaire inférieure.

3.4 Accéder

Pour accéder aux éléments d'une matrice, on utilise la notation standard de l'algèbre linéaire : `A(i,j)` fait référence à l'élément de la ligne `i` et de la colonne `j` de `A`. Dans l'exemple précédent, l'instruction `A(2,3)` retourne la valeur `7`. Seul un entier positif non nul est accepté comme indice de matrice ou de vecteur. On peut extraire plusieurs éléments simultanément, par exemple :

```
>> A([1,3],2), A([1,3],[2,1,4])
```

On utilise l'opérateur deux-points pour extraire une ligne, une colonne ou une sous-matrice. L'instruction `A(2,:)` extrait la deuxième ligne, `A(:,1)` la première colonne et `A(2:4,1:3)` retourne le bloc

```
ans =
     5     6     7
     9    10    11
    13    14    15
```

L'instruction `A(:)` crée un vecteur colonne contenant les éléments de `A` énumérés colonne par colonne. Notons que l'instruction `A(7)` retourne la valeur `10`, car les éléments d'une matrice sont numérotés selon ce même ordre d'énumération. Ainsi, l'élément `A(i,j)` est numéroté `i+m*(j-1)`, où `m` est le nombre de lignes de `A`.

$$\begin{array}{cccc} 1 & m+1 & 2m+1 & \cdots \\ 2 & m+2 & 2m+2 & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ m & 2m & 3m & \ldots \end{array}$$

Cette numérotation permet d'extraire des éléments avec un tableau d'indices. Par exemple, l'instruction

```
>> A([1 2 3; 4 5 6])
```

retourne

```
ans =
     1     5     9
    13     2     6
```

3.5 Modifier

On peut modifier les éléments d'une matrice en leur affectant de nouvelles valeurs :

```
>> A(4,4) = 0, A(1,:) = A(1,:)*10, A = [A; [1  1  1 1]]
```

Matlab possède aussi un éditeur de tableau qui permet de modifier les dimensions et les entrées d'une matrice. Pour utiliser cet éditeur, sélectionner **Workspace** dans le menu **View**. Une fenêtre contenant les variables en mémoire apparaît, puis cliquer deux fois sur la variable à éditer. On peut aussi ouvrir `A` dans l"éditeur avec l'instruction `open A` (figure 4).

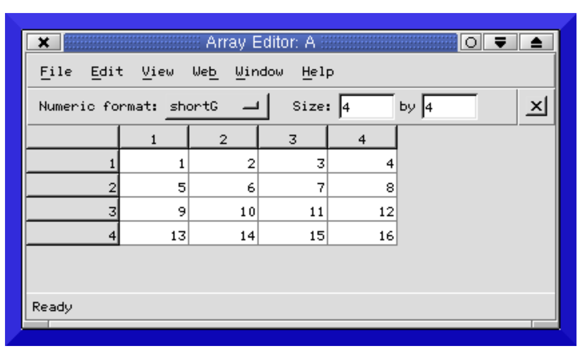

Fig. 4 – Éditeur de tableau.

3.6 Aide

L'instruction `help` permet d'obtenir de l'aide. Par exemle,

```
>> help diag
```

renvoie l'aide en ligne de la fonction `diag`. Si on ne cconnait pas le nom de la fonction qu'on veut utiliser, on peut faire une recherche par mot clé avec la fonction `lookfor`.

```
>> lookfor identity
EYE Identity matrix.
SPEYE  Sparse identity matrix.
```

L'utilisation de l'aide est un peu plus détaillée au paragraphe 6

3.7 Exercices

Exercice 1 *Soit* `A = [1:4 ; 5:8 ; 9:12 ; 13:16]`*. Quel est le résultat de l'instruction* `A(10:-1:5)` *? Comment créer un vecteur contenant les éléments de* `A` *dans l'ordre suivant :* `16 12 8 4 15 11 ...` *? Quel est le résultat de l'instruction* `[A(1,:),5]` *? Comment créer un vecteur contenant les éléments de* `A` *dans l'ordre suivant :* `1 2 3 4 ...` *? Quel est le résultat de* `diag(diag(A))` *?*

Exercice 2 *Créer une matrice de ce type*

```
0         0         0         0         0
0   94.4043  107.8118   97.3439         0
0  104.4365  105.6896   88.1222         0
0   90.5010   91.7829   77.9768         0
0         0         0         0         0
```

les éléments de la matrice 3×3 *centrale étant choisit suivant une loi normale d'espérance 100 et d'écart type 10* [2]*.*

Exercice 3 *Créer une matrice carrée de taille 10 avec 1 pour les éléments* $(i, 11-i)$*, pour* $i = 1, \ldots, 10$ *et des 0 ailleurs.*

Exercice 4 *Soit* `M = magic(4)`*. Comment expliquer le résultat de l'instruction* `M(M)` *?*

4 Opérations sur les matrices

4.1 Opérations algébriques

Les opérations algébriques usuelles sont :

[2]Soit X une variable aléatoire et soit (a, b) un couple de nombres réels. Alors $E(aX+b) = aE(X)+b$ et $\sigma(aX+b) = |a|\sigma(X)$.

`+`	addition
`-`	soustraction
`*`	produit
`'`	transposition (complexe conjuguée)
`^`	puissance
`\`	division à gauche
`/`	division à droite.

Toutes ces opérations doivent respecter les règles usuelles de l'algèbre linéaire.

```
>> M = [1,2] * [3;4], P = [1;2] * [3,4], Q = [1,2] * [3,4]
M =
    11
P =
     3     4
     6     8

??? Error using ==> *
Inner matrix dimensions must agree.
```

Quand les opérandes sont une matrice et un scalaire (par exemple `A+1, A-1, A*2 ou A/2`), l'opération est effectuée entre le scalaire et chaque élément de la matrice.

```
>> twos = ones(2) + 1
twos =
     2     2
     2     2
```

4.2 La division

Les opérateurs de division permettent de calculer une solution d'une équation linéaire. Lorsque `A` est inversible, `A\b` calcule la solution de l'équation `Ax=b`. C'est la même chose que `inv(A)*b`, la solution étant calculée sans inversion explicite de la matrice. Matlab ustilise une méthode de Gauss avec pivotage partiel. Si la matrice est singulière, Matlab retourne un message d'avertissement. Par exemple,

```
>> [1 3;2 6]\[1;1]

Warning : Matrix is singular to working precision.
ans =
   Inf
   Inf
```

Lorsque `A` n'est pas carrée, `A\b` retourne une solution calculée au sens des moindres carrés. Par exemple, s'il y a plus de lignes que de colonnes et que `A` est de plein rang, le résultat est une solution du problème de minimisation min $\|Ax - b\|$, pour la norme euclidienne usuelle (voir les exercices 5 et 6). Matlab calcule la solution de l'"équation normale $A^\top Ax = A^\top b$.

L'opérateur de division à droite est défini par `b/A = (A'\b')'`. Si `A` est inversible, c'est la même chose que `b*inv(A)`.

4.3 Opérations éléments par éléments

Matlab utilise un type d'opérations particulières appelées "array operations". Ces opérations concernent `*`,`^`, / et \. Lorsque un de ces opérateurs est précédé d'un point, l'opération est effectuée sur chaque élément de la matrice.

```
>> [1:10].^2
ans =
     1     4     9    16    25    36    49    64    81   100

>> [1:4]./[2:5]
ans =
    0.5000    0.6667    0.7500    0.8000

>> [1 2 3 ; 4 5 6 ; 7 8 9] .* eye(3)
ans =
     1     0     0
     0     5     0
     0     0     9
```

Ces opérateurs sont très utiles pour tracer des graphiques.

