République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 8 Mai 1945 Guelma

Faculté des Mathématiques et de l'Informatique
et des Sciences de la Matière
Département de Mathématiques

Polycopié de Cours

Première année Master en Mathématiques

Option : **Equations aux Dérivées Partielles**

Et analyse numérique

Module

Outils d'informatique

Présentée par :

Dr. FRIOUI. Assia

Année Universitaire
2023/2024

Table des matières

Introduction

Matlab est un logiciel puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant calcul numérique et visualisation. C'est un environnement performant, ouvert et programmable qui permet de remarquables gains de productivité et de créativité. En d'autre terme Matlab trouve ses applications dans de nombreuses disciplines. Il constitue un outil numérique puissant pour la modélisation de systèmes physiques.

Ce cours s'adresse aux étudiants en master 1 de mathématiques dans le but de leur apprendre à écrire des programmes de calcul scientifique en utilisant le logiciel Matlab.

Ce cours est structuré comme suit :

On explique dans le premier chapitre intitulé : introduction à l'environnement Matlab, l'utilité des différents composants de l'interface Matlab, l'espace de travail, l'aide en utilisant la fonction *help,* en évoquant aussi quelques commandes usuelles de Matlab.

A travers le chapitre 2, on traite les objets et classes de Matlab concernant les valeurs litterales, les variables, les listes et vecteurs, les tableaux plus précisément la manipulation de ces derniers et enfin les fonctions opérant sur les éléments d'un tableau.

Au chapitre 3, nous introduisons les scripts et fonctions, Nous allons voir dans ce chapitre, comment utiliser Matlab comme un véritable langage de programmation, en passant des fichiers de commandes (que l'on peut sauvegarder et donc réutiliser), en écrivant nos propres fonctions et en utilisant des structures de contrôle telles que par exemple les nstructions conditionnelles if et $switch$, les boucles for et $while$. Ce chapitre se termine par un ensemble d'exercices corrigés et tester en des séances de Travaux Pratiques .

L'objectif du chapitre 4 est de présenter quelques fonctions de Matlab pour traiter des problèmes typiques de calcul numérique. Plus principalememt, et à l'issue de ce chapitre nous présentons la résolution numérique de certains problèmes mathématiques traités dans le module d'analyse numérique niveau Master 1 concernant la résolution des équations différentielles

ordinaires et aux dérivées partielles par la méthode des différences finies. Cette présentation se trouve complétée par un travail d'implémentation et d'application réalisé par les étudiants en des séances de TP avec le logiciel Matlab.

Finalement, dans le dernier chapitre, nous expliquons les différentes fenêtres graphiques. On aborde en particulier la visualisation des fonctions en deux et trois dimensions à l'aide des deux fonctions plot et plot3.

Chapitre 1

Introduction à Matlab

1.1 Qu'est-ce que Matlab

Le logiciel Matlab (MATtrix LABoratory) est un environnement interactif de programmation scientifique basé sur le calcul matriciel et la visualisation graphique. Matlab est une console d'exécution, il permet d'exécuter des fonctions, d'effectuer des opérations mathématiques, de manipuler des matrices, d'attribuer des valeurs à des variables, de tracer ou de représenter des graphiques ect...

Matlab est un interpréteur : les instructions sont interprétées et exécutées ligne par ligne.

Il existe deux modes de fonctionnement :

► Mode interactif : le logiciel exécute les instructions au fur et à mesure qu'elles sont entrées par l'utilisateur dans la fenêtre de commande.

► Mode exécutif : le logiciel exécute ligne par ligne un "fichier M" (programme en langage Matlab).

Matlab comprend de nombreuses fonctions, de calcul ou de traitements de données, d'affichage, de tracés de courbes, de résolution de systèmes et d'algorithmes de calculs numériques au sens large du terme.

Considéré comme l'un des meilleurs langage de programmation, Matlab n'est pas le seul environnement de calcul scientifique, il existe d'autres concurents dont les plus importants sont Maple et Mathematica. Il existe même des logiciels libres qui sont des clones de Matlab comme Scilab et Octave.

1.2 Une session Matlab

L'interface utilisateur de Matlab varie légèrement en fonction de la version installée. Cependant, elle est constituée d'une fenêtre de commandes et un éditeur de texte permettant d'écrire des scripts et des fonctions voir figure 1 ci dessous.

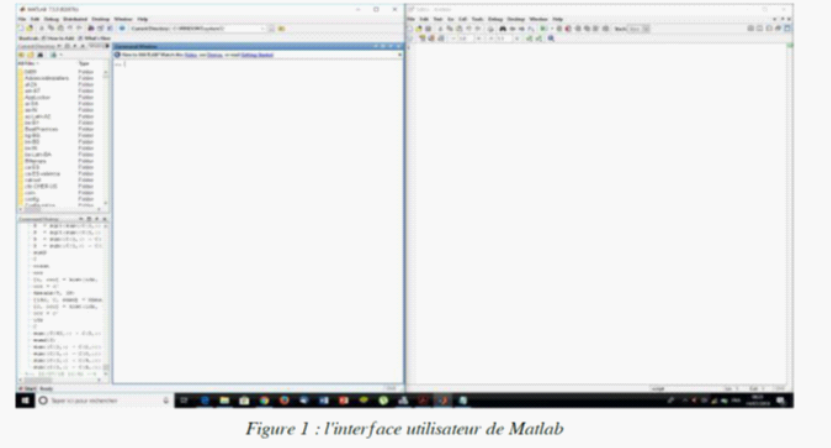


Figure 1 : l'interface utilisateur de Matlab

1.2.1 Lancer, quitter Matlab

Pour lancer Matlab, il suffit de double cliquer sur l'icône du logiciel. La fenêtre de commande de Matlab s'ouvre alors et on tape les commandes ou les expressions à évaluer à droite du prompt ">>" Le processus d'évaluation est déclenché par la frappe de la touche *« Enter »*.

On quitte Matlab en tapant tout sinplement *quit* dans la fenêtre de commande.

1.2.2 La fenêtre de commandes

La fenêtre de commandes est la fenêtre principale de l'environnement Matlab. Elle donne accès à l'interpréteur de commandes et exécute les commandes Matlab saisies par l'utilisateur.

La fenêtre de commandes donne aussi accès à l'historique de toutes les commandes tapées.On peut les retrouver et les modifier grâce aux touches de direction. Il est possible d'utiliser les touches ⇑ et ⇓ pour parcourir les commandes exécutées précédemment, puis les éditer. Pour relancer une commande, il suffit d'appuyer sur la touche « *Enter* » .

On peut retrouver toutes les commandes commençant par un groupe de lettres. Pour cela, il suffit de taper les premières lettres de la commande recherchée, puis appuyer plusieurs fois sur ⇑ pour parcourir les commandes correspondant à cette recherche.

1.2.3 Fonctions et commandes

Certaines fonctions de MATLAB ne calculent pas de valeur numérique ou vectorielle, mais effectuent une action sur l'environnement de la session en cours. Ces fonctions sont alors appelées commandes. Elles sont caractérisées par le fait que leurs arguments (lorsqu'ils existent) ne sont pas placés entre parenthèses.

Dans de nombreux cas, fonctions ou commandes peuvent être appelées avec des arguments qui différent soit par leur nombre, soit par leur nature (nombre, vecteur, matrice, . . .). Le traitement effectué par ces fonctions dépend principalement de leurs arguments. Par exemple nous verrons plus loin que la fonction *diag* appelée avec une matrice pour argument retourne sa diagonale principale alors que lorsqu'elle est appelée avec un vecteur, elle retourne une matrice diagonale dont le vecteur diagonal est le vecteur donné comme argument.

Donc une fonction (ou une commande) n'est pas caractérisée seulement par son nom, mais par sa signature i.e (nom + arguments).

1.2.4 Historique

Matlab conserve l'historique des commandes. Il est donc possible à l'aides des flèches du clavier de remonter dans la liste des instructions déjà entrées pour retrouver une instruction particulière pour la réutiliser et éventuellement la modifier avant de l'utiliser à nouveau.

1.3 Aide et documentation

Matlab dispose d'une aide fournie et fonctionnelle. Elle est accessible de différentes manières :

▶ par la commande *help,* en tapant directement *help* nom de la commande dans la fenêtre de commandes.

▶ par la commande *doc*, en tapant directement *doc* nom de la commande dans la fenêtre de commandes.

▶ en consultant la documentation en ligne à l'adresse **https ://fr.mathworks.com/help/.**

La commande *help* est à privilégier pour une aide rapide. Elle donne accès à la syntaxe de toute fonction, par contre la commande *doc* est plus complète. Elle reprend les informations de la commande *help*, en ajoutant des exemples des différents usages de la fonction, ainsi que des informations complémentaires.

Enfin la différence entre les deux commandes est que *help* affiche ces messages dans la fenêtre de commandes alors que *doc* les affiche dans une fenêtre de navigation.

Pour terminer ce chapitre donnons la liste des functions Matlab usuelles - *helpwin :*

▶ *helpwin elfun* affiche la liste des fonctions mathématiques élémentaires,

▶ *helpwin specfun* affiche la liste des fonctions mathématiques avancées,

▶ *helpwin elmat* affiche la liste des fonctions matricielles élémentaires,

1.4 Quelques commandes usuelles de Matlab

- *who x* : % Pour avoir une information sur une variable x .
- *clear x* : % Pour effacer la variable x de l'espace de travail.
- *clear all :* % Pour effacer toutes les variables de l'espace de travail.
- *what :* % Pour lister tous les m-fichiers dans le répertoire courant.
- *dir/lst :* % Pour lister tous les fichiers du répertoire courant.
- *pwd* : % Pour afficher le répertoire courant.
- *whos :* % Pour avoir toutes les variables connues avec plus de détails.
- *clc* : % Pour effacer la fenêtre de commande.
- le prompt matlab '»' qui indique que matlab attend des instructions.

Pour plus de détail voir chapitre deux.

Chapitre 2

Les" objets" de Matlab-Listes, vecteurs, tableaux

2.1 Objets et classes de Matlab

La classe fondamentale de Matlab est la classe double, c'est à partir de cette classe que sont définis des types de données plus comme : les nombres complexes et les tableaux. Une autre classe qui est moins fréquemment utilisée est la classe *char* qui modélise des caractères (alphabets, chiffres et caractères spéciaux).

Dans les nouvelles versions de Matlab (à partir de la version 7) une classe logical a été introduite. Cette classe est utilisée pour modéliser des valeurs logiques : vrai ou faux (true or false).

2.2 Valeurs littérales

Le terme valeur littérale désigne les valeurs (de tout type de données) qu'on peut directement saisir à partir du clavier. Les types de données qui nous intéressent sont :

- ▶ Les types numériques.
- ▶ Les tableaux.
- ▶ Les chaînes de caractères.

2.2.1 Les types numériques

Les nombres réels ou entiers sont écrits soit sous la forme décimale (en utilisant le point **"."** comme séparateur décimal) soit en utilisant la notation scientifique usuelles comme exemple : 4, 3.214, 1.65E33.

Les nombres complexes sont écrits sous la forme "a + bi" comme dans 1+2i.

Citons quelques fonctions relatives aux nombres complexes :

- ▶ *real* et *imag* renvoient respectivement la partie réelle et la partie imaginaire du complexe,
- ▶ *abs* et *arg* renvoient respectivement le module et l'argument du complexe.
- ▶ *conj* renvoie le complexe conjugué du nombre complexe.

2.2.2 Les tableaux de nombres

La syntaxe utilisée par Matlab pour saisir un tableau à une ou deux dimensions (autrement dit : matrice) est la suivante

- ▶ le tableau est délimité par deux crochets ([.])
- ▶ les éléments de la même ligne sont séparés par des espaces ou par des virgules ;
- ▶ les lignes sont séparées par des points-virgules ou des retours à la ligne ;
- ▶ toutes les lignes doivent contenir le même nombre d'éléments.

Exemple 2.1 *Initialiser dans Matlab les tableaux suivant :*

>> [1 2 0 5] % *ou* [1, 2, 0, 5] *pour initialiser un vecteur ligne*

ans =

1 2 0 5

>> [1; 2; 3; 2 + i] % pour initialiser un vecteur colonne

ans =

1.0000

2.0000

3.0000

2.0000 + 1.0000i

>> [1, 2; 0, 3] % pour initialiser une matrice

ans =

1 2

0 3

>> [1, 2; 0, 3, 5] % les lignes doivent contenir le même nombre d'elements

??? Error using ==> vertcat

CAT arguments dimensions are not consistent.

1) L'opérateur colon " : "

Avant de poursuivre notre cours, il est impérativement nécessaire de comprendre une notation que l'on retrouve partout en Matlab, celle de l'opérateur " : " (en anglais colon). En Matlab, quand l'on ecrit " 1 : 8" par exemple, cela signifie tous les nombres de "1 à 8" (c.a.d. 1 2 3 4 5 6 7 8), et par exemple "10 :-2 :1" cela signifie encore tous les nombres de 10 à 1 avec un pas de -2 (et donc les nombres : 10 8 6 4 2), le pas étant unitaire par défaut i.e égal à 1.

