DÉPARTEMENT STPI
TROISIÈME ANNÉE
PRÉORIENTATION ICBE

Analyse Numérique

Initiation à MATLAB

F. Deluzet, A. Huard, A. Liné, J. Morchain, P. Poncet, G.Quinio

2006/2007

Table des matières

Chapitre 1

Initiation à Matlab

MATLAB est un logiciel de calcul et de visualisation, dont les entités de base sont des matrices : MATLAB est une abréviation de Matrix Laboratory.

MATLAB est un langage interprété : il propose des facilités de programmation et de visualisation, ainsi qu'un grand nombre de fonctions réalisant diverses méthodes numériques.

La meilleure façon d'apprendre à utiliser ce logiciel est de l'utiliser vous même, en faisant des essais, en commettant des erreurs et en essayant de comprendre les messages d'erreur qui vous seront renvoyés. Ces messages sont en anglais !

Ce document est destiné à vous aider pour quelques premiers pas avec MATLAB. Les sections et sous-sections signalées par (*) donnent des compléments qui pourront vous être utiles à l'avenir, mais peuvent être réservées pour une lecture ultérieure.

1.1 Pour commencer

Nos salles de TP utilisent un environnemment `Linux`. Pour accéder au logiciel MATLAB, vous avez plusieurs possibilités :

- Vous pouvez utiliser les icônes du menu déroulant de `KDE`, et suivre les arborescenses à partir du menu `CRI`. Vous aurez accès à plusieurs versions de MATLAB. Les sessions complètes mettent en œuvre une *machine virtuelle java*. Elles peuvent se révéler un peu trop lourdes lorsqu'il y a de nombreux utilisateurs. Elles ouvrent une fenêtre du type de celle représentée à la figure 1.1.
- Vous pouvez accéder aux mêmes versions à partir d'un terminal. **C'est cette façon de procéder que nous vous recommandons**. Elle vous permet en effet de mieux organiser votre travail. Vous devez :
  - ouvrir un terminal (icône sur la barre de tâches de votre écran),
  - **créer un répertoire** pour le TP en cours. Aujourd'hui, vous aller créer le répertoire `InitMatlab` en entrant la commande :

    ```
    mkdir InitMatlab
    ```

  - vous rendre dans ce répertoire :

    ```
    cd InitMatlab
    ```

  - lancer par exemple une session complète :

```
matlab -jvm
```

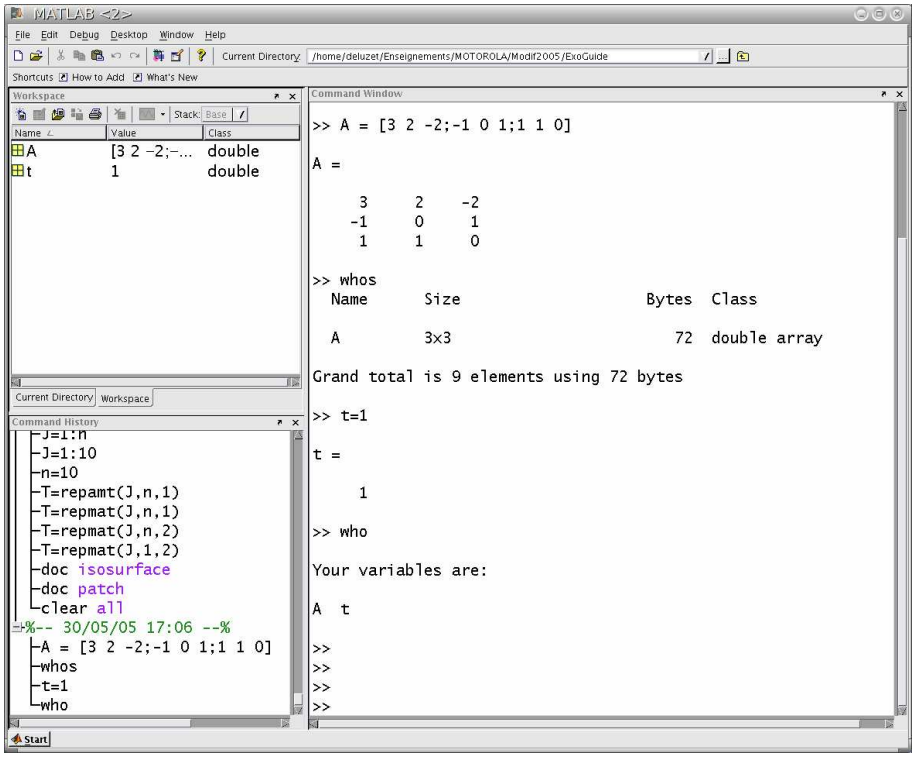

FIG. 1.1 – Exemple de session matlab avec le support *java* (`matlab -jvm`).

Vous reconnaîtrez votre fenêtre de travail par le "*prompt*" :
`>>`
Vous pourrez quitter MATLAB en tapant dans cette fenêtre :
`>> quit`

Dans la suite de cette séance, vous êtes invités à tester les instructions présentées en les tapant après le "*prompt*". Les instructions et variables MATLAB sont données dans une typographie de machine à écrire. Les opérations mathématiques et valeurs représentées par ces variables sont écrites en italiques.

Un certain nombre de fichiers vous sont fournis. Vous pourrez les récupérer en utilisant un navigateur et en vous rendant à l'adresse :

`http ://www-gmm.insa-toulouse.fr/~huard/InitMatlab/`

1.2 Utiliser les aides en ligne

1.2.1 La recherche par mot clef

La commande `lookfor` est une commande de recherche par mot clef. Elle permet de retrouver toutes les fonctions MATLAB dont les commentaires introductifs contient une chaine de caractères donnés.

Si, par exemple, je cherche une fonction permettant de tracer un histogramme, je peux entrer l'instruction :

```
lookfor histogram
```

En réponse, j'obtiens la liste suivante :

```
HIST  Histogram.
HISTC Histogram count.
ROSE   Angle histogram plot.
```

On utilise ensuite la commande `help` pour plus de précisions. Cette commande affiche les commentaires de la fonction indiquée. Ces commentaires contiennent l'objet de la fonction ainsi que les différents formats d'appel. Ils contiennent également une rubrique "See Also" qui donne les noms de fonctions dont l'objet est en rapport avec la fonction considérée. Ces commentaires sont affichés sur la fenêtre de travail.

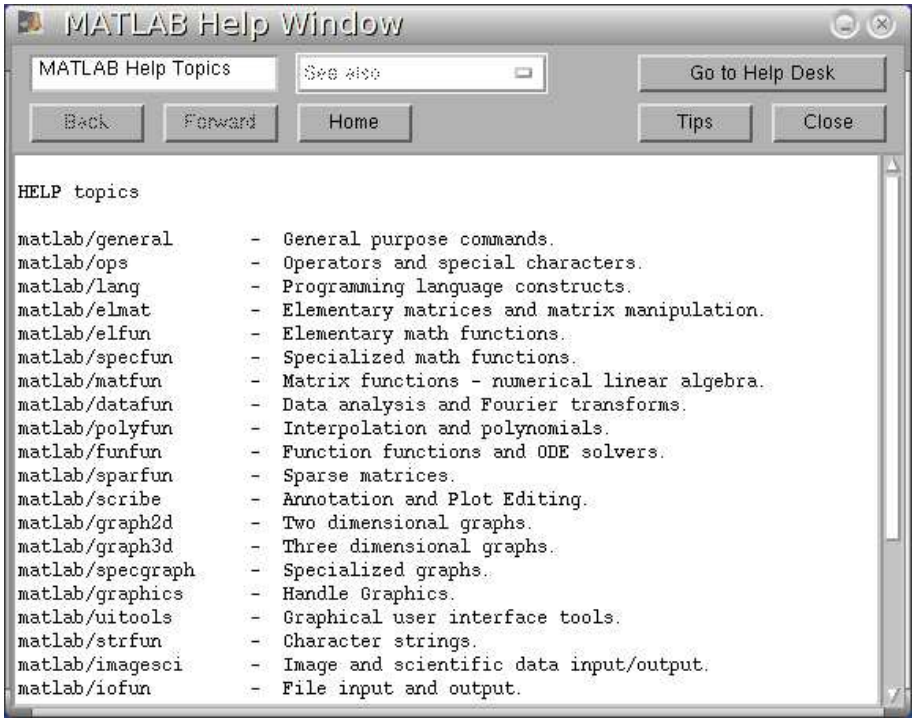

FIG. 1.2 – Page d'accueil de la fenêtre ouverte par `helpwin`.

1.2.2 La recherche par navigation

La fonction `helpwin` ouvre une fenêtre qui donne accès à ces commentaires sur une fenêtre séparée. L'accès est arborescent, partant d'un classement par domaine d'application jusqu'à l'aide de chaque fonction. Le lien est fait avec les aides des renvois suggérés par la rubrique "See Also".

La figure 1.2 nous montre sa page d'accueil. On peut cliquer sur une ligne pour continuer sa recherche...

La commande `helpdesk` donne accès à une documentation hypertexte complète supportée par le navigateur internet. Il suffit alors de se laisser guider dans sa recherche. Cette aide n'est pas accessible lorsque l'on utilise MATLAB sans le support *java*.

1.3 Variables et Matrices

1.3.1 Matrices

Dans MATLAB, toutes les variables représentent des matrices. Par exemple, on multiplie deux matrices `a` et `b` par `a*b,` et le produit de deux scalaires s'écrit de la même façon : ils sont interprétés comme des matrices `1X1`.

On peut définir une matrice dans MATLAB de plusieurs façons :

- par la liste de ses éléments,
- en la générant par une suite d'instructions et de fonctions,
- en la lisant dans un fichier extérieur.

Si l'on représente la touche `enter` de votre clavier par le symbole ↩, les instructions suivantes :

```
>> A = [3 2 -2 ;-1 0 1 ;1 1 0] ↩
```

et

```
>> A = [ ↩
 3 2 -2 ↩
-1 0 1 ↩
 1 1 0 ]  ↩
```

créent la même matrice `3X3`.

Dans la suite, le symbole ↩ ne sera plus indiqué. Le symbole `=` désigne l'opérateur d'**affectation**. Il permet de fixer ou de changer la valeur d'une variable.