Lorsqu'on effectue des calculs avec des nombres complexes, la transposition non complexe conjuguée est obtenue avec l'opérateur `.'`.

```
>> Z1 = [1 ; i]' , Z2 = [1 ; i].'
Z1 =
   1.0000                  0 - 1.0000i
Z2 =
   1.0000                  0 + 1.0000i
```

4.4 Opérateurs relationnels

Les opérateurs relationnels sont

==	égal
~=	différent
>	strictement plus grand
>=	plus grand ou égal
<	strictement plus petit
<=	plus petit ou égal.

Les deux opérandes doivent être de même dimension, sauf si un des deux opérandes est un scalaire. Le résultat est une matrice de même dimension n'ayant que des 0 (faux) et des 1 (vrai), la comparaison étant effectuée élément par élément. Lorqu'un des deux opérandes est un scalaire,la comparaison est effectuée entre chaque élément de la matrice et le scalaire.

```
>> R = randn(1,6), R>0
R =
    1.1892   -0.0376    0.3273    0.1746   -0.1867    0.7258
ans =
     1     0     1     1     0     1
```

4.5 Opérateurs logiques

Les opérateurs logiques sont

&	et
|	ou
$\sim$	non.

Calculons une table logique :

```
>> a = [0;0;1;1]; b = [0;1;0;1]; EtOuDonc = [a,b,a&b,a|b,(~a)|b]
EtOuDonc =
     0     0     0     0     1
     0     1     0     1     1
     1     0     0     1     0
     1     1     1     1     1
```

Une valeur numérique est considérée comme vraie ($= 1$) si elle est non nulle, sinon elle est fausse ($= 0$). Par exemple, ∼`A` renvoie une matrice dont les éléments valent 1 là où `A` a des éléments nuls et 0 ailleurs.

4.6 Le type logical

Le résultat d'une opération relationnelle ou logique est du type `logical` (figure 5). Un tableau de type `logical` peut être utilisée comme tableau d'indice d'une matrice de même dimension.

Dans l'exemple ci-dessous, `R = randn(3,4)` est une matrice de dimension 3×4 dont les valeurs sont choisies selon une loi normale. L'opération `Rpos=R>0` retourne une matrice de même dimension telle que `Rpos(i,j)=1` si `R(i,j)>0` et `Rpos(i,j)=0`. La matrice `Rpos` est utilisée ensuite pour extraire les valeurs positives de `R`.

```
>> R = randn(3,4), Rpos=R>0, R(Rpos)
R =
   -2.1707    0.6145    0.5913   -1.0091
   -0.0592    0.5077   -0.6436   -0.0195
```

```
   -1.0106    1.6924    0.3803   -0.0482
Rpos =
     0     1     1     0
     0     1     0     0
     0     1     1     0
ans =
    0.6145
    0.5077
    1.6924
    0.5913
    0.3803
```

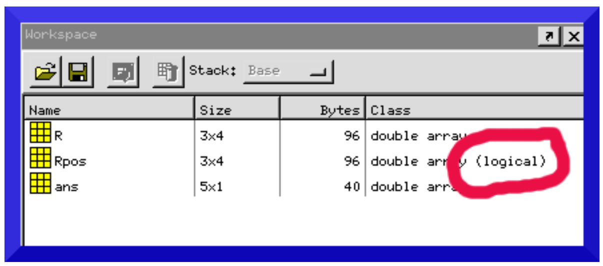

FIG. 5 – Type logical.

La fonction `logical` permet d'affecter le caractère logique à un tableau de données numériques. Cette action permet d'utiliser le tableau comme d'un tableau d'indices (voir l'exercice 11).

```
>> A = [0,1;2 3]; A(logical(eye(2)))
ans =
     0
     3
```

4.7 Exercices

Exercice 5 *On considère les instructions suivantes*

```
>> M = [eye(2);ones(1,2)], b = ones(3,1), x = M\b
```

Montrer que la solution calculée correspond à la valeur minimale de la fonction de deux variables $f(x_1, x_2) = (x_1 - 1)^2 + (x_2 - 1)^2 + (x_1 + x_2 - 1)^2$. *Vérifier qu'on obtient la même valeur avec l'instruction* `(M'*M)\(M'*b)`.

Exercice 6 *Soit* $(x_i, y_i)_{i=1...n}$*,* n *points de* $\mathbb{R}^2$ *tels que* $x_i = i$ *et* $y_i = x_i + \epsilon_i$*, où* ϵ_i *est une variable aléatoire suivant une loi normale centrée et d'écart type* $\sigma = 0.5$*. Calculer la droite d'ajustement linéaire au sens des moindres carrés de* y *en* x*. Rappel : cette droite est définie par l'équation* $y = ax + b$*, où* a *et* b *sont solutions du problème* $\min \sum_i (y_i - ax_i - b)^2$.

Exercice 7 *Créer une matrice M magique d'ordre 3, puis calculer a, b et c tels que*

$$M = a \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} + b \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} + c \begin{pmatrix} -1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

Exercice 8 *Créer un vecteur des puissances de 2 de 0 à 16. Créer une matrice 10×4, contenant les puissances de 0 à 9 des nombres 2, 3, 4 et 5.*

Exercice 9 *Quel est le résultat de l'instruction suivante ?*

```
>> M = [1 2 3; 4 5 6; 7 8 9]; M ~= diag(diag(M))
```

Exercice 10 *Créer une matrice triangulaire supérieur de dimension 10×10 avec seulement des 1 dans sa partie supérieure.*

Exercice 11 *Soit* `A = [1 2 ;3 4]`. *Expliquer les résultats des deux instructions* `A(A)` *et* `A(logical(A))`.

Exercice 12 *Soit X une variable aléatoire de loi normale centrée et réduite. La probabilité que X prenne des valeurs comprises entre -2 et 2 est supérieure à 0.95. Vérifier expérimentalement si le générateur de nombre aléatoire de Matlab satisfait cette assertion.*

5 Fonctions usuelles

5.1 Fonctions scalaires

Ce sont les fonctions mathématiques usuelles.

`sin`	`cos`	`tan`	`acos`	`asin`	`atan`
`sinh`	`cosh`	`tanh`	`acosh`	`asinh`	`atanh`
`exp`	`log`	`log10`	`abs`	`sqrt`	`sign`
`rem`	`round`	`floor`	`ceil`	`fix`	
`real`	`imag`	`conj`	`angle`		

L'argument d'entrée peut être un scalaire, un vecteur ou une matrice. L'argument de sortie est une matrice de même dimension, la fonction étant appliquée sur chacun de ses éléments.

```
>> log(0:5)
Warning: Log of zero.
ans =
      -Inf         0    0.6931    1.0986    1.3863    1.6094
```

5.2 Fonctions vectorielles

Lorsque l'argument d'entrée de ces fonctions est un vecteur, le résultat est un scalaire.

`min`	`max`	`sum`	`prod`	`sort`
`median`	`mean`	`std`	`any`	`all`

```
>> [min(rand(1,10000)), max(rand(1,10000))]
ans =
    0.0001    0.9999
```

Lorsqu'une telle fonction est appliquée à une matrice, elle agit colonne par colonne et retourne un vecteur ligne.

```
>> sum(magic(5))
ans =
    65    65    65    65    65
```

Les fonctions `any` et `all` sont très utiles. Si `x` est un vecteur, `any(x)` renvoie la valeur logique 1 si au moins une composante du vecteur est non nulle, sinon elle renvoie la valeur logique 0 ; `all(x)` renvoie la valeur logique 1 si toutes les composantes de `x` sont non nulles, sinon elle renvoie 0.

5.3 Fonctions matricielles

Ce sont des fonctions usuelles de l'analyse numérique matricielle. L'argument d'entrée est une matrice.