Exemple 2.2 *Observer les résultats des exemples suivants :*

>> X = [0 : pi/11 : pi] % [valeur-initiale : incrément : valeur-finale]

X =

0 0.2856 0.5712 0.8568 1.1424 1.4280 1.7136 1.9992 2.2848 2.5704 2.8560 3.1416

>> 1 :10 ,

ans =

1 2 3 4 5 6 7 8 9 10

>> 10 :-1 :1,

ans =

10 9 8 7 6 5 4 3 2 1

>> 0 :pi/4 :pi, >> 2 :-1 : 8, >> 8 :-1 : 2, % Tester ces exemples.

Par exemple pour créer un vecteur de valeurs équidistantes de 0.1 entre 0 et 1

>> x = 0 :0.1 :1

x =

Columns 1 through 7

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000

Columns 8 through 11

0.7000 0.8000 0.9000 1.0000

Remarque 2.1 *Il est possible d'utiliser le même opérateur pour générer des vecteurs colonnes en transposant le résultat obtenu avec l'opérateur apostrophe* $(.)'$.

Exemple 2.3 *Commenter le résultat suivant :*

>> (1 :3)'

2) La fonction linspace

La fonction *Linspace* (v_i, v_f, n) renvoie un vecteur ligne de n valeurs régulièrement espacées entre v_i *et* v_f

Exemple 2.4 *Exprimer les instructions suivantes :*

>> Linspace (1, 2, 6)

ans =

1.0000 1.2000 1.4000 1.6000 1.8000 2.0000

>> Linspace (1, -2, 6)

ans =

1.0000 0.4000 -0.2000 -0.8000 -1.4000 -2.0000

>> Linspace (1, 2, 6)'

ans =

1.0000

1.2000

1.4000

1.6000

1.8000

2.0000

Remarque 2.2 *Par défaut, i.e. en l'absence du troisième argument, le vecteur généré est de taille 100. Pour avoir un vecteur ligne avec des composantes réparties suivant une échelle logarithmique, on utilise logspace. la syntaxe est la même que pour linspace sauf que par défaut, le vecteur généré est de taille 50.*

2.2.3 Caractères et chaînes de caractères

Les chaînes de caractères sont des séquences finies de caractères placées entre des apostrophes i.e entre guillemets.

Une chaîne de caractères est considérée par Matlab comme un vecteur ligne dont chaque élément contient un caractère.

D'autre part, pour Matlab, chaînes de caractères et liste de caractères sont des objets de même nature :

Exemple 2.5 *Commenter le résultat des exemples suivants :*

>> ['M' 'a' 't' 'l' 'a' 'b'] % liste de caractères

ans =

Matlab

>> 'Matlab' % chaînes de caractères

ans =

Matlab

donc la liste de caractères ['M' 'a' 't' 'l' 'a' 'b'] est identique à la chaînes de caractères ['Matlab']

>> ['Matl' 'ab']

ans =

Matlab % Mieux encore, ['Matl' 'ab'] est identique à 'Matlab'

Cet exemple donne un idée du rôle des crochets [.]. Les crochets sont le symbole de l'opérateur de concaténation :

▶ concaténation "en ligne" lorsque le séparateur est un espace ou une virgule ;

▶ concaténation "en colonne" lorsque le séparateur est un point-virgule comme dans les tableaux de nombres (il est alors nécessaire que les listes de nombres ou de caractères ainsi concaténées possèdent le même nombre d'éléments).

Exemple 2.6 *>>['123'; '1234']*

??? Error using ==> vertcat

CAT arguments dimensions are not consistent.

2.2.4 Cellules et tableaux de cellules - cell array

Une cellule est un conteneur dans le quel on peut placer toute sorte d'objets : nombre, chaîne de caractères, tableau et même tableau de cellules. Les tableaux de cellules permettent de regrouper dans une même structure des éléments de nature très différente. La syntaxe des tableaux de cellules est voisine de celle des tableaux usuels, les crochets étant remplacés par des accolades.

Exemple 2.7 *Observer le résultat suivant :*

```
>>{'somme' 4+2; 'difference' 6-23; 'produit' 4*3;' division' 5/0 }
ans =
    'somme'       [ 6]
  'difference'   [-17]
   'produit'      [ 12]
  ' division'    [Inf]
```

2.2.5 Calculs élémentaires

L'utilisation la plus basique de Matlab consiste à utiliser l'interpréteur de commandes comme une calculatrice.

On peut alors utiliser les opérateurs arithmétiques les plus courants : +, -, *, /, ^. Les parenthèses s'utilisent de manière classique pour tenir compte des priorités des opérations.

Exemple 2.8 *tapez une expression mathématique quelconque et appuyez sur «Entrée»*

```
>> (3*4)/(5+6)
ans =
    1.0909
```

Le résultat est mis automatiquement dans une variable appelée *ans.* Celle-ci pourrait être utilisée dans un autre calcul comme nous allons le voir dans les prochains exemples.

2.2.6 Fonctions classiques

Dans Matlab, de nombreuses fonctions de calcul sont prédéfinies :

- sqrt() : calcule la racine carrée.
- cos(), sin(), tan() et cotg() : pour les fonctions trigonométriques de base.
- acos(), asin() et atan() : pour leurs réciproques.

▶ cosh(), sinh(), tanh(), acosh(), asinh() et atanh() : pour les fonctions trigonométriques hyperboliques .

▶ log() : pour le logarithme népérien (ln)

▶ log10() : pour le logarithme en base 10.

▶ log2() : pour le logarithme en base 2.

▶ exp() : pour la fonction exponentielle.

▶ floor() : pour l'arrondi vers l'entier inférieur.

▶ ceil() : pour l'arrondi vers l'entier supérieur.

▶ fix() : pour la troncature (arrondit à l'entier inférieur en valeur absolue).

▶ round() : pour l'arrondi à l'entier le plus proche (0,5→ 1).

▶ abs() : pour la valeur absolue.

▶ rand() : pour avoir un nombre aléatoire entre 0 et 1, suivant une loi uniforme.

Exemple 2.9 *Exemple sur les fonctions floor, ceil, fix et round.*

```
>> x = [-3.9   -2.5  -1.2   0   1.2  2.5   3.9]
x =
  -3.9000  -2.5000   -1.2000   0   1.2000   2.5000   3.9000
>> floor(x)
ans =
  -4  -3  -2  0   1  2  3
>> x = [-3.9   -2.5   -1.2  0   1.2   2.5   3.9]
x =
 -3.9000   -2.5000   -1.2000   0   1.2000   2.5000   3.9000
>> ceil(x)
ans =
   -3   -2   -1  0  2   3   4
>> x = [-3.9   -2.5   -1.2   0   1.2   2.5  3.9]
```

```
x =
  -3.9000   -2.5000   -1.2000    0    1.2000    2.5000    3.9000
>> fix(x)
ans =
    -3   -2   -1  0   1   2   3
>> x = [-3.9  -2.5   -1.2  0   1.2   2.5   3.9]
x =
  -3.9000   -2.5000   -1.2000    0    1.2000    2.5000    3.9000
>> round(x)
ans =
    -4   -3   -1   0   1   3   4
```

2.2.7 Format d'affichage

Par défaut, Matlab affiche les résultats en *format short*. On peut modifier le format d'affichage en utilisant la

commande format :

▶ *format short* : valeur par défaut, notation fixe à 4 décimales.

▶ *format long* : notation fixe à 7 décimales pour un réel au format simple, 14 ou 15 décimales pour un réel au format double.

▶ format *short e* ou format *long e* : notation scientifique exponentielle à virgule flottante

Exemple 2.10 *tapez les expressions mathématiques suivantes :*

1. *>> pi*

   ans =

   3.1416

2. *>> format long*

   >> pi

ans =

3.141592653589793

3. >> format short e

>> pi^3

ans =

3.1006e+001

2.2.8 Ponctuation, commentaires, interruption

Une ligne de commande Matlab peut comporter plusieurs commandes séparées par des virgules (,) ou des points-virgules (;).

Exemple 2.11 *Effectuer ces opérations*

>> x=3/8, y=2.2678/37.80 ; z=(x+y)/sqrt(-4) → *retourne*

x =

0.3750

z =

0 -2.1750e-001i

Exemple 2.12 *>> x=3/8 ; y=2.2678/37.80 ; z=(x+y)/sqrt(-4) % Mieux encore*

z =

0 -2.1750e-001i

On rappele qu'une ligne de commande Matlab peut comporter des commentaires signalés par le symbole %. Tout ce qui suit ce symbole est alors ignoré par l'interpréteur Matlab.

Si une expression est trop longue pour tenir sur une ligne de commande, alors on termine la ligne par... (3 points).

Exemple 2.13 *>> y=cos(2*pi/3)*(sin(5.78*pi/4)^2+...*

cos(pi/5)^2) → retourne

y =

-8.1247e-001

2.3 Variables

Une caractéristique de Matlab est que les variables n'ont pas à être déclarées, elles sont créées systématiquement à la première opération d'affectation (=). Le type et la dimension de la variable sont déterminés de manière automatique à partir de l'expression mathématique ou de la valeur affectée à la variable.

Une variable est désignée par un identificateur qui est formé d'une combinaison de lettres et de chiffres. Le premier caractère de l'identificateur doit nécessairement être une lettre. Cet identificateur doit être unique pour distinguer les différentes variables utilisées par le programme.

Il est à noter que Matlab différencie majuscules et minuscules, c'est-à-dire que la variable avec l'identifiant x est différente de la variable X .

Exemple 2.14 *>> x1=3.8675, X1=pi^2 → retourne*

x1 =

3.8675

X1 =

9.8696

2.3.1 Quelques variables spéciales

Voici quelques identificateurs prédéfinis :

► La variable *ans* : cette variable garde la valeur de la dernière expression non-affecté à une variable.

▶ La variable *pi* : cette variable comporte la valeur de π (= 3.146...).

▶ La variable i ou j : représentent tous deux le nombre imaginaire unité $\sqrt{-1}$, attention à ne pas utiliser i et j comme indices pour accéder aux éléments d'un tableau

▶ *inf* : désigne l'infini au sens d'une évaluation du type $(1/0)$;

▶ *NaN* signifie "Not a Number" - peut être le résultat d'une évaluation du type $(0/0)$;

Exemple 2.15 *Commenter les résultats des exemples suivants :*

>> x=i^2, >> x=j^2,

2.3.2 Affectation

Comme dans la plupart des langages de programmation, le signe = correspond à l'affectation d'une valeur dans une variable.

Exemple 2.16 *>> x=[1 2 3]*

```
x =
    1   2   3
>> x= 'bon'
 x =
    bon
```

L'exemple ci-dessus montre bien que dans Matlab les variables ne sont ni déclarées ni typées.

2.3.3 Espace de travail - workspace

Les variables sont définies au fur et à mesure que l'on donne leurs noms et leurs valeurs numériques ou leurs expressions mathématiques. Les variables ainsi définies sont stockées dans l'espace de travail et peuvent être utilisées dans les calculs subséquents.

Les commande who et whos

La commande **who** donne la liste des variables actives dans l'espace de travail. Mais elle est moins précise. Pour avoir plus d'informations, il est préférable d'utiliser la commande **whos** qui donne non seulement les variables mais aussi leur type, leur taille (comme tableau) et l'espace mémoire occupé.

Exemple 2.17 *>> u = pi; v = 'Matlab'; w = [1 2 3; 5 3 4];*

>> who u v w

Your variables are :

u v w

>> whos u v w

Name	*Size*	*Bytes*	*Class*
u	*1x1*	*8*	*double*
v	*1x6*	*12*	*char*
w	*2x3*	*48*	*double*

Les fonctions size, size(,1) et size(,2)

La fonction size retourne le couple (nl, nc) formé du nombre de lignes nl et du nombre de colonnes nc du tableau associé à la variable donnée comme argument.

Exemple 2.18 *Considérons les trois variables u, v et w de l'exemple précédent. La fonction size produit l'affichage suivant*

>> size(u)

ans =

1 1

>> size(v)

ans =

1 6

Où plus facilement pour accéder au nombre de lignes et au nombre de colonnes, on peut affecter la valeur retournée par size à un tableau à deux éléments $[nl, nc]$:

Exemple 2.19 *>> size(w)*

>> [nl, nc] = size(w)

nl =

2

nc =

3

Enfin size $(, 1)$ et size $(, 2)$ permettent l'accès direct au nombre de lignes et au nombre de colonnes d'un tableau.

La fonction class

La fonction class retourne le nom de la classe à laquelle appartient la variable donné comme argument.

Exemple 2.20 *Reprenons u et v définies dans l'exemple précédent*

>> cu = class(u)

cu =

double

>> cv = class(v)

cv =

char

2.3.4 Les commandes save, load

Les commandes save, load permettent d'intervenir directement sur l'environnement de travail :

– save permet de sauver tout ou partie de l'espace de travail sous forme de fichiers binaires appelés mat-files ou fichiers .mat, plus précisément :

– save : enregistre la totalité de l'espace de travail dans le fichier matlab.mat ;

– save nom de fichier : l'espace de travail est enregistré dans le fichier nom de fichier ;

– save nom de variable . . . nom de variable : enregistre les variables indiquées (et les objets qui leurs sont associés) dans un fichier .mat qui porte le nom de la première variable ;

– save nom de fichier nom de variable . . . nom de variable : enregistre les variables dans le fichier dont le nom a été indiqué.