En ayant éxecuté l'une ou l'autre de ces commandes, vous aurez remarqué que MATLAB affiche la valeur de la variable `A` que vous venez de définir. Vous pouvez éviter l'affichage en faisant suivre la commande d'un point-virgule. Ne la retapez pas pour le vérifier ; en utilisant la flèche ascendante de votre clavier ↑, vous pouvez rappeler la commande

```
>> A = [3 2 -2;-1 0 1;1 1 0]
```

et ajouter le point-virgule

```
>> A = [3 2 -2;-1 0 1;1 1 0] ;
```

MATLAB distingue les majuscules et les minuscules ; si vous tapez :

```
>> a
```

vous recevrez le message :

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

L'ensemble des variables actives peut être consulté grâce aux commandes `whos` et `who` :

```
>> t = 1;
>> whos
  Name      Size                    Bytes  Class
  A         3x3                        72  double array
Grand total is 9 elements using 72 bytes
```

La commande `clear` permet de détruire une variable.

```
>> who
Your variables are:
A  t
>> clear t
>> who
Your variables are:
A
```

La commande `clear all` réinitialise l'environnement (en détruisant toutes les variables actives).

L'énumération des éléments d'une matrice ou d'un vecteur peut être implicite. Essayez par exemple :

```
>> m = [-3:3]
>> x = [1:-0.25:-1]
```

Les matrices `m` et `x` sont des matrices à une ligne (vecteurs-lignes). MATLAB vous fournit les dimensions d'une variable par la fonction `size` :

```
>> size(m)

ans =
     1     7
```

Si l'on veut un vecteur, on peut taper

```
>> m = m'
```

ou

```
>> x = x(:)
```

L'apostrophe définit la transposition : `A'` représente la matrice A^T. Les deux points imposent le format colonne :

```
>> format compact % pour éviter les sauts de ligne à l'affichage.
>> A'
ans =
     3    -1     1
     2     0     1
    -2     1     0
>> A(:)'
ans =
     3    -1     1     2     0     1    -2     1     0
```

On peut aussi créer directement un vecteur en séparant les composantes par un point-virgule :

```
>> b = [3 ; 0 ;  2 ] ; size(b)
```

Un coefficient quelconque d'une matrice est référencé par ses indices de ligne et de colonne, dans cet ordre. Par exemple, l'instruction :

```
>> A(1,2)
```

renvoie la réponse :

```
ans =
        2
```

`ans` est le nom d'une variable qui reçoit le dernier résultat d'une instruction ou d'une suite d'instructions lorsqu'il n'a été affecté à aucune variable. On peut faire s'afficher la valeur d'une variable sans affichage de son nom ni de `ans`

```
>> disp(A(1,2))
```

Certaines fonctions de MATLAB créent des matrices particulières. Les quelques matrices suivantes sont parmi les plus couramment utilisées :

- `eye(n)` renvoie la matrice identité, habituellement notée I_n en mathématiques ; essayez `eye(4)`,
- `ones(n,m)` renvoie une matrice à n lignes et m colonnes dont tous les coefficients sont des 1,
- `zeros(n,m)` matrice à n lignes et m colonnes dont tous les coefficients sont des 0 ; elle sert beaucoup pour les initialisations.
- `linspace(a,b,n)` crée une matrice ligne de n points régulièrement espacés sur l'intervalle $[a,\ b]$ (bornes comprises).
- `rand` crée une matrice aléatoire. Essayez `>> B = rand(3,4)`. Dans quel intervalle sont choisis les coefficients de cette matrice ?

La commande `help` permet de recevoir des informations (en anglais !) sur une fonction ; essayez :

```
>> help linspace, >> help rand
```

1.3.2 Opérations usuelles

MATLAB permet d'effectuer les opérations usuelles en étendant leur définition aux matrices selon les règles de l'algèbre linéaire.

L'addition et la soustraction ne méritent aucun commentaire particulier, sinon que les tailles des matrices doivent être égales, essayez :

```
>> 3+5
>> ans -2
>> A + ones(3)
>> m + x
```

La dernière commande aura entrainé cette réponse cinglante :

```
??? Error using ==> +
Matrix dimensions must agree.
```

MATLAB autorise que les dimensions ne soient pas égales lorsqu'une des variables est un scalaire. `A + 1` fournit la même valeur que `A + ones(3)`.

La multiplication de deux variables sera possible si les matrices qu'elles représentent respectent les règles de concordance des dimensions :

```
>> 3*4
>> c = A*b
>> C = A*B
>> b'*c
>> b*c'
```

Comme les variables `A` et `B` représentent respectivement des matrices 3×3 et 3×4, la variable `C` représente une matrice 3×4, mais la commande `B*A` susciterait un message d'erreur de dimensions.

On peut aussi remarquer que le produit scalaire des vecteurs b et c s'obtient directement par `b'*c` , alors que `b*c'` définira une matrice 3×3 (de rang 1 !).

MATLAB autorise que les dimensions ne concordent pas lorsqu'une des variables représente un scalaire.

MATLAB distingue la division à gauche et la division à droite :

```
>> 3/4
ans =
    0.7500
>> 3\4
ans =
    1.3333
```

Ainsi cette division doit elle s'interpreter comme le produit par l'inverse de la variable située du côté vers lequel penche la barre : `3/4` représente $3 \times \frac{1}{4}$ alors que $3 \backslash 4$ représente $\frac{1}{3} \times 4$. Cette idée se généralise aux variables représentant des matrices :

```
>> x = A\b ;
>> A*x - b
```

Comment interpréter le dernier résultat renvoyé par la machine ? Vous aurez compris que la variable `x` contient la solution du système linéaire $Ax = b$. Au passage, vous remarquerez que cette nouvelle **affectation** de la variable `x` écrase la précédente.

Quelle reflexion vous suggère le résultat de :

```
>> b/A
```

On peut se demander ce que représente la matrice M définie par

```
>> M = A\eye(3)
```

MATLAB autorise le produit et les divisions par un scalaire.

L'élévation à une puissance donnée est définie par le symbole `^` : essayez

```
>> 2^3
>> A^2
>> A^0
>> A^(-1)
```

Vous avez retrouvé la matrice M.

Les règles de priorité des opérations sont conformes à celles qui sont utilisées dans la plupart des langages de programmation. Elles sont données par le tableau suivant :

Niveau de priorité	Opération
1	puissance
2	multiplication et division
3	addition et soustraction

A niveau de priorité égale, les opérations sont effectuées de la gauche vers la droite. Les parenthèses vous permettent d'organiser les priorités dans vos calculs. Leur utilisation est recommandée pour éviter les ambiguités et risques d'erreurs.

```
>> 2 + 3^2
>> 2 + 3*3^2
>> 2 + (3*3)^2
>> (2 + 3*3)^2
>> A^2\A
>> A*A\A
```

1.3.3 Opérations sur les tableaux

Une matrice peut être vue comme un tableau de valeurs, indépendament de l'algèbre linéaire. MATLAB permet de travailler avec les valeurs contenues dans un tableau. Pour l'addition et la soustraction, il n'y a pas de différence entre matrice et tableau de valeurs.

Pour la multiplication, les divisions et l'élévation à une puissance, on fait précéder le symbole d'opération d'un point. Les tableaux doivent avoir la même taille :

```
>> A.^2
>> M.*A
>> b./c
>> C.*B
>> A.*B
>> A.^[2 1 2 ; 2 0 1 ; 1 1 1]
```

On remarquera que `A.^ 2` élève à la puissance 2 chaque élément de A. C'est un raccourci pour `A.^ (2*ones(3))`.

1.3.4 Manipulations sur les matrices

MATLAB permet de manipuler les matrices par blocs : Essayez par exemple :

```
C = [ones(3), rand(3,2) ; rand(2,3), eye(2)]
```

On peut ensuite en extraire la sous-matrice formée des trois dernières colonnes :

```
C1 = C(:,3:5);
```

et on retrouve `ones(3)` en faisant afficher `C(1 :3,1 :3)`.

MATLAB propose quelques fonctions qui facilitent certaines manipulations.

La fonction `diag` extrait la diagonale d'une matrice, sous forme d'un vecteur, ou crée une matrice diagonale, à partir d'un vecteur ; essayez par exemple les instructions suivantes :

```
>> R = rand(5);
>> diag(R)
>> diag(ans)
>> diag(diag(R))
```

L'argument général de `diag` est double ; entrez :

```
>> help diag
```

Vous constatez que l'on peut ainsi extraire d'une matrice ses éléments d'une "parallèle" à la diagonale. Essayez :

```
>> diag(R,1)
>> diag(R,-2)
>> d = ones(4,1);
>> M = 2*eye(5)-(diag(d,1)+diag(d,-1))
```

`tril` renvoie la partie triangulaire inférieure d'une matrice, éléments diagonaux compris. Elle admet aussi plusieurs arguments

```
>> tril(R)
>> tril(R,1)
```

`triu` renvoie la partie triangulaire supérieure d'une matrice.

`reshape` reformate une matrice, c'est-à-dire change le nombre de lignes et de colonnes, en prenant les éléments colonne par colonne.

`repmat` crée une "grande" matrice en recopiant une matrice donnée selon le format fournit.

1.4 Boucles et tests

Les principales instructions de contrôle proposées par MATLAB sont `for, while` et `if` ; elles fonctionnent à peu près comme leurs équivalents dans les autres langages de programmation.

1.4.1 La boucle for

La boucle `for` doit respecter la syntaxe suivante :

`for` *compteur* = *expression*
    instructions
`end`

Generalement, *expression* est un vecteur de la forme `début :incrément :fin` et *compteur* prend successivement toutes les valeurs de *expression* pour éxecuter *instructions*.

Les instructions suivantes permettent de calculer une valeur approchée de e^x, pour $x = 10$, en utilisant la formule :

$$e^x \simeq \sum_{k=0}^{n} \frac{x^k}{k\,!} \tag{1.1}$$

```
>> x = 10;
>> n = 50;
>> s = 1;
>> terme = 1;
>> for k = 1:n, terme = x*terme/k ; s = s + terme ; end;
>> s
```

On pourra vérifier plus tard que la valeur obtenue, $s \simeq 2.2026 \times 10^4$ est assez proche de e^{10}.

