

Initiation sur MatLab

Par **Rafic YOUNES**

Enseignant – Chercheur à la Faculté de Génie – Université Libanaise
Responsable Scientifique du Master de Recherche en Mécanique 3M

R.Y. 10. 04

1. Introduction

- 1.1 Démarrage d'une session MatLab
- 1.2 Type des fichiers. Compatibilité entre DOS et Unix. Edition
- 1.3 Documentation. Aide en ligne
- 1.4 Information sur l'espace de travail
- 1.5 Coopération de MatLab avec des logiciels externes

2. Généralités

- 2.1 Entrées et traitement des données
- 2.2 Instructions de contrôle
- 2.3 Les fonctions
- 2.4 Notations et raccourcis
- 2.5 Les fichiers
- 2.6 Commandes systèmes
- 2.7 Chaînes de caractères et messages
- 2.9 Copie d'écran texte
- 2.10 Les graphiques

3. Résumé des commandes MatLab

4. Applications

- 4.1 Systèmes d'équations non linéaires
- 4.2 Traitement des polynômes
- 4.3 Interpolation linéaire et non linéaire
- 4.4 Intégration numérique
- 4.5 Différentielles totales
- 4.6 Equations aux dérivées partielles
- 4.7 Optimisation linéaire et non linéaire
- 4.8 Calcul symbolique
- 4.9 Graphiques 2D et 3D

5. Références bibliographiques

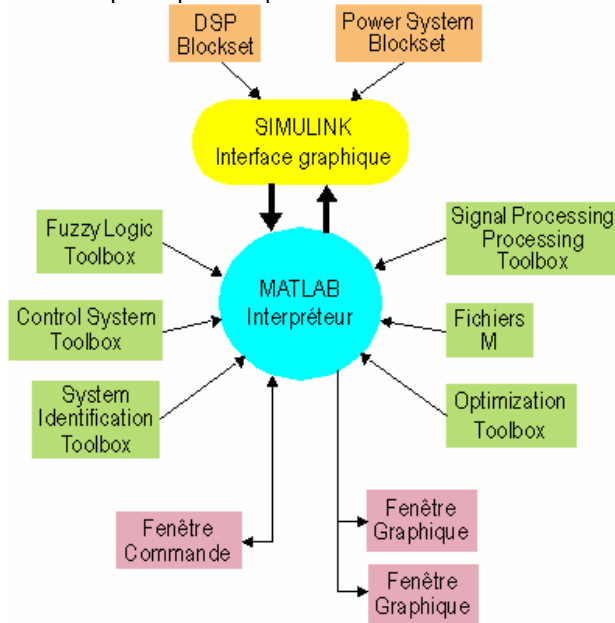
Le logiciel MatLab met à la disposition de l'utilisateur un environnement performant pour mener à bien des calculs numériques ou symboliques. Cet article constitue une initiation sur MatLab et s'adresse aux débutants qui désirent prendre rapidement connaissance des fonctionnalités de ce logiciel.

Les thèmes abordés couvrent l'essentiel de ce qu'il faut connaître pour utiliser efficacement MatLab : types de données, fonctions MatLab, ... ainsi qu'un aide-mémoire regroupant la majorité des commandes MatLab par thématique.

Enfin, une série des applications dans des domaines diverses ont été présentées. Elles sont convenables pour un utilisateur débutant ou un ingénieur confirmé.

1. Introduction

MatLab est un puissant outil de calcul numérique, de programmation et de visualisation graphique. Son nom signifie *matrix laboratory*, c'est à dire un environnement interactif de travail avec des matrices. La facilité de développement des applications dans son langage fait qu'il est pratiquement devenu le standard dans son domaine. Actuellement, on trouve des nombreuses boîtes à outils (Toolboxes) qui contiennent des fonctions spécialisées permettant d'utiliser l'environnement MatLab pour résoudre des classes spécifiques de problèmes.



Avec ses fonctions spécialisées, MATLAB peut être aussi considéré comme un langage de programmation adapté pour les problèmes scientifiques. MATLAB est un interpréteur : les instructions sont interprétées et exécutées ligne par ligne. MATLAB fonctionne dans plusieurs environnements tels que X-Windows, Windows, Macintosh.

Il existe deux modes de fonctionnement :

Mode interactif: MATLAB exécute les instructions au fur et à mesure qu'elles sont données par l'utilisateur.

Mode exécutif: MATLAB exécute ligne par ligne un "fichier M" (programme en langage MATLAB).

Les blocks de la figure précédente sont présentés comme suit :

Fenêtre Commande: Dans cette fenêtre, l'utilisateur donne les instructions et MATLAB retourne les résultats.

Fenêtres Graphique: MATLAB trace les graphiques dans ces fenêtres.

Fichiers M: Ce sont des programmes en langage MATLAB (écrits par l'utilisateur).

Toolboxes: Ce sont des collections de fichiers M développés pour des domaines d'application spécifiques (Signal Processing Toolbox, System Identification Toolbox, Control System Toolbox, u-Synthesis and Analysis Toolbox, Robust Control Toolbox, Optimization Toolbox, Neural Network

Toolbox, Spline Toolbox, Chemometrics Toolbox, Fuzzy Logic Toolbox, etc.)

Simulink: C'est l'extension graphique de MATLAB permettant de travailler avec des diagrammes en blocs.

Blocksets: Ce sont des collections de blocs Simulink développés pour des domaines d'application spécifiques (DSP Blockset, Power System Blockset, etc.).

1.1 Démarrage d'une session MatLab.

Le lancement de MatLab sur un PC se fait tout simplement en l'appelant avec la commande **MatLab** dans une fenêtre. Cette fenêtre deviendra alors la fenêtre de commande MatLab. Sous Windows, il suffit de cliquer sur l'icône correspondante pour qu'une fenêtre similaire s'ouvre dans votre environnement. Il s'agit en effet d'une interface en mode texte avec l'interpréteur de langage MatLab dans laquelle on peut exécuter des instructions spécifiques ou lancer des programmes.

L'affichage des résultats numériques se fera en mode texte dans cette même fenêtre ou dans une autre fenêtre en mode graphique (en faisant appel aux fonctions de visualisation de MatLab).

1.2 Types des fichiers. Compatibilité entre DOS et UNIX. Edition.

Les programmes MatLab sont des fichiers ASCII qui contiennent des commandes spécifiques. La convention est que le nom d'un tel fichier doit se terminer en **.m**. Pour exécuter un tel programme en MatLab il suffit de taper son nom dans la fenêtre de commande, mais sans l'extension par défaut **.m**. Les fichiers sont parfaitement compatibles entre les stations et les PC. C'est à dire, en récupérant par ftp des fichiers qui étaient sur une station UNIX, ces fichiers resteront toujours exécutables par MatLab PC (même procédure pour le transfert PC -> UNIX). Une seule petite observation : il faut configurer le transfert ftp en mode ASCII, sinon (en binaire) la convention de fin de ligne qui est différente sous UNIX et DOS va faire un programme illisible. Il est possible d'employer le mode de transfert binaire seulement pour les fichiers qui ont des noms terminés en **.mat**. Ce sont des fichiers des données MatLab, qui utilisent le format binaire pour des raisons d'efficacité et de taille.

L'édition de programmes **.m** est un processus totalement indépendant de MatLab. L'utilisateur peut favoriser son propre logiciel d'édition.

1.3 Documentation. Aide en ligne.

La documentation de MatLab est particulièrement riche. Le logiciel est livré en standard avec plusieurs livres dont :

Users Guide - destiné surtout aux débutants. Cette documentation guide lentement dans les méandres du langage MatLab. Pour un débutant, le passage par cette documentation est quasiment inévitable.

Reference Guide - tout comme son nom l'indique il s'agit d'un document qui décrit tous les éléments de langage MatLab (instructions, fonctions et leurs paramètres). Grâce à son index très complet, il permet de retrouver facilement des informations.

Building a Graphical User Interface - il s'agit d'un document qui traite les fonctions MatLab d'interaction

graphique avec l'utilisateur. Parmi toutes les documentations, c'est de loin le livre le moins complet. Heureusement, il contient de nombreux exemples qui peuvent conduire, par le biais de l'expérimentation, à une parfaite maîtrise de l'interface graphique (si cela est vraiment nécessaire).

External Interface Guide - c'est un des ouvrages les plus utiles parce qu'il traite l'interaction de MatLab avec des programmes externes, notamment celles écrites en langage C.

Signal Processing Toolbox User's Guide - le guide d'utilisation de la boîte à outils de traitement de signal. Il contient deux parties : *Tutorial* et *Reference*, conçues de même manière que les deux documentations correspondantes MatLab.

Image Processing Toolbox User's Guide - le guide d'utilisation de la boîte à outils de traitement d'image.

Statistical Toolbox User's Guide - le guide d'utilisation de la boîte à outils de statistique.

De même, l'aide en ligne est très complète. Il suffit de taper dans la fenêtre de commande :

» help

et le résultat sera une énumération de ce type :

HELP topics:

apps\MatLab - Establish MATLAB session paramètres.

MatLab\general - General purpose commands.

MatLab\ops - Operators and special characters.

MatLab\lang - Language constructs and debugging.

MatLab\elmat - Elementary matrices and matrix manipulation.

MatLab\specmat - Specialized matrices.

MatLab\elfun - Elementary math functions.

MatLab\specfun - Specialized math functions.

MatLab\matfun - Matrix functions - numerical linear algebra.

MatLab\datafun - Data analysis and Fourier transform functions.

MatLab\polyfun - Polynomial and interpolation functions.

[...]

For more help on directory/topic, type "help topic".

Il s'agit d'une énumération de toutes les sections de l'aide en ligne. Pour une aide sur une certaine section il suffit de taper **help** suivi du nom de la section. Par exemple, la commande

» help ops

aura comme résultat une liste de tous les opérateurs utilisés par MatLab.