`cond`	conditionnement
`det`	déterminant
`norm`	norme 1, 2, Frobenius, ∞
`rank`	rang
`trace`	trace
`chol`	factorisation de Cholesky
`inv`	inverse
`lu`	factorisation LU
`qr`	factorisation QR
`eig`	valeurs propres et vecteurs propres
`poly`	polynôme caractéristique
`svd`	décomposition en valeurs singulières
`expm`	exponentielle de matrice
`sqrtm`	racine carrée de matrice

Dans l'exemple ci dessous, on calcule les vecteurs propres (`V`) et les valeurs propres (diagonale de `D`) de la matrice $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

```
>> [V,D] = eigs(ones(2,2))
V =
    0.7071   -0.7071
    0.7071    0.7071
D =
     2     0
     0     0
```

5.4 Exercices

Exercice 13 *Comment déterminer l'élément maximal d'une matrice quelconque ?*

Exercice 14 *Comment tester si une matrice carrée est magique avec une seule instruction ?*

Exercice 15 *Avec la fonction* `diag`*, créer la matrice suivante*

$$M = \begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & (0) & \\ & \ddots & \ddots & \ddots & \\ & (0) & \ddots & \ddots & 1 \\ & & & 1 & 2 \end{pmatrix}.$$

Calculer ses valeurs propres et sa décomposition de Cholesky.

6 Aide

6.1 Aide en ligne et recherche par mot clé

La commande `help fonc` permet d'obtenir de l'aide sur la fonction `fonc`. Pour un affichage page par page du texte à l'écran utiliser la l'instruction `more on`. Pour désactiver ce mode d'affichage utiliser `more off`. Quand on ne connaît pas le nom de la fonction à utiliser, la commande `lookfor` permet de faire une recherche par mot clé. Cette recherche se fait dans la première ligne d'aide des fonctions, la ligne H1. Par exemple, `lookfor sort` affiche le nom des fonctions qui contiennent la chaîne "sort" dans leur ligne H1.

```
>> lookfor sort
CPLXPAIR Sort numbers into complex conjugate pairs.
SORT   Sort in ascending order.
SORTROWS Sort rows in ascending order.
EIGFUN Function to return sorted eigenvalues (used in GOALDEMO).
V2SORT Sorts two vectors and then removes missing elements.
SORT   Sort for cell arrays of strings.
```

6.2 Fonctions prédéfinies

Une fonction prédéfinie de Matlab est soit une fonction intégrée (built-in function), soit une fonction écrite en Matlab sauvegardée dans une des bibliothèques. La commande `which fonc` indique si `fonc` est une variable, une fonction intégrée ou bien retourne le chemin d'accès au fichier `fonc.m`.

```
>> which min, which bench
min is a built-in function.
/usr/local/matlabr12/toolbox/matlab/demos/bench.m
```

Nous verrons au paragraphe 10 comment créer ses propres fonctions.

6.3 Contenu de l'espace de travail

La commande `whos` renvoie le nom des variables de l'espace de travail, leur taille, leur occupation en mémoire et leur classe. La fenêtre "workspace" (voir la figure 5) permet aussi d'acceder à ces informations. La commande `clear` permet de libérer la place mémoire occupée par certaines variables.

6.4 Contenu d'un répertoire

La commande `dir` retourne la liste des fichiers du répertoire courant. La commamde `pwd` permet de connaître le nom de ce répertoire. Pour changer de répertoire utiliser `cd` (change directory). La commande `type` permet d'afficher le contenu d'un fichier et `delete` permet de détruire fichier et objet graphique. Pour passer une commande Unix, il suffit de commencer la ligne de commande avec un point d'exclamation. Par exemple,

```
>> !emacs /usr/local/matlabr12/toolbox/matlab/demos/bench.m&
```

ouvre le fichier bench.m avec l'éditeur emacs.

6.5 Exercices

Exercice 16 *Rechercher une fonction Matlab permettant de générer toutes les permutations de k nombres pris parmi n.*

7 Textes et chaînes de caractères

7.1 Chaine = vecteur de caractères

Une chaînes de caractères est encadrée par deux apostrophes.

```
>> c = 'La variable c contient une chaine de caracteres'
```

Les textes sont représentés par des tableaux de caractères. La variable `c` est un vecteur de 48 caractères. La plupart des opérations vues dans la section 3 s'appliquent. Les instructions

```
>> c = [c,'.'], c(length(c):-1:1)
c =
La variable c contient une chaine de caracteres.
ans =
.seretcarac ed eniahc enu tneitnoc c elbairav aL
```

Certaines fonctions permettent de transformer chaines en valeurs numériques et vice versa (voir la section 12).

```
>> two = 2, second = num2str(two)
two =
     2
second =
2
```

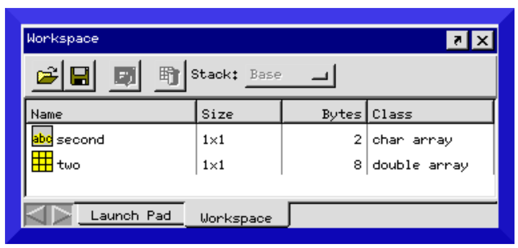

FIG. 6 – Type caractère.

7.2 Affichage, lecture et évaluation

Certaines fonctions utilisent une chaine de caractère comme argument d'entrée. Par exemple, `disp` affiche un message à l'écran, `error` termine l'exécution d'une fonction et retourne un message, `input` affiche un message et attend une entrée au clavier.

Les fonctions `eval` et `feval` permettent d'exécuter une instruction à partir d'une chaîne de caractères. Par exemple, l'instruction

```
>> for i=2:5, eval(['I',num2str(i),'=eye(i)']), end
```

retourne les matrices identités de dimension 2, 3, 4 et 5 dans des variables `I2`, ..., `I5`. La fonction `feval` permet de passer des arguments d'entrée. Un exemple d'utilisation est présenté à la section 10.

8 Fichiers script

8.1 Exemple

Un script est un fichier contenant une suite d'instructions qui sont exécutées séquentiellement à l'appel du fichier. L'utilisation d'un script est très courante, cela evite d'avoir à saisir plusieurs fois de longues suites d'instructions. Pour éditer un script, on peut utiliser son éditeur préféré ou bien l'éditeur par défaut de Matlab avec la commande `edit`. Le fichier doit être sauvegardé avec l'extension `.m`. L'exemple suivant est un fichier script sauvegardé dans le répertoire courant sous le nom `durer.m`. Les lignes commençant par le caractère `%` sont des commentaires.

```
% durer.m  version du 23 septembre 2002
% calculs dans l'espace des matrices magiques d'ordre 4

% base de l'espace
C1 = [1 0 0 0; 0 0 0 1; 0 1 0 0; 0 0 1 0]
C2 = [0 1 0 0; 0 0 1 0; 1 0 0 0; 0 0 0 1]
C3 = [0 0 0 1; 0 1 0 0; 1 0 0 0; 0 0 1 0]
C4 = [0 1 0 0; 0 0 0 1; 0 0 1 0; 1 0 0 0]
C5 = C4'
C6 = C3'
C7 = C1'

% matrice de la gravure de Dürer
D = [16 3 2 13;5 10 11 8 ; 9 6 7 12; 4 15 14 1]

% décomposition de D sur la famille {C1, ..., C7}
C = [C1(:),C2(:),C3(:),C4(:),C5(:),C6(:),C7(:)]
x = C\D(:)
```

Pour exécuter les instructions de ce fichier, entrer le nom du fichier, sans l'extension, dans la fenêtre des commandes de Matlab.

```
>> durer
```

8.2 Chemin d'accés

Pour que Matlab associe correctement la commande `durer` avec le fichier `durer.m`, il faut que ce fichier soit dans le répertoire courant (le nom du répertoire courant est obtenu avec la commande `pwd`) ou bien dans un des répertoires de la liste PATH obtenue avec la commande `path`. Si la réponse de Matlab est

```
??? Undefined function or variable 'durer'.
```

cela signifie que Matlab ne trouve pas le fichier `durer.m`. Dans ce cas, il faut en premier s'assurer que le fichier existe dans un des répertoires. Si le contenu du fichier est visible dans la fenêtre de l'éditeur, soit il n'a pas été sauvegardé, soit il a été sauvegardé dans un répertoire différent du répertoire courant. Pour rechercher un fichier on peut utiliser la commande Unix suivante (à saisir dans une fenêtre de terminal) :

```
find ~ -name durer.m
```

On peut aussi saisir la commande dans la fenêtre de Matlab en la faisant précéder d'un `!` :

```
>> ! find ~ -name durer.m
/users/math/armand/ens/matlab/doc/durer.m
```

Unix va retourner la liste des répertoires qui contiennent un fichier `durer.m`. Si le fichier n'existe pas, la liste est vide. Si le fichier existe, c'est que le répertoire courant de Matlab ne correspond pas au répertoire qui contient le fichier. Il suffit alors de changer de répertoire courant avec la commande `cd` de Matlab. Il est aussi possible de modifier la liste PATH, avec la commande `path` elle-même ou bien en sélectionnant **Set Path** dans le menu **File** du bureau Matlab.