Plusieurs boucles peuvent être emboîtées. Comme MATLAB est un language interprété, il faut essayer d'éviter les boucles quand c'est possible. Pour cela on peut utiliser des boucles implicites. Pour construire la matrice $H = (h_{i,j}) \in M_3(\mathbb{R})$ avec $h_{i,j} = \dfrac{1}{i+j-1}$, on peut utiliser les instructions suivantes :

```
>> n = 3;
>> H = zeros(3);
>> for i = 1:n
       for j = 1:n
           H(i,j) = 1/(i+j-1);
       end;
  end;
>> H
```

Notez que l'on a **initialisé** la variable `H`. C'est recommandé lorsque c'est possible. Notez également que l'on a **indenté** les boucles : c'est recommandé pour une meilleure lisibilité !

On peut aussi construire `H` de la façon suivante :

```
>> J = 1:n;
>> J = repmat(J,n,1);
>> I = J';
>> E = ones(n);
>> H = E./(I+J-E);
```

Remarque 1 *Dans le premier exemple d'utilisation de* `for`*, la boucle tenait dans une ligne. Dans ce cas, une virgule doit séparer l'initialisation de la boucle de l'instruction suivante.*

Dans la second exemple, puisque l'on va à la ligne, le prompt de MATLAB *ne réapparait que lorsque la boucle est correctement terminée.*

1.4.2 La boucle while

L'instruction `while` respecte la syntaxe suivante :

`while` *expression*
    instructions
`end`

expression désigne le résultat d'une **opération logique**. Elle est construite en utilisant des opérateurs relationnels et logiques.

Opérateurs relationnels		Opérateurs logiques	
Opérateur	Symbole	Opérateur	Symbole
égal	==	et logique	&
différent	~=	non logique	~
supérieur	>	ou logique	|
inférieur ou égal	<=		

Les *instructions* seront éxécutées tant que *expression* sera "vraie". Toute valeur différente de 0 est assimilée au 1 logique, c'est-à-dire au "vrai". Essayez

```
>> x = [1 -1 1] ; y = [1 1 -1];
>> x>0, ~(x>0),~x>0
>> x>=y
>> x>0 & y>0
>> x>0 | y>0
```

Les instructions suivantes permettent de trouver approximativement le plus petit nombre positif représenté dans l'arithmétique utilisée par MATLAB :

```
>> x = 1 ;  while x>0 , xm = x; x = x/2; end; xm
```

1.4.3 Le choix conditionnel if

L'instruction `if` respecte la syntaxe suivante :

`if` *expression*
    instructions
`end`

Les *instructions* ne seront éxécutées que si *expression* est vraie. Il est possible de proposer une alternative, en indiquant les instructions à éxecuter lorsque *expression* est fausse :

`if` *expression*
    instructions 1
`else`
    instructions 2
`end`

Il est également possible d'emboîter les choix proposés :

`if` *expression1*
    instructions 1
`elseif` *expression2*
    instructions 2

```
else
      instructions 3
end
```

1.4.4 Utiliser un fichier de commandes

Les exemples d'utilisation de la boucle `for` nous ont amenés à écrire plusieurs lignes de commandes. Lorsque l'on veut recommencer l'exécution d'un de ces exemples en modifiant la valeur d'un paramètre, il faudra rappeler toutes les lignes qui suivent sa définition : c'est le cas si l'on veut évaluer (1.1) pour une autre valeurs de x ou de n. Cette opération devient très désagréable lorsque l'on a emboité des boucles !

De même que vous utilisez un brouillon avant de rendre un devoir, de façon à pouvoir faire des ratures, changer les expressions où vous avez détecté une erreur, vous pouvez éditer un fichier de commandes MATLAB dont vous vous servirez comme d'un brouillon.

Un tel fichier s'appelle un M-fichier. Il devra être enregistré sous le nom de votre choix **avec le suffixe** `.m`.

Pour écrire ce fichier, vous devez utiliser un **éditeur** : MATLAB vous en propose un auquel vous pouvez accéder en vous servant des icônes de la fenêtre.

Nous allons illustrer cette possibilité en traitant un exemple qui utilise la boucle `while` et le branchement `if` : il s'agit de tester la conjecture de Syracuse.

On part d'un nombre entier différent de 1. S'il est impair, on le multiplie par 3 et on ajoute 1 ; sinon, on le divise par 2. On recommence avec ce nouveau nombre tant qu'il n'est pas égal à 1. Dans ce cas, en effet, on reproduira indéfiniment le cycle $\{\cdots,\ 1, 4, 2, 1, \cdots\}$.

Quelque soit le nombre entier dont on part, on finit toujours par obtenir la valeur 1. C'est en tous cas ce que montre l'expérience, car personne n'a jamais trouvé de contre exemple. Mais, on a jamais pu démontrer qu'il devait en être ainsi !

On vous propose de tester cette conjecture. L'objectif de l'exercice est d'écrire un programme MATLAB qui propose à l'utlisateur de donner un nombre entier différent de 1, et à partir de ce nombre, de compter le nombre d'itérations de la suite de Syracuse avant qu'elle n'atteigne la valeur 1. On prendra la précaution de **limiter le nombre d'itérations** à 1000 pour éviter que le programme ne tourne indéfiniment. Cela pourrait se produire en cas d'erreur de saisie.

Pour cela, vous allez commencer par éditer le fichier dans lequel vous écrirez ce programme. Vous pouvez utiliser le fichier qui vous est fournit sous le nom de `Syracuse.m` et qui contient le squelette de programme qui est donné au tableau 1.1.

Il vous faut remplacer les points de suspension par des instructions appropriées. Voici quelques indications :

- La fonction `input` permet de proposer à l'utilisateur de rentrer un entier `n` initial.

```
% Syracuse :  Tester la conjecture de Syracuse

n = input('Entrez un entier superieur a 1 : ') ;
compteur = 1 ;         % Compteur des itérations

while   ... &  ...
  if rem(n,2) ...
     n =  ...  ;
  else
     n =  ...  ;
  end
  compteur = compteur +1;
end
compteur
```

TAB. 1.1 – Programme `Syracuse.m`

- Le symbole `%` permet de placer des commentaires.
- Le nombre d'itérations sera donné par la variable `compteur`.
- la boucle `while` sera cotrôlée par deux expressions logiques, qui autoriseront son éxécution tant que `n` restera différent de 1 et `compteur` inférieur ou égal à 1000.
- Le test sur la parité de n utilise la fonction `rem` (abréviation de remainder) qui donne le reste de la division entière par 2 : ce reste vaut 1 si `n` est impair et 0 sinon.

Remarque 2 *On ne s'est pas trop bridé en limitant le nombre des itérations à* 1000. *Pour des valeurs initiales inférieures ou égales à 10 000, la suite la plus longue est obtenue pour une valeur initiale de* 6171. *Sa longueur est* 262.

1.5 Fonctions de base

1.5.1 Fonctions scalaires

Ce sont les fonctions usuelles ; par exemple :

`sin`	`exp`	`abs`	`round`
`cos`	`log`	`sqrt`	`tanh`
`tan`	`rem`	`sign`	`acosh`

Ces fonctions font partie des fonctions élémentaires proposées par MATLAB. Pour en avoir la liste, vous pouvez taper :

```
>> help elfun
```

Comme la liste est longue, vous pouvez contrôler son défilement :

```
>> more on, help elfun, more off
```

Le défilement se fait ligne à ligne (en tapant une touche quelconque) ou par pages de 20 lignes (en tapant la barre d'espacements). Vous pouvez maintenant comparer la variable `s` calculée en utilisant la formule (1.1) avec e^{10} ;

```
>> s - exp(10)
```

Quelle commande devrez-vous entrer pour obtenir la valeur absolue de l'erreur relative ?

Ces fonctions traitent les variables matricielles comme des tableaux de nombres ; si la variable `A` représente une matrice A, la variable `S = sin(A)` représentera une matrice S dont les coefficients sont $s_{i,j} = \sin(a_{i,j})$. Pour l'exponentielle et la racine carrée, qui possèdent une version "matricielle", MATLAB propose les fonctions `expm` et `sqrtm`.

Le cas des polynômes est un peu particulier : un polynôme peut être défini par ses coefficients : on utilise dans ce cas une matrice ligne : par exemple, le polynôme $Q(x) = x^3 + 2x^2 - 3$ sera représenté par :

```
>> Q = [1 2 0 -3];
```

Sa valeur au point $x = 1.2$ sera fournie par :

```
>> polyval(Q,1.2)
```

Les racines du polynôme Q seront fournies par la fonction `roots`. A l'inverse, on peut aussi déterminer les coefficients d'un polynôme à partir de ses racines en utilisant la fonction `poly` :

```
>> r=[1 2 3]; K = poly(r)
>> polyval(K,r)
```

Ces instructions assignent à `K` la valeur `[1 -6 11 -6]` qui définit le polynôme $K(x) = x^3 - 6x^2 + 11x - 6$. Le polynôme ainsi calculé a toujours le coefficient de son terme de plus fort degré égal à 1.

Comme les fonctions usuelles `polyval` traite les matrices comme des tableaux, Il faudra utiliser la fonction `polyvalm` pour évaluer un polynôme de matrice.

Comparez `polyval(K,A)` et `polyvalm(K,A)`.

Certaines fonctions spéciales sont également proposées par Matlab : par exemple, les fonctions de Bessel de première espèce notées $J_\nu(x)$ sont données par la fonction `besselj` dont le format d'appel le plus courant est, pour le paramètre entier `nu` et la variable `x` : `y = besselj(nu,x) ;`

Vous pouvez retrouver l'information sur le format d'utilisation par :

```
>> help besselj
```

La liste de ces fonctions spéciales vous sera renvoyée par la commande :

```
>> more on, help specfun, more off
```

1.5.2 Fonctions vectorielles

Ces fonctions sont plutot destinées à agir sur des vecteurs, lignes ou colonnes. Elles sont très utiles à l'analyse des données. Citons par exemple :

```
max        sum        mean        sort
min        prod       std         find
```

dont vous trouverez l'objet par la commande `help` lorsqu'il n'est pas évident.

Elles agissent aussi sur les matrices, mais colonne par colonne. Essayez par exemple :

```
>> b = rand(3,3) ; c = max(b) ; d = max(max(b));
>> b,c,d
>> sort(c)
>> sort(b)
```

Vous pouvez modifier le fichier `Syracuse.m` pour retrouver le résultat annoncé à la remarque 2 en utilisant une boucle `for` et la fonction `max`.