Pour une aide encore plus pointue, la syntaxe est **help** suivi par le nom de la commande. Supposant qu'on veut voir l'utilisation de la commande **fft2**, il suffit d'écrire:

» help fft2

Une autre commande utile est

» lookfor strings

qui recherche toutes les fonctions MatLab ou utilisateur qui contiennent dans leur help la série de caractères 'string'.

L'information fournie par l'aide en ligne est plus ou moins équivalente avec celle contenue dans le *Reference Guide*.

Une autre source d'informations extrêmement riche est le programme de démonstration de MatLab. On peut le lancer avec la commande **demo**. Les sources du demo sont disponibles dans le répertoire C:\MATLAB\TOOLBOX\MATLAB\DEMOS\ (pour les

PC) ou /u/MatLab/toolbox/MatLab/demos/ (pour les stations).

1.4 Information sur l'espace de travail

Pour obtenir une liste des variables dans l'espace de travail, on utilise les instructions suivantes:

who Affichage des variables dans l'espace de travail.

whos Affichage détaillé des variables dans l'espace de travail.

1.5 Coopération de MatLab avec des logiciels externes

1.5.1 Accès aux fichiers de données externes

En dehors du mécanisme classique de sauvegarde et du chargement des données de type **load/save**, MatLab possède des mécanismes alternatifs.

La première variante est implantée toujours en utilisant les commandes **load/save**. Il s'agit de sauver une matrice sur le disque dur, en format ASCII. Il faut seulement rajouter à la fin le commutateur **-ascii**. Exemple :

» save toto -ascii

Dans le fichier de données, les éléments seront séparés par un espace, et les lignes par un saut de fin de ligne. Ce format a plusieurs avantages et plusieurs défauts. Comme avantage, on peut citer le fait que le format ASCII est interprétable par la simple inspection visuelle du fichier avec un éditeur de texte. Nul besoin de connaître le format interne MatLab, et c'est extrêmement facile de programmer des procédures C pour écrire ou lire un tel fichier ou même pour le convertir dans le format Maple ASCII. Par contre, au fur et à mesure que la matrice devient grande, la manipulation d'un tel fichier s'alourdit (ça peut constituer un gros problème surtout pour ceux qui ont des petits désagréments avec leur quota de disque dur).

Enfin, MatLab sait utiliser de fonctions de manipulation des fichiers façon C. Il s'agit des fonctions classiques genre **fopen**, **fclose**, **fwrite**, **fseek**, **fprintf**, etc. On peut ainsi lire ou générer un format binaire ou n'importe quel format exotique.

1.5.2 Utiliser des procédures C dans l'environnement MatLab

La méthode la plus simple est d'utiliser le "escape shell" !. La commande

»!mon_prog;

indique à MatLab qu'il faut exécuter un logiciel externe qui s'appelle **mon_prog**. Une fois le logiciel externe fini, l'exécution suit son cours en MatLab. La communication entre l'environnement et **mon_prog** peut se réaliser par l'intermédiaire de fichiers des données (voir la section précédente). Généralement, on essaye d'implanter avec **mon_prog** des sections critiques du programme qui prendrait beaucoup plus de temps et/ou de mémoire en MatLab. L'avantage est qu'à la fin on garde sur le disque dur tous les résultats intermédiaires et on peut les examiner tranquillement. Parmi toutes les méthodes de collaboration avec des programmes externes, celle ci est la plus inefficace.

Un peu plus avancée, mais plus complexe, est l'utilisation des fichiers **.mex**. Ce sont des fichiers

écrits en langage C, compilées avec un compilateur spécial **cmex**, qui peuvent ultérieurement être appelées comme n'importe quelle fonction. Ces sources doivent inclure le fichier entête "**mex.h**". Cette démarche est particulièrement lourde, car elle comporte aussi la création d'une fonction pont (*gateway routine*). Des nombreux problèmes de compatibilité de **.mex** entre différentes versions de MatLab se posent. "Ne pas utiliser les MEX que si vous avez absolument besoin" - je cite la documentation de MatLab.

1.5.3 Utiliser les fonctions MatLab dans le cadre d'un logiciel C

C'est la modalité la plus efficace de faire coopérer MatLab et un logiciel externe. Il s'agit de lancer en tâche de fond en même temps que votre logiciel C, le noyau *MatLab Engine*. Vous pouvez faire appel à ce noyau pour lire des fichiers des données en format MatLab, pour inverser des matrices ou faire des graphiques 3D. Une licence MatLab sera occupée seulement pendant que le noyau tourne en tâche de fond et sera libérée dès que le contrôle total revient au logiciel principal.

Pour utiliser les fonctions de MatLab dans les programmes en C il faut modifier le **Makefile** en rajoutant aux lignes de définitions du compilateur et de l'éditeur de liaisons :

```
CC = cc ..... -I/u/MatLab/extern/include
```

Sous SunOS:

```
LD = cc ..... /u/MatLab/extern/lib/sun4/libmat.a -lm
sous Solaris:
```

```
LD = cc ..... /u/MatLab/extern/lib/sol2/libmat.a -lm
```

Dans le logiciel C il faut inclure la ligne d'entête suivante

```
#include "engine.h"
```

Le noyau MatLab s'appelle avec la procédure **engOpen()** et il se ferme avec la procédure **engClose()**. Une liste détaillée de toutes les fonctions de communication avec le noyau MatLab se trouve dans *External Interface Guide*, page 2-1.

En ce qui concerne le PC, Borland C++ ne fait pas partie des compilateurs indiqués par Mathworks. Le seul compilateur qui pourrait éventuellement coopérer avec MatLab est le **mcc**.

2. Généralités sur MatLab

2.1 Entrée et traitement des données

2.1.1 Matrices

Comme il a déjà été mentionné MATLAB travaille essentiellement sur des objets de type matriciel qui peuvent être réels ou complexes. Par conséquent un scalaire est une matrice 1×1 et un vecteur une matrice N×1.

Les matrices peuvent être définies de plusieurs façons:

- sous forme d'une liste explicite: $A = [12 \ 1 \ 4; 8 \ 5 \ 13; 7 \ 9 \ 2]$ est la matrice :

$$A = \begin{bmatrix} 12 & 1 & 4 \\ 8 & 5 & 13 \\ 7 & 9 & 2 \end{bmatrix}$$

- qui peut aussi être définie par:

$$A = \begin{bmatrix} 12 & 1 & 4 \\ 8 & 5 & 13 \\ 7 & 9 & 2 \end{bmatrix}$$

Les éléments d'une même ligne sont séparés par un espace ou une virgule, les lignes sont elles-mêmes séparées par un point virgule ou un retour chariot.

Une matrice à éléments complexes sera définie de manière très simple :

Soit $A = [12 \ 1; 4 \ 8] + i*[5 \ 3; 9 \ 7]$

Soit $A = [12+5i \ 1+3i; 4+9i \ 8+7i]$

Les imaginaires purs i ou j peuvent être utilisés indifféremment. *Attention de ne pas insérer d'espace dans la définition d'un nombre complexe.*

- générées par une suite d'instructions :

Certaines fonctions de MATLAB génèrent automatiquement des matrices, ce sont par exemple les fonctions **magic** ou **randn**. Ecrire **A=magic(n)** va créer un carré magique **A** de dimension $n \times n$. Ecrire **A=randn(m,n)** va créer une matrice $m \times n$ dont les éléments seront aléatoirement distribués suivant une loi normale centrée.

- chargées à partir d'un fichier.

Chaque élément d'une matrice est accessible par ses indices écrits entre parenthèses. L'élément de la 3^{ème} ligne et 4^{ème} colonne est $A(3,4)$. Pour un vecteur x la 2^{ème} composante est $x(2)$. *Attention les indices sont forcément strictement positifs. L'indexation des éléments d'un tableau commence toujours à 1.*

2.1.2 Lignes d'instructions

MATLAB utilise un langage interprété. Chaque expression écrite est interprétée et évaluée. La syntaxe est généralement de la forme:

- $variable = expression ;$
- $expression ;$

Le point virgule de terminaison de ligne indique si le résultat de l'évaluation est affiché ou non à l'écran selon qu'il est absent ou présent. Cela facilite la mise au point en offrant la possibilité d'obtenir très simplement des résultats intermédiaires de calcul.

Sous la première forme l'expression est évaluée et le résultat assigné à la variable définie. Sous la deuxième forme l'expression est évaluée et le résultat est assigné à une variable interne appelée **ans**.

Différentes expressions, séparées par des virgules ou des points virgules, peuvent apparaître sur une même ligne.

Une ligne d'instructions est généralement terminée par un retour chariot, cependant pour des lignes trop longues on peut répartir sur plusieurs lignes en utilisant comme indicateur de continuation une suite d'au moins 3 points.

Ainsi: $x = [1 \ 2 \ 3 \ 4]$ est équivalent à: $x = [1 \ 2 \ \dots \ 3 \ 4]$

Il y a distinction entre majuscule et minuscule. Ainsi **variable** est différent de **Variable**.

L'exécution peut être arrêtée par l'utilisation de CTRL_C ou CTRL_BREAK.

Pour visualiser l'état d'une session, la commande **who** ou **whos** renvoie la liste des variables existantes ainsi que leur type (réel ou complexe) et leur taille. Pour libérer de l'espace mémoire on peut éliminer une variable par **clear nom_de_la_variable**.

Si l'on veut sauvegarder simplement la totalité des variables avant de quitter une session MATLAB, la commande **save** sauvegarde l'ensemble dans un fichier appelé *matlab.mat* que l'on pourra recharger avec la commande **load** à la session suivante.