– load permet d'ajouter le contenu d'un fichier .mat à l'espace de travail actuel ;

2.4 Manipuler des tableaux

Comme il a été cité plus haut Matlab est un outil de calcul matriciel, il considère chaque nombre comme une matrice de taille (1×1), disons un vecteur ligne de n éléments est pris comme une matrice de taille $(1\times\ n)$ et un vecteur colonne de m éléments est considéré comme une matrice de taille $(m\ \times 1)$.

Pour deux matrices A et B, si le nombre de lignes de B est égale au nombre de colonnes de A, alors l'opération

A * B effectue une multiplication matricielle de A par B. L'opérateur ^ peut également être utilisé pour multiplier une matrice carrée par elle même.

Exemple 2.21 *Commenter les résultats.*

>> A= [1 2;3 4];

>> B= [5 6;7 8];

*>> A*B ??? >> A.*B??? >> A+B??? >> A-B??? >> A+3???*

L'opérateur " + "est utilisé pour faire l'addition de deux matrices qui ont la même dimension, ou pour additionner un scalaire à tous les composants d'une matrice. L'opérateur " − "

peut être utilisé de la même façon pour effectuer des soustractions.

2.4.1 Accéder aux éléments d'un tableau

Pour accéder à un élément particulier d'un tableau ou d'une matrice, il suffit d'entrer le nom du tableau ou de la matrice suivi entre parenthèses du ou des indices dont on veut lire ou écrire la valeur.

Exemple 2.22 *Accéder aux éléments d'un tableau*

>> A = 2 :2 :10 → retourne

A =

2 4 6 8 10

>> A (3) → retourne

ans =

6

>> B= [5 6 ;7 8]

B =

5 6

7 8

>> B (2, 1)

ans =

7

>> B (2, end)

ans =

8

>> B= [1 2 0 1 ;3 1 4 7; 2 3 5 3]; Un exemple, si je veux afficher la valeur de B_{32}

>> B(3,2)

ans = 3

Remarque 2.3 *L'accès en lecture à un élément dont les indices seraient négatifs ou dont la valeur serait strictement supérieure au nombre de lignes ou au nombre de colonnes du tableau, conduit à une erreur.*

Exemple 2.23 *Commenter ces exemples*

>> A(1,6) ; >> B(3,5) ;

? ? ? Index exceeds matrix dimensions.

2.4.2 Extraire un sous-tableau

Soit T un tableau de dimension $(L \times C)$ et soit $l_1 < l_2 < L$, $c_1 < c_2 < C$. La commande $T(:, c_1)$ extrait la colonne c1 du tableau T, alors que la commande $T(l_1, :)$ extrait la ligne l_1. La commande $T(l1 : l2, c1 : c2)$ extrait un sous-tableau formé par les éléments de T dont l'indice de ligne appartient à $[l_1, l_2]$ et l'indice de colonne appartient à $[c_1, c_2]$. L'identificateur *end* peut être utilisé pour accéder à la dernière ligne ou la dernière colonne d'un tableau.

Exemple 2.24 *Extraction des sous-tableaux*

>>A= [1 2 0 1 ;3 1 4 7; 2 3 5 3 ;1 2 3 4] ;

>> A(:, 3)

ans =

0

4

5

3

>> A(3 : end ,2 :3) → retourne

ans =

3 5

2 3

2.4.3 Concaténation de tableaux - [ ]

L'opérateur [ ] permet la concaténation de tableaux :

- si les tableaux $\{T_k\}_{k=1,2,.....n}$ possèdent le même nombre de lignes les expressions équivalentes $[T_1, T_2, ...T_p]$ ou $[T_1 T_2, ...T_p]$ créent un tableau :

- qui a le même nombre de lignes que les tableaux composants ;

- dont le nombre de colonnes est la somme des nombres de colonnes de chacun des tableaux composants :

- qui est obtenu en concaténant "en ligne" les tableaux composants

- si les tableaux T_k ont le même nombre de colonnes l'expression $[T_1, T_2, ...T_p]$ crée un tableau :

- qui a le même nombre de colonnes que les tableaux composants ;

- dont le nombre de lignes est la somme des nombres de lignes de chacun des tableaux composants ;

- qui est obtenu en concaténant "en colonne" les tableaux composants.

Exemple 2.25 *effectuer ces exemples*

>>T1 = [1 2 ; 2 3]

T1 =

1 2

2 3

>>T2 = [3 4 ; 6 7]

T2 =

3 4

6 7

>> T3 = [T1 , T2] ou [T1 T2]

T3 =

1 2 3 4

2 3 6 7

>> T4 = [T1 ; T2]

T4 =

1 2

2 3

3 4

6 7

2.4.4 Sous-matrices spéciales

Pour extraire les éléments de la diagonale principale d'une matrice, on utilise la fonction *diag*. La même fonction peut être paramétrée pour extraire les éléments de la $k^{ième}$ diagonale. La fonction *diag* retourne un vecteur-colonne formé des éléments de la diagonale qu'elle extrait. Selon le type de son argument, la fonction *diag* se comporte de deux façon différentes.

▶ si le paramètre de la fonction *diag* est une matrice elle retourne un vecteur-colonne contenant les éléments de la diagonale de son paramètre.

▶ si le paramètre de la fonction *diag* est un vecteur, elle retourne une matrice diagonale dont le vecteur diagonal est composé à partir de son paramètre.

Exemple 2.26 *effectuer les calculs suivants*

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

>> diag(A,1)

ans =

5

5

5

>> diag(A,-2)

ans =

2

2

On peut construire à l'aide de la commande *diag* très simplement des matrices tridiagonales. Par exemple la matrice correspondant à la discrétisation par différences finies du problème de Dirichlet en dimension 1 s'obtient ainsi

```
>> N=5 ;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1)
   A =
        2   -1    0    0    0
       -1    2   -1    0    0
        0   -1    2   -1    0
        0    0   -1    2   -1
        0    0    0   -1    2
```

On dispose également de la commande *tril* qui permet d'obtenir la partie triangulaire inférieure (l pour lower) d'une matrice. La commande *triu* permet d'obtenir la partie triangulaire supérieure (u pour upper) d'une matrice.

Comme pour la commande diag, les commandes *triu* et *tril* admettent un second paramètre k . On peut ainsi obtenir la partie triangulaire supérieure (ou inférieure) à partir de la k diagonale. Ainsi,

Exemple 2.27 *Commenter les résultats*

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

>> tril(A)

ans =

$$\begin{matrix} 4 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 2 & 3 & 4 & 0 \\ 1 & 2 & 3 & 4 \end{matrix}$$

Exemple 2.28 *>> tril(A,-1)*

ans =

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 3 & 0 \end{matrix}$$

Exemple 2.29

>> tril(A,1)

ans =

$$\begin{matrix} 4 & 5 & 0 & 0 \\ 3 & 4 & 5 & 0 \\ 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 \end{matrix}$$

Remarque

tril $(A, 0)$ s'écrit aussi *tril* (A), de même, *triu* $(A, 0)$ s'écrit aussi *triu* (A).

2.4.5 La structure sparse

On appelle matrice creuse (le terme anglais est sparse matrix) une matrice comportant une forte proportion de coefficients nuls. L'intérêt de telles matrices résulte non seulement de la réduction de la place mémoire (on ne stocke pas les zéros) mais aussi de la réduction du nombre d'opérations (on n'effectuera pas les opérations portant sur les zéros).

Si A est une matrice, la commande $sparse\ (A)$ permet d'obtenir la même matrice mais stockée sous la forme sparse. Si l'on a une matrice stockée sous la forme sparse, on peut obtenir la même matrice stockée sous la forme ordinaire par la commande *full*. Aussi, il est possible de visualiser graphiquement la structure d'une matrice grâce à la commande *spy*.

Si A est une matrice, la commande $spy\ (A)$ ouvre une fenêtre graphique et affiche sous forme de croix les éléments non-nuls de la matrice. Le numéro des lignes est porté sur l'axe des ordonnées, celui des colonnes en abscisse.

On obtient également le nombre d'éléments non-nuls de la matrice. La commande nnz permet d'obtenir le nombre d'éléments non-nuls d'une matrice.

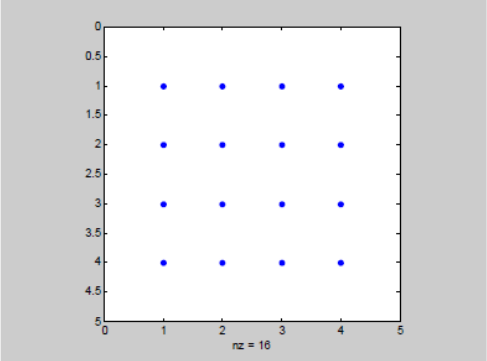
Exemple 2.30 *Appliquer ces instructions à l'exemple vu précédemment consernant la discrétisation par différences finies du problème de Dirichlet en dimension 1.*

```
>> N=4 ;
>> A=diag(2*ones(N,1)) - diag(ones(N-1,1),1) - diag(ones(N-1,1),-1)
   A =

        2   -1    0    0
       -1    2   -1    0
        0   -1    2   -1
        0    0   -1    2
```

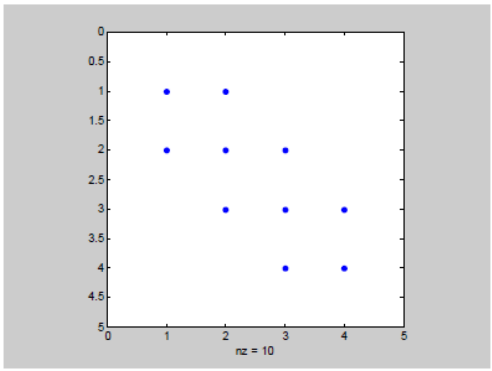
Exemple 2.31

```
>> nnz(A)
      ans =
           10
>> spy(A)
```



Reprsentation graphique spy(A)

```
>> B = sparse(A)
        B =
            (1,1)       2
            (2,1)      -1
            (1,2)      -1
            (2,2)       2
            (3,2)      -1
            (2,3)      -1
            (3,3)       2
            (4,3)      -1
            (3,4)      -1
            (4,4)       2
>> spy(B)
```



Reprsentation graphique spy(B)

2.4.6 Matrices par blocs

La construction des matrices par blocs est particulièrement facile avec Matlab. Soit à construire une matrice D de la forme

$$D = \begin{pmatrix} A & I \\ O & A \end{pmatrix}$$

où A est une matrice $n \times n$, O la matrice nulle d'ordre n et I_n la matrice identité d'ordre n.

$>> A = [1, 2, 3; 3, 5, 6; 1, 0, 1]\,;$

$>> I3 = eye\,(3)\,;$ % matrice identité d'ordre 3,

$>> Zero = zeros\,(3)\,;$ % matrice nulle d'odre 3

$>> D = [A \;\; I3\,; Zero \;\; A]$

$D =$

$$\begin{matrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 3 & 5 & 6 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 3 & 5 & 6 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{matrix}$$

2.4.7 Matrices spéciales

Certaines matrices se construisent très simplement grâce à des commandes dédiées. Citons les plus utilisées :

$eye(n)$: la matrice identité de dimension n

$ones(m,n)$: la matrice à m lignes et n colonnes dont tous les éléments valent 1

$zeros(m,n)$: la matrice à m lignes et n colonnes dont tous les éléments valent 0

$rand(m,n)$: une matrice à m lignes et n colonnes dont les éléments sont générés de manière aléatoire entre 0 et 1

Exemple 2.32 *Donner la valeur de la matrice A aprés exécution de la séquence d'instructions suivante :*

1)
```
n=4 ;
S= [ eye(n)    zeros(n,1)] ;
S= S( :, 2 : n+1) ;
A=2*eye(n) -S -S'
```

2)
```
n=4 ;
D=diag(ones(n,1)) ;
S=diag(ones(n-1,1),1) ;
A=2*D-S -S'
```

2.4.8 L'indice "end "

"end" est un mot-clé de Matlab qui peut être employé comme opérateur d'indexage. Dans le cas d'un vecteur, le

dernier élément est retourné, alors que dans le cas d'une matrice (ou un tableau à plusieurs dimensions) il retourne le dernier élément d'une de ces dimensions.

Exemple 2.33 *Application de l'indice "end "*

>> A=[4 5 6 7];

>> A(end) → retourne

ans = 7

>> A = [4 5 6 7; 3 4 5 6; 2 3 4 5];

>> A(end,1) → retourne

ans = 2,

>> A(2,end) → retourne

ans = 6,

>> A(end,1 :end)

ans = 2 3 4 5 ,

>> A(2 :end,1 :end)

ans =

3 4 5 6

2 3 4 5

2.5 Fonctions opérant sur les éléments d'un tableau

Les fonctions présentées ci-dessous effectuent des opérations arithmétiques itérativement sur les éléments d'une liste ou d'un vecteur. Appliquées à un tableau, elles effectuent ces mêmes opérations sur les colonnes du tableau.

2.5.1 Fonctions sum et prod

– Appliquée à une liste ou un vecteur, la fonction *sum* (respectivement *prod*) calcule la *somme* (respectivement le *produit)* des éléments la liste ou du vecteur.