1.5.3 Fonctions matricielles

Citons en quelques-unes :

`eig` : valeurs et vecteurs propres d'une matrice,
`inv` : inverse,
`expm` : exponentielle de matrice,
`size` : dimensions de la matrice,
`norm` : norme (2 par défaut, mais aussi 1 ou ∞),
`rank` : rang.

Une liste un peu plus complète, mais non exhaustive, de ces fonctions vous est fournie en annexe. Une liste complète est fournie en réponse à la commande

```
>> more on, help matfun, more off
```

1.6 Graphiques

1.6.1 Visualisation des courbes en 2D

MATLAB permet de tracer facilement le graphe de fonctions scalaires. Cela se fait à l'aide de la fonction `plot` . Elle prend en argument des paires de vecteurs de même dimension. Voici par exemple le moyen d'obtenir le graphe de la fonction $\cos 3x$ sur l'intervalle $[0, 2\pi]$:

```
>> x = [0:0.01:2*pi] ; y =cos(3*x) ; plot(x,y);
```

Le format de base de la fonction `plot` est en fait le suivant :

```
plot(x,y,s) ;
```

où `x` contient les absisses, `y` les ordonnées, et `s` est une chaine de 1 à 3 caractères : `s='ctm'`, pour la couleur (c) et le style (tm), qui peuvent être choisis dans le tableau suivant :

Couleur		Style	
`y`	jaune	`-`	trait continu (t)
`m`	magenta	`:`	pointillés (t)
`c`	cyan	`-.`	trait-point (t)
`r`	rouge	`--`	tirets (t)
`g`	vert	`+`	plus (m)
`b`	bleu	`o`	cercles (m)
`k`	noir	`*`	étoiles (m)
`w`	blanc	`x`	croix (m)

Les style de tracé suivis de (t) sont des tracés continus, avec une interpolation entre les points fournis, alors que pour les autres on utilise des marqueurs pour ne représenter que les points (x_i, y_i). On peut combiner ces deux styles.

On peut tracer les graphes de plusieurs fonctions simultanément. Essayez maintenant :

```
>> z = sin(2*x);plot(x,y,x,z);
```

On peut alors légender notre graphique de la façon suivante :

```
>> legend('cos(3x)','sin(2x)');
```

On peut également obtenir ces deux courbes par la succession de commandes :

```
>> plot(x,y);
>> hold on;
>> plot(x,z);
```

La seule différence est que cette fois, les deux graphes sont tracés avec la même couleur. `hold on` gère le graphique courant de façon que les commandes graphiques à suivre vont s'ajouter à ce graphique. `hold off` est a valeur par défaut : dans ce cas une nouvelle commande `plot` effacera le graphique existant.

Les axes sont définis automatiquement ; on peut choisir les bornes des coordonnées du graphe à l'aide de la fonction `axis` . Essayez par exemple :

```
>> axis([-1 5 -1.5 1.5]);
```

Pour la suite de notre travail, il est préférable que vous entriez maintenant la commande :

```
>> hold off;
```

Le format de la fonction `plot` permet bien sûr de représenter facilement des courbes paramétriques :

```
>> plot(y,z);
```

On obtient, pour les valeurs définies plus haut, une courbe connue sous le nom de courbe de Lissajoux. Les instructions suivantes vont réduire légèrement l'échelle, donner un titre et un nom aux axes, et faire apparaître une grille sur le fond :

```
>> axis([-1.1 1.1 -1.1 1.1])
>> title('Courbe de Lissajoux');
>> xlabel('x : axe des absisses');
>> ylabel('y : axe des ordonnees');
>> grid
```

Vous obtiendrez l'impression sur papier d'un graphe en utilisant la commande `print` qui enverra l'impression sur l'imprimante à laquelle est relié votre ordinateur. C'est la dernière de vos figures qui sera imprimée.

Exercice

On considère la fonction définie par

$$F(x) = 0.01 \exp x + 10 \cos x - 3\,x. \tag{1.2}$$

On vous demande d'écrire un M-fichier `ExempleCourbe.m` pour tracer la courbe représentant cette fonction sur l'intervalle $[-1, 10]$, en utilisant ses valeurs aux points $x_k = -1 + \dfrac{k}{100}$, $0 \leq k \leq 1100$. Vous fixerez la taille de la fenêtre de visualisation de façon que les absisses soient comprises entre -1 et 11 et les ordonnées entre -50 et 200.

Vous pourrez compléter cette figure en y ajoutant les axes et un titre. Ces axes peuvent être tracés "à la main" en utilisant les fonctionnalités de la fenêtre graphique (voir les icônes). Vous les obtiendrez aussi en terminant votre programme par les instructions données dans l'encadré ci-dessous.

```
 ...
hold on ;
plot([0,9],[0,0],9,0,'>')
plot([0,0],[-20,150],0,150,'^')
hold off ;
```

Exercice

Vous allons créer un fichier qui représente la fonction de Bessel de paramètre $\nu = 0$ sur l'intervalle $[0,\ 50]$, et vérifier graphiquement que ses racines sur cet intervalle sont bien approchées par les expressions

$$r_j \simeq (2\,j-1)\frac{\pi}{2} + \frac{\pi}{4},\ 1 \leq j \leq 15. \tag{1.3}$$

Vous appellerez ce fichier `VisuRacBess.m`. Il devra contenir des instructions pour :

- définir les absisses x (utiliser la fonction `linspace`),
- définir le paramètre ν et calculer les $y = J_\nu(x)$, en utilisant la fonction `fzero`. Cette fonction résoud les équations non linéaires. L'algorithme utilisé combine plusieurs méthodes, dont celles de la sécante et de bissection. On doit fournir une valeur pour initialiser les itérations, et il est bon de disposer d'une valeur assez voisine de la solution cherchée. On se servira donc des expressions (1.3).
- représenter sur la même figure les y et l'axe des x,
- calculer les r_j, (définir un tableau `R` en utilisant une variable `J = [1 :15] ;`),
- représenter sur la figure précédente les points $(r_j,\ 0)$ par le symbole $*$.

On voudrait pouvoir donner un titre à cette figure, et pour cela représenter les lettres grecques ϕ et ϵ : cela peut se faire de la façon suivante :

```
title('La fonction J_{\nu } pour \nu = 0');
```

On peut souhaiter des caractères un peu plus grands pour ce titre :

```
title('La fonction {\nu } pour \nu = 0','Fontsize',12);
```

La taille de fonte est précisée par la propriété `'Fontsize'`, suivie de la valeur choisie.

Cette instruction utilise les règles du langage LaTeXqui est un langage de programmation de traitement de texte particulièrement bien adapté à la gestion des formules mathématiques. Elle appelle quelques commentaires :

- Le caractère _ permet de mettre en position d'indice les caractères entre accolades qui le suivent.
- Le caractère \ permet à LaTeX de reconnaître ses commandes. Vous en trouverez quelques-unes dans le tableau 1.2.

1.6.2 Visualisation des courbes en 3D (*)

La fonction qui permet de représenter une courbe dans l'espace $\mathbb{R}^3$ est `plot3`. Sa syntaxe est

```
plot3(x,y,z,s) ;
```

et cette fois `x`, `y` et `z` sont des vecteurs de même dimension contenant les trois coordonnées. Essayez :

minuscules grecques		**majuscules grecques**		**symboles mathématiques**	
Commande	Symbole	Commande	Symbole	Commande	Symbole
`\alpha`	α	`\Delta`	Δ	`\leq`	$\leq$
`\beta`	β	`\Theta`	θ	`\geq`	$\geq$
`\gamma`	γ	`\Gamma`	Γ	`\int`	$\int$
`\omega`	ω	`\Omega`	Ω	`\infty`	∞
`\lambda`	λ	`\Lambda`	Λ	`\sim`	$\sim$
`\pi`	π	`\Phi`	Φ	`\partial`	∂
`\rho`	ρ	`\Psi`	Ψ	`\in`	$\in$
`\nu`	ν	`\Sigma`	Σ	`\rightarrow`	$\rightarrow$

TAB. 1.2 – Principaux symboles mathématiques reconnus par MATLAB.

```
>> theta = pi*[-4:0.04:4]; r = linspace(1,6,201);
>> x = r.*(1+cos(theta)); y = r.*sin(theta); z = r;
>> plot3(x,y,z,'k*'); grid;
```

1.6.3 Visualisation des surfaces (*)

MATLAB permet de représenter des surfaces données en coordonnées cartésiennes ou sous forme paramétrique. La forme paramétrique peut s'écrire :

$$\begin{aligned} x &= x(s,t) \\ y &= y(s,t) \\ z &= z(s,t) \end{aligned}$$

où les paramètres s et t parcourent un certain domaine. Plusieurs fonctions permettent cette représentation, parmi lesquelles `mesh` représente un treillis, et `surf` une surface pleine. Elles ont le même format d'appel :

```
surf(X,Y,Z,C) ;
```

`X,Y` et `Z` sont des matrices de mêmes dimensions contenant les coordonnées de points de la surface. `C` permet de définir les couleurs et peut être omis. Nous allons représenter le cône d'equation $x^2 + y^2 - 2xz = 0$ en utilisant la représentation paramétrique donnee pour $(\theta,\, \rho) \in]-\pi,\, \pi[\times]\, 0,\, 12[$ par :

$$\begin{aligned} x &= \rho\,(1+\cos\theta), \\ y &= \rho\,\sin\theta, \\ z &= \rho. \end{aligned}$$

On peut procéder de la façon suivante :

```
>> n = 31 ; theta = pi*[-n:2:n]/n ; r = linspace(0,12,n);
>> X = r'*(1+cos(theta)) ;
>> Y = r'*sin(theta) ;
>> Z = r'*ones(size(theta));
>> surf(X,Y,Z);
```

Il est important de remarquer que `X`, `Y` et `Z` sont des matrices de même taille !