2.1.3 Opérations sur les Matrices

Les opérations suivantes sont directement accessibles:

+	addition
-	soustraction
*	multiplication
^	élévation à la puissance
'	transpose conjugué
\	division à gauche
/	division à droite

Il y a vérification des dimensions des éléments mis en jeu pour chaque opération et un message d'erreur est délivré en cas de problème. Un seul cas échappe à cette règle c'est celui d'une opération entre un scalaire et une matrice pour lequel l'opération (+, -, *, /) a lieu entre le scalaire et chacun des éléments de la matrice. Les deux divisions possibles sont définies comme suit: Si A est une matrice carrée inversible et b est un vecteur colonne (resp. ligne) de dimension compatible, alors:

$x = A \setminus b$ résout le système $A * x = b$ (resp. $x = b / A$ résout le système $x * A = b$).

Dans le cas de la division à gauche si la matrice A n'est pas carrée, elle est factorisée selon la méthode d'orthogonalisation de Householder et les facteurs sont utilisés pour résoudre le système sur ou sous dimensionné au sens des moindres carrés.

On pourra aisément vérifier que les divisions à gauche et à droite sont liées par: $b / A = (A \setminus b)'$.

Les opérations *, , et / peuvent agir élément par élément si on les fait précéder d'un point. On pourra vérifier la différence de résultat entre $A * B$ et $A . * B$.

2.2 Instructions de contrôle

Comme dans la plupart des langages il existe des instructions de contrôle de la forme **for**, **while** ou **if**. On notera ici qu'en raison du caractère interprété du langage il faut, dans la mesure du possible, éviter de les utiliser et les remplacer par la notion de boucle implicite que l'on verra plus loin.

2.2.1 FOR

La syntaxe est de la forme:

```
for compteur = début : pas : fin, ou for Matrice
expression,
expression;
end
```

La séquence suivante:

```
>> x = []; for k = 1:n, x = [x, 2*k], end
```

ou de manière équivalente:

```
x = [];
for k = 1:n,
x = [x, 2*k],
end
```

va créer un vecteur x de longueur n .

Le vecteur renversé serait créé par:

```
x = [];
```

```
for k = n:-1:1,
```

```
x = [x, 2*k],
```

```
end
```

On remarquera ici qu'il existe des fonctions MATLAB permettant de manipuler simplement les lignes ou colonnes de matrices. Ainsi renverser les composantes d'un vecteur s'effectue simplement en écrivant $y = \text{flipud}(x)$. **flipud** étant la contraction de flip up down. On trouvera de même **fliplr** (pour left right) etc...

Si on utilise une matrice au lieu d'un compteur la boucle est exécutée autant de fois que le nombre de colonnes de la matrice. Exemples:

$s = 0;$	$s = 0;$
$A = \text{randn}(3)$	$A = \text{randn}(2)$
$\text{for } \text{cmpt} = A$	$\text{for } \text{cmpt} = A$
$s = s+1$	$s = s+\text{sum}(\text{cmpt})$
end	end

2.2.2 WHILE

La boucle est répétée tant que la relation reste vraie. La syntaxe est de la forme:

```
while relation
expression;
end
```

Supposons que l'on dispose d'une observation de signal représentée par un vecteur x de taille N et que l'on veuille en calculer le spectre sur M points où $M = 2^n \geq N$. M peut être calculé par la séquence suivante:

```
n = 0;
while 2^n >= size(x)
n = n+1;
end
M = 2^n
```

Il suffira alors d'exécuter $y = \text{fft}(x, M)$ pour obtenir le résultat désiré, la fonction **fft** complétant le vecteur x par $M-N$ zéros.

2.2.3 IF

La syntaxe peut revêtir les formes:

1. **if** relation
expression;
end
2. **if** relation
expression;
else
expression;
end
3. **if** relation
expression;
elseif relation
expression;
else
expression;
end

L'expression n'est exécutée que si la relation est vraie. La notion de relation au sens de MATLAB est donnée dans le paragraphe suivant.

2.2.4 Relations

Les opérateurs de relation sont:

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

Les évaluations de relations peuvent elles-mêmes être manipulées par des opérateurs logiques tels que & (et), | (ou) et ~ (non). Quand une relation intervient entre des scalaires, le résultat est un scalaire qui vaut 1 ou 0 suivant que la relation est vraie ou fausse. Quand une relation intervient entre deux matrices de même dimension, le résultat est une matrice constituée de 1 et de 0 suivant que la relation entre les éléments correspondants des matrices est vraie ou fausse.

Si on utilise une relation entre matrices dans un **while** ou un **if** l'interprétation sera vraie si tous les éléments de la matrice résultante de l'évaluation de la relation sont à 1. Suite à cette interprétation il faudra prendre quelques précautions d'usage. Ainsi le programme suivant:

- if A ~= B
 expression
end

résultera dans l'exécution de *expression* si et seulement si tous les éléments de mêmes indices dans A et B sont différents. Ce qui n'est pas forcément le résultat escompté.

Pour se ramener à la différence au sens mathématique du terme il faudra utiliser l'instruction **any** et le programme devient:

- if **any**(**any**(A ~= B))
 expression
end

On applique deux fois **any**, car **any** est un opérateur vectoriel qui s'applique aux matrices dimension par dimension.

2.3 Les Fonctions

On peut en distinguer trois types: les fonctions scalaires, les fonctions vectorielles et les fonctions matricielles.

2.3.1 Les fonctions scalaires

Comme leur définition l'indique elles agissent principalement sur des scalaires. On peut citer sans les commenter: **sin asin cos tan atan exp log abs sqrt sign round** etc...

Cependant on peut aussi les faire agir sur des matrices. Dans ce cas elles agissent élément par élément, ainsi **sin(A)** est la matrice où chaque élément est la racine carrée de chaque élément de A.

2.3.2 Les fonctions vectorielles

Elles sont prévues pour opérer sur des vecteurs lignes ou colonnes. On peut citer sans les commenter: **max min sum prod mean std any** etc...

De même que pour les fonctions scalaires, on peut les faire opérer sur des matrices. Dans ce cas elles agissent sur chacune des colonnes de la matrice. Pour obtenir une action ligne par ligne il suffit d'utiliser l'opérateur de transposition. Ainsi si A est une matrice,

max(A) renvoie un vecteur ligne dont chaque composante correspond au maximum de chacune des colonnes de A, alors que **max(max(A))** renvoie l'élément maximum de A.

2.3.3 Les fonctions matricielles

La plupart des fonctions de MATLAB opèrent sur des matrices et c'est ce qui fait en grande partie la puissance du logiciel. On peut citer **eig** (vecteur propre, valeur propre), **chol** (décomposition de Cholesky), **svd** (décomposition en valeurs singulières), **lu** (factorisation LU), **qr** (factorisation QR), **det** (déterminant), **rank** (rang) etc...

Ces fonctions peuvent renvoyer un ou plusieurs arguments. Ainsi **[X, V] = eig[A]**, renvoie une matrice X dont les colonnes sont les vecteurs propres de A et une matrice diagonale V constituée des valeurs propres de A. Alors que **X = eig[A]** renvoie un vecteur colonne X dont les composantes sont les valeurs propres de A.

2.3.4 Matrices creuses

MATLAB est particulièrement gourmand en place mémoire, aussi a-t-il été prévu de ne stocker que les éléments non nuls des matrices. La manipulation s'effectue avec les commandes **sparse** et **full**. Ainsi **sA = sparse(A)** génère une matrice sA qui ne comporte (en mémoire) que les éléments non nuls de A. On peut revenir à A par **A = full(sA)**.

On consultera l'aide de **spdiags**, **speye**, **sparse**, **spones** et **sprandn**.

2.4 Notations et raccourcis

2.4.1 Boucle implicite

MATLAB étant un langage interprété l'utilisation de boucles ralentit considérablement l'exécution. Pour pallier cet inconvénient il existe la notion de boucle implicite qui, elle, est exécutée vectoriellement donc plus rapidement. Ainsi écrire **n = 1:5**, revient à créer le vecteur ligne **n = [1 2 3 4 5]**. On peut changer le pas d'incrémentement en écrivant **n = 1:2:7**, ce qui équivaut à **n = [1 3 5 7]**. Ceci s'étend aux nombres non entiers (**x = .1:.01:1.**) et le pas d'incrémentement peut être négatif.

Exemples:

Soit le vecteur **t = [-1;0;1;3;5]** et la matrice A de Vandermonde construite à partir de t. La construction directe en est:

```
n = length(t);
for j = 1:n
for i = 1:n
A(i,j) = t(i)^(n-j);
end
end
```

2.4.2 Utilisation du ':'

La notation ':' est très utile pour travailler sur des sous matrices. Nous en donnons quelques variantes ci-dessous.

- **A(1:3,4)** est le vecteur colonne dont les composantes sont les trois premiers éléments de la quatrième colonne de la matrice A.
- **A(:,2)** est la deuxième colonne de la matrice A
- **A(1:3,:)** est la matrice constituée des trois premières lignes de A
- **A(:,[3 6])** est la matrice à deux colonnes qui correspondent aux colonnes 3 et 6 de A.

- `A(:,[3 6]) = B(:,1:2)` remplace les colonnes 3 et 6 de A par les deux premières colonnes de B.
etc...

2.5 Les fichiers

MATLAB peut exécuter une suite d'instructions figurant dans des fichiers. De tels fichiers sont appelés fichiers M ou '*M-files*' car leur extension est toujours *.m*. Ces fichiers sont créés à l'aide d'un éditeur de texte standard. Il en existe deux types: les fichiers textes ou *scripts* et les fichiers fonctions.

De même pour les fichiers de données, on trouve des fichiers d'un format spécifique à MATLAB comportant un en-tête que l'on crée par la commande **save** et que l'on charge par la commande **load**.

2.5.1 Fichiers script

Un fichier *script* est une suite d'instructions MATLAB. Taper le nom du fichier sans son extension sur la ligne de commande MATLAB résulte en l'exécution des instructions contenues dans le fichier. Un *M-file* peut en appeler un autre ou s'appeler récursivement.