8.3 Exercices

Exercice 17 *Reproduire le script* `durer.m` *et le faire exécuter. Quelles sont les valeurs des coefficients* α_i *tels que* $D = \sum_{i=1}^{7} \alpha_i C_i$ *? Vérifier que* D *est bien une matrice magique. Vérifier que la somme des nombres des quatres cases centrales est aussi égale à 34.*

9 Boucles et contrôles

9.1 Branchement conditionnel (`If...then...else`)

L'instruction `if` permet d'exécuter un bloc d'instructions en fonction de la valeur logique d'une expression. Sa syntaxe est :

```
if expression
  instructions
end
```

Le groupe d'instructions est exécuté si seulement si l'expression est vraie. Il est possible d'utiliser des branchements multiples.

```
if length(x) ~= 1
  'non scalaire'
elseif isnan(x)
  'NaN'
elseif x > 0
  'positif'
elseif x < 0
  'negatif'
else
  'nul'
end
```

Si `A` et `B` sont deux matrices, il est possible d'utiliser l'expression `A == B` comme test logique. Le bloc d'instructions venant à la suite du `if A == B` est exécuté si et seulement si les deux matrices sont égales. Attention cependant à ce test d'égalité, car il renvoie un message d'erreur si les deux matrices n'ont pas les mêmes dimensions. Il est préférable d'utiliser la fonction `isequal` pour tester l'égalité entre plusieurs matrices.

9.2 Branchement multiple (Switch)

C'est une instruction de branchement conditionnel. Sa syntaxe est :

```
switch expression
  case valeur 1
    instructions 1
  case valeur 2
    instructions 2
  ...
  otherwise
    instructions
  end
```

où `expression` doit être un scalaire ou une chaîne de caractère. Dans le cas scalaire, `expression` est comparée successivement avec `valeur 1`, `valeur 2`, etc. Dès que le premier test `expression==valeur` retourne la valeur vraie, le bloc d'instruction qui suit est exécuté. Si aucun test n'est validé, le bloc qui suit `otherwise` est exécuté.

```
switch mod(x,3)
 case 0
  'multiple de 3'
 case 1
  'x = 3 [1]'
 case 2
  'x = 3 [2]'
 otherwise
  'autre'
end
```

L'exemple ci-dessous présente un cas où `expression` est une chaîne de caractères. Noter l'utilisation des points de suspension pour continuer une instruction sur la ligne suivante.

```
switch lower(car)
 case {'a','e','i','o','u','y'}
  disp([car ' est une voyelle'])
 case {'b','c','d','f','g','h','j','k','l','m','n','p',...
       'q','r','s','t','v','w','x','z'}
  disp([car ' est une consonne'])
 case {'0','1','2','3','4','5','6','7','8','9'}
  disp([car ' est un chiffre'])
 otherwise
  'autre'
end
```

9.3 Boucle finie (`For`)

Une boucle `for` permet de répéter un certain nombre de fois un bloc d'instructions. Sa syntaxe est :

```
for variable = expression
  instructions
end
```

L'exemple suivant permet d'éditer une table de multiplication.

```
n = 10; t = 1:n;
T = [];
for i = 1:n
  T = [T;t*i];
end
T
```

Plusieurs boucles peuvent être imbriquées, comme dans l'exemple qui suit (calcul d'une matrice de Hilbert).

```
H = []; n = 10; m=n;
for i = 1:n
  for j = 1:m
    H(i,j) = 1/(i+j-1);
  end
end
H
```

La variable de contrôle peut être un vecteur et l'expression une matrice. Par exemple,

```
>> for u = eye(4), u, end
```

énumère les vecteurs de la base canonique de l'espace euclidien de dimension 4.

9.4 Boucle infinie (`While`)

Une boucle `while` permet d'exécuter un bloc d'instructions tant qu'une expression logique est vraie. Sa syntaxe est :

```
while expression
  instructions
end
```

L'exemple suivant permet de calculer le epsilon machine (= plus petite distance entre 1.0 et le nombre flottant suivant).

```
e = 1;
while 1+e > 1
  e = e/2
end
epsilon = 2*e
```

Attention à modifier la valeur de la variable de contrôle dans le bloc d'instructions, sinon la boucle n'a pas de fin. Pour forcer l'interruption d'une boucle en cours d'exécution, utiliser la combinaison de touches Ctrl–C.

L'instruction `break` permet de terminer l'exécution d'une boucle `while` ou `for`. Elle peut être utile lorsqu'on veut placer le test de contrôle ailleurs qu'en début de boucle. L'instruction `continue` permet de passer à la prochaine itération de la boucle `while` ou `for` qui la contient, en sautant les instructions qui lui font suite dans le corps de cette boucle.

Des habitudes de programmation nous poussent souvent à utiliser une ou plusieurs boucles `for` ou `while` lorqu'il s'agit d'effectuer une opération sur chaque élément d'un tableau. Avec Matlab il faut toujours avoir présent à l'esprit que la plupart des opérations peuvent agir sur tous les éléments d'une matrice ou bien sur tous ses vecteurs colonnes simultanément. Il ne faut pas hésiter à remplacer une boucle par quelques instructions simples à chaque fois que cela est possible, il en résultera un gain en rapidité d'exécution. Prenons l'exemple du calcul d'une table de multiplication. Dans l'exemple précédent, la boucle `for` peut avantageusement être remplacée par l'instruction

```
n = 10; t = 1:n;
T = t'*t;
```

Avec `n = 10` on obtient un gain en rapidité d'exécution d'environ 3, avec `n = 500` le gain est supérieur à 350.

9.5 Exercices

Exercice 18 *Avec les fonctions de chronométrage* `tic` *et* `toc`*, comparer les temps de calcul nécessaires pour créer une table de multiplication selon les deux méthodes exposées ci-dessus.*

Exercice 19 *Recopier dans un fichier appelé* `erathostene.m` *le script suivant. Faire executer ce script et expliquer l'algorithme.*

```
n = 49;
T = ones(1,n);
for k = 2:sqrt(n)
  if T(k)==1
    T(2*k:k:n)=0;
  end
end
find(T)
```

Exercice 20

On considère les deux suites (x_n) et (y_n) définies par :

$$x_n = \begin{cases} 1 & si\ 0 \leq n \leq 1, \\ x_{n-1} + x_{n-2} & si\ n \geq 2 \end{cases}$$

et

$$y_n = \frac{5+\sqrt{5}}{10}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{5-\sqrt{5}}{10}\left(\frac{1-\sqrt{5}}{2}\right)^n, n \geq 0.$$

1. *Montrer que pour tout $n \geq 0$, $x_n = y_n$.*
2. *Ecrire un programme de calcul des valeurs x_n et y_n pour $0 \leq n \leq 25$ avec les formules indiquées. Ces valeurs seront mémorisées dans deux vecteurs* `x` *et* `y`*.*
3. *Que peut-on constater ?*

10 Les fonctions

10.1 Déclaration, arguments

Une fonction est un fichier texte dont la première ligne contient un en-tête de la forme suivante :

```
function [s1, s2, ...] = nom_fonction (e1, e2, ...)
```

Le reste du fichier contient la suite des instructions qui sont exécutées lors de l'appel de la fonction. Les variables `e1`, `e2`, ..., sont les arguments d'entrée de la fonction. Les variables `s1`, `s2`, ..., sont les arguments de sortie.