La surface représentée laisse apparaitre les traces d'une grille qui la recouvre. L'aspect de la surface est contrôlé par la variable `shading`, dont les valeurs possibles sont `faceted` (valeur par défaut), `flat` (supprime le treillis), et `interp` (adoucit les transitions). Essayez :

```
>> shading interp;
```

Les couleurs sont définies par une matrice `C` de même taille que celles des coordonnées, et par défaut, cette matrice est `C=Z`, de sorte que d'une certaine façon, la couleur est "proportionnelle" à l'altitude du point de la surface. Tout cela dépend du choix d'une palette graphique. Essayez :

```
>> C = rand(size(Z)) ; surf(X,Y,Z,C);
```

La palette de couleurs de la fenêtre de visualisation est définie par la variable `colormap`. Sa valeur par défault est `'jet'` . Essayez :

```
>> colormap('cool');
```

La liste des palettes disponibles est fournie avec celle de toutes les fonctions ou variables de contrôle d'un graphique 3D par la commande :

```
>> more on, help graph3d, more off
```

Examinons à présent le tracé d'une surface donnée par son équation cartésienne $z = f(x, y)$. En général, on dispose d'une famille de valeurs de x et d'une famille de valeur de y stockées dans 2 vecteurs, respectivement `x` et `y`. On va devoir construire deux matrices `X` et `Y` telles que les colonnes de `X` et les lignes de `Y` soient constantes : ces matrices seront utilisees pour le calcul des valeurs de f. On peut utiliser la fonction `meshgrid` .

Ainsi, pour représenter la surface d'équation $z = \exp(-x^2 - y^2)$, $-2 < x, y < 2$, on pourra procéder comme suit :

```
>> [X,Y] = meshgrid(-2:.2:2, -2:.2:2);
>> Z = X .* exp(-X.^2 -Y.^2);
>> surf(Z);
```

Si on ne souhaite que le treillis, on remplace la fonction `surf` par `mesh`. Si l'on souhaite des isolignes, on utilise la fonction `contour3` ; par exemple :

```
>> v  = linspace(0,max(max(Z)),20); contour3(Z,v);
```

1.6.4 La notion d'objets graphiques (*)

Différents objets et leur identification

Lorsque que l'on trace une figure, par exemple la courbe représentant la fonction $y = \sin x$ sur l'intervalle $[-3.5, 3.5]$, on utilise une fonction graphique de MATLAB. On peut le faire simplement de la façon suivante :

```
>> x = linspace(-3.5,3.5) ;
>> y = sin(x);
>> plot(x,y);
```

Les commandes précédentes auront créé la figure 1.3

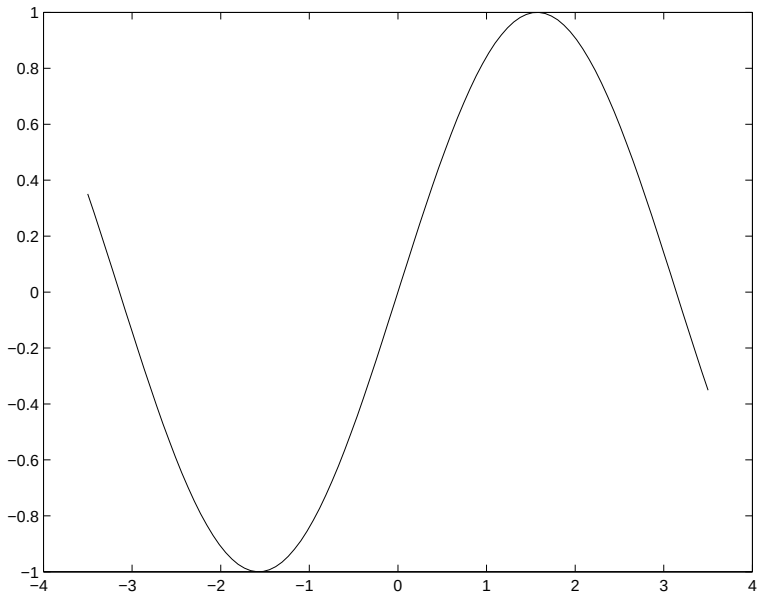

FIG. 1.3 – Une figure obtenue par la commande de base `plot`.

La fonction `plot` que l'on a utilisé a généré un assez grand nombre de données. Ces données sont les caractéristiques de notre figure. Le système de représentation graphique de MATLAB est un système orienté objet, et les données qui caractérisent une figure sont relatives aux différents objets qui la constituent.

Certaines de ces données peuvent être précisées lors de l'appel de la fonction : on peut choisir le type de tracé, sa couleur ... D'autres pourraient l'être en utilisant des fonctions qui le permettent : `legend, xlabel...`

Si l'on avait voulu modifier la position la taille de la fenêtre graphique, il aurait cependant fallu utiliser la fonction `figure` avant la fonction `plot` et fixer les valeurs de la propriété `'Position'`.

Pour gérer une fenêtre graphique, il convient de pouvoir identifier ces objets, avant de fixer ou de modifier leurs propriétés. Cette identification se fait par les handles, littéralement les “poignées” pour les attraper !

La première étape consiste à récupérer ces handles. La fonction `findobj` le permet. Voici ce que peut fournir son appel :

```
>> h = findobj
h =
         0
    1.0000
  151.0015
  152.0022
```

Il ne s’agit pas ensuite d’utiliser les valeurs affichées, mais de se rappeler que `h` est un vecteur de 4 composantes dont chacune est un handle. Quels sont les objets ainsi identifiés ?

```
>> get(h,'type')
ans =
    'root'
    'figure'
    'axes'
    'line'
```

La première composante dont la valeur est 0 est la racine, c’est-à-dire l’écran. La seconde est le fenêtre. Son handle est également désigné par la fonction `gcf` (“get current figure”).

Cette structure est **arborescente** : à partir de la racine, on peut trouver une ou plusieurs fenêtres ; dans chaque fenêtre, un ou plusieurs systèmes d’axes, et dans chacun d’entre eux, un ou plusieurs courbes.

Modifier les propriétes

Chaque type d’objet gère certaines propriétés. On va commencer par modifier la taille de la fenêtre. Pour cela, il faut modifier la propriété `'Position'`.

Pour voir comment elle est définie, nous récupérons sa valeur.

```
>> get(h(2),'Position')
ans =
   181   457   560   420
```

Vous n’aurez pas les mêmes valeurs. Ces valeurs représentent respectivement la distance la fenêtre au bord gauche de l’écran, sa distance au bord inférieur de l’écran, sa largeur et sa hauteur. Elles sont données en Pixels.

Pour modifier la taille de cette fenêtre, il faut connaître la taille de l’écran. C’est une propriété de la racine :

```
>> get(h(1),'ScreenSize')
ans =
   1          1        1280        1024
```

On peut préférer travailler en coordonnées relatives avant de fixer cette taille :

```
>> set(h(2),'Units','normalized','Position',[0.1 0.3 0.6 0.4])
```

Pour connaître l'ensemble des propriétés relatives à l'objet `figure`, il suffit de taper `get(h(2))`.

Pour obtenir la liste de ces propriétés avec les valeurs autorisées, lorsqu'elles sont prédéfinies, on tape `set(h(2))`.

L'objet `h(3)` est de type `axes`. On peut lister le champ de ses propriétés avec leur valeur actuelle :

```
>> get(h(3))
```

Nous allons supprimer l'encadrement, modifier les limites de l'axe des y, fixer les positions et valeurs des "ticks" de l'axe des x :

```
>>set(h(3),'Box','off')
>>set(h(3),'Ylim',[-1.1 1.1])
>>set(h(3),'Ytick',[])
>>set(h(3),'Xtick',[-pi -pi/2 0 pi/2 pi])
>>set(h(3),'XtickLabel','-pi|-pi/2|0|pi/2|pi')
```

On remarquera que les symboles LaTeX ne sont pas pris en compte pour étiqueter les ticks.

Une propriété du nom de `Title` figure dans la liste des propriétés de l'objet `axes` et sa valeur est un nombre réel. Il s'agit là encore d'un handle, car le titre est lui même un objet graphique.

```
>>htit = get(h(3),'Title')
>>set(htit)
>>set(htit,'string','Fonction sinus','FontSize',12)
>>set(htit,'Color',[1 0 0])
```

La couleur obtenue est le rouge. La définition des couleurs peut se faire par un tableau de 3 valeurs comprises entre 0 et 1 qui donnent respectivement les poids relatifs du rouge, du vert et du bleu. (système rgb). Le blanc est [1 1 1] et le noir [0 0 0].

On peut aussi modifier certaines propriétés de l'objet de type `line` On en obtient la liste par `get(h(4))`.

Pour connaître les possibilités offertes au niveau des types de tracé,

```
>>set(h(4),'Linestyle')
>>set(h(4),'Marker')
```

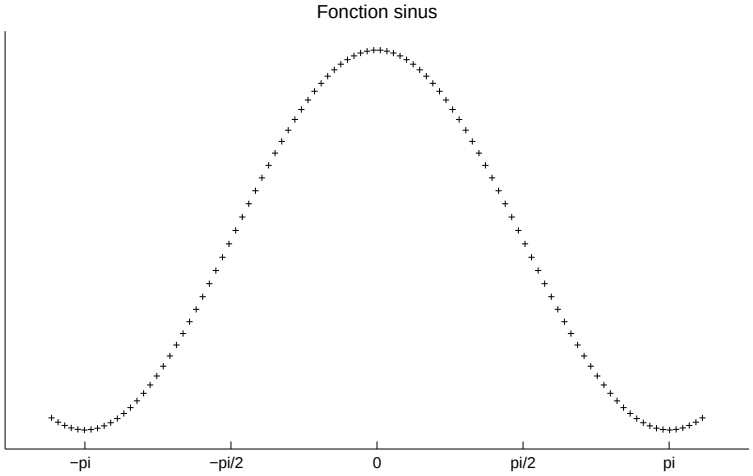

FIG. 1.4 – Figure modifiée grâce à la manipulation des *hanlde*.

On peut alors proposer

```
>>set(h(4),'Linestyle','none','Marker','+')
>>set(h(4),'Markersize',4)
>>set(h(4),'Ydata',cos(x))
```

La figure après modification ressemblera, sauf pour les couleurs, à la figure 1.4. Il faudrait quand même modifier son titre ...

1.7 M-fichiers

1.7.1 Scripts et fonctions

Bien que l'on puisse faire beaucoup de choses en travaillant directement dans la fenêtre de commandes de MATLAB, nous avons déja constaté que qu'il était souvent plus pratique d'utiliser des M-fichiers. Ils permettent

- de corriger plus facilement ses erreurs,
- de garder un travail en cours pour le reprendre plus tard,
- d'exécuter plusiers fois un programme en modifiant certains paramètres,
- d'échanger son travail avec des camarades.