Il est cependant plus courant d'utiliser des fichiers fonctions.

2.5.2 Fichiers fonction

Les fichiers fonctions permettent d'étendre les possibilités de MATLAB. Ils sont l'équivalent des SUBROUTINE FORTRAN, des fonctions C etc... Ces fonctions utilisent des variables qui par défaut sont locales, mais peuvent être déclarées **global**. La syntaxe est de la forme:

- `function [out1, out2, ...] = function_name(in1, in2, ...)`

où *out_i* désigne une variable de sortie, et *in_i* désigne une variable d'entrée.

Exemple 1:

```
function a = aleat(m,n,min,max)
% aleat(m,n) renvoie une matrice m×n dont les
% éléments suivent
% une loi normale centrée.
% aleat(m;n;min,max) renvoie une matrice m×n dont
% les éléments
% suivent une loi uniforme entre min et max.
if nargin 3,
a = randn(m,n);
else
a = (max - min + 1)*rand(m,n) + min;
end
```

Ce script sera sauvegardé dans un fichier nommé `aleat.m`, le nom du fichier et de la fonction doit être le même. Les lignes précédées d'un % qui suivent la déclaration de la fonction sont utilisées comme aide. On y accède par **help aleat**.

Cette fonction est appelée sous la forme `x = aleat(2,3)`, ou sous la forme `x = aleat(2,5,-2,2)`. La fonction `nargin` est utilisée pour tester le nombre d'arguments d'entrée. On peut également utiliser `nargout` pour traiter le nombre d'arguments de sortie.

Exemple 2:

```
function [moyenne, ecart-type] = stat(x)
% Si x est une matrice elle est traitée colonne par
% colonne
% moyenne est le vecteur des moyennes
```

% ecart-type est le vecteur des ecart-type.

```
[m,n] = size(x);
if m == 1
m = n;
end
moyenne = sum(x) / n;
ecart-type = sqrt(sum(x.2)/m - moyenne.2);
```

On peut facilement obtenir le listing d'un *M-file* en exécutant **type file_name** sur la ligne de commande MATLAB.

2.5.3 Visibilité des variables

Seules les variables d'un script sont affichables à l'aide de **who** ou **whos**, les variables internes à une *function* sont locales au même titre que dans le langage C. On peut cependant rendre des variables *globales*, ce qui évitent de les transmettre à éventuellement plusieurs fonctions.

2.5.4 Fichiers de données

Les fichiers MATLAB

Il est possible à tout instant de sauvegarder une ou plusieurs variables dans un fichier spécifique MATLAB dont l'extension sera *.mat*. La commande à utiliser est **save**. Les différents formats et options sont listés ci-dessous:

save fname X sauve seulement X.

save fname X Y Z sauve X, Y, et Z.

save fname X Y Z -ascii utilise le format ASCII 8-digit au lieu du binaire.

save fname X Y Z -ascii -double utilise le format ASCII 16-digit.

save fname X Y Z -ascii -double -tabs utilise le format ASCII 16-digit délimité par des *tabs*.

Pour recharger un tel fichier on utilisera **load fname**. Chaque variable ayant été sauvegardée par son nom, les variables rechargées auront les noms, dimensions et valeurs au moment de la sauvegarde.

2.5.5 Les fichiers standards

On appelle fichier standard tout fichier crée à partir d'un langage de programmation que ce soit un fichier texte ou binaire. Ces fichiers sont accessibles en lecture ou écriture à l'aide d'une syntaxe proche de celle du langage C. L'ouverture s'effectue en allouant un pointeur de fichier par la fonction **fopen**. Les lecture et écriture s'effectuent à l'aide des fonctions **fread** et **fwrite**. La fermeture du fichier est réalisée par **fclose**.

- Ouverture de fichier: `fid = fopen('fname', permission)`

'*fname*' est une chaîne de caractères spécifiant le nom du fichier. Permission prend les formes suivantes:

'**r**' lecture '**w**' écriture (création si nécessaire)

'**a**' ajout (création si nécessaire)

'**r+**' lecture et écriture (pas de création)

'**w+**' troncation ou création pour lecture et écriture

'**a+**' lecture et ajout (création si nécessaire)

Par défaut les fichiers sont ouverts en mode binaire.

Pour les ouvrir en mode texte il faut utiliser '**rt**' ou '**wt**'.

- Lecture d'un fichier: `[a, count] = fread(fid,size,precision)`

Lecture des données binaires du fichier spécifié par `fid` et écriture dans la matrice `a`.

L'argument de sortie optionnel `count` renvoie le nombre d'éléments lus.

L'argument *size* est optionnel; s'il n'est pas spécifié ou si il vaut **inf**, le fichier est lu entièrement; s'il est spécifié, les formes valides sont: **N** pour lire N éléments dans un vecteur colonne, **inf** pour lire tout le fichier, **[M,N]** pour remplir une matrice colonne par colonne. Dans ce dernier cas **N** peut être **inf**, mais pas **M**.

L'argument *precision* désigne le type de données à lire. C'est une chaîne de caractères qui prend les formes suivantes:

'char', 'schar', 'short', 'int', 'long', 'float', 'double', 'uchar', 'ushort', 'uint', 'ulong' etc...

Écriture d'un fichier: count = fwrite(fid,a,precision)

Écriture des données binaires de la matrice *a* dans le fichier spécifié par *fid*.

L'argument de sortie optionnel *count* renvoie le nombre d'éléments effectivement écrits.

L'argument *precision* désigne le type de données à lire. C'est une chaîne de caractères qui prend les mêmes formes que pour la lecture.

Fermeture de fichier: fclose(fid)

On utilisera l'aide MATLAB pour l'utilisation de fonctions telles que **fseek**, **fscanf**, **fprintf**, etc...

2.6 Format d'affichage

Tous les calculs effectués dans MATLAB le sont en double précision. Cependant le format d'affichage des résultats peut être contrôlé par la commande **format**. Le tableau ci-dessous résume les différents formats utilisables:

format short	4 décimales (défaut)
format long	14 décimales
format short e	4 décimales notation scientifique
format long e	14 décimales notation scientifique
format rat	approximation par une fraction rationnelle
format hex	format hexadécimal
format bank	correspond à une sortie en dollars et cents
format +	+, -, espace

Un format reste actif tant qu'il n'a pas été redéfini par la commande **format**, à l'exception des commandes **format compact** et **format loose** qui servent à supprimer (resp. restaurer) les lignes blanches qui pourraient apparaître à l'affichage.

2.7 Chaînes de caractères et messages

Les chaînes de caractères sont définies entre simples 'quotes'. Ainsi **c = 'Ceci est une chaîne'** définira la variable **c** comme une chaîne de caractères que l'on pourra utiliser pour afficher un message.

L'affichage de messages s'effectue de plusieurs manières. On utilise pour cela les fonctions **disp**, **error** ou **input**. Par exemple **disp(c)** affichera la chaîne **c** à l'écran. **disp('Ceci est une chaîne')** aura le même effet.

Si on utilise **error**('Erreur dans la fonction func'), Le texte sera affiché et de plus l'exécution du programme sera arrêtée.

Enfin **Z = input**('Entrez la valeur de Z: ') affichera le texte et attendra l'entrée de la valeur qui sera validée lors de la frappe sur la touche Entrée ou Enter.

2.8 Commandes système

Certaines commandes équivalentes à des commandes système sont intégrées à MATLAB. Ainsi on pourra utiliser **delete**, **type**, **dir** etc... MATLAB étant multi-plateforme certaines commandes UNIX sont acceptées sous MS-WINDOWS c'est le cas par exemple de **pwd**, **what**, **ls**, Cependant toutes les commandes ne sont pas accessibles directement. Pour exécuter n'importe quelle commande système on la fera précéder d'un point d'exclamation !.

Il en est de même pour lancer un exécutable (fichier .exe) à partir d'une ligne de commande MATLAB.

2.9 Copie d'écran texte

La commande **diary file_name** permet de sauvegarder dans le fichier *file_name* toute la partie active de l'écran à l'exception des graphiques. La sauvegarde s'effectue sous forme de texte ce qui permet une édition ultérieure du fichier.

2.10 Les graphiques

MATLAB peut afficher des courbes planes, des courbes 3D, des surfaces maillées 3D ou des surfaces à facettes 3D. Les commandes respectives sont **plot**, **plot3**, **mesh** et **surf**.

2.10.1 Courbes planes

La commande **plot(y)** trace la courbe correspondant au vecteur *y* en fonction du numéro des composantes. **plot(x,y)** tracera le vecteur *y* en fonction du vecteur *x* si ils sont de même taille. La commande **plot** ouvre une fenêtre graphique pour tracer la courbe. Les fenêtres graphiques sont numérotées, la fenêtre active étant par défaut la fenêtre 1. Pour rendre active une fenêtre graphique on fera précéder le tracé par la commande **figure(n)**, où *n* est le numéro de la fenêtre.
 >> x = -pi : .01 : pi; y = sin(x); plot(x,y)
 tracera la fonction sinus(x) dans l'intervalle $[-\pi, \pi]$ avec un pas de 0.01.

Une commande **plot** suivante effacera le premier graphique pour tracer le nouveau dans la même fenêtre si la commande **figure** n'a pas été utilisée entre temps. Cependant il est possible de tracer des graphiques en surimpression à l'aide de la commande **hold on**. Cette commande a pour effet de conserver le graphe présent avant de tracer le suivant. La commande **hold off** annule cette possibilité.

MATLAB permet également de tracer directement à partir d'une fonction avec la commande **fplot**. Soit la fonction sinus définie dans le fichier sinus.m par:

function y = sinus(x)

y = sin(x);

En exécutant: **fplot**('sinus', [-pi pi]), on obtiendra le graphe.