– Appliquée à un tableau la fonction *sum* (respectivement *prod*) retourne une liste dont chacun des éléments est la *somme* (respectivement le *produit*) des éléments de chaque colonne.

Exemple 2.34 *Effectuer ces calculs*

>> s = [5 2 3 1 7]; p = prod(s) , s = sum(s)

p = 210, s= 18,

2.5.2 Fonctions max et min

Appliquée à une liste ou un vecteur, la fonction *max* (respectivement *min*) détermine le plus grand élément (respectivement le plus petit élément) de la liste ou du vecteur et éventuellement la position de cet élément dans la liste ou le vecteur.

– Appliquée à un tableau la fonction *max* (respectivement *min*) retourne la liste des plus grands (respectivement plus petit éléments) de chaque colonne.

Exemple 2.35 *>> s = [5 2 3 1 7]; [ma, ind] = max(s)*

ma =

7

ind =

5

>> [mi, ind] = min(s)

mi =

1

ind =

4

Exemple 2.36 *>>T=[1 3 0; 5 3 2; 8 9 4]*

>> [ma, ind] = max(T)

ma =

8 9 4

ind =

3 3 3

Pour obtenir la valeur de l'élément maximal du tableau, il suffit d'appliquer deux fois la fonction max :

Exemple 2.37 *>> m = max(max(T))*

m =

9

2.5.3 Fonctions statistiques - mean , cov et norm

Appliquée à une liste ou un vecteur, la fonction *mean* détermine la moyenne des élément de la liste ou du vecteur.

Appliquée à une liste ou un vecteur, *cov* détermine la variance des élément de la liste ou du vecteur.

norm : Calcul les normes vectorielles ou matricielles usuelles ($||.||_1$, $||.||_2$, $||.||_\infty$ ... etc.).

Par exemple les 3 normes vectorielles usuelles de $\mathbb{R}^n$

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|\,, \qquad \|x\|_2 = \left(\sum_{i=1}^{n} (x_i)^2\right)^{1/2}, \quad \|x\|_\infty = \max_{i=1,,,,n} |x_i|$$

sont calculées respectivement avec $norm\ (x,1)$, $norm\ (x,2)$ et $norm\ (x,inf)$.

Exemple 2.38 *>> x = [5 0 3 1 8]*

>> norm $(x,1)$

ans =

17

>> norm $(x, 2)$

ans =

9.9499

>> norm $(x, \inf)$

ans =

8

2.5.4 Fonctions abs

La fonction *abs* appliquée a un tableau retourne un tableau de même format dont les éléments sont les valeurs absolues des éléments du tableau argument.

Exemple 2.39 *>> A = [4 -5 6 7; -3 4 5 6; 2 3 -4 5]*

>> abs(A)

ans =

4	*5*	*6*	*7*
3	*4*	*5*	*6*
2	*3*	*4*	*5*

2.5.5 Réorganisation des éléments d'un tableau - reshape et sort

Soit T un tableau de format (m, n). Si$m \times n = p \times q$, *reshape* (T, p, q) retourne un tableau de format (p, q) dont les éléments sont pris dans T en le parcourant colonne par colonne ; si $m \times n \neq p \times q$,*reshape* (T, p, q) retourne une erreur

Exemple 2.40 *>>T =*

1	*4*	*7*	*13*

2 5 8 11

6 4 9 1

>> P=reshape (T,4,3)

P =

1 6 5 7 9 11

2 4 4 8 13 1

Si u est un vecteur ou une ligne, la fonction *sort* ordonne les éléments de u par ordre croissant.

Exemple 2.41 *>>u = [3 0 1 -5 4];*

>>v=sort (u)

v =

-5 0 1 3 4

Si T est un tableau, la fonction *sort* (T, dim) ordonne par ordre croissant :

– les colonnes de T si la variable *dim* a la valeur 1 ;

– les lignes de T si la variable *dim* a la valeur 1

Exemple 2.42 *>>A = [4 6 7; 1 2 8];*

>>U=sort (A,1) % on ordonne le tableau A selon les colonnes

U =

1 2 7

4 6 8

>> V= sort (A, 2) % on ordonne le tableau A selon les lignes

V =

4 6 7

1 2 8

2.5.6 Opérations sur les ensembles

Matlab dispose de fonctions spécialisées effectuant les opérations sur les ensembles telles que *l'union*, *l'intersection*, le *complémentaire*, etc.

La fonction *union* réalise la réunion de deux vecteurs considérés comme des ensembles. Le vecteur résultat est trié.

Exemple 2.43 *>> x=union ([2 6 1 0],[1 7 3 2 10 5])*

```
x =
     0   1   2   3   5   6   7   10
>> x=union ([2 6 1 0 2 2],[1 7 3 2 10 5],'rows')
x =
     1    7    3    2    10    5
     2    6    1    0    2     2
```

Dans la deuxième forme (i.e. avec l'option 'rows') la fonction *union* retourne une combinaison de ligne des arguments d'entrée sans répétition. Dans ce cas les deux arguments doivent avoir le même nombre de colonnes. Avec l'argument optionnel '*rows*', la fonction *union* accepte aussi des matrices en arguments (avec le même nombre de colonnes).

La fonction *intersect* réalise l'intersection de deux vecteurs ou de deux matrices, considérés comme des ensembles, dans les mêmes conditions (et les mêmes options) que la fonction union.

Exemple 2.44 *>>x=intersect([2 6 1 0],[1 7 3 2 10 5])*

```
x =
     1    2
```

La fonction *setdiff (x,y)* retourne le complémentaire du vecteur y dans le vecteur x. En d'autres termes, la fonction *setdiff(x,y)* retourne les éléments de x qui ne sont pas dans y. Avec deux matrices ayant le même nombre de colonnes, la fonction retourne les lignes de x qui ne sont pas dans y.

Exemple 2.45 *>>x=setdiff ([2 5 3 0 4],[3 2 1])*

x =

0 4 5

Exemple 2.46 *>> A=[2 4 3 ; 1 0 2 ; 0 3 4], B=[1 1 1 ; 2 4 3]*

A =

2 4 3
1 0 2
0 3 4

B =

1 1 1
2 4 3

>> C=setdiff (A,B,'rows')

C =

0 3 4
1 0 2

La fonction *setxor(x,y)* retourne les éléments qui ne sont pas dans l'intersection de x et y dans les mêmes conditions (et options) que les fonctions précédentes

Exemple 2.47 *>>u=setxor([2 6 1 0],[1 7 3 2 10 5])*

u =

0 3 5 6 7 10

Chapitre 3

Expressions, scripts et fonctions

3.1 Qu'est ce qu'un script

Un script est une séquence d'instructions et de commandes Matlab. Les différentes instructions et commandes doivent être séparées par une virgule, un point-virgule ou saut de ligne. Les instructions s'exécutent dans leur ordre d'écriture, et seules les valeurs des expressions qui ne sont pas suivies d'un point-virgule sont affichées.

Matlab permet d'enregistrer les scripts dans des fichiers texte avec l'extension ".m" appelés m-files et de les réutiliser même après fermeture et réouverture de Matlab.

3.1.1 Écrire un script

Pour écrire un script, on peut utiliser l'éditeur fourni par l'environnement Matlab (ou un autre éditeur comme : gedit, notepad ... etc.). Une fois terminé, un script doit être enregistré dans un fichier avec l'extension ".m" (appelé m-file dans quelques références).

Pour exécuter un script, il suffit de l'appeler par son nom dans la fenêtre de commandes, sans préciser l'extension **'.m'**. C'est pourquoi il est conseillé de choisir des noms significatifs, permettant une identification rapide de l'utilité des scripts créés.

3.2 Instructions dans un script

Cette section présente les différentes catégories d'instructions qu'on peut utiliser dans un script

3.2.1 Les instructions d'entrée-sortie

Les fonctions *input* et *disp* (ou *display*) permettent respectivement à l'utilisateur d'introduire la valeur d'une variable ou d'afficher cette dernière sur l'écran. La syntaxe suivante est utilisée pour la fonction *input* :

variable = *input* ('message indicatif') ;

L'instruction *input* a pour effet d'attendre que l'utilisateur saisisse une valeur au clavier et la valide en appuyant sur la touche **entrée**. La valeur saisie par l'utilisateur est par la suite affectée à la variable à gauche de l'opération d'affectation. Le message passé comme paramètre s'affiche à l'écran pour indiquer à l'utilisateur ce qu'il doit faire, mais il n'a pas d'influence sur le déroulement du programme (ou script).

Pour la fonction *disp* nous utilisons la syntaxe suivante : *disp (variable) ;*

Exemple 3.1 *Script - add.m*

a= input('introduisez la valeur de a : ') ;

b= input('introduisez la valeur de a : ') ;

somme = a + b ;

disp (somme) ;

Le script dans cet exemple utilise la fonction *input* pour demander à l'utilisateur de saisir la valeur des deux variables (le message indicatif lui sera affiché) a et b, puis il calcule et affiche la somme de ces deux nombres en utilisant la fonction *disp*. Nous montrons ci-dessous l'exécution de ce script.

>> add

introduisez la valeur de a ?

introduisez la valeur de b ?

Les deux fonctions *disp* et *display* ont pour mission d'afficher un message (ou une valeur) à l'écran. La seule différence entre les deux est que la fonction *disp* affiche seulement la valeur de la variable passée comme argument, alors que la fonction *display* afficher sa valeur, ainsi que son identifiant.

Cependant il existe une autre fonction d'affichages qui est *num2str*. La fonction *num2str*(x) où x est un nombre, retourne la valeur littérale de ce nombre.

Exemple 3.2 *>> x=5 ;*

disp(x)

5

display(x)

x =

5

>>s = ['la valeur de pi est : ' num2str(pi)] ;

s =

la valeur de pi est : 3.1416

>> a = [1 2 ;3 4] ;

>> disp(a) ?

>> disp(['ordre de la matrice a : ' num2str(size(a,1))]) ;

ordre de la matrice a : 2

Exemple 3.3 *Ecrire un programme Matlab qui construit la matrice tridiagonale A pour n quelconque. Résoudre ensuite le système linéaire $AU = B$ où B est un vecteur colonne unitaire de dimension n.*

n=input ('donner la valeur n :') ;

S=[eye(n) zeros(n,1)] ;

S=S (:, 2 :n+1) ;

*A =2*eye(n) -S -S' ;*

display(A)

% Résolution de AU=B

B=ones(n,1) ;

U=A\B

Aprés exécution

donner la valeur n : 3

A =

2	-1	0
-1	2	-1
0	-1	2

U =

1.5000
2.0000
1.5000

3.2.2 Structures de contrôle

Les structures de contrôle sont des structures qui permettent le contrôle de l'exécution des commandes. Ces structures permettent une exécution conditionnelle de lignes de commandes. Ces lignes de commandes forment ce que l'on appelle communément un bloc.

Les structures de contrôle forment ainsi des blocs, délimités par un mot clé spécifique (donnant le sens de la structure de contrôle) et le mot clé *end.*

Citons quelques symboles utilisés par Matlab et leurs significations

- = Affectation

- == Test d'égalité
- ~= Test d'inégalité (équivalent de $\neq$)
- <= L'équivalent de $\leq$
- >= L'équivalent de $\geq$
- | L'équivalent du *ou* logique
- & L'équivalent du *et* logique
- ~ L'équivalent de la négation logique

3.3 Les instructions conditionnelles (if ... else ... end)

3.3.1 L'instruction if ... end

La commande *if* permet de tester une condition avant d'exécuter une séquence d'instruction. La syntaxe est la suivante :

if condition

bloc d'instructions

end

Le bloc d'instructions entre le test de la condition et la commande *end* n'est exécuté que si la condition testée est satisfaite, sinon les instructions ne sont pas exécutées. Il arrive souvent qu'un programme doit exécuter un bloc d'instructions si une condition est vraie, et un autre bloc si cette même condition est fausse. Plutôt que de tester une condition puis son contraire, il est possible d'utiliser la structure *if ... else ...end*, dont la syntaxe est la suivante :

if condition

bloc d'instructions 1

else

bloc d'instructions 2

end

Notez qu'un seul des deux blocs est exécuté(bloc d'instructions 1 et bloc d'instructions 2).

Le premier bloc est exécuté si la condition est satisfaite, et le deuxième est exécuté sinon. Cette structure fonctionne de la manière suivante :

- ▶ si la condition testée est satisfaite, alors le premier bloc d'instructions est exécuté ;
- ▶ sinon, le deuxième bloc d'instructions est exécuté ;
- ▶ les instructions après la commande *end* (s'il y en a) sont exécutées par la suite.

Exemple 3.4 *Dans cet exemple nous allons écrire un algorithme qui demande à l'utilisateur de saisir deux nombres : a et b, puis il calcule et affiche le résultat de la division de a par b. Évidement la division par zéro ne peut pas être effectuée, donc si l'utilisateur donne la valeur 0 à la variable b l'opération de division ne doit pas être exécutée et le résultat n'est pas affiché.*

```
% Script - division.m
x = input ('Donnez la valeur du numerateur ');
y = input ('Donnez la valeur du denominateur ');
if (y ~= 0)
  res x / y;
  disp(res);
end
```

Le script précédent n'effectue la division par y lorsque ce dernier est différent de 0. Autrement la divisionn'est pas effectuée et rien n'est affiché par le programme. Nous allons maintenant améliorer ce script pour qu'il affiche un message d'erreur indiquant qu'il est impossible de diviser par 0. Pour cela nous allons utiliser la commande *else*.