Il existe des différences importantes entre un script et une fonction. Une fonction possède son propre espace de travail. Une variable utilisée à l'intérieur du corps de la fonction possède un caractère local, elle n'existe que pendant l'exécution de la fonction, sauf si elle appartient à la liste des arguments de sortie ou bien si elle a été préalablement déclarée comme variable globale (mot-clé `global`). De même, une variable de l'espace de travail courant, ne peut pas être référencée depuis une fonction, sauf si elle est passée en argument d'entrée ou bien si elle est globale.

Voici un exemple très simple. La fonction `caracter` retourne la valeur de la fonction caractéristique de l'intervalle $[0, 1]$ ($f(x) = 1$ si $x \in [0, 1]$, $= 0$ sinon). Si l'argument n'est pas de type numérique elle renvoie un message d'erreur.

```
function [y] = caracter(x)
%CARACTER  Fonction caracteristique de [0,1]
%   CARACTER(X) retourne la valeur de la fonction caracteristique
%   de [0,1] evaluee sur les elements de X

if isnumeric(x)
  y = x>=0 & x <=1;
else
  error('Argument non numerique')
end
```

Ce fichier est sauvegardé dans le répertoire courant sous le nom `caracter.m`. L'en-tête contient le mot-clé `function`, une seule variable de sortie, le nom de la fonction suivi de l'unique argument d'entrée. Les trois lignes de commentaires qui suivent constituent les lignes d'aide qui seront affichées avec la commande `help caracter`. Les lignes d'aide sont les premières lignes de commentaires contiguës situées après l'en-tête, les autres lignes de commentaire sont ignorées par le `help`. La première ligne de commentaire constitue la ligne H1. C'est dans cette ligne que la fonction `lookfor` recherche un mot-clé. Par exemple, `lookfor car`, renvoie

```
CARACTER  Fonction caracteristique de [0,1]
CART2POL Transform Cartesian to polar coordinates.
CART2SPH Transform Cartesian to spherical coordinates.
...
```

Ensuite viennent les instructions qui constituent la partie exécutable de la fonction.

L'appel de cette fonction peut se faire dans le corps d'une autre fonction, dans un script ou directement dans la fenêtre des commandes, sous réserve que le chemin d'accés au fichier `caracter.m` soit le répertoire courant ou bien qu'il soit indiqué dans le `PATH`. L'association entre l'appel à une fonction et le fichier où est sauvegardé le texte de la fonction, obéit aux mêmes règles que pour un fichier script (voir le paragraphe 8).

```
>> a = [0, 3, .4, -5, 1/2, pi], b = caracter(a)
```

L'instruction ci-dessus renvoie

```
b =
     1     0     1     0     1     0
```

10.2 Exemple : résolution d'une équation non linéaire

Dans l'exemple ci-dessous, nous montrons comment passer une fonction comme argument d'entrée d'une autre fonction. Il s'agit de programmer une méthode de résolution numérique d'une équation non linéaire

$$g(x) = 0,$$

où $g : \mathbb{R}^n \to \mathbb{R}^n$. La méthode que nous voulons programmer et tester est due à Barzilai et Borwein. C'est une méthode itérative qui génère une suite de vecteurs $\{x_k\}_{k\geq 1}$ par la récurrence

$$x_{k+1} = x_k - \alpha_k g(x_k),$$

où

$$\alpha_k = \begin{cases} 1 & \text{si } k = 1, \\ \frac{\|x_k - x_{k-1}\|^2}{\langle x_k - x_{k-1}, g(x_k) - g(x_{k-1})\rangle} & \text{si } k > 1. \end{cases}$$

ALGORITHME BARZILAI-BORWEIN

Entrée : ε, x_1. **Sortie** : x_*, tel que $\|g(x_*)\| \leq \varepsilon$.

1. Poser $k = 1$.
2. Calculer $g(x_k)$.
3. Si $\|g(x_k)\| \leq \varepsilon$, stop.
4. Calculer α_k.
5. $x_{k+1} = x_k - \alpha_k g(x_k)$.
6. $k = k + 1$ et retour en 2.

```
function  x = abb(fonc,x,epsilon,maxiter)
%ABB  Resolution d'une equation non lineaire, methode de Barzilai-Borwein
%
% Entree : fonc    = fonction test
%          x       = point de départ
%          epsilon = tolérance du test d'arrêt, valeur par défaut 1e-8
%          maxiter = nombre maximum d'itérations, valeur par défaut 100
%
% Sortie : x       = solution

% vérification du nombre d'arguments d'entrée et valeurs par défaut
if nargin < 2, error('Nombre d''arguments d''entree trop petit'); end
if nargin < 3, epsilon = 1e-8; end
if nargin < 4, maxiter = 100; end

iter = 1; % compteur itération

while iter <= maxiter
  g = feval(fonc,x);
  if norm(g) <= epsilon, break, end
  if iter == 1, a = 1; else a = norm(x-x_)^2/dot(x-x_,g-g_); end
  x_ = x;
  g_ = g;
  x = x-a*g;
  iter = iter + 1;
end
if iter > maxiter, warning('nombre maximum d''iterations atteint'), end
```

La fonction `abb` est sauvegardée dans un fichier appelé `abb.m`. Un contrôle du nombre d'arguments d'entrée est effectué grâce à la variable `nargin`. Cette fonction indique le nombre d'arguments d'entrée présents à l'appel de la fonction `abb`. Si par exemple les valeurs des deux premiers arguments entrée sont données, les variables `epsilon` et `maxiter` sont initialisées à leur valeurs par défaut. Supposons que l'on veuille tester la méthode sur la fonction

$$x \in \mathbb{R}^n \to (e^{x_1} - 1, \ldots, e^{x_n} - 1).$$

Il faut d'abord créer une nouvelle fonction Matlab qui va calculer les valeurs de la fonction définie ci-dessus.

```
function g = expon(x)
%EXPON  Fonction test

if isnumeric(x), g = exp(x)-1; else  error('Argument non numerique'), end
```

Supposons que l'on fasse un essai avec $n = 10$, un point de départ x_1 choisi au hasard (dans un voisinage de 0) et une précision de $10^{-10} \times \|g(x_1)\|$.

```
>> x = rand(1,10); x = abb(@expon,x,1e-10*norm(expon(x)))
```

Le passage de la fonction comme argument d'entrée de `abb` se fait en préfixant le nom de la fonction `expon` avec le caractère `@`. Dans la fonction `abb`, l'appel de `expon` se fait grâce à la fonction `feval`. L'instruction `feval(fonc,x)` est équivalente à l'instruction `expon(x)`.