Le graphe obtenu peut être agrémenté d'un titre, de légendes ou de texte. Les commandes MATLAB correspondantes sont:

- **title**('Ceci est le titre de la figure');
- **xlabel**('Légende des abscisses');
- **ylabel**('Légende des ordonnées');

- **gtext** et **text** positionnement et écriture dans la zone graphique.

Un graphe possède des attributs qui peuvent être modifiés à l'aide de commandes, on consultera l'aide de **axis** et **plot**.

Il est également possible de tracer plusieurs graphes dans la même fenêtre à l'aide de la commande **subplot** (voir aide). Enfin des représentations types sont obtenues à l'aide des commandes **polar**, **bar**, **hist**, etc...

Exemple:

On utilisera la fonction **peaks** (**help peaks**). Définir $M = \text{peaks}(20)$.

plot(M) trace 20 courbes représentant les colonnes de M . Les numéros de lignes servent d'abscisse. Rajouter un titre, une légende en abscisse et une légende en ordonnée.

Le script suivant permet un affichage multiple :

```
>> y1 = M(:,1); y2 = M(:,2);
>> subplot(211), plot(y1)
>> subplot(212), plot(y2)
```

2.10.2 Courbes 3D

La commande **plot3** est identique à la commande **plot**, elle ne fait appel qu'à une coordonnée supplémentaire. La syntaxe de base est **plot3**(x,y,z). Les axes, titres et légendes se traitent comme dans le cas des courbes planes, la commande **zlabel** est disponible..

Exemple:

```
>> t = 0:pi/50:10*pi ;
>> plot3(sin(t), cos(t), t).
```

2.10.3 Maillages et surfaces 3D

La commande **mesh** permet de visualiser des courbes 3D sous forme de surfaces maillées. La surface maillée est définie par les cotes (coordonnées z) de points situés au-dessus d'une grille rectangulaire du plan (x,y).

La commande **surf** s'utilise de la même façon mais produit une surface à facettes colorées.

Le coloriage des facettes est modifiable à l'aide de la commande **shading**. Il existe trois possibilités **shading faceted**, **shading interp** et **shading flat**. La commande **shading** s'utilise après la commande **surf**.

Le profil de couleurs d'une surface est défini par la commande **colormap**(argument) qui possède un certain nombre d'arguments prédéfinis tels que **hsv** (défaut), **hot**, **cool**, **jet**, **pink**, **copper**, **flag**, **gray** et **bone**.

L'angle de vue est géré par la commande **view**.

D'autres fonctions sont reliées à la visualisation 3D telles que **meshz**, **surfc**, **surfl**, **contour** et **pcolor**.

3. Résumé des commandes MatLab

On a regroupé par sujet d'intérêt les différentes commandes et fonctions de MatLab. Il en existe d'autres que l'on pourra trouver dans les différentes boîtes à outils (toolbox) qui viennent compléter le logiciel de base.

Gestions des commandes et des fonctions	
help	aide
what	Listing du nom des M_files présents
type	impression d'un M_file
lookfor	recherche d'une entrée dans le help
which	localise les fonctions et fichiers
demo	lance la démonstration
path	Défini les chemins d'accès aux fichiers et fonctions
cedit	paramètres d'édition d'une ligne de commande
whatsnew	affiche les fichiers README de la toolbox
info	information sur MATLAB et The MathWorks
why	renvoie une réponse aléatoire non 'neutre'

Gestion des variables et de l'espace de travail	
who	affiche les variables courantes
whos	affiche les variables courantes, format long
save	Sauve l'espace de travail sur disque
load	restaure l'espace de travail à partir du disque
clear	efface les variables et fonctions de la mémoire
pack	réorganise la mémoire
size	renvoie la taille d'une matrice
length	renvoie la longueur d'un vecteur
disp	affiche une matrice de texte

Fenêtre de commande MatLab	
clc	efface la fenêtre de commande
home	curseur en haut de l'écran
format	définit le format d'affichage
echo	affiche les instructions exécutées par un script
more	contrôle de l'affichage paginé

Commandes système	
cd	change le directory courant
pwd	affiche le directory courant
dir, ls	liste les fichiers
delete	suppression de fichiers
getenv	renvoie la variable d'environnement
!	appelle et exécute une commande système
diary	sauvegarde le texte d'une session MATLAB

Démarrer et quitter MatLab	
quit, exit	quitter MATLAB
startup	M_file de lancement de MatLab
matlabrc	M_file principal de lancement

Opérateurs sur les Matrices		Opérateurs sur les Tableaux	
+	addition	+	addition
-	soustraction	-	soustraction
*	multiplication	.*	multiplication
^	puissance	.^	puissance
/	division à droite	./	division à droite
\	division à gauche	.\	division à gauche
'	transpose conjugué		
.'	transpose		
kron	produit de Kronecker		

Opérateurs Relationnels		Opérateurs Logiques	
<	inférieur à	&	et
>	supérieur à		ou
<=	inférieur ou égal à	~	non
>=	Supérieur ou égal à	xor	ou exclusif
==	égal à		
~=	différent de		

Variables prédéfinies	
ans	réponse à une expression sans assignation
eps	précision de la virgule flottante
realmax	plus grand nombre flottant
realmin	plus petit nombre flottant positif
pi	π
i, j	$[\sqrt{-1}]$
inf	∞
NaN	Not a Number
flops	nombre d'opérations flottantes par seconde
nargin	nombre d'arguments d'entrée d'une fonction
nargout	nombre d'arguments de sortie d'une fonction
computer	type du calculateur

Caractères spéciaux	
=	assignation
[]	définition de matrices ou vecteurs;
()	gère la priorité des opérations arithmétique
.	point décimal
..	directory parent
...	indique une ligne suite
,	séparateur d'arguments ou d'instructions
;	fin de lignes (matrices) ou suppression de l'affichage
%	commentaires
:	manipulation de sous matrices ou génération de vecteurs

Intégration numérique	
quad	intégrale de Simpson
quad8	intégrale de Newton-Cotes
trapz	méthode des trapèzes

Matrices prédéfinies	
zeros	matrice de 0
ones	matrice de 1
eye	matrice identité
diag	matrice diagonale
toeplitz	matrice de Toeplitz
magic	carré magique
compan	matrice compagnon
linspace	vecteurs linéairement espacés
logspace	vecteurs logarithmiquement espacés
meshgrid	grille pour les graphiques 3D
rand	nombres aléatoires à répartition uniforme
randn	nombres aléatoires à répartition normale
hilb	Hilbert
invhilb	inverse de Hilbert (exact)
vander	Vandermonde
pascal	Pascal
hadamard	Hadamard
hankel	Hankel
rosser	matrice test pour le calcul des valeurs propres
wilkinson	matrice test pour le calcul des valeurs propres
gallery	deux matrices test spéciales

Durées et Date	
date	date courante
clock	horloge
etime	durée d'exécution
tic, toc	affiche le début et la fin d'exécution
cputime	temps CPU écoulé

Manipulation de Matrices	
diag	création ou extraction de la diagonale
rot90	rotation de 90°
fliplr	retournement gauche-droit
flipud	retournement haut-bas
reshape	redimensionnement
tril	partie triangulaire inférieure
triu	partie triangulaire supérieure
.'	transposition
:	conversion matrice → vecteur

Fonctions logiques	
exist	teste l'existence d'une variable ou d'une fonction
any	vrai si un élément est vrai
all	vrais si tous les éléments sont vrais
find	cherche l'indice des éléments non nuls
isnan	vrai si l'élément n'est pas un nombre
isinf	vrai pour tout élément infini
finite	vrai pour tout élément fini
isieee	vrai si la représentation est au format IEEE
isempty	vrai pour une matrice vide
issparse	vrai pour une matrice creuse
isstr	vrai pour une chaîne de caractères
strcmp	comparaison de deux chaînes

Instruction de contrôle	
if	test conditionnel
else	complète if
elseif	complète if
end	terminaison de if, for et while
for	instruction de répétition avec compteur
while	instruction de répétition avec test
break	interrompt une boucle for ou while
return	retour
error	affiche un message et interrompt l'exécution

Instructions spécifiques	
input	indicateur d'attente d'entrée
keyboard	considère le clavier comme un fichier script
menu	génère un menu de choix pour l'utilisateur
pause	attente
function	définition de fonction
eval	exécute un chaîne de caractère
feval	exécute une fonction définie dans une chaîne
global	définit les variables comme globales
nargchk	valide le nombre d'arguments d'entrée

Traitement du son	
saxis	modification de l'échelle d'amplitude
sound	convertit un vecteur en son
auread	lit un fichier audio au format SUN
auwrite	écrit un fichier audio au format SUN
lin2mu	conversion loi linéaire vers loi μ
mu2lin	conversion loi μ vers loi linéaire

Textes et chaînes	
string	à propos des chaînes dans MATLAB
abs	convertit une chaîne en valeur numérique
blanks	une chaîne d'espaces
eval	évalue une chaîne contenant du code MATLAB
num2str	convertit un nombre en chaîne
int2str	convertit un nombre entier en chaîne
str2num	convertit une chaîne en nombre
isstr	vrai si l'élément est une chaîne
strcmp	comparaison de chaînes
upper	conversion en majuscule
lower	conversion en minuscule
hex2num	convertit une chaîne hexadécimale en flottant
hex2dec	convertit une chaîne hexadécimale en entier
dec2hex	convertit un entier en une chaîne hexadécimale

Mise au point (debug)	
dbstop	met un point d'arrêt
dbclear	supprime un point d'arrêt
dbcont	reprend l'exécution
dbdown	change le contexte local
dbstack	affiche qui appelle qui
dbstatus	liste des points d'arrêt
dbstep	exécute une ou plusieurs lignes
dbtype	affiche un M_files avec lignes numérotées
dbup	inverse de dbdown
dbquit	sortie du mode debug