Exemple 3.5

```
% Script - division.m
x = input ('Donnez la valeur du numerateur ');
y = input ('Donnez la valeur du denominateur ');
if (y == 0)
```

disp (' On ne peut pas diviser par zero') ;

else

res = x / y ;

disp (res) ;

end

Cependant, lorsqu'il y a plus de deux alternatives, la commande *elseif* est utilisée pour tester plusieurs possibilités. La

syntaxe de l'instruction elseif est la suivante :

if (condition 1)

bloc d'instructions 1

elseif (condition 2)

bloc d'instructions 2

elseif (condition 3)

⋮

elseif condition n)

bloc d'instructions n

else

bloc d'instructions n + 1

end

Chaque bloc d'instructions i est exécuté si et seulement si la condition i est vraie, et toutes les conditions j pour

$j < i$ sont fausses. Le dernier bloc d'instructions (n + 1) n'est exécuté que si toutes les conditions testées sont fausses.

Comme exemple nous allons écrire un script Matlab permettant de lire un entier et puis faire le traitement suivant :

1) Afficher le message "positif" si le nombre saisi est strictement supérieur à zéro.

2) Afficher le message "négatif" si le nombre saisi est strictement inférieur à zéro.

3) Afficher le message "nul" si le nombre saisi vaut zéro.

Exemple 3.6 *% Script - signe.m*

```
n = input ('Veuillez saisir un nombre : ');
if  n > 0
disp ('positif ');
elseif  n < 0
disp ('négatif ');
else
disp('nul');
end
```

3.3.2 Les structures itératives

Certains algorithmes nécessitent la répétition de certaines instructions plusieurs fois avant d'obtenir le résultat voulu. Cette répétition est réalisée en utilisant une structure de contrôle de type itératif, nommée boucle.

Une boucle est une structure qui permet d'exécuter un certain nombre de fois un même bloc d'instructions. Nous distinguons deux types de boucles, la boucle *for* et la boucle *while*.

Dans le cas d'une boucle *for*, l'ensemble des valeurs pour lesquelles le bloc est effectué est un ensemble fini, déclaré en début de structure et sa syntaxe est telle que :

```
for variable = ensemble de valeurs
    instructions
end
```

Exemple 3.7 *tracé d'un réseau de courbes avec une boucle for*

```
clear all
```

*x=-pi :2*pi/100 :pi ;*

ligne = 1 ;

for n = [1,2,3,5,7]

*y(ligne, :) = exp(-n*x.^2) ;*

ligne = ligne + 1 ;

end

plot(x,y,'-.o')

En Matlab, la boucle *for* peut s'effectuer sur une variable réelle, et pas seulement sur des entiers.

Exemple 3.8 *>> for v = 1.0 : -0.25 : 0.0*

display(v)

end

La boucle *while* est une boucle qui exécute un bloc d'instructions tant qu'une condition logique est vraie (vaut 1 ou true), et sa syntaxe est telle que :

while condition logique

instructions

end

Le fonctionnement est classique. Un *while* est une boucle dont on ne connaît pas a priori le nombre de termes... et cet ensemble de termes peut être infini.

Exemple 3.9 *>> n = 1 ;*

>> while (2^n <= 100)

n = n + 1 ;

end

>> disp(n-1)

3.4 Fonctions

A coté des fonctions prédéfinies, Matlab offre à l'utilisateur la possibilité de définir ses propres fonctions. Trois méthodes sont possibles : les *m-fonctions* qui sont associées à des *m-files* auxiliaires, les fonctions *anonymes* et les fonction *Inline*. Ces deux derniers types de fonction peuvent être définies directement dans l'espace de travail où ces fonctions vont être utilisées.

3.4.1 m-Fonctions

Comme pour les scripts, une fois que la fonction est définie, elle doit être enregistrée dans un fichier portant le même nom de la fonction avec l'extension *'.m'*. La syntaxe suivante est utilisée pour définir une fonction :

function sortie = nomFonction (listeEntrées)

bloc d'instructions

Une fonction qui a été définie peut être utilisée à partir de la fenêtre de commande, ou bien dans un script de la même façon qu'on utilisait les fonctions prédéfinies.

Exemple 3.10 *% Fonction - moyenne.m*

function res = moyenne(a, b)

res = (a + b) / 2;

Parfois pour définir une fonction courte et simple il est souhaitable d'utiliser les fonctions *inline* et les fonctions *anonymes*.

3.4.2 Fonctions Inline

La fonction *inline* permet de définir directement une fonction sans avoir à créer un fichier portant le nom de cette fonction c'est à dire qu'on peut créer la fonction directement dans

l'espace de travail courant, sans utiliser un m-file auxiliaire. La syntaxe des fonctions *inline* est simple

nom_fonction = *inline* ('expression mathématique de la fonction')

Ou avec précision des arguments, comme suit :

nom = *inline* ('expression mathématique de la fonction', 'arg1', 'arg2')

Exemple 3.11 *Par exemple pour définir la fonction* $f(x) = x^3 + 2x^2 - x + 7$

>> f = inline $('x\hat{}3 + 2 * x\hat{}2 - x + 7', 'x')$ *ou tout simplement*

>> f = inline $('x\hat{}3 + 2 * x\hat{}2 - x + 7')$

>>f(4)

ans =

66

>> f = inline $('x.\hat{}2 + y\hat{}2\ ', 'x', 'y')$

f =

Inline function :

f(x,y) = x.^2+y^2

>> f(2,3)

ans =

13

3.4.3 Fonctions anonymes

Une fonction anonyme est une fonction Matlab qui est définie directement, sans la création préalable d'un fichier spécifique. La syntaxe générale pour définir une fonction anonyme est :

nomFonction = @(entrées) expression

Généralement, on utilise les fonctions anonymes pour créer des raccourcis syntaxiques.

À titre d'exemple, on peut définir une fonction anonyme qui calcule le carré d'une valeur ou d'un tableau :

Exemple 3.12 *carre = @(x) x.^2 ;*

>>carre(5)

ans =

25

De même, on peut définir une fonction plus générale qui calcule la puissance d'une valeur ou d'un tableau :

Exemple 3.13 *puissance = @(x,n) x.^n ;*

>> puissance(3,3)

ans =

27

Comment définir une fonction h(x)=x.*sin(pi/2 * x) sous forme de fonction anonyme ?

Exemple 3.14 *h = @(x) x.*sin(pi/2 * x) ; Cette fonction est alors directement utilisable :*

>> h(3)

ans =

-3

>> ezplot(h) : trace la courbe définie par la fonction h(x) sur l'intervalle par défaut $[-2\pi \;\; 2\pi]$ *pour x.*

Nous verrons plus de detail dans le chapitre 5.

3.4.4 Instructions conditionnelles switch

La structure *switch* est une structure conditionnelle comme la structure *if*, mais, il n'y a pas de conditions logiques, le critère de choix est la valeur d'une expression (ou d'une variable). Cette valeur, que l'on appelle cas (case), permet la sélection du bloc à exécuter. *otherwise* est executée seulement si aucun case n'est pas exécuté.

Sa syntaxe est la suivante :

switch *expression du choix*
case *expression du cas*
instructions
.....
case *expression du cas*
instructions
.....
otherwise
instructions
end

Exemple 3.15 *Exemple sur l'instruction switch*

n=input('entrer une valeur :')
switch rem(n,4) % reste de la division de n par 4
case 0, disp('Multiple de 4')
case 1, disp('1 modulo 4')
case 2, disp('2 modulo 4')
otherwise, disp('Nombre non multiple')
end
>>entrer une valeur :25
1 modulo 4
>>entrer une valeur :35
Nombre non multiple

3.4.5 Instructions de rupture de séquence

Il existe dans Matlab plusieurs instructions de rupture de séquences

La commande *break* termine l'exécution d'une boucle *for* ou *while*. Dans le cas de boucles imbriquées, *break* permet de sortir de la boucle la plus proche.

Pour passer à l'itération suivante, dans une boucle *for* ou *while*, on utilise l'instruction *continue*.

La commande *return* termine l'exécution de la fonction ou du script en cours et permet de revenir au ficher appelant ou au clavier.

La commande *pause* interrompt momentanément l'exécution du programme. Il existe deux variantes

- *pause* attend que l'utilisateur appuie sur une touche du clavier pour continuer l'exécution du programme ;

- *pause(n)* le programme s'arrête pendant n secondes avant de se poursuivre.

3.4.6 La fonction modulo

La fonction modulo qui s'écrit *mod* permet de récupérer le reste de la division euclidienne d'un nombre par un autre

Exemple 3.16 *>> u=mod(27,5)*

>> disp(u)

2

Notons que le modulo est trés pratique pour tester si un nombre est divisible par un autre dans ce cas, le reste de la division est 0. On l'utilise souvent dans ue condition pour tester la divisibilité d'un nombre par un autre.

Exemple 3.17 *Écrire un script Matlab qui permet de tester si le nombre x est divisible par 6*

```
% Script - divisibilité.m
x=input(' saisir une valeur : ');
if mod(x,6) ==0
```

disp (' x est divisible par 6') ;

else

disp (' x n est pas divisible par 6') ;

end

Exemple 3.18 *Ce script testant la parité d'un nombre*

% Script - parité.m

x=input(' saisir une valeur : ') ;

if mod(x,2) ==0

disp (' x est un nombre pair') ;

else

disp (' x n est un nombre impair') ;

end

3.4.7 Commandes et fonctions nargin et nargout

La commande *nargin* qui s'utilise à l'intérieur d'une fonction, retourne le nombre d'arguments d'entrée passés lors de l'appel à cette fonction. Elle permet par exemple de donner des valeurs par défaut aux paramètres d'entrée manquants.

Quand à la commande *nargout* cette dernière retourne le nombre de variables de sortie auxquelles la fonction est affectée lors de l'appel.

Exemple 3.19 *Exemple utilisant la fonction nargin*

function c = test(a,b)

switch nargin

case 2

c = a + b ;

case 1

```
        c = a + a;
    otherwise
        c = 0;
    end
end
>> test(12,34)   % Appel à la fonction avec deux arguments
      ans =
            46
>> test(17)       % Appel à la fonction avec un argument
       ans =
             34
```

Terminons ce chapitre par un ensemble d'exercices corrigés.

3.4.8 Exercices

Exercise 1 *Écrire un script Matlab qui calcule le plus grand de 3 nombres.*

```
T = input('Entrer les trois nombres :','s');
T = str2num(T);
Max = T(1);     %initialiser Max au premier nombre
for i=2 :3
if T(i)>Max     %comparer Max avec chaque nombre suivant
Max = T(i);
end
end
disp(['Le plus grand nombre est : ',num2str(Max)]);
```

Aprés exécution

>>Entrer les trois nombres :12 67 43

Le plus grand nombre est : 67

Exercise 2 *Écrire un script Matlab qui permet de faire la somme des éléments d'un vecteur.*

```
X = input('Entrer le vecteur de nombres :','s') ;
X = str2num(X) ;
Taille = length(X) ;
if Taille==0   %  si aucun élément rentré, afficher un message
disp('pas d"éléments dans le vecteur') ;
else
Som = 0 ;  % initialiser la somme à 0
for i=1 :Taille   %  sommer un à un les éléments du vecteur
Som = Som+X(i) ;
end
disp(['la somme des éléments du vecteur est :',num2str(Som)]) ;
end
```

Aprés exécution

>> Entrer le vecteur de nombres :[5,20,3,10,7]

la somme des éléments du vecteur est :45

Exercise 3 *Écrire un script Matlab permettant de calculer un diviseur commun à x et y en partant de la plus petite de ces deux valeurs i.e le PGCD(x,y).*

Solution

```
% Script - pgcd.m
x=input(' saisir une valeur : ') ;
y=input(' saisir une valeur : ') ;
n=min (x,y) ;
```

while ~(mod(x,n) ==0 && mod(y,n)==0)

n=n-1 ;

end

disp(n)

Exercise 4 *Ecrire une fonction fact qui demande un nombre de départ et qui permet de calculer sa factorielle.*

Solution

function y=fact(n)

y=1 ;

for i=2 :n,

*y=y*i ;*

end

On peut évidemment vectorser ce script et éviter la boucle for :

x=input(' saisir une valeur : ') ;

p=prod(1 :x) ;

disp(p)

Exercise 5 *Ecrire un script Matlab qui demande à l'utilisateur de saisir un nombre entier positif n, et puis calcule et affiche les n premiers termes de la suite U_n définie comme suit :*

$$U_1 = 2, \quad U_2 = 3, \; U_i \; = 2/3 \; U_{i-1} \; - 1/4 \; U_{i-2}$$

Solution

n= input('Veuillez saisir le nombre de termes :') ;

U(1)= 2 ;

display(U(1)) ;

U(2)= 3 ;

display(U(2)) ;

for i= 3 :n

*U(i) = (2/3) * U(i-1) -(1/4) * U(i-2) ;*

display(U(i)) ;

end ;

Exercise 6 *Ecrire un programme Matlab qui effectue le calcul des coeffcients binomiaux*

$$C_k^n = \frac{n!}{k!\,(n-k)!}$$

où n et k sont des entiers positifs avec $k \leq n$.