Supposons maintenant que l'on veuille faire un test avec la fonction

$$x \in \mathbb{R}^n \to (x_1, 2x_2, \ldots, nx_n).$$

On crée une nouvelle fonction, appelée `prodn.m`.

```
function g = prodn(x)
%PRODN  Fonction test

if isnumeric(x),
  [n,m] = size(x);
  if n==1
    g = (1:m).*x;
  elseif m==1
    g = (1:n)'.*x;
  else
    error('L''argument n''est pas un vecteur')
  end
else
  error('Argument non numerique')
end
```

Le test se fait comme précédemment :

```
>> x = rand(1,10); x = abb(@prodn,x,1e-10*norm(prodn(x)))
```

10.3 Exercices

Exercice 21 *On considère la fonction Matlab suivante :*

```
function s = fonc(p,a)
%FONC
%    Entree : p = vecteur
%             a = matrice
%    Sortie : s = matrice de dimension size(a)

s = p(1)*ones(size(a));
for k = 2:length(p)
   s = s.*a+p(k);
end
```

1. *Créer cette fonction.*
2. *Quel est le résultat de l'appel* `fonc([1,-4,3],5)` *? Expliquer.*
3. *Même question avec l'instruction* `fonc([1,-4,3],[0,1;2,3])`.
4. *Sachant que* `p` *représente le vecteur des coefficients d'un polynôme, que calcule la fonction* `fonc` *et quel est l'algorithme utilisé ?*

Exercice 22 *On considère l'équation non linéaire*

$$g(x) = 0,$$

où $g : \mathbb{R}^n \to \mathbb{R}^n$ est une fonction différentiable, dont la jacobienne $g'(x) := (\frac{\partial g_i}{\partial x_j})_{i,j=1\dots n}$ est supposée inversible dans un voisinage de la solution. La méthode de Newton est une méthode itérative qui génère une suite de vecteurs $\{x_k\}$ avec la récurrence

$$x_{k+1} = x_k - (g'(x_k))^{-1} g(x_k).$$

Ecrire une fonction Matlab similaire à la fonction `abb` *en utilisant l'algorithme de Newton. Comparer les méthodes de Barzilai-Borwein et de Newton en comptant le nombre d'itérations pour chaque exemple* `expon` *et* `prodn`.

11 Graphiques

Matlab possède un vaste ensemble de fonctionnalités graphiques. Nous ne présenterons seulement que quelques principes de base qui serviront à visualiser courbes et surfaces. Que ce soit pour tracer le graphe d'une fonction dans le plan ou dans l'espace, la technique peut se décomposer en trois temps.

1. Discrétiser le domaine de représentation.
2. Evaluer la fonction en chaque point de ce domaine discrétisé.
3. Exécuter l'instruction graphique avec les données précédentes.

11.1 Graphiques 2d

La fonction `plot` permet de tracer des graphes de fonctions en deux dimensions. Prenons un exemple avec la fonction $x \to x\sin(x)$.

```
n = 5;
x = -n*pi:2*n*pi/200:n*pi;
y = x.*sin(x);
plot(x,y)
```

Noter l'utilisation de l'opérateur `.*` pour évaluer la fonction en chaque point de l'intervalle discrétisé. L'instruction `plot(x,y)` dessine une courbe passant par les points `(x(i),y(i))`. Les vecteurs `x` et `y` doivent avoir la même longueur. On peut ajouter des légendes sur le graphique.

```
title('Courbe de la fonction y = x sin(x)')
xlabel(['x = -',num2str(n),'\pi:',num2str(n),'\pi'])
ylabel('y')
```

Le résultat est sur la figure 7.

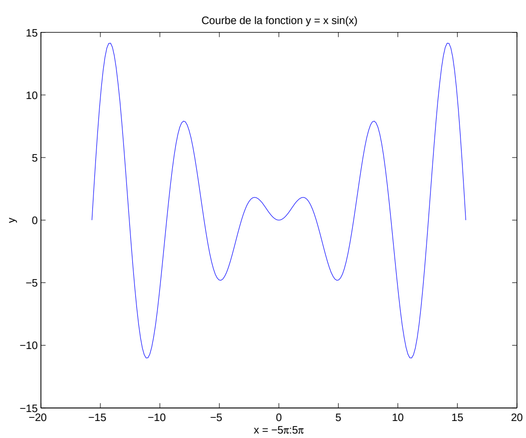

FIG. 7 – Graphe de $x \to x\sin(x)$.

On peut dessiner plusieurs courbes sur un même graphique.

```
figure(2)
n = 5;
x = -n*pi:2*n*pi/200:n*pi;
y = x .* sin(x);
plot(x,y,x,x,x,-x)
legend('y = x*sin(x)','y = x','y = -x')
```

L'instruction `figure(2)` ouvre une deuxième fenêtre graphique. Le résultat est sur la figure 8.

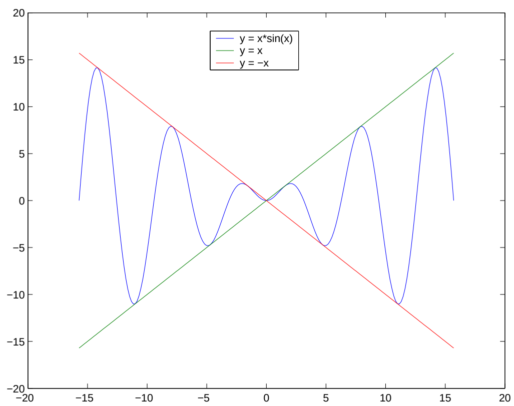

FIG. 8 – Graphes de $x \to x\sin(x)$, $x \to x$ et $x \to -x$.

Une nouvelle instruction graphique va en principe effacer le graphique courant et le remplacer par un nouveau tracé. L'instruction `hold on` permet d'ajouter un ou plusieurs tracés à un graphique déjà existant. Les tracés sont ajoutés jusqu'au retour du mode par défaut avec `hold off`. Supposons qu'on veuille visualiser les zéros de la courbe sur le premier graphique.

```
figure(1)
hold on
plot(-n*pi:pi:n*pi,zeros(1,2*n+1),'or')
hold off
grid
```

L'instruction `figure(1)` permet de choisir la première fenêtre graphique comme fenêtre courante. On peut choisir des styles variés pour réaliser les tracés, ici ce sont des ronds rouges (voir la figure 9). Une grille est ajoutée avec l'instruction `grid`.

11.2 Graphiques 3d

Pour tracer le graphe d'une fonction $z = f(x, y)$ il faut d'abord générer une grille de points, évaluer la fonction en chaque point, puis tracer le graphe.

```
x = -1.5:.1:1.5;
y = x;
[X,Y] = meshgrid(x,y);
Z = sin(3*X.^2+2*Y.^2)./(X.^2+Y.^2+eps);
mesh(Z)
title('Cowboy Hat')
```

Le tracé est en figure 10. La fonction `meshgrid` génère une grille de points représentée par les matrices `X` et `Y`. La matrice `X` possède `length(y)` lignes, chaque ligne est une

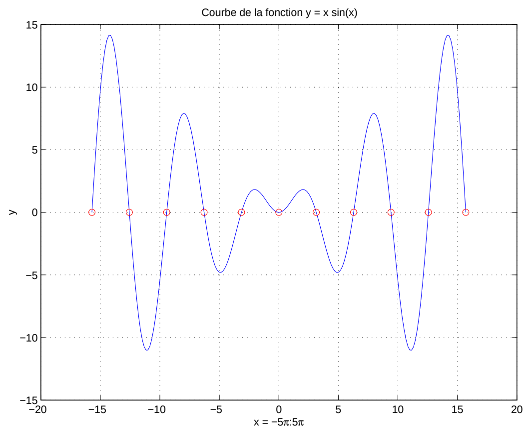

FIG. 9 – Graphes de $x \to x\sin(x)$ et zéros de la fonction.

copie de `x`. La matrice `Y` possède `length(x)` colonnes, chaque colonne est une copie de `y`. Un couple `(X(i),Y(i))` repère ainsi un point de la grille. Pour évaluer la fonction sur la grille, on utilise les opérateurs élément par élément. Le tracé d'une surface peut se faire sous des formes diverses et variées (voir la figure 11).