Fonctions mathématiques élémentaires	
abs	valeur absolu ou module
angle	argument d'un complexe
sqrt	racine carrée
real	partie réelle
imag	partie imaginaire
conj	complexe conjugué
gcd	PGCD
lcm	PPCM
round	arrondi à l'entier le plus proche
fix	troncature
floor / ceil	arrondi vers $-\infty$ / vers $+\infty$
sign	signe de
rem	reste de la division
exp	exponentiel
log / log10	log népérien / log décimal

Sauvegarde et copie graphique	
print	imprime ou sauve dans un fichier
printopt	configuration de l'imprimante
orient	orientation paysage ou portrait

Fonctions trigonométriques	
sin, asin, sinh, asinh	
cos, acos, cosh, acosh	
tan, atan, tanh, atanh	
cot, acot, coth, acoth	
sec, asec, sech, asech	1./cos(z), acos(1./z), 1./cosh(z), acosh(1./z)
csc, acsc, csch, acsch	1./sin(z), asin(1./z), 1./sinh(z), asinh(1./z)

Fonctions prédéfinies	
bessel	fonction de Bessel
beta	fonction beta
gamma	fonction gamma
rat	approximation par un rationnel
rats	format de sortie pour rat
erf	fonction erreur erf
erfinv	inverse de erf
ellipke	intégrale elliptique complète
ellipj	fonction elliptique de Jacobi
expint	fonction intégrale exponentielle pour n=1
log2	logarithme base 2 ou décomposition mantisse, exposant
pow2	calcule 2 puissance y

Analyse de données par colonne	
max	valeur max
min	valeur min
mean	valeur moyenne
median	valeur médiane
std	écart type
sort	tri en ordre croissant
sum	somme des éléments
prod	produit des éléments
cumsum	vecteur des sommes partielles cumulées
cumprod	vecteur des produits partiels cumulés
hist	histogramme

Animations	
moviein	initialise l'espace mémoire pour l'animation
getframe	enregistre une image pour l'animation
movie	joue l'animation

Décompositions et factorisations de Matrices	
inv	inversion
lu	décomposition LU
rref	réduction de lignes
chol	factorisation de Cholesky
qr	décomposition QR
nnls	moindres carrés non-négatifs
lscov	moindres carrés avec covariance connue
null	noyau
orth	orthogonalisation
eig	valeurs et vecteurs propres
hess	forme de Hessenberg
schur	décomposition de Schur
cdf2rdf	forme complexe diagonale vers forme réelle diagonale par blocs
rsf2csf	forme réelle diagonale par blocs vers forme complexe diagonale
balance	mise à l'échelle pour le calcul des valeurs propres
qz	valeurs propres généralisées
polyeig	polynôme aux valeurs propres
svd	décomposition en valeurs singulières
pinv	pseudo-inverse

Traitement de signal	
corrcoef	coefficients de corrélation
cov	matrice de covariance
filter	filtrage monodimensionnel
filter2	filtrage bidimensionnel
cplxpair	tri en paires complexes
unwrap	suppression des sauts de phase
nextpow2	puissance de 2 immédiatement supérieure
fft	FFT monodimensionnel (fréquences de 0 à 1)
fft2	FFT bidimensionnel
ifft	FFT inverse
ifft2	FFT inverse
fftshift	FFT (fréquences de -1/2 à 1/2)

Conditionnement	
cond	suivant norme L_2 ($(\lambda_{\max})/(\lambda_{\min})$)
rcond	$\cong 1$ bon, $\cong 0$. mauvais
condest	suivant norme L_1
norm	normes L_1, L_2, L_p et L_∞
normest	estimateur de la norme L_2
rank	rang

4. Applications

4.1 Systèmes d'équations non linéaires :

On se propose de travailler pour la recherche des racines de la fonction non linéaire suivante :

$$f(x) = e^x - 2 \cdot \cos(x)$$

Dans MATLAB, on peut obtenir la même solution avec la fonction 'fzero'. Pour cela, il faut créer le fichier.m MATLAB ('f.m', par exemple) dans lequel sera programmé **f(x)**.

```
% *****
function f=f(x)
f=exp(x)-2*cos(x)
```

```
% *****
```

Pour obtenir la solution de **f(x) = 0** au voisinage de **x = 0,5**, on exécute la commande 'fzero('f',0.5)'. Cette fonction affiche les valeurs obtenues de **f(x)** à chaque itération pour donner à la fin la solution recherchée **x***.

```
>>d=fzero('f',0.5)
f =
-8.8818e-016
```

```
d =
0.5398
```

La boîte à outil 'Optimisation Toolbox' propose les fonctions 'fsolve' et 'fsolve2' qui permettent respectivement, la recherche des racines d'une fonction dans un intervalle donné et les solutions d'équations non linéaires et de systèmes d'équations non linéaires.

Nous allons appliquer ces fonctions à la résolution d'un système de deux équations non linéaires à 2 inconnues, par exemple :

$$\begin{aligned} 2 \cdot x_1 - x_2 - e^{-x_1} &= 0 \\ -x_1 + 2 \cdot x_2 - e^{-x_2} &= 0 \end{aligned}$$

à partir de l'itération $x_0 = [-5 \ -5]$.

Il faut, d'abord, créer le fichier myfun.m MATLAB

```
% *****
function F = myfun(x)
```

```
F = [2*x(1) - x(2) - exp(-x(1));
     -x(1) + 2*x(2) - exp(-x(2))];
```

```
% *****
```

Après, on fait appel à la routine suivante.

```
>> x0 = [-5; -5];
>> [x,fval] = fsolve(@myfun, x0)
```

Après 33 évaluations de la fonction myfun, un zéro est trouvé.

```
x =
0.5671
0.5671
```

```
fval =
1.0e-006 *
-0.4059
-0.4059
```

4.2 Traitement des polynômes :

Dans MATLAB, les polynômes sont représentés sous forme de vecteurs lignes dont les composantes sont données par ordre des puissances décroissantes. Un polynôme de degré n est représenté par un vecteur de taille $(n+1)$.

Le polynôme :

$$f(x) = 8 \cdot x^5 + 2 \cdot x^3 - 3 \cdot x^2 + 4 \cdot x - 2$$

est représenté par :

```
>>f=[8 0 2 -3 4 -2]
f=8 0 2 -3 4 -2
```

La fonction 'conv' donne le produit de convolution de deux polynômes. L'exemple suivant montre l'utilisation de cette fonction. Soient :

$$\begin{cases} f(x) = 3 \cdot x^3 + 2 \cdot x^2 - x + 4 \\ g(x) = 2 \cdot x^4 - 3 \cdot x^2 + 5 \cdot x - 1 \end{cases}$$

Le produit de convolution : **h(x) = f(x) g(x)** est donné par :

```
>>f=[3 2 -1 4];
>>g=[2 0 -3 5 -1];
>>h=conv(f,g)
h =
6 4 -11 17 10 -19 21 -4
```

Ainsi, le polynôme **h(x)** obtenu est :

$$h(x) = 6 \cdot x^7 + 4 \cdot x^6 - 11 \cdot x^5 + 17 \cdot x^4 + 10 \cdot x^3 - 19 \cdot x^2 + 21 \cdot x - 4$$

La fonction 'deconv' donne le rapport de convolution de deux polynômes (déconvolution des coefficients du polynôme). L'exemple suivant montre l'utilisation de cette fonction.

Soient les mêmes fonctions précédentes **f(x)** et **g(x)** :

$$\begin{cases} f(x) = 3 \cdot x^3 + 2 \cdot x^2 - x + 4 \\ g(x) = 2 \cdot x^4 - 3 \cdot x^2 + 5 \cdot x - 1 \end{cases}$$

La division de **g(x)** par **f(x)** : **h(x) = $\frac{g(x)}{f(x)}$** est donnée

par la fonction 'deconv' :

```
>>f=[3 2 -1 4];
>>g=[2 0 -3 5 -1];
>>h=deconv(g,f)
h =
0.6667 -0.4444
```

et le polynôme **h(x)** obtenu est :

$$h(x) = 0,6667 \cdot x - 0,4444$$

Soit le polynôme suivant :

$$P(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \dots + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

Après factorisation, on a :

$$P(x) = a_n \cdot (x - r_1)(x - r_2)(x - r_3) \dots (x - r_n)$$

où r_1, r_2, \dots, r_n sont les racines du polynôme $P(x)$.

Exemple :

$$P(x) = x^4 + 2 \cdot x^3 - 7 \cdot x^2 - 8x + 12$$

Ce polynôme est équivalent à :

$$P(x) = (x - 1)(x - 2)(x + 2)(x + 3)$$

Un polynôme d'ordre n possède n racines qui peuvent être réelles ou complexes.

Le polynôme :

$$P(x) = 2 \cdot x^3 + x^2 + 4x + 5$$

qui est représenté dans MATLAB par :

```
>>P=[2 1 4 5];
```

a pour racines r_i . Pour trouver ces racines, on doit exécuter la fonction '**roots**'.

D'où :

```
>>r=roots(P);
```

et le résultat donné est :

```
>> r
r =
0.2500 + 1.5612i
0.2500 - 1.5612i
-1.0000
```

Les trois racines de ce polynôme (dont 2 sont complexes) sont données sous forme d'un vecteur colonne. Quand les racines sont connues, les coefficients peuvent être recalculés par la commande '**poly**'.

Exemple :

```
>>poly(r)
ans =
1.0000 0.5000 2.0000 2.5000
```

La fonction '**poly**' accepte aussi une matrice comme argument dont elle retourne le polynôme caractéristique.