Jusqu'a présent, ces fichiers étaient des fichiers d'instructions, ou fichiers "scripts". Un fichier script est un fichier contenant une suite d'instructions MATLAB directement éxécutables. Ces instructions sont éxécutées en utilisant les variables de l'espace de travail, ou en créant des variables dans cet espace.

MATLAB utilise également des M-fichiers. En fait, un grand nombres des fonctions que nous avons rencontrées sont écrites dans des M-fichiers. Il s'agit de fichiers "fonction". Vous pouvez également créer vos propres fonctions.

Un fichier "fonction" a sa première ligne d'instruction éxécutable qui commence par le mot clef `function`. Il peut recevoir des variables en entrée, mais utilise ces

variables en dehors de l'espace de travail. Aucune des variables créées ou modifiées lors de son éxécution n'est visible depuis l'espace de travail à moins qu'elle ne soit une variable en sortie de la fonction !

Nous allons illustrer la différence entre ces deux types de fichiers sur un exemple. Si le fonction f est suffisament régulière, la formule de Taylor permet d'obtenir une valeur approchée de sa dérivée en un point x, pour un pas h, suivant

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{h}{2} f''(x) + \mathcal{O}(h^2) \simeq f'(x). \tag{1.4}$$

```
% Fichier ScriptDn1 pour calculer
% la derivee numerique de sinus
% au point 1 pour le pas
% h = 0.01.

x = 1;
h = 0.01;
Dn = (sin(x+h) -sin(x))./h;
```

```
function Res = Fonc1Dn1(h);
% Derivee numerique de sinus
% au point 1 en fonction du
% pas h.

x = 1;
Dn = (sin(x+h) -sin(x))./h;
Res = Dn;
```

TAB. 1.3 – Fichier "script" et fichier "fonction".

Le tableau 1.3 montre le contenu du fichier "script" `ScriptDn1.m` et du fichier "fonction" `FoncDn1.m` Une fois ces deux fichiers sauvegardés dans votre espace de travail, vous pourrez tapez les commandes quivantes :

```
>> clear all
>> ScriptDn1
>> who
>> Err = abs(Dn-cos(1))
>> clear all
>> FoncDn1
```

MATLAB n'est pas content et il vous le dit :

```
??? Input argument "h" is undefined.

Error in ==> FoncDn1 at 6
Dn = (sin(x+h) -sin(x))./h;
```

L'erreur est commentée ; MATLAB vous indique le numéro de la ligne où elle s'est produite. A présent, nous fournissons le pas en appel de la fonction

```
>> pas = 0.01 ; FoncDn1(pas)
ans =
    0.5361
>> Err = abs(Dn-cos(1))
??? Undefined function or variable 'Dn'.
>> who
Your variables are:

ans  pas

>> Der = FoncDn1(pas);
>> Der-ans
```

On peut faire quelques remarques :

- Le fichier script s'éxécute sans qu'on lui fournisse d'arguments. Le fichier fonction a besoin qu'on lui fournisse les arguments qu'il ne crée pas, ici le pas.
- Le nom de la variable affectée de la valeur de ce pas dans l'espace de travail n'a pas d'importance ; c'est sa valeur qui est passée.
- Les variables créées par l'éxécution du script sont vues dans l'espace de travail. Celles qui sont créées lors de l'éxécution de la fonction ne le sont pas. Ce sont des variables **locales**. L'argument en sortie prend le nom qu'on lui donne dans la session MATLAB.

1.7.2 Fonctions

Fonctions simples

Nous avons déjà rencontré la fonction $F(x) = 0.01\, e^x + 10\, \cos x - 3\, x$.

Dans ce cas, on dispose de deux façons pour créer une fonction MATLAB qui calcule les valeurs de F :

- comme la fonction F peut se définir en une seule ligne, on peut le faire avec la fonction `inline`
  ```
  >> F1 = inline('0.01*exp(x)+10*cos(x)-3*x')
  ```
- de façon plus générale, on peut éditer un fichier que l'on va appeler `F2.m` qui devra commencer par le mot-clef `function` : vous pouvez le définir de la façon suivante
  ```
  function y = F2(x) ;

  y = 0.01*exp(x)+10*cos(x)-3*x;
  ```

La première définition est à utiliser lorsque l'on n'envisage pas d'utilisation ultérieure de cette fonction. La définition au moyen d'un fichier permet de la conserver. Cependant, ainsi qu'elle est rédigée, cette fonction a encore le défaut que son objet n'est pas immédiatement lisible ; on risque de l'avoir oublié quand on aura besoin de la réutiliser. Il faut donc la commenter.

Cela se fait sur des lignes qui commencent par le symbole % . Ce symbole transforme en commentaires tous les caractères qui le suivent dans la ligne courante. Entre autres commentaires, vous devrez toujours indiquer le format d'appel de la fonction. Voici comment je vous suggère d'écrire ce fichier :

```
function y = F2(x) ;
% fichier fonction definissant la fonction F2(x) qui sert d'exemple
% au polycopie d'initiation a Matlab.
% L'appel se fait selon :
%                     >> y = F2(x);

y = 0.01*exp(x)+10*cos(x)-3*x;
```

Ces commentaires sont placés immédiatement après la première ligne (de déclaration). De cette façon, ils constitueront la réponse fournie par MATLAB à la commande :

```
>> help F2
```

Les variables internes d'un fichier fonction sont locales ; elles ne sont pas vues depuis l'espace de travail. Corollairement, les variables de l'espace de travail que vous souhaitez utiliser dans une fonction doivent être passées en argument. La commande `global` permet de définir des **variables globales**. Elles doivent être déclarées globales au début de la fonction **et** dans l'espace de travail.

Exercice

La formule (1.4) est dite d'ordre 1 car l'erreur d'approximation de la dérivée est de la forme $\mathcal{O}(h)$. On peut obtenir une formule d'ordre 2, c'est-à-dire avec une erreur en $\mathcal{O}(h^2)$ en utlisant la formule

$$\frac{f(x+h)-f(x-h)}{2\,h} = f'(x) + \frac{h^2}{6}\,f'''(x) + \mathcal{O}(h^3) \simeq f'(x). \tag{1.5}$$

On vous demande d'écrire une fonction `Fonc2Dn1.m` qui calcule la dérivée approchée de la fonction sinus au point $x = 1$ pour le pas h en utilisant la formule (1.5)

Vous écrirez également un script `TestDn.m` dans lequel :

- vous définirez le vecteur des pas, `pas = 10.^(-[1 : 16]);`
- vous calculerez les dérivées approchées pour ces pas en utilisant les deux fonctions ;
- vous calculerez les erreurs `Err1` et `Err2` commises dans chaque cas ;
- vous représenterez ces erreurs en fonction du pas avec des coordonnées logarithmiques :
  `loglog(pas, Err1, 'b-+', pas, Err2, 'r-*') ;`
- vous pourrez réfléchir et commenter cette figure !

Fonctions de plusieurs variables

On peut passer plusieurs arguments à une fonction. Supposons que l'on souhaite créer une fonction MATLAB qui calcule la fonction $F_{(a,b)}$ présentant un "pic" peu marqué au point a et un pic plus marqué au point b :

$$F_{(a,b)}(x) = \frac{2}{(x-a)^2 + 0.01} + \frac{1}{(x-b)^2 + 0.002}.$$

Les paramètres a et b seront des variables de la fonction que l'appellera par exemple `Foncapics`.

On peut encore procéder de deux façons.

- la première utilise la fonction `inline` : Essayez.
- la seconde consiste à créer le fichier `Foncapics.m`, par exemple :

```
function y = Foncapics(x,a,b);

% Cette fonction présente un pic peu marqué pour
%  x=a et un pic plus marqué pour x= b.
% L'appel se fait selon
%         >> y = Foncapics(x,a,b);
% x est une matrice ; a et b sont des scalaires.

y = 2./((x-a).^2+0.01) + 1./((x-b).^2+0.002);
```

Exercice

Crééz les fonctions `Fonc1Dn2.m` et `Fonc2Dn2.m` qui calcule la dérivé de la fonction sinus en un point x et pour un pas h à fixer par l'utilisateur.

Exercice (*)

Modifiez les fonctions précédentes de façon que l'on puisse passer un vecteur x **et** un vecteur h en arguments. Le format d'appel devra préciser :

```
%  Pour un vecteur x de longueur nx et un vecteur h
% de longueur nh les dérivées seront calculées dans
% une matrice Dn telle que size(Dn) =  [nx, nh].
```

Vous pouvez utiliser la fonction `repmat`.

Passage d'une fonction en argument

On veut maintenant calculer l'approximation de la dérivée d'une fonction à choisir par l'utilisateur.

La fonction `Fonc1Dn3.m` réalise cet objectif. Les restrictions d'utilisation sont données dans le commentaire.

```
function Dn = Fonc1Dn3(fonc,x,h);

% Approximation à l'ordre 1 de la dérivée de la
%  fonction fonc au(x) point(s) x pour le pas h.
% Appel se fait selon :
%          >> Dn = Fonc1Dn3(fonc, x, h)
% Variables en entree  :
%         - fonc est le nom de la fonction,
%         - x est scalaire ou vecteur,
%         - h, scalaire,  est le pas du calcul.

xp = x + h;
y = feval(fonc,x);
yp = feval(fonc,xp);
Dn = (yp - y)/h;
```

Le passage d'une fonction en argument mérite quelques commentaires :

- si `fonc.m` est une fonction prédéfinie de MATLAB (par exemple `sin`) ou le nom d'un fichier de fonction (par exemple `F2`), on pourra écrire l'un ou l'autre des deux appels ci-dessous :
  `>> Dn = Fonc1Dn3('F2', x, pas);`
  `>> Dn = Fonc1Dn3(@F2, x, pas);`
- si `fonc` est le nom d'une fonction définie par `inline`, (par exemple `F1`), on devra formuler l'appel selon :
  `>> Dn = Fonc1Dn3(F1, x, pas);`

A vous de créer la fonction `Fonc2Dn3.m` pour une approximation à l'ordre 2.