11.3 Exercices

Exercice 23 *Dessiner le nuage de points* $(x_i, y_i)_{i=1\ldots n}$ *de l'exercice 6 et la droite d'ajustement linéaire.*

Exercice 24 *On considère la fonction* $f : \mathbb{R}^2 \to \mathbb{R}^3$*, définie par*

$$f(x, y) = (x^2, -y^2, \sin(xy)).$$

Dessiner la surface de $\mathbb{R}^3$*, image du rectangle* $[0, 5] \times [-1.5, 1.5]$ *par la fonction* f*.*

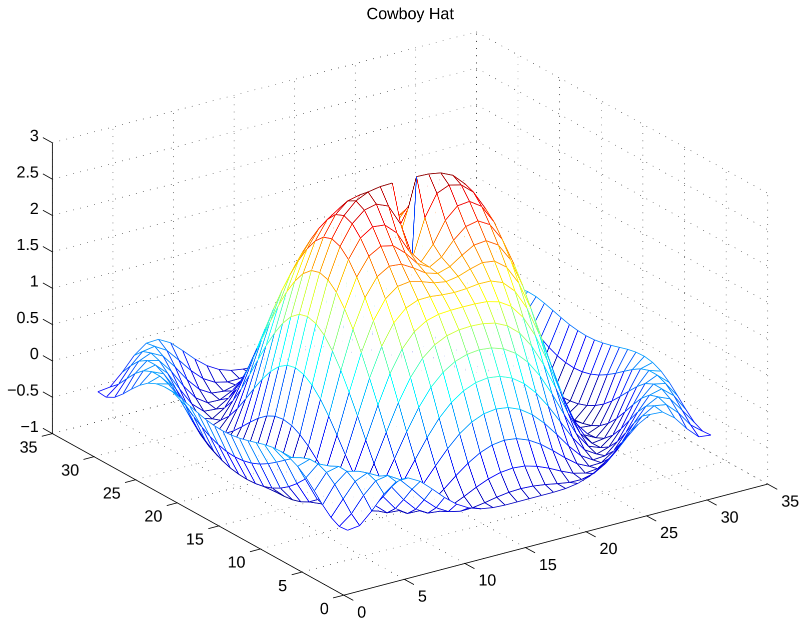

Fig. 10 – $f(x,y) = \frac{\sin(3x^2+2y^2)}{x^2+y^2}$

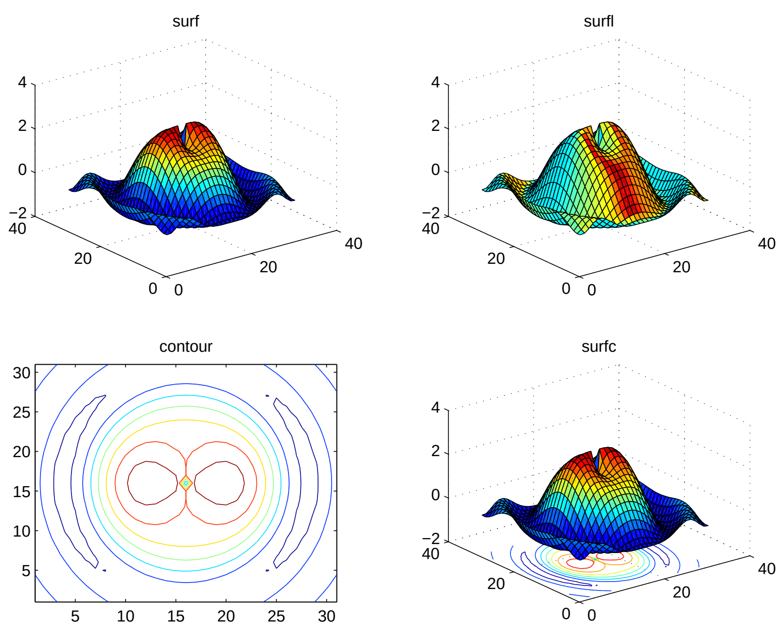

Fig. 11 – subplot

12 Références

Cette section regroupe une partie des commandes et fonctions de Matlab.

Commandes générales

Aides sur les commandes et fonctions	
`demo`	Programme de démonstration
`help`	Aide en ligne
`info`	Informations sur Matlab et The MatWorks
`lookfor`	Recherche par mot clé dans les textes d'aide en ligne
`path`	Contrôle du chemin de recherche des commandes
`type`	Affiche le contenu d'un fichier M
`what`	Liste des fichiers M, MAT et MEX du répertoire courant
`which`	Localise fonctions et fichiers

Variables et espace de travail	
`clear`	Efface variables et fonctions de l'espace de travail
`disp`	Affiche texte et matrice
`length`	Longueur d'un vecteur
`load`	Charge le contenu de variables sauvegardées sur disque
`pack`	Défragmente la mémoire
`save`	Sauve le contenu de variables sur disque
`size`	Dimensions d'une matrice
`who`	Affiche la liste des variables de l'espace de travail
`whos`	Identique à `who` avec plus de détails

Commandes système	
`cd`	Change de répertoire
`delete`	Détruit un fichier
`diary`	Sauvegarde la session en cours dans un fichier texte
`dir`	Affiche le contenu du répertoire courant
`getenv`	Retourne la valeur d'une variable d'environement
`unix`	Exécute une commande système et retourne le résultat
`!`	Exécute une commande système

Contrôle de la fenêtre des commandes	
`clc`	Efface la fenêtre des commandes
`echo`	Renvoie les commandes d'un fichier M
`format`	Contrôle le format d'affichage
`home`	Place le curseur en haut de la fenêtre des commandes
`more`	Affichage page par page dans la fenêtre des commandes

Démarrer et quitter Matlab	
`exit`	Quitte Matlab
`matlabrc`	Script de démarrage
`quit`	Exécute le script `finish.m` et quitte Matlab

Opérateurs et caractères spéciaux

Opérateurs et caractères spéciaux	
`+`	Plus
`-`	Moins
`*`	Produit
`.*`	Produit élément par élément
`^`	Puissance de matrice
`.^`	Puissance élément par élément
`kron`	Produit tensoriel de Kronecker
\	Backslash ou division à gauche
.\	Division à gauche élément par élément
`/`	Slash ou division à droite
`./`	Division à droite élément par élément
`:`	Indices ou génération de vecteur
`( )`	Parenthèses, arguments d'entrée de fonction
`[ ]`	Crochets, arguments de sortie de fonction
`.`	Point décimal
`..`	Répertoire parent
`...`	Continue l'instruction sur la ligne suivante
`, ;`	Séparateurs
`%`	Commentaires
`!`	Commande système
`'`	Transposition (complexe conjugué), chaine de caractère
`.'`	Transposition non complexe
`=`	Affectation
`==`	Egalité
`< >`	Opérateurs relationnels
`&`	ET logique
`|`	OU logique
~	NON logique
`xor`	OU EXCLUSIF logique

Fonctions logiques	
`all`	Vrai si tous les éléments d'un vecteur sont non nuls
`any`	Vrai si au moins un élément d'un vecteur est non nul
`exist`	Vérifie si une variable ou une fonction existe
`find`	Retourne les indices des élélements non nuls
`isempty`	Vrai pour matrice vide
`isinf`	Vrai pour élément infini
`isnan`	Vrai pour NaN (Not-a-Number)
`issparse`	Vrai pour matrice creuse
`isstr`	Vrai pour chaine de caractère

Programmation

Fonctions logiques	
`eval`	Exécute une chaine de caractères comme instruction
`feval`	Appel de function
`function`	Crée une nouvelle fonction
`global`	Définition d'une variable globale
`narchk`	Valide le nombre d'arguments d'entrée

Instructions de contrôle	
`break`	Termine une boucle
`else`	sinon, utilisé avec `if`
`elseif`	sinon si, utilisé avec `if`
`end`	Termine `for`, `if` et `while`
`error`	Retourne un message d'erreur et termine une fonction
`for`	Répétition
`if`	Instruction conditionnelle
`return`	Retour à la fonction appelante
`while`	Boucle tant que

Instructions interactives	
`input`	Attente d'une entrée au clavier
`keyboard`	Donne la main à l'utilisateur juqu'à un `return`
`menu`	Crée un menu
`pause`	Attente de réponse

Création de matrice

Matrices usuelles	
`eye`	Matrice identité
`linspace`	Génère des vecteurs espacés arithmétiquement
`logspace`	Génère des vecteur espacés logarithmiquement
`meshgrid`	Génère une grille pour les graphes en 3D
`ones`	Matrice de 1
`rand`	Générateur aléatoire selon une loi uniforme
`randn`	Générateur aléatoire selon une loi normale
`zeros`	Matrice de 0