Solution

function coeff = binomial(n,k)

k = fix(k) ; n = fix(n) ; if k > n

disp(' tnombres non compatibles'.. :-(') ;

return ;

end

if k > n/2

k = n-k ;

end

if k <= 1

coeff = n^k ;

else

numerateur = (n-k+1) :n ;

denominateur = 1 :k ;

coeff = prod(numerateur ./ denominateur) ;

end.

Exercise 7 *Ecrire un script Matlab qui demande à l'utilisateur de saisir un nombre entier positif n, et puis affiche s il est divisible par 3 ou 5 .*

Solution

```
n=input('Donner un nombre positif ');
if rem(n,3)==0
disp('Ce nombre est divisible par 3')
elseif rem(n,5)==0
disp('Ce nombre est divisible par 5')
else
disp('Ce nombre n'n'est pas divisible par 3 ou par 5')
end
```

Chapitre 4

Matlab et l'analyse numérique

4.1 Fonctions "numériques"

Les fonctions numériques de Matlab généralisent les fonctions numériques usuelles, avec une différence cependant elles sont vectorisées. C'est à dire qu'elles s'appliquent aussi bien à des nombres qu'à des tableaux. Lorsqu'une de ces fonctions à pour argument un tableau, la fonction est appliquée à chacun de ces éléments. Le résultat obtenu est donc un tableau du même format que le tableau donné comme argument

4.2 Polynômes

Pour Matlab, un polynôme est une liste : la liste des coefficients ordonnés par ordre décroissant :

Exemple 4.1 *Le polynôme p(x) = 1 - 2x + x^3 est représenté par la liste :*

```
>> p = [1   0   -2   1]
     p = 1    0   -2    1
```

4.2.1 Manipuler les polynômes

La fonction la plus utilisée sur les polynômes est évidemment celle qui permet d'évaluer le polynôme p (la fonction polynômiale) en des points donnés. Sa syntaxe est *polyval* (p, x) où x est une valeur numérique ou un vecteur. Dans le second cas on obtient un vecteur contenant les valeurs de la fonction polynômiale aux différents points spécifiés dans le vecteur x.

Exemple 4.2 *Dans cet exemple nous allons évaluer le polynôme P(x) = 2x² - 3x + 1 pour x = 2.*

```
>> P=[ 2 -3  1];
>> polyval (P, 3)
ans =
    10
>> polyval(p,[-2,-1,0,1])
ans =
     1     5     6     1     0
```

Les fonctions usuelles du calcul polynomial sont les suivantes :

roots(p) : Trouve les *racines* d'un polynôme.

conv(p,q) : Effectue le *produit* des polynômes p et q.

deconv(p,q) : Effectue le *division* des polynômes p et q.

poly(racines) : Trouve le *polynôme* à partir des ses racines.

polyder(p) : Calcule la *dérivée* du polynôme p.

polyint(p) : Calcule *l'intégrale* du polynôme p.

Exemple 4.3 *Supposons qu'on souhaite calculer les racines de l'équation :*

$$x^5 - 2x^4 + 2x^3 + 3x^2 + x + 4 = 0$$

Tout d'abord, entrons les coefficients du polynôme :

>> P = [1 −2 2 3 1 4];

>> solution = roots(P)

solution =

1.5336 + 1.4377i

1.5336 - 1.4377i

-1.0638

-0.0017 + 0.9225i

-0.0017 - 0.9225i

Étant donné un tableau de racines, la fonction poly renvoie les coefficients du polynôme avec ces racines ordonnées par puissances décroissantes. Ainsi, dans cet exemple, on peut récupérer le polynôme d'origine comme suit :

>> poly(solution)

ans =

1.0000 -2.0000 2.0000 3.0000 1.0000 4.0000

4.2.2 Calcul matriciel

Voici quelques-unes des fonctions usuelles du calcul matriciel :

eig : calcul des valeurs et des vecteurs propres

poly : polynôme caractéristique

det : calcul du déterminant

trace : calcul de la trace

inv : calcul de l'inverse

Exemple 4.4 *a = [1 2 ; 3 4]*

>> vp = eig(a) ?

vp =

-0.3723

```
    5.3723
>> [p, d] = eig(a) % p est la matrice des vecteurs propres
                et % d est la matrice diagonalisée
p =
   -0.8246   -0.4160
    0.5658   -0.9094
d =
   -0.3723         0
         0    5.3723
>> p = poly(a)  % le polynôme caractéristique est p(t)
p =
    1.0000 -5.0000 -2.0000
>> roots(p) % Comparer les racines de p avec les valeurs propres
ans =
    5.3723
   -0.3723
```

4.3 Fonctions d'une variable

4.3.1 Recherche du minimum d'une fonction

La fonction *fminbnd* permet de trouver le minimum d'une fonction à une variable.

Il faut fournir d'une part la fonction f elle-même et d'autre part les bornes inférieure et supérieure de l'intervalle dans lequel le minimum est recherché. f peut être définie dans un fichier ou par l'intermédiaire d'une fonction anonyme.

Exemple 4.5 *On cherche le minimum de la fonction suivante : sin(x.^2) - x.*cos(x) dans l'intervalle* $(-2\pi, 2\pi)$

```
>> f= @(x) sin(x.^2) - x.*cos(x);
>> x_minimum = fminbnd (f,-2*pi,2*pi)
x_minimum =
    0.3919
```

4.3.2 Recherche de racines - fzero

La syntaxe de la fonction *fzero* est voisine de celle de la fonction fmin. La fonction *fzero* prend pour arguments le nom de la fonction à étudier écrite sous forme d'une chaîne de caractères et une valeur initiale voisine de celle d'une racine. La fonction peut être une fonction prédéfinie de Matlab ou une fonction définie par l'utilisateur, mais elle doit impérativement être une fonction de la variable x.

Exemple 4.6 *Considérons la fonction :* $f(x) = exp(x) - 2 - x$ *et essayons d'en trouver les zeros*

```
>> f = @(x) exp(x)-2-x;
>>fzero(f,1)
ans =
        1.1462
>> f(1.1462)
 ans =
     1.4550e-005  %  la solution est proche de zero
>> fzero(f,-1)
ans =
       -1.8414
>> f(-1.8414)
 ans =
     -4.7627e-006    %  la solution est proche de zero
```

Exemple 4.7 *Soit* $f(x) = exp(x) - 2\cos(x)$*.Pour trouver la solution de f(x) = 0 au voisinage du point x = 0,5, on lance la commande 'fzero('f',0.5)'. Cette fonction affiche les valeurs de f(x) calculées à chaque itération jusqu'à ce que la solution souhaitée x* soit trouvée. Pour cela nous devons créer un fichier (par exemple 'f.m') contenant la fonction.*

function f=f(x)

*f=exp(x)-2*cos(x)*

>>d=fzero('f',0.5)

f =

-0.1064

-0.1430

-0.0692

-0.1579

-0.0536

-0.1788

-0.0313

-0.2080

5.8950e-004

-3.0774e-005

-4.1340e-009

2.2204e-016

-8.8818e-016

d =

0.5398

4.3.3 Intégration - trapz, quad et quad8

Matlab propose plusieurs fonctions pour calculer numériquement la valeur de l'intégrale d'une fonction d'une variable, sur un intervalle fermé.

– *trapz* - La fonction *trapz* utilise la méthode des trapèzes. les arguments de *trapz* sont deux listes, dans l'ordre :

– une liste x qui est une subdivision de l'intervalle d'intégration ;

– une liste y dont les valeurs sont les images des valeurs de de la liste x par la fonction à intégrer (y(k) = f(x(k))).

Exemple 4.8 *>> x = 0 : pi/100 : pi ; y = sin(x) ;*

>> z = trapz(x, y)

z =

1.9998

quad et *quad8* - Ces deux fonctions sont fondées respectivement sur la méthode de Simpson et sur la méthode de Newton-Cotes.

Exemple 4.9 *>> z = quad('sin', 0, pi)*

z =

2.0000

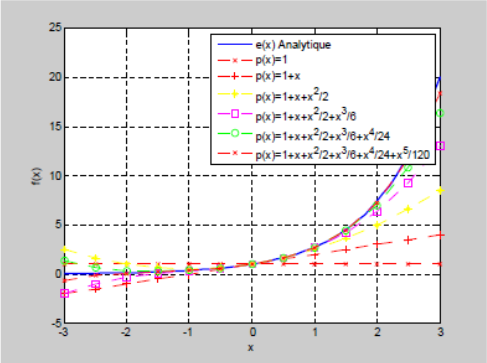
Pour terminer, et suite à ce qu'il a été indiqué dans l'introduction du chapitre 4 voici quelques programmes avec le logiciel Matlab consernant la discrétisation de certaines équations différentielles et partielles par la méthode des différences finies réalisés par les étudiants en des séances de TP.

D'abord pour commencer voici un code Matlab qui permet de visualiser l'approximation de la fonction exponentielle par la série de Taylor où on a considéré des approximations d'ordre 0, 1, 2,3, 4 et 5

code Matlab

```
% Approximation de la fonction exponentielle par série de Taylor
% Nettoyage
clear
clc
% la pseudo analytique
x=(-3 :0.1 :3) ;
Yexact=exp(x) ;
plot(x,Yexact,'LineWidth',2)
xlabel('x')
ylabel('f(x)')
hold on
grid on
legend('e(x) Analytique')
pause
% Approximation d'ordre 0
x=(-3 :0.5 :3) ;
Y1=x.^0 ;
plot(x,Y1,'--xr')
hold on
legend('e(x) Analytique','p(x)=1')
pause
% Approximation d'ordre 1
Y2=1+x ;
plot(x,Y2,'--+r')
hold on
legend('e(x) Analytique','p(x)=1','p(x)=1+x')
```

```
pause
% Approximation d'ordre 2
Y3=1+x+x.^2/2;
plot(x,Y3,'–*y')
hold on
legend('e(x) Analytique','p(x)=1','p(x)=1+x','p(x)=1+x+x^2/2')
pause
% Approximation d'ordre 3
Y4=1+x+x.^2/2+x.^3/6;
plot(x,Y4,'–sm')
hold on
legend('e(x) Analytique','p(x)=1','p(x)=1+x','p(x)=1+x+x^2/2','p(x)=1+x+x^2/2+x^3/6')
pause
% Approximation d'ordre 4
Y5=1+x+x.^2/2+x.^3/6+x.^4/24;
plot(x,Y5,'–og')
hold on
legend('e(x) Analytique','p(x)=1','p(x)=1+x','p(x)=1+x+x^2/2','p(x)=1+x+x^2/2+x^3/6',...
'p(x)=1+x+x^2/2+x^3/6+x^4/24')
pause
%% Approximation d'ordre 5
Y6=1+x+x.^2/2+x.^3/6+x.^4/24+x.^5/120;
plot(x,Y6,'–xr')
hold on
legend('e(x) Analytique','p(x)=1','p(x)=1+x','p(x)=1+x+x^2/2','p(x)=1+x+x^2/2+x^3/6',..
'p(x)=1+x+x^2/2+x^3/6+x^4/24','p(x)=1+x+x^2/2+x^3/6+x^4/24+x^5/120')
pause
```



Visualisation de l'approximation de

exp(x)

Programme 1 : Résolution Numérique de l'équation $4u"(x) - 2u'(x) + u(x) + x = 0$ avec $u(0) = 1$ et $u(20) = 10$ par la métode des Différences finies pour $h = 5$

Le problème discrét utilisé dans ce code Matlab est donné par :

$$\begin{cases} 4\frac{u_{i-1}-2u_i+u_{i+1}}{h^2} - 2\frac{u_{i+1}-u_{i-1}}{2h} + u_i + x_i = 0, \ \ i = 1, 2, 3 \\ u_0 = 1, \quad u_4 = 20 \end{cases}$$

Code Matlab

clear all ; clc ; close all ;

n=input('le nombre de noeuds intérieurs') ;

x§ ;

u§ ;

u§ ;

h=§ ;

x=(§,x§,n+2) ;

B=§§§§(§) ;

A=§*(§) ;

for§

A(k,§)=-h+§ ;

A(§,k)=h§ ;

B§ ! ! ! !x§) ;

end

B(1)§§§§§ ;

B(n)§§§§§ ;

u=§§§§ ;

u=[§§§§]

disp§§§§§§ ;

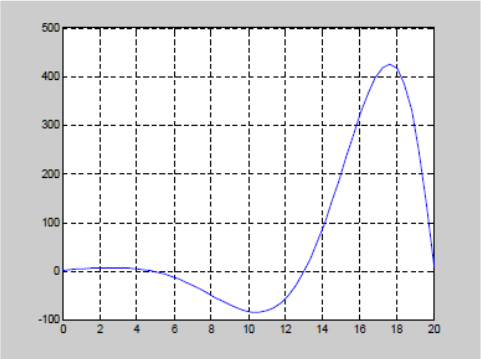
§§§§§§

plot§§§§§

Résultats pour n=3

x'	u
0	1.0000
5.0000	-8.5355
10.0000	-11.1042
15.0000	-15.5919
20.0000	10.0000

Visualisation graphique pour n=50



visualisation graphique n=50

Programme 2 : Résolution Numérique de l'équation : $au'(t) = bu(t) + ct$ par la méthode d'Euler explicite avec $u(0) = 1$.