Arguments optionnels en entrée (*)

Le nombre de variables en entrée n'est pas nécessairement fixé. La fonction `FoncDn4` propose de choisir l'ordre de la formule d'approximation de la dérivée que l'on va utiliser. On fixe par défaut cet ordre à 1. On utilise les fonctions précédentes pour calculer les approximations de la dérivée.

```
function Dn = FoncDn4(fonc, x, h, ordre);

% Approximation à l'ordre 1 ou 2 de la dérivée de la
% fonction fonc au(x) point(s) x pour le pas h.
% Appel :
%           >> Dn = FoncDn4(fonc, x, h);
%      ou   >> Dn = FoncDn4(fonc, x, h, ordre);
% Variables en entree  :
%         - fonc est le nom de la fonction,
%         - x est scalaire ou vecteur,
%         - h, scalaire,  est le pas du calcul.
```

```
%         - ordre est optionnel ; par defaut, ordre = 1.

if nargin ==3
  ordre = 1;
end

switch ordre
   case 1
      Dn = Fonc1Dn3(fonc, x, h);
   case 2
      Dn = Fonc2Dn3(fonc, x, h);
   otherwise
      Dn = [];
      disp('Ordre non valable');
      return
end
```

Cette fonction appelle quelques commentaires :
- `nargin` renvoie le nombre de variables utilisées dans l'appel à la fonction.
- le branchement définit par `switch, case, end` n'a pas été présenté précédemment. Il permet de choisir parmi plusieurs instructions suivant la valeur d'une variable, ici `ordre`. Essayez de le remplacer par un test multiple utilisant `if, elseif ...`
- les fonctions appelées, ici `Fonc1Dn3.m` et `Fonc2Dn3.m` doivent être définies dans le même répertoire que la fonction qui les appelle.

Nombre variable d'arguments (*)

La fonction dont on veut calculer la dérivée peut dépendre de paramètres, comme la fonction $F_{(a,b)}$ ci-dessus. On utilisera dans ce cas la fonction `varargin`, comme le montre l'exemple ci-dessous :

```
function Dn = Fonc1Dn5(fonc,x,h,varargin);

% Approximation à l'ordre 1 de la dérivée de la
% fonction fonc au(x) point(s) x pour le pas h.
% Appel  :
%          >> Dn = Fonc1Dn5(fonc, x, h, p1, p2,...) ;
% Variables en entree  :
%         - fonc est le nom de la fonction,
%         - x est scalaire ou vecteur,
%         - h, scalaire,  est le pas du calcul.
%         - p1, p2,  eventuels parametres de fonc

xp = x + h;
y = feval(fonc,x,varargin{:});
```

```
yp = feval(fonc,xp,varargin{:});
Dn = (yp - y)/h;
```

`varargin` désigne une liste de longueur quelconque de variables qui peuvent être de tout format ; elle est gérée comme une **cellule de tableaux**.

Les cellules de tableaux sont un des moyens proposés par MATLAB pour gérer une collection de variables de types et formats différents. Les lignes suivantes définissent une cellule de tableaux et font s'afficher quelques-uns de ses éléments :

```
>> C ={1:3, rand(2),'coucou'};
>> C{1}
ans =
     1     2     3
>> c = C{3}
c =
coucou
```

Les éléments d'une cellule sont ordonnés comme ceux d'une matrice, par ligne et par colonne ; on y fait référence par l'utilisation d'accolades au lieu de parenthèses.

Exercice

Ecrivez une fonction `Fonc2Dn5.m` qui fournisse une approximation à l'ordre 2 de la dérivée de la fonction argument en autorisant cette fonction à dépendre de paramètres.

Ecrivez ensuite une fonction `FoncDn6.m` qui permette de choisir l'ordre, mais ne l'impose pas. Son appel se fera selon

```
>> Dn = FoncDn6( fonc, x, h, ordre, p1, p2, ...)
```

Lorsque l'on ne veut pas fixer l'ordre et que la fonction `fonc` dépend de paramètres, on passera la matrice vide `[]` pour la variable `ordre`.

Plusieurs arguments en sortie

On peut redéfinir la fonction `F2` de façon qu'elle fournisse aussi la dérivée F' de la fonction F :

$$F'(x) = 0.01\, e^x - 10\, \sin x - 3.$$

Cela peut se faire de la façon suivante :

```
function [y, yprime] = F2bis(x);
% fichier fonction définissant la fonction
% F(x) = 0.01*exp(x)+10*cos(x)-3*x;
% ainsi que sa derivee.
% L'appel se fait selon :
%                       y = F2bis(x);
```

```
% et renvoie alors seulement les valeurs de F, ou selon
%                    [y, yprime] = F2bis(x);

y = 0.01*exp(x)+10*cos(x)-3*x;
yprime  = 0.01*exp(x) -10*sin(x) -3;
```

Les deux arguments en sortie sont placés entre crochets. Dans les deux formats d'appel, on calcule $F(x)$ et $F'(x)$. On peut ne faire calculer que $F(x)$ si l'on ne veut pas les valeurs de la dérivée :

```
function [y, yprime] = F2ter(x);
% fichier fonction définissant la fonction
% F(x) = 0.01*exp(x)+10*cos(x)-3*x;
% ainsi que sa derivee.
% L'appel se fait selon :
%                      y = F2ter(x);
% et renvoie alors seulement les valeurs de F, ou selon
%                    [y, yprime] = F2ter(x);

y = 0.01*exp(x)+10*cos(x)-3*x;
if nargout==2
  yprime  = 0.01*exp(x) -10*sin(x)-3;
end
```

1.7.3 Fichiers de sauvegarde

Il est possible de sauvegarder des variables de l'espace de travail en vue d'une utilisation ultérieure. Cela se fait à l'aide de l'instruction `save`. Le format le plus simple est le suivant :

`save nomfichier X Y Z`

Les variables `X, Y` et `Z` seront sauvegardées dans un fichier `nomfichier.mat`. On peut les retrouver par l'appel :

`load nomfichier`

Les fichiers que vous avez ainsi sauvegardés apparaissent dans le répertoire courant sous le nom `nomfichier.mat`. Ces fichiers ne sont lisibles que par MATLAB. Vous pouvez souhaiter exporter vos données pour les utiliser dans une autre circonstance. Il faut alors créer un fichier ASCII. On peut le faire en simple (par défaut) ou double précision. Essayez :

```
>> A = rand(1,3);
>> save data1 A
>> save data2.txt A -ascii
>> save data3.txt A -ascii -double
>> clear all
>> load data1
```

```
>> load data2.txt
>> whos
>> data2 - A
>> load data3.txt
>> data3 - A
```

Exercice

Nous aurons besoin, pour un prochain TD, de disposer d'un certain nombre de racines de l'équation $J_0(x) = 0$. Nous allons créer un fichier contenant les valeurs approchées des 200 plus petites racines de cette équation. Pour ce faire, vous modifierez le fichier `VisuracBess`.

1.7.4 Lecture de données extérieures

MATLAB peut lire des fichiers créés par d'autres logiciels. Nous allons prendre l'exemple de fichiers créés par EXCELL. Ces fichiers peuvent vous être fournis sous plusieurs formats :

- pour les fichiers `.xls`, vous pourrez utiliser les fonctions `xlsread` ou `importdata` ;
- pour les fichiers `.txt`, vous pourrez utiliser les fonctions `textread` ou `importdata`.

Exercice

Vous pouvez recopier à l'adresse indiquée en introduction les fichiers `Traction.xls` et `Bruits.txt`.

Vous allez lire ces fichiers depuis MATLAB. Pour chacun d'eux, vous allez récupérer une matrice de deux colonnes. La première colonne est constituées d'absisses, et la seconde d'ordonnées. Vous représenterez les courbes représentant chacun de ces échantillons.

Chapitre 2

Annexe : lexique non exhaustif

Caractères spéciaux et opérateurs logiques	
Symbole	**Usage**
=	instruction d'affectation
()	utilisées pour marquer la priorité d'une expression arithmétique ; contiennent les variables en entrée des fonctions
[ ]	utilisé pour former matrices et vecteurs ; contient les variables en sortie des fonctions
.	point décimal
..	répertoire au-dessus du répertoire courant
...	continue une suite d'instructions à la ligne suivante
;	termine une ligne ; evite l'affichage
,	sépare les éléments ou instructions d'une ligne et les variables d'une fonction
%	commentaires
:	utilisés pour la génération de vecteurs lignes d'éléments régulièrement espacés, par défaut avec un incrément entier, sinon avec un incrément à préciser ; désigne l'ensemble des indices d'un tableau selon une de ses dimensions
(:)	force le format vecteur colonne
'	se place avant et après une chaine de caractères
"	permet de définir ' à l'intérieur d'une chaine de caractères
@	identificateur de fonction depuis la version 6
!	permet l'execution d'une commande du système d'exploitation
&	ET logique ; opère élément par élément pour les tableaux avec "faux" valant 0 et toute valeur non nulle valant "vrai"
|	OU logique
~	NON logique
`xor`	OU exclusif, s'emploie sous la forme `xor(A, B)`
`any`	fonction vectorielle logique qui vaut 1 si un élément du vecteur est non nul et 0 sinon ; elle opère sur chaque colonne pour une matrice et s'emploie sous la forme `any(A)`
`all`	fonction vectorielle logique qui vaut 1 si chaque élément du vecteur est non nul ; s'emploie et opère comme la précédente pour les tableaux

Opérateurs arithmétiques	
Symbole	**Usage**
+	addition
-	soustraction
*	produit matriciel ; les dimensions doivent être compatibles (sauf le cas d'un scalaire)
.*	produit élément par élément ; les deux variables doivent avoir les mêmes dimensions
\	division à gauche, ou solution de système(s) linéaire(s) : `C = A \ B` si `A*C=B`
.\	division à gauche, élément par élément
/	division à droite, ou solution de système(s) linéaire(s) : `C = B/A` si `C*A=B`
./	division à droite, élément par élément
^	puissance de scalaire ou de matrice
.^	puissance de chaque élément d'un tableau
'	conjuguée transposée
.'	transposée

Opérateurs relationnels	
Symbole	**Usage**
<	inférieur, compare élément par élément des tableaux de même taille et prend pour valeur un tableau de cette taille , de coefficients 1 ("vrai") ou 0 ("faux")
<=	inférieur ou égal, opére comme le précédent
>	supérieur
>=	supérieur ou égal
==	égal
~=	différent