Variables et constantes prédéfinies	
`ans`	Variable d'affectation par défaut
`computer`	Type d'ordinateur et de système
`eps`	Epsilon machine
`i, j`	Unités imaginaires
`inf`	Infini
`NaN`	Not-a-number
`nargin`	Nombre d'arguments d'entrée d'une fonction
`nargout`	Nombre d'arguments de sortie
`pi`	3.14159265358979
`realmax`	Plus grand nombre flottant
`realmin`	Plus petit nombre flottant

Horloges	
`clock`	Date et heure sous forme de vecteur
`cputime`	Temps CPU
`date`	La date d'aujourd'hui
`etime`	Mesure un intervalle de temps
`tic, toc`	Chronomètre

Manipulation de matrice	
`diag`	Crée ou extrait une matrice diagonale
`fliplr`	Permutte les colonnes d'une matrice
`flipud`	Permutte les lignes d'une matrice
`reshape`	Modifie la taille d'une matrice
`rot90`	Rotation de 90° dans le sens trigonométrique
`tril`	Extrait une matrice triangualaire inférieure
`triu`	Extrait une matrice triangualaire supérieure

Matrices spéciales	
`compan`	Matrice compagnon
`hadamard`	Matrice de Hadamard
`hankel`	Matrice de Hankel
`hilb`	Matrice de Hilbert
`invhilb`	Inverse d'une matrice de Hilbert
`magic`	Carré magique
`pascal`	Matrice de Pascal
`rosser`	Matrice test pour le calcul de valeurs propres
`toeplitz`	matrice de Toeplitz
`vander`	matrice de Vandermonde
`wilkinson`	Matrice test pour le calcul de valeurs propres

Fonctions élémentaires

Fonctions mathématiques élémentaires	
`abs`	Valeur absolue
`acos`	Cosinus inverse
`acosh`	Cosinus hyperbolique inverse
`angle`	Argument
`asin`	Sinus inverse
`asinh`	Sinus hyperbolique inverse
`atan`	Tangente inverse
`atanh`	Tangente hyperbolique inverse
`ceil`	Plafond
`conj`	Conjugué
`cos`	Cosinus
`cosh`	Cosinus hyperbolique
`exp`	Exponentielle
`fix`	Arrondi vers 0
`floor`	Plancher
`imag`	Partie imaginaire
`log`	Logarithme népérien
`log10`	Logarithme décimal
`real`	Partie réelle
`rem`	Reste de la division euclidienne
`round`	Arrondi vers l'entier le plus proche
`sign`	Signe
`sinh`	Sinus hyperbolique
`sqrt`	Racine carrée
`tan`	Tangente
`tanh`	Tangente hyperbolique

Analyse numérique matricielle

Analyse	
`cond`	Conditionnement
`det`	Déterminant
`norm`	Norme
`null`	Base orthonormale du noyau
`orth`	Base orthonormale de l'image
`rank`	Rang (= dimension de l'image)
`trace`	Somme des éléments diagonaux

Equations linéaires	
`chol`	Factorisation de Cholesky
`inv`	Inverse
`lscov`	Moindres carrés avec matrice de covariance donnée
`lu`	Factorisation LU
`nnls`	Moindres carrés avec contrainte de positivité
`pinv`	Pseudoinverse
\ /	Résolution d'équations linéaires

Valeurs propres	
`eig`	Valeurs propres et vecteurs propres
`poly`	Polynôme caractéristique

Analyse de données

Opérations vectorielles élémentaires	
`cumprod`	Produits cumulés des composantes
`cumsum`	Sommes cumulées des composantes
`max`	Plus grande composante
`mean`	Moyenne des composantes
`median`	Médiane des composantes
`min`	Plus petite composante
`prod`	Produit des composantes
`sort`	Tri des composantes
`std`	Ecart type des composantes
`sum`	Somme des composantes
`trapz`	Intégration numérique, méthode des trapèzes

Dérivées approchées	
`del2`	Approximation du Laplacien
`diff`	Différences finies
`gradient`	Approximation du gradient

Corrélation	
`corrcoef`	Coefficients de corrélation
`cov`	Matrice de covariance

Polynômes

Polynômes	
`conv`	Produit de polynômes
`deconv`	Division de polynômes
`poly`	Définit un polynôme à partir des racines
`polyder`	Polynôme dérivé
`polyfit`	Interpolation polynomiale
`polyval`	Valeur d'un polynôme
`polyvalm`	Valeur d'un polynôme avec argument matriciel
`residue`	Décomposition de fraction rationnelle
`roots`	Racines d'un polynôme

Méthodes numériques

Méthodes numériques	
`fmin`	Minimise une fonction d'une variable
`fmins`	Minimise une fonction de plusieurs variables
`fplot`	Graphe de fonction
`fzero`	Zéro d'une fonction d'une variable
`ode43`	Intégration e.d.o.
`ode45`	Intégration e.d.o., méthodes d'ordre supérieur
`quad`	Intégration numérique
`quad8`	Intégration numérique, méthodes d'ordre supérieur

Chaines de caractères

Commandes générales	
`abs`	Convertit une chaine en une valeur numérique
`eval`	Exécute une chaine de caractères comme instruction
`isstr`	Vrai pour une chaine de caractères
`setstr`	Convertit une valeur numérique en chaine de caractères
`str2mat`	Crée des matrices de chaines de caractères

Comparaisons	
`lower`	Conversion en minuscules
`strcmp`	Comparaison de chaines de caractères
`upper`	Conversion en majuscules

Conversions	
`int2str`	Entier → Chaine
`num2str`	Valeur numérique → Chaine
`sprintf`	Ecriture formatée
`sscanf`	Lecture formatée
`str2num`	Chaine → Valeur numérique

Fichiers

Ouverture et fermeture de fichier	
`fclose`	Ferme un fichier
`fopen`	Ouvre un fichier

Entrée-sortie non formatées	
`fread`	Lecture de données binaires dans un fichier
`fwrite`	Ecriture de données binaires dans un fichier

Entrée-sortie formatées	
`fgetl`	Lecture de ligne, pointeur en fin de ligne
`fgets`	Lecture de ligne, pointeur en début de ligne
`fprintf`	Ecriture de données formatées
`fscanf`	Lecture de données formatées

Position du pointeur de fichier	
`ferror`	Nature d'une erreur d'entrée-sortie
`frewind`	Pointeur en début de fichier
`fseek`	Positionne le pointeur
`ftell`	Renvoie la position du pointeur

Lecture et écriture de chaine	
`sprintf`	Ecriture de données formatées en chaine de caractères
`sscanf`	Lecture de chaine de caractères selon un format

Graphiques

2D	
`bar`	Diagramme en barres
`fplot`	Graphe de fonction
`grid`	Quadrillage
`gtext`	Placement de texte à la souris
`hist`	Histogramme
`plot`	Graphe 2D
`polar`	Graphe en coordonnées polaires
`stairs`	Graphe en escalier
`text`	Placement de texte
`title`	Titre de graphique
`xlabel`	Légende axe horizontal
`ylabel`	Légende axe vertical

3D	
`contour`	Lignes de niveau
`grid`	Quadrillage
`gtext`	Placement de texte à la souris
`mesh`	Graphe 3D
`surf`	Graphe 3D avec effet d'ombre
`text`	Placement de texte
`title`	Titre de graphique
`xlabel`	Légende x
`ylabel`	Légende y
`zlabel`	Légende z

Commandes diverses	
`axes`	Création d'axes
`axis`	Contrôle des axes de coordonnées
`clf`	Efface la figure courante
`close`	Ferme une fenêtre graphique
`figure`	Ouvre une nouvelle fenêtre graphique
`ginput`	Renvoie la position de la souris sur un graphique
`subplot`	Plusieurs figures dans un même fenêtre