On rappelle que la formule d'Euler explicite est donnée par : $u_{i+1} = u_i + hf(t_i, u_i)$ où $t_i = t_0 + ih$

Code Matlab

% Coefficients de l'équation au'=bu+t

a=§§§§. ;

b=§§§§. ;

c=§. ;

% T§§§§§§§

tinit§§§§§. ;

tmax§§§§. ;

% Nombre §§§§§§§

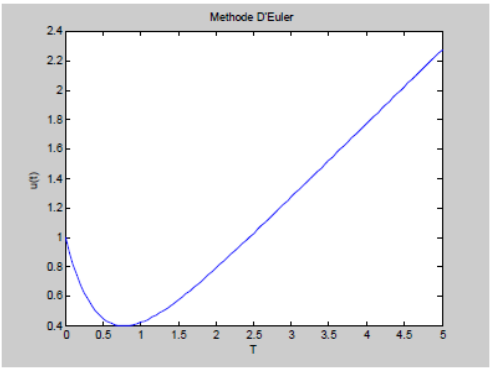
maxt§ ;

dt = (§§§§§§§§§)

% §§§§§§§initiale

u(1)§. ;

```
t(1)=§;
% Boucle§§§§§§
for j=§§§§§
u§§§§§=§ :
t§§§§§§§=§
end;
% Figure
plot(t,u)
title('§§§§§§§§ ')
xlabel('§§§§')
ylabel('§§§§§)')
```



Visualisation graphique de la solution

Programme 3 : Le code Matlab suivant permet le calcul de la solution approchée du problème

$$\begin{cases} -u''(x) = \pi^2 \sin(\pi x), \quad x \in [0,1] \\ u(0) = u(1) = 0 \end{cases}$$

par la méthode des différences finies avec conditions aux limtes de Dirichlet homogène et compare à la fin la solution nmérique avec la solution analytique évaluée par $\sin(\pi x)$.

Rappelons qu'en utilisant un schéma centré pour la dérivée seconde l'équation discrète s'écrit :

$$\begin{cases} -\frac{u_{i-1}-2u_i+u_{i+1}}{h^2} = \pi^2 \sin(\pi x_i), \quad i = 1, ...N \\ u_0 = u_{N+1} = 0 \end{cases}$$

Code Matlab

```
function [xi,u]=SolveLaplacian(N)
%  Discretization
h=§
x§ ;
f=§§§§§
%  §§§§§§§§§§ ;
u§,
u§ ;
% Matrice,§§§§§§§§§
A=2*diag§-diag§)-diag§ ;
A=§§§§§§§§ ;
A
b=§§§§§§f§ :
b
u=§§§§§§§ ;
u
u=[u0 ; § ; §]  % §§§§§§§§(N)
% Exact §§§§§§§
u§§§§§§=sin§ ;
```

disp(' §§§§§§§§§ :')

uexact

plot(xi,§§§§§§') ;§§§§§ ; plot(xi, §§§§§§§§') ;

xlabel(§§§§§§ ; ylabel(§§§§§§)

legend(§§§§§§§§', 'solution exacte')

% [§§§§§]=ErreurLaplacian(§)

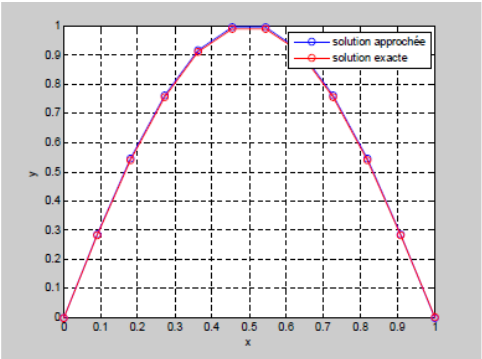
u§§§§§§=sin§ ;

§§§§§§§= norm§ ;

disp('§§§§§§§ § :')

erreur

end



Visualisation graphique pour N=10

Résultats pour N=10

u = [0 , 0.2837, 0.5443, 0.7609, 0.9158, 0.9966, 0.9966, 0.9158, 0.7609, 0.5443, 0.2837, 0]′

uexact =[0, 0.2817, 0.5406, 0.7557, 0.9096, 0.9898, 0.9898, 0.9096, 0.7557, 0.5406, 0.2817, 0]′

La valeur de l'erreur est :

erreur =

0.0068

la valeur de l erreur pour N=100

erreur =

8.0620e-005

Programme 4 : Le code Matlab suivant permet le calcul de la solution approchée de l'équation $u''(x) - u'(x) + xu(x) = xe^x$ sur l'intervalle $[0, 1]$ par la méthode des différences finies avec des conditions aux lmites de Dirichlet non homogènes : $(0) = 1$ et $u(1) = e$ et compare à la fin la solution exacte $u(x) = e^x$ avec la solution numérique.

Notons qu'en utilisant le schéma centré pour la dérivée seconde et le schéma en arrière pour la dérivée première l'équation discrète utilisée dans ce code Matlab est donnée par :

$$\begin{cases} \frac{u_{i-1}-2u_i+u_{i+1}}{h^2} - \frac{u_i-u_{i-1}}{h} + x_i u_i = x_i e^{x_i}, \ i = 1, ...N \ -1 \\ u_0 = 1, \quad u_N = e \end{cases}$$

Code Matlab

clear all ; clc ; close all ;

n§

h§

x=§

b§§§§§§§§ ;

b§§§§§§§§ ;

for§§§§§§§§

b(i)=§§§§§§§ ;

end

for i=1 :§

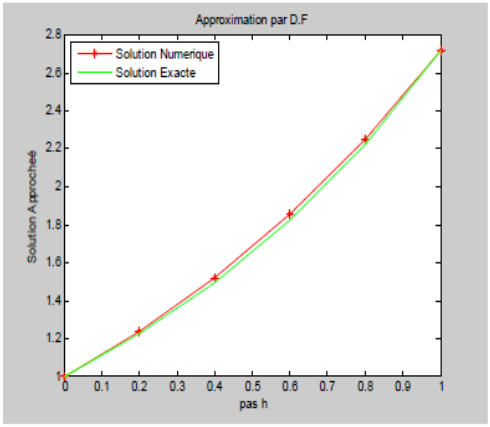
A(i,i)=x(i)*§ ;

end

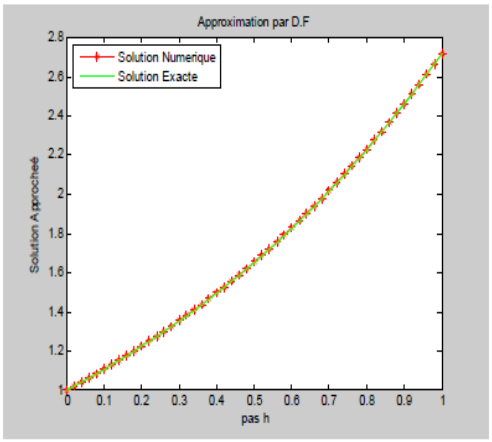
for j=1 :§

A(§§§§§§)= 1 ;

```
A(j+1,j)= § ;
end
disp ('La matrice est')
A
inv(A)
disp ('§§§§§§§§')
b'
x=§§§§§§§ ;
u= inv§*§' ;
u=[§ ;§ ;exp§]'
disp ('La s§§§§§§§§ est')
Uex=§
erreur = §§§§§§§§ ;
disp('la valeur de l erreur est :')
erreur
plot (x,u,'*r-§) ;
legend (§§§§§§§§§','Solution Exacte',2) ;
title ('Approximation par D.F') % figure
xlabel(' pas h') ;
ylabel('Solution Approcheé')
```



Visualisation graphique pour N=5



Visualisation graphique pour N=50

Résultats pour N=5

u =

1.0000 1.2352 1.5173 1.8536 2.2513 2.7183

La solution exact est

Uex =

1.0000 1 .2214 1.4918 1.8221 2.2255 2.7183

la valeur de l erreur est :

erreur =

0.0315

la valeur de l erreur pour N=50

erreur =

0.0045

Programme 5 : Résolution Numérique de léquation $u"(x) - u'(x) = 0$ avec $u(0) = 1$ et $u(1) = 2;718$ par la métode des Différences finies et comparison de la solution discrète avec la solution exacte. Ce qu il y a de plus important dans ce programme est qu il s'agit là

d'une grande boucle **for** qui fait varier le nombre de noeuds à chaque fois qu'on appuie sur la touche entrer (*Enter*) puis fait systématiquement la résolution du système linéaire au fur et à mesure que le nombre d'inconnues et le nombre d'équations augmentent et il compare à la fin la solution nmérique avec la solution analytique évaluée par $exp(x)$.

L'équation discrète utilisée dans ce code Matlab est donnée par :

$$\begin{cases} \frac{u_{i-1}-2u_i+u_{i+1}}{h^2} - \frac{u_i-u_{i-1}}{h} = 0, \ i = 1, ...N \ -1 \\ u_0 = 1, \quad u_N = 2.718 \end{cases}$$

Code Matlab

clear all ; clc ; close all ;

b ;a=§ ;

for §§§§§§§

deltax=(§)/(§) ;

% Preparation des Matrices

A=§ ;

B=§ ;

% Remplissage des Matrices

for i=§

A(i§)=§§§§ ;

A(§)=§ ;

A(§)=§§§§§

end

A(§)=§§§§ ;

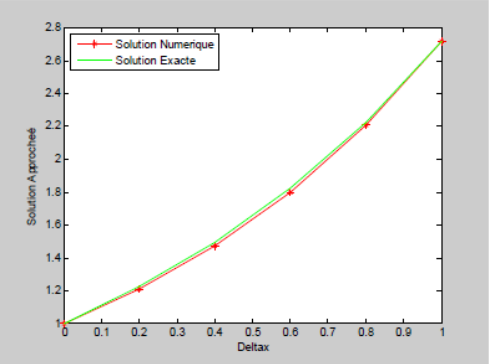
A(§)=§§§§ ;

B(§)=§§§§§ ;

B(§)=§§§§§

% Resolution

```
u=§;
disp(u);
% §§§§§§§§§
X=§;
u_Analytique=exp(X);
disp§§§§§§§§§§);
% Figure
plot(X, u,'*r-',X, u_Analytique,'g');
legend ('§§§§§§§§§§','Solution Exacte',2);
xlabel(' §§§§§§§');
ylabel('Solution Approcheé')
hold §
hold §
pause
end
```



Visualisation pour N de 3 :6

Programme 6 : Résolution numérique de $-u^{"} = (x+1)\,e^{3x^2}$ avec $u(0) = u(1) = 0$ par la méthode des différences finies en utilisant cette fois-ci la fonction anonyme " @,"

L'équation discrète utilisée dans ce code Matlab est donnée par :

$$\begin{cases} -\frac{u_{i-1}-2u_i+u_{i+1}}{h^2} = (x_i+1)\exp(3x_i^2), \ i = 1, ...N \\ u_0 = 1, \quad u_{N+1} = 2;718 \end{cases}$$

Code Matlab

n = §; h =§; xi = §; % Discrétisation du domaine

fx = @§*(xi + 1);

sur_diag = diag§; % Sur-diagonale

des_diag = diag§; % Sous-diagonale

in_diag = diag§; % Diagonale principale

A = §+§+!; % Matrice A

display(A)

B =h^2* §; % Matrice B

display(B)

U = inv§'; % §§§§§§§§§§§§

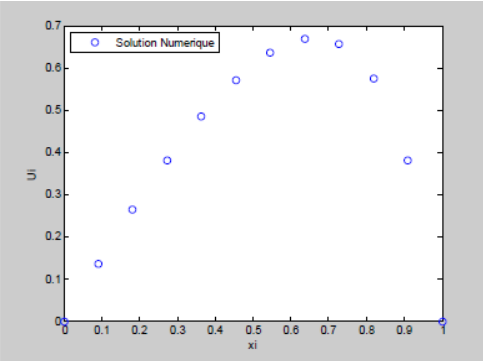
display(U)

U = [0 § 0]; % Vecteur §§§§§§§§§§§§

plot(§,§,'o');

xlabel('xi'); ylabel('Ui');

legend ('§§§§§§§§§§', 2)



Visualisation graphique pour N=10

Programme 7 : Résolution numérique de $-u^{"}(x) + u(x) = x^2 + x - 1$ avec maintenant des conditions aux limites de Neumman : $u'(0) = 2$ et $u(1) = 4$ par la méthode des différences finies en utilisant le schéma discret suivant :

$$\begin{cases} -\frac{u_{i-1}-2u_i+u_{i+1}}{h^2} + u_i = x_i^2 + x - 1 \\ \frac{-3u_0-+4u_1-u_{i2}}{2h} = 2, \qquad u_n = 4 \end{cases}$$

Code Matlab

clear all ; clc ; close all ;

n§

h§

x§

b§

b(n)=(h^2)§ ;

for§§§§§§§§

b§=§§§§§§§§) ;

end

for§§§§§§§§

A(i§)=2+§ ;

end

for§§§§§§§

A(§)= §§§§§§ ;