Variables et fonctions d'intérêt général	
Symbole	**Usage**
`helpdesk`	accès à une documentation hypertexte de MATLAB (version complète)
`helpwin`	fenêtre d'aide arborescente (version simple)
`help`	aide en ligne
`doc`	documentation relative à une fonction (version complète)
`what`	fournit la liste des M-fichiers du repertoire courant
`type`	affiche le contenu d'un M-fichier
`lookfor`	indique les occurences d'une chaine de caractères dans l'aide en ligne
`ans`	résultat de la dernière instruction lorsqu'il n'est pas affecté
`whos`	affiche la liste des variables courantes, et leur format
`who`	affiche la liste des variables courantes
`save`	sauvegarde une ou plusieurs variables de la session dans un fichier
`load`	retrouve les variables sauvegardées précédemment
`clear`	supprime une ou plusieurs variables de la session

Variables et fonctions d'intérêt général	
Nom	**Usage**
`function`	définit une fonction ;
`feval`	évalue une fonction dont le nom est donné en argument
`nargin`	nombre de variables en entrée d'une fonction
`nargout`	nombre d'arguments en sortie d'une fonction
`varargin`	liste d'arguments variable en entrée ; cellule de tableaux
`varargout`	liste d'arguments variable en sortie ; une cellule de tableaux
`global`	définit des variables globales
`pause`	arrète la session en attente d'une réponse de l'utilisateur
`disp`	affiche une variable sans donner son nom, ou un texte
`input`	renvoie l'utilisateur au "prompt" pour entrer une valeur
`find`	fournit les indices des éléments non nuls d'un tableau
`for`	boucle de répétition d'une suite d'instructions gérée par un compteur
`while`	boucle de répétition d'une suite d'instructions gérée par une expression logique
`if`	instructions éxécutées sous condition ; s'emboîte avec `else, elseif`
`switch`	branchement conditionné par les valeurs d'une variable ; gérée par `case`
`end`	clôt le corps d'instructions de `for`, `while` et `if`
`break`	arrète l'éxécution des boucles `for` ou `while`
`return`	renvoie vers la fonction d'appel ou l'espace de travail
`error`	affiche un message d'erreur et quitte une fonction
`size`	dimensions d'un tableau
`length`	longueur d'un vecteur
`linspace`	crée un vecteur de composantes uniformément réparties entre deux valeurs
`logspace`	crée un vecteur de composantes réparties logarithmiquement entre deux valeurs
`reshape`	change les dimensions d'un tableau (avec les mêmes éléments)
`repmat`	crée un tableau en reproduisant une matrice selon les dimensions spécifiées
`cat`	crée une matrice par concatenation
`cell`	crée une cellule de tableaux
`format`	précise le format d'affichage
`echo`	controle l'affichage des commandes éxécutées
`more`	controle le nombre de lignes de chaque page affichée
`tic`	ouvre le compteur de durée d'execution d'une suite de commandes
`toc`	ferme le compteur ouvert par `tic`
`cputime`	compteur de temps CPU
`num2str`	convertit des données numériques en chaine de caractères
`sprintf`	écrit des données formattées dans une chaine de caractères
`textread`	lit des données d'un fichier texte
`importdata`	lit des données d'un fichier de l'espace de travail
`cd`	change le repertoire de travail
`quit`	termine une session Matlab

Arithmétique, polynômes et manipulation de données	
Nom	**Usage**
`eps`	précision relative de l'arithmétique virgule-flottante utilisée
`pi`	π
`i,j`	unité imaginaire des nombres complexes
`abs`	valeur absolue ou module d'un nombre complexe
`angle`	argument d'un nombre complexe
`sqrt`	racine carrée
`real`	partie réelle
`imag`	partie imaginaire
`conj`	conjugué complexe
`gcp`	plus grand diviseur commun
`lcm`	plus petit multiple commun
`round`	arrondi à l'entier le plus proche
`fix`	arrondi vers 0
`ceil`	arrondi vers ∞
`floor`	arrondi vers $-\infty$
`rem`	reste après division entière : `rem(x,y)= x-fix(x./y).*y`
`mod`	reste signé : `mod(x,y)=x-floor(x./y).*y`
`max`	plus grande composante d'un vecteur ; pour une matrice, la fonction renvoie un vecteur ligne formé des maxima de chaque colonne
`min`	plus petite composante d'un vecteur
`sort`	trie les composantes d'un vecteur, par défaut dans l'ordre croissant
`fliplr`	inverse l'ordre des colonnes d'un tableau
`flipud`	inverse l'ordre des lignes d'un tableau
`mean`	moyenne des composantes d'un vecteur
`sum`	somme des composantes d'un vecteur
`std`	écart-type des composantes d'un vecteur
`cov`	matrice de covariance
`prod`	produit des composantes d'un vecteur
`cumsum`	sommes cumulées des éléments d'une matrice selon une dimension à préciser
`poly`	construit un polynôme de racines données
`roots`	racines d'un polynômes
`polyval`	evalue un polynôme donné par ses coefficients
`conv`	produit de polynômes, ou convolution de vecteurs
`deconv`	division polynômiale ou déconvolution de vecteurs
`polyder`	fournit le polynôme dérivé
`diff`	opérateur aux différences ; permet de construire les différénces finies ou divisées
`fft`	transformée de Fourier rapide en dimension 1
`ifft`	transformée de Fourier rapide inverse en dimension 1
`fft2`	transformée de Fourier rapide en dimension 2

Algèbre linéaire	
Nom	**Usage**
`zeros`	matrice de 0
`ones`	matrice de 1
`eye`	matrice identité
`rand`	matrice aléatoire, de coefficients uniformément distribués sur $[0, 1]$
`randn`	matrice aléatoire, de coefficients normalement distribués
`diag`	extrait la diagonale d'une matrice, ou crée une matrice diagonale ou bande
`tril`	extrait la partie triangulaire inférieure d'une matrice
`triu`	extrait la partie triangulaire supérieure d'une matrice
`inv`	inverse d'une matrice (par résolution de systèmes linéaires)
`lu`	factorisation lu d'une matrice, avec stratégie de pivot partiel
`chol`	factorisation de Choleski d'une matrice symétrique définie positive
`qr`	factorisation orthogonale-triangulaire d'une matrice
`null`	fournit une base orthonormée du noyau
`orth`	fournit une base orthonormée de l'image
`rank`	rang d'une matrice
`eig`	tous les éléments propres d'une matrice carrée
`svd`	valeurs et vecteurs singuliers d'une matrice
`pinv`	pseudo-inverse d'une matrice
`det`	déterminant
`cond`	nombre de conditionnement pour la résolution des systèmes linéaires, en norme 2
`condest`	estimation du conditionnement d'une matrice carrée, en norme 1
`norm`	norme d'une matrice (plusieurs normes sont proposées)
`trace`	trace d'une matrice
`expm`	exponentielle de matrice
`sqrtm`	racine carrée matricielle d'une matrice carrée

Analyse	
Nom	**Usage**
`exp, log, log10`	exponentielle de base e, logarithme néperien et de base 10
`sin, asin, sinh, asinh`	sinus, arcsinus, sinus hyperbolique et argsh
`cos, acos, cosh, acosh`	cosinus,...
`tan, atan, tanh, atanh`	tangente,...
`cot, acot, coth, acoth`	cotangente,...
`besselj, bessely, besselh`	fonctions de Bessel
`gamma, gammainc`	fonctions gamma
`beta, betainc`	fonctions beta
`erf, erfinv`	fonction d'erreur (ou fonction de Gauss) et sa réciproque

Graphiques et visualisations	
Nom	**Usage**
`figure`	crée une nouvelle fenêtre de visualisation
`plot`	graphe linéaire en dimension 2
`fplot`	graphe d'une fonction entre deux valeurs
`bar`	graphe en rectangles verticaux
`hist`	histograme
`pie`	graphe en camembert
`polar`	graphe en coordonnées polaires
`stem`	graphe en segments marqeé de données discrètes
`subplot`	permet de partitionner une fenêtre de visualisation
`hold`	conserve le graphique courant
`axis`	définit les bornes des axes
`title`	affiche un titre (chaine de caractères à fournir)
`legend`	affiche une légende sur la figure
`text`	affiche un commentaire (chaine de caractères à fournir) en un point donné par ses coordonnées
`xlabel`	précise le nom de la variable en absisse
`semylogy`	graphique avec une echelle logarithmique en ordonnée
`plot3`	graphe linéaire en dimension 3
`bar3`	graphe en parallèlogrames verticaux
`meshgrid`	construit un jeu de coordonnées pour la visualisation tridimensionnelle d'une fonction de 2 variables
`mesh`	visualise une surface maillée en dimension 3
`hidden`	fait apparaitre (`off`) ou disparaitre (`on`) les parties cachées d'une surface maillée
`surf`	visualise une surface ombrée (en couleurs) en dimension 3
`surfnorm`	représente les normales aux surfaces en dimension 3
`contour`	représente les isolignes d'une surface en dimension 2
`meshc`	combine `mesh` et `contour`
`clabel`	fait apparaitre les valeurs des isolignes
`quiver`	représente des gradients par des flèches
`contour3`	représente les isolignes d'une surface en dimension 3
`view`	spécifie le point de vue d'un graphe en dimension 3
`colormap`	définit le jeu de couleurs
`colorbar`	affiche une échelle des couleurs
`getframe`	forme un vecteur colonne à partir d'un graphique en vue d'une animation
`movie`	execute l'animation visuelle avec la matrice précédente
`image`	crée une image en interpretant les valeurs des coefficients d'une matrice
`print`	imprime un graphique, ou le sauvegarde en format pdf, post-script ou jpeg
`findobj`	récupère les handles des objets graphique de la session
`get`	récupère les valeurs de propriètés d'un objet identifié par son handle
`get`	fixe les valeurs de propriètés d'un objet identifié par son handle

Chapitre 3

Bibliographie

Matlab Guide, D. J. Higham, N. J. Higham, SIAM, 2000.

Introduction to Scientific Computing, A Matrix-Vector Approach Using MATLAB, C.F. Van Loan, MATLAB curriculum Series, 1997.

Apprendre et Maîtriser Matlab, versions 4 et 5 et SIMULINK^r, M. Mokhtari, A. Mesbah, Springer, 1997.