Exemple :

```
>>A=[3 1;2 4];
>>p=poly(A)
p =
1 -7 10
```

Ainsi, le polynôme caractéristique de la matrice A est :

$$p(x) = x^2 - 7 \cdot x + 10$$

Les racines de ce polynôme sont les valeurs propres de la matrice A . ces racines peuvent être obtenues par la fonction '**eig**' :

```
>>Val_prop=eig(A)
Val_prop =
2
5
```

Pour évaluer le polynôme $P(x)$ en un point donné, on doit utiliser la fonction '**polyval**'. On évalue ce polynôme pour $x=1$, par exemple :

On veut évaluer en $x=2.5$ le polynôme suivant :

$$y = 3 \cdot x^4 - 7 \cdot x^3 + 2 \cdot x^2 + x + 1$$

```
>>C=[3 -7 2 1 1];
```

```
>>x=2.5;
```

```
>>y=polyval(C,x)
```

```
y =
23.8125
```

Si x est un vecteur contenant plusieurs valeurs, y sera aussi un vecteur qui contiendra le même nombre d'éléments que x .

4.3 Interpolation linéaire et non linéaire

4.3.1. Interpolation au sens des moindres carrés

Dans le domaine de l'analyse numérique des données, on a souvent besoin d'établir un modèle mathématique liant plusieurs séries de données expérimentales. L'interpolation polynomiale consiste à approcher la courbe liant les deux séries de mesures par un polynôme. Les coefficients optimaux de ce polynôme sont ceux qui minimisent la variance de l'erreur d'interpolation. Ce principe (voir cours) est connu sous le nom de la méthode des moindres carrés. La fonction '**polyfit**' retourne le polynôme P de degré n permettant d'approcher la courbe $y = f(x)$ au sens des moindres carrés.

Exemple :

```
>>x=[1.1 2.3 3.9 5.1];
>>y=[3.887 4.276 4.651 2.117];
>>a=polyfit(x,y,length(x)-1)
a =
-0.2015 1.4385 -2.7477 5.4370
```

Ainsi, le polynôme d'interpolation de y (d'ordre $length(x)-1=3$) est :

$$P(x) = -0,2015 \cdot x^3 + 1,4385 \cdot x^2 - 2,7477 \cdot x + 5,4370$$

Pour déduire l'erreur entre les valeurs expérimentales et le modèle obtenu par la fonction '**polyfit**', on dispose de la fonction '**polyval**' qui retourne la valeur du polynôme P pour toutes les composantes du vecteur (ou de la matrice) x .

Ainsi, cette fonction donne :

```
>>yi=polyval(a,x)
yi =
3.8870 4.2760 4.6510 2.1170
```

On remarque ici que les valeurs expérimentales de y sont bien restituées ($y=y_i$). Dans ce cas, on a un

coefficient de corrélation qui est égal à 1 (voir la définition de ce coefficient dans le cours).

Pour mieux comprendre ces fonctions prédéfinies dans MATLAB, on va simuler une courbe expérimentale par une sigmoïde à laquelle on superpose un bruit du type *Gaussien*. Cette courbe sera donnée par :

$$y = \frac{1}{1 + e^{-x}} + 0,05 \cdot \text{randn}(1, \text{length}(x))$$

Le programme correspondant à l'approximation des ces données dans MATLAB (*regres.m*, par exemple) est le suivant :

```
%*****
% Génération de données expérimentales: *
% y=1./(1+exp(-x))+0.05*randn(1,length(x))*

clc; % Effacer l'écran
clear all; % Effacer des variables de l'espace de travail
x=-5:0.1:5; % Intervalle de définition et de calcul de la sigmoïde

% Fonction sigmoïde bruitée
y=1./(1+exp(-x))+0.05*randn(1,length(x));
plot(x,y); % Tracé de la sigmoïde bruitée
title('Fonction sigmoïde bruitée - Polynôme d''interpolation');
xlabel('x');ylabel('y');

% Polynôme d'interpolation d'ordre 1
P=polyfit(x,y,1);

% Valeurs du polynôme d'interpolation
Vp=polyval(P,x);

% Tracé du polynôme d'interpolation
hold on;

plot(x,Vp,'--');
% Calcul de l'erreur d'interpolation
erreur=y-Vp;
% Tracé de la courbe de l'erreur
plot(x,erreur,':')
grid

gtext('Mesures')
gtext('Erreur')
gtext('Modèle')
hold off
% Affichage du polynôme d'interpolation
disp('Polynôme d''interpolation')
P
Var_erreur=num2str(std(erreur).^2);
disp(['La variance de l''erreur d''interpolation est : ',Var_erreur])
%*****
```

Après exécution du programme, on obtient à l'ordre 1 (droite affine : *fig. 1* ci-dessous) les résultats suivants :

```
>> regres
Polynôme d'interpolation
P =
0.1309 0.5008
```

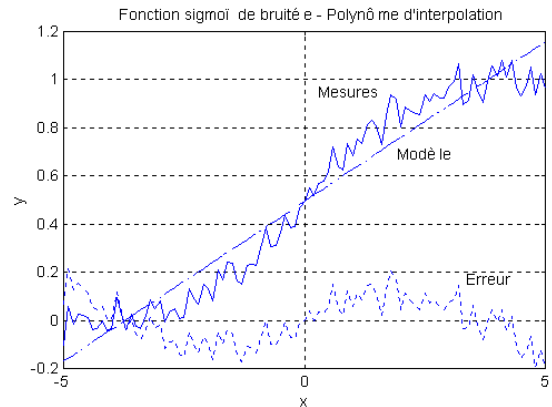
La variance de l'erreur d'interpolation est : 0.011277

On remarque ici que le polynôme d'interpolation d'ordre 1 n'est pas une bonne approximation pour ces

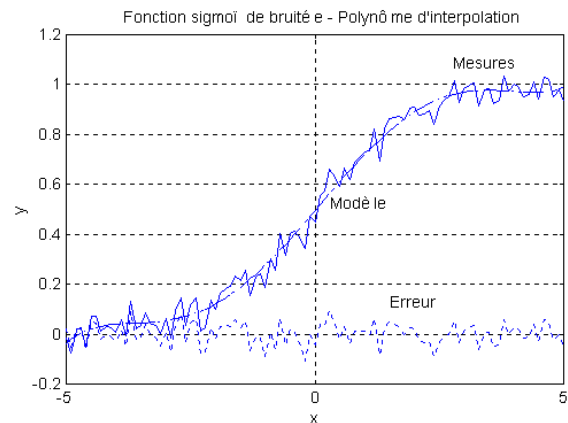
données. En effet, un polynôme d'ordre 5 donne une meilleure approximation de la sigmoïde. Ainsi, dans le programme précédent (*sigreg.m*), on change dans le '1' par '5' dans la fonction '*polyfit*', et on obtient les résultats suivants :

```
>> regres
Polynôme d'interpolation
P =
0.0002 -0.0000 -0.0111 0.0008 0.2326 0.4844
```

La variance de l'erreur d'interpolation est : 0.002279



Interpolation linéaire



Interpolation par un polynôme d'ordre 5

4.3.2. Interpolation non linéaire

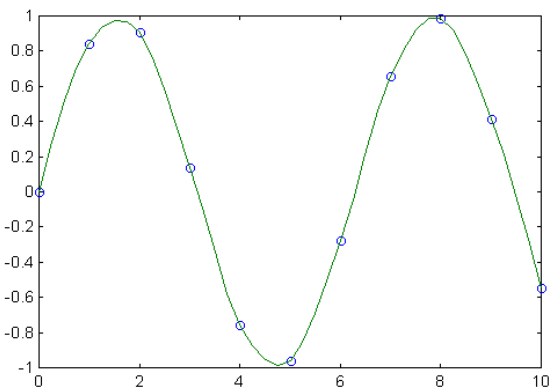
Une interpolation consiste à relier les points expérimentaux par une courbe sous forme de segments de droites ou de courbes polynomiales. Ceci peut être réalisé par la fonction '*interp1*'. La commande '*interp1(x,y,x_i,'type')*' retourne un vecteur de mêmes dimensions que x_i et dont les valeurs correspondent aux images des éléments de x_i déterminées par interpolation sur x et y . Si f est l'interpolation de y , la chaîne '*type*' spécifie alors le type d'interpolation qui doit être parmi les suivants :

'*linear*' : interpolation linéaire, '*spline*' : interpolation par splines cubiques, '*cubic*' : interpolation cubique.

Si on ne spécifie pas le type, l'interpolation linéaire est choisie par défaut.

Exemple 1 :

```
>>x = 0:10; y = sin(x); xi = 0:.25:10;
yi = interp1(x,y,xi,'cubic'); plot(x,y,'o',xi,yi)
```



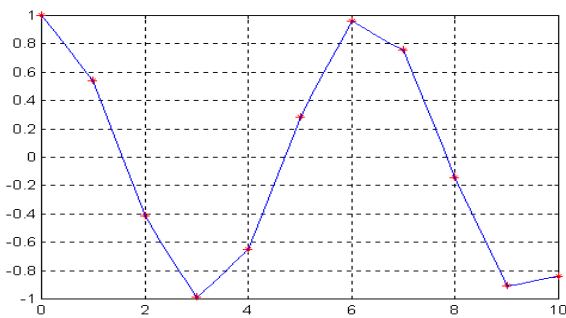
Interpolation cubique

Exemple 2 :

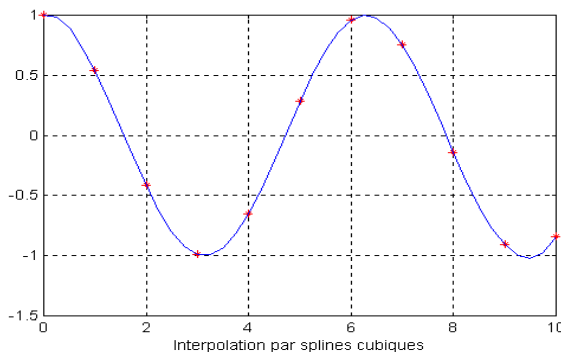
Dans le cas suivant, on étudiera ces différents types d'interpolation sur un même exemple de valeurs discrètes de la fonction 'cosinus'. On appellera l'algorithme : 'interp1.m'.

```
% *****
% Utilisation de la fonction interp1 *
clear all; clc;
x=0:10;
y=cos(x); % Points à interpoler
z=0:0.25:10; % Le pas du vecteur z est< à celui de x
% Interpolation linéaire
figure(1);
f=interp1(x,y,z);
% Tracé des valeurs réelles et de la courbe
d'interpolation
plot(x,y,'*r',z,f);
grid on; xlabel('Interpolation');
% Interpolation par splines cubiques
figure(2);
f=interp1(x,y,z,'spline');
plot(x,y,'*r',z,f);
grid on;
xlabel('Interpolation par splines cubiques');
% *****
```

En exécutant ce programme, on obtient les courbes suivantes :



Interpolation linéaire



Interpolation par splines cubiques

La fonction 'interp2' réalise l'interpolation dans l'espace trois dimensions (3D).