A(§)= §§§§§§ ;

end

A(§)=-§

A(§)=§

disp ('§§§§§§§ §')

A

disp ('Le vecteur §')

b'

x=§ ;

u= inv§' ;

u=[§ ;§]'

disp (§§§§§§§§ § est')

uex=§

erreur = norm(§ ;

erreur

plot(x,u,'*r-'§) ;

legend ('§§§§§§§§','§§§§§§§§§§',2) ;

title ('Approximation par D.F') % figure

xlabel(' pas h') ;

ylabel('Solution Approcheé')

Résultat pour n=4

n =

4

```
h =
    0.2500
x =
         0    0.2500    0.5000    0.7500    1.0000
```

La matrice A est

```
A =
   -3.0000    4.0000   -1.0000         0
   -1.0000    2.0625   -1.0000         0
         0   -1.0000    2.0625   -1.0000
         0         0   -1.0000    2.0625
```

Le vecteur b est

```
b =
    1.0000
   -0.0273
    0.0156
    4.0664
```

La solution qpprochée est

```
u =
    1.0000    1.5625    2.2500    3.0625    4.0000
```

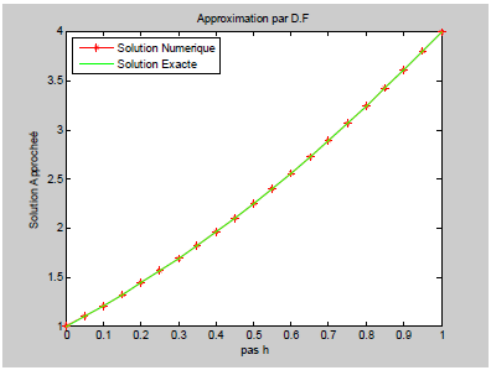
La solution exact est

```
uex =
    1.0000    1.5625    2.2500    3.0625    4.0000
```

la valeur de l erreur est :

```
erreur =
    2.2204e-015
```

On remaque que la solution approchée coincide avec la solution exacte ce qui'est encore bien justifié graphiquement pour n=20.



Visualisation graphique pour n=20

Programme 8 : Reprenons l'exemple 6 mais avec les conditions aux limites suivantes : $u(0) = 1$ et $u'(1) = 4$.

Code Matlab

clear

clc

n=.§

h=§

x=§

b§§§§§§§§§§

b§§§§§§§

for§§§§§§

b§§§§§§§§

end

for§§§§§§

A(§)=§ ;

end

```
for §
A(§)= § ;
A(§)= §§§§§ ;
end
A(§) ;
A§
x=§ ;
u= §' ;
u=§'
uex=§
plot (x,u,'*r-'§') ;
legend ('u',§§§§§§',2) ;
title ('§§§§§§§§§')
xlabel('Le pas ') ;
ylabel('Solution Approchee') ;
```

Programme 9 : Résolution numérique de l'équation de la chaleur

$$\begin{cases} \frac{\partial}{\partial t}u\left(x,t\right)-\alpha\frac{\partial^2}{\partial t^2}u\left(x,t\right)=0, \quad t\in\left]0,T\right[\,,\ x\in\left]0,1\right[\\ u(x,0)=\sin\left(\pi x\right)=u_0\left(x\right),\ x\in\left]0,1\right[\,, \\ u(0,t)=u(1,t)=0, \quad t\in\left]0,T\right[\,, \end{cases}$$

Une discrétisation par la méthode d'Euler explicite du problème conduit à :

$$\begin{cases} u_i^{n+1}=\lambda u_{i-1}^n+\left(1-2\lambda\right)u_i^n+\lambda u_{i+1}^n,\ \forall i=\overline{1,N} \\ u_i^0=\sin\left(\pi x_i\right),\ u_0^n=u_{N+1}^n=0 \end{cases}$$

où $\lambda=\alpha\frac{\Delta t}{\Delta x^2}$. Le code Matlab suivant calcul les trois premières itérations pour $h=\Delta x=0.2$, $k=\Delta t=0.02$. avec $T=0.06$.

Code Matlab

```
clc ;
clear all ;
close all ;
format short
L§ ;                % §§§§§§§§§§§§ 0<x<1
T=§ ;             % §§§§§§§§§§§
§=1 ;            % §§§§§§§§§§§§§
§=0.2 ;           % §§§§§§§§§
dt=§          %§§§§§§§§§
lambda=§          % §§§§§§§§§§§§
N§ ;                              % Noeuds§§§§§§§§§
M§ ;                              % §§§§§§§§§§
x=(§) ;
t=§ ;
for §
x(i)§ ;
end
for §
U=§ ;
U( :,1)§ § ;                      % Conditions aux limites
U§=sin§) ;        % Condition Initiale
W=(§) ;
W
fprintf (' § § §')
for n=§
for i§
U(n+1,i)=§    §     §        §
```

```
fprintf(' %.4f    §    %.4f\n ', §, §, §))
end
fprintf('   t     §     §')
end
U
% figure
plot(x,U(1, :),'-rs',§),'-gs' § :),'-bs',!),'-ys')
title('x § en t=0.06 ')
xlabel('x')
ylabel('Temperature')
grid on
```

Résultats

```
lambda =
            0.5000
W =
       0.5878      0.9511      0.9511      0.5878
t                x            U
0.0200     0.2000       0.4755
0.0200     0.4000       0.7694
0.0200     0.6000       0.7694
0.0200     0.8000       0.4755
t                x            U
0.0400      0.2000      0.3847
0.0400      0.4000      0.6225
0.0400      0.6000      0.6225
0.0400      0.8000      0.3847
 t                x            U
```

0.0600 0.2000 0.3112

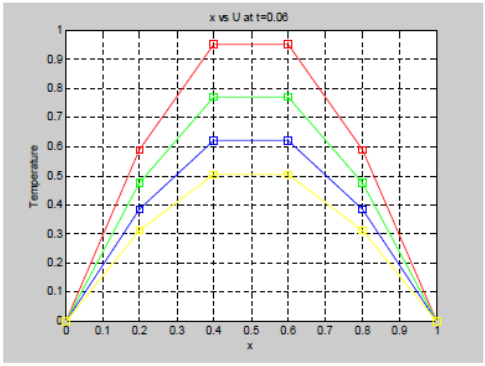
0.0600 0.4000 0.5036

0.0600 0.6000 0.5036

0.0600 0.8000 0.3112

U =

0	0.5878	0.9511	0.9511	0.5878	0
0	0.4755	0.7694	0.7694	0.4755	0
0	0.3847	0.6225	0.6225	0.3847	0
0	0.3112	0.5036	0.5036	0.3112	0



Visualisation de la solution

Programme 10 : Résolution numérique de l'équation de la chaleur

$$\begin{cases} \frac{\partial}{\partial t}u\left(x,t\right)-\alpha\frac{\partial^2}{\partial t^2}u\left(x,t\right)=0, \quad t\in\left]0,T\right[,\ x\in\left]0,1\right[\\ u(x,0)=x\left(1-x\right)\exp(x)=u_0\left(x\right),\ x\in\left]0,1\right[,\ \text{condition initiale} \\ u(0,t)=u(1,t)=0, \quad t\in\left]0,T\right[,\ \text{condition aux limtes} \end{cases}$$

Une discrétisation par la méthode d'Euler implicite du problème conduit à :

$$\begin{cases} u_i^n=-\lambda u_{i-1}^{n+1}+\left(1+2\lambda\right)u_i^{n+1}-\lambda u_{i+1}^{n+1},\ \forall i=\overline{1,N} \\ u_i^0=x_i*\left(1-x_i\right)*\exp(x_i),\ u_0^n=u_{N+1}^n=0 \end{cases}$$

où $\lambda = \alpha \frac{\Delta t}{\Delta x^2}$. Le code Matlab suivant calcul les trois premières itérations pour $T = 0.04$.

Code Matlab

clc ;

clear all ;

close all ;

format short

N§§§§§§§§§

N=§ ;

h=§ ;

T§§§§§§§ ;

M§ ; ;

t=§ ;

lambda=§

A= diag(§)*§§§§§§§§§+diag(§),1)+...

diag(§),-1) ;

A

x=§§§§§§§§§§ ;

W=(§.*§) ;

W

U(§=§ ;

B=inv(§) ;

B

for§

U(§)=B*§)' ;

U

end

Ustart=§§§§§§§§) ; Uend=§§§§§§§(§) ;

U=[§ §§§§ §] ;

U

x=[§ § §] ;

plot(x,U(1, :),'-rs',x,§),'-gs',x§,'-bs',x !),'-ms')

title('x vs U §§§§§§§ ')

xlabel('§§§§§§')

ylabel('§§§§§§§§')

grid §§§§§§§§

Résultats pour N=5

N=5

lambda =

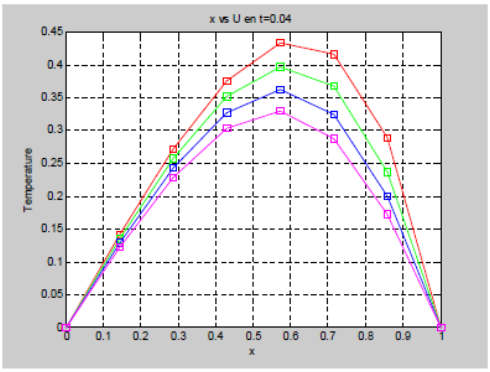
0.4900

W =

0.1413 0.2716 0.3759 0.4337 0.4169 0.2885

U =

0	0.1413	0.2716	0.3759	0.4337	0.4169	0.2885	0
0	0.1351	0.2577	0.3519	0.3970	0.3674	0.2366	0
0	0.1284	0.2429	0.3274	0.3618	0.3245	0.1998	0
0	0.1212	0.2277	0.3031	0.3290	0.2879	0.1722	0



Visualisation de la solution

Chapitre 5

Courbes et surfaces

Matlab est un langage réputé pour la facilité d'utilisation de ses fonctions graphiques. Les vecteurs et les matrices peuvent être visualisé sous forme de courbes en 2D ou bien des courbes et surfaces en 3D ou des histogrammes. Ces graphiques peuvent être légendées et manipulées avec des commandes Matlab soit à partir d'une fenêtre de commandes, d'un script ou même directement à partir de la fenêtre graphique.

5.1 Fenêtres graphiques

Pour créer une fenêtre graphique, nous utilisons la fonction Matlab qui affiche figure (.) une fenêtre vide. La valeur retournée par la fonction figure (.) est le numéro de la fenêtre (à utiliser par la suite pour afficher des composants sur cette fenêtre).

5.1.1 Courbes du plan

La fonction *plot* est sans aucun doute la plus utilisée pour tracer simplement des données contenus dans des tableaux.

L'utilisation la plus simple de la commande *plot* est la suivante : *plot (Vx, Vy)*

où $Vx = [x_1\ x_2\ \ldots x_n]$ est le vecteur d'abscisses, et $Vy = [y_1\ \ y_2\ \ldots y_n]$ le vecteur d'ordonnées.

Exemple 5.1 ***1*** *>> x = 0 :2*pi/100 :2*pi ;*

>> plot(x,sin(x)) ;

Exemple 5.2 *2 Dans cet exemple nous allons voir trois façons d'écrire un script qui trace la courbe de la fonction f(x)=x^3+2*x-1 sur* $[0, 5]$.

1) *clear ; clc ;*

*f=@(x) x^3+2*x-1 ;*

fplot(f, [0 5])

grid on

2) *clear ; clc ;*

f = [1 0 2 -1] ;

x= linspace(0, 5, 100) ;

y =polyval (f, x) ;

*plot(x, y) ; title('f(x)=x^3+2*x-1')*

grid on

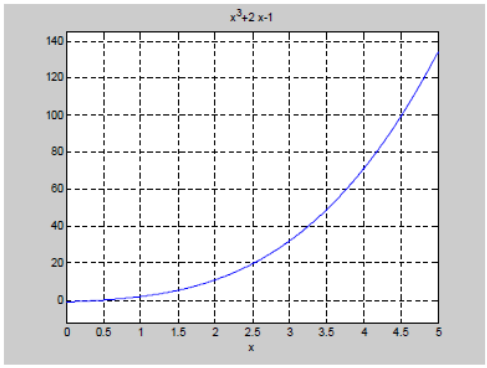
3) *clear ; clc ;*

syms x

*f= x^3+2*x-1 ;*

ezplot(f, [0 5])

grid on



Graphe f(x)=x^3+2*x-1

Remarque : La forme ezplot(f, [0 5]) génère un graphe sur l'intervalle [0 5]. On notera que l'expressin formelle (symbolique f), fonction d'une seule variable x est automatiquement placée en haut du graphe et que l'étiquette de la variable indépendante est automatiquement positionnée sur l'axe voir figure ci dessus.

Matlab définit automatiquement un système d'axes. Pour personnaliser le système d'axes il faut utiliser la fonction axis, pour plus de détails utiliser la commande *help*

Nous présentons ici quelques fonction utiles pour pour manipuler les courbes, les légendes, les axes ... etc. sur une figure.

xlabel : Définir une légende pour l'axe des abscisses d'une figure.

ylabel : Définir une légende pour l'axe des ordonnées d'une figure.

title : Donner un titre à la figure.

axis : Définir un système d'axes.