4.4. Intégration numérique

Dans Matlab (Toolbox), il existe 2 fonctions appelées 'quad' et 'quad8' pour l'intégration numérique. La fonction 'quad' utilise la méthode de Simpson et la fonction 'quad8' utilise les formules de Newton-Cotes à l'ordre 8. Ces deux fonctions sont utilisées de la façon suivante :

- quad('fonction_f',a,b)
- quad('fonction_f',a,b,tol)
- quad('fonction_f',a,b,tol,trace)

Dans la première forme, la tolérance 'tol' qui correspond à l'erreur relative **E**, est considérée égale à 0,001 par défaut. Ainsi, le calcul quadratique de l'intégrale est réitéré jusqu'à ce que la tolérance soit satisfaite. Si la 3^{ème} forme est utilisée avec une valeur non nulle de 'trace', un graphique d'évolution des itérations sera affiché sur l'écran. Quand à la fonction 'quad8', elle est utilisée de la même manière que la fonction 'quad' dans les 3 formes d'expressions précédentes.

Exemple :

Soit à calculer l'intégrale suivante :

$$I = \int_0^{\infty} (x-1) \cdot e^{-x \cdot (x-2)} \cdot dx$$

avec la transformation suivante :

$$y = x \cdot (x-2) = (x-1)^2 - 1$$

Dans ce cas, on a :

$$dx = \frac{dy}{2 \cdot \sqrt{y+1}}$$

On obtient donc :

$$I = \frac{1}{2} \cdot \int_0^{\infty} e^{-y} \cdot dy$$

Or, on connaît la fonction $\Gamma(\mathbf{x})$ (fonction Gamma) eulérienne du deuxième espèce qui est définie par :

$$\Gamma(x) = \int_0^{\infty} t^{(x-1)} \cdot e^{-t} \cdot dt$$

où $f(x)$ est un réel.

Pour un entier n strictement positif, $\Gamma(n)$ a la propriété suivante :

$$\Gamma(n) = (n-1)!$$

On pourra donc s'en servir pour calculer la factorielle d'un entier naturel n :

$$n! = \Gamma(n+1)$$

Dans Matlab, cette fonction est notée 'gamma'.

Exemple :

La factorielle de 10 est donnée par `gamma(10+1)=gamma(11)` :

```
>> fact=gamma(11)
fact =
3628800
```

Cette propriété est à l'origine du nom de la 'fonction factorielle', attribuée souvent à la fonction 'Gamma'. La fonction 'gamma' est donc prédéfinie dans Matlab. Cette fonction permet, de part sa définition pour tous les arguments réels positifs, d'introduire 'la factorielle' des arguments non entiers.

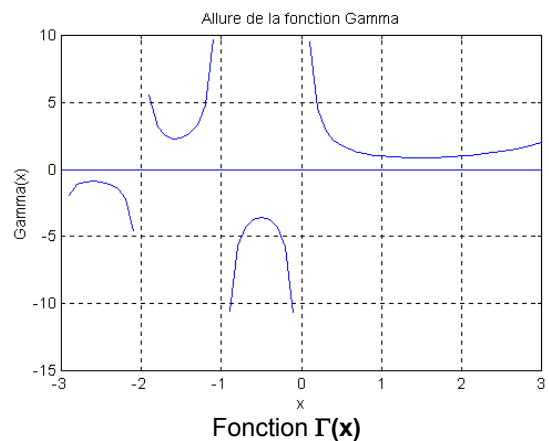
Exemple :

```
Gamma(0,5) = 0,5! = sqrt(pi)
>> format long
>> gamma(0.5)
ans =
1.77245385090552
>> racine_de_pi=sqrt(pi)
racine_de_pi =
1.77245385090552
>> gamma(0)
Warning: Divide by zero.
ans =
Inf
>> gamma(-2)
Warning: Divide by zero.
ans =
-Inf
```

Tracé de la fonction Gamma

Tracer la fonction Gamma pour $x \in [-3, 3]$.

```
>> x=-3:0.1:3;
>> gx=gamma(x);
>> plot(x,gx);grid;title('Allure de la fonction Gamma');
>> xlabel('x');ylabel('Gamma(x)');
>> hold on;plot(x,zeros(size(x)))
```



Revenons maintenant à l'intégration de la fonction $f(x)$:

$$I = \int_0^{\infty} (x-1) \cdot e^{-x \cdot (x-2)} \cdot dx$$

D'après le changement de variable effectué, cette intégrale s'écrit :

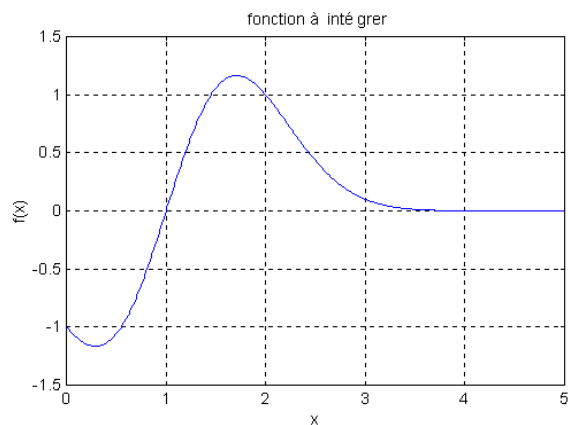
$$I = \frac{1}{2} \cdot \int_0^{\infty} e^{-y} \cdot dy = \frac{1}{2} \Gamma(1)$$

Calculons cette intégrale par les fonctions Matlab 'quad8' et 'gamma'. La fonction 'quad8' nécessite l'écriture de la fonction à intégrer dans un fichier contenant l'extension '.m' (programme Matlab).

```
function f=fct(x);
% fonction à intégrer
f=(x-1).*exp(-x.*(x-2));
```

Tracé de la fonction à intégrer :

```
>> x=0:0.01:5;
>> ft=fct(x);
>> plot(x,ft);
>> grid;title('fonction à intégrer');
>> xlabel('x');ylabel('f(x)');
```



$$f(x) = (x-1) \cdot e^{-x \cdot (x-2)}$$

Dans le programme donné ci-dessous (*quadgamm.m*), on calcule l'intégrale précédente par les fonctions 'quad8' et 'gamma', pour lesquelles on compare les performances en terme de temps de calcul et du nombre d'opérations en virgule flottante.

```
% *****
% Calcul de l'intégrale de f(x) par la *
% la fonction 'quad8' sur un K6-233 *
% *****
clear all; clc;
flops(0); % Mise à zéro du compteur d'opérations
tic; % Déclenchement du compteur temps de calcul
lquad=quad8('fct',0,5)
Nbre_op1=flops, %Arrêt du compteur d'opérations
tps_q=toc, %Arrêt du compteur de temps de calcul

% *****
% Calcul de l'intégrale de f(x) par la *
% la fonction 'gamma' *
% *****
flops(0);
tic;
lgamma=gamma(1)/2
Nbre_op2=flops
tps_g=toc
```

En exécutant le programme *quadgamm.m*, on obtient les résultats suivants :

```
>> format long
>> quadgamm
lquad =
0.49999987773178
Nbre_op1 =
686
tps_q =
0.06000000000000
lgamma =
0.50000000000000
Nbre_op2 =
40
tps_g =
0.05000000000000
```

D'après ces résultats, on remarque bien que le nombre d'opérations en virgule flottante et le temps d'exécution dans le cas de l'utilisation de la fonction 'gamma' sont nettement inférieurs à ceux que l'on obtient en utilisant la fonction 'quad8'.

La fonction 'gamma' prend des valeurs très grandes au voisinage de zéro et des entiers négatifs. Afin d'éviter un dépassement de capacité, MATLAB dispose de la fonction 'gamma1n' qui retourne le logarithme népérien.

4.5 Equations différentielles

On renvoie à [3] pour une présentation des équations différentielles et des méthodes numériques pour les résoudre.

Matlab possède plusieurs solveurs approchés pour les équations différentielles du type

$$y'(t) = f(t, y(t)), \quad t \in \mathbb{R},$$

$$y(t_0) = y_0 \in \mathbb{R}^n.$$

Le solveur choisi, par exemple ode45, ode23, ode113, ode15s, ode23s, ode23t, ou ode23tb, résout le problème de Cauchy ci-dessus en utilisant une méthode et une précision choisies par l'utilisateur. La syntaxe de base est

```
[T,Y] = solver(odefun,tspan,y0)
% odefun: fonction au second membre du système
% différentiel ci-dessus f(t,y)
% tspan : [t0,t1,...,tf] temps où on veut calculer la
% solution y
% y0 : Condition initiale y(t0) (vecteur colonne n
composantes)
% T : En sortie les temps où est calculée la solution
% Y : Les valeurs de la solution aux temps T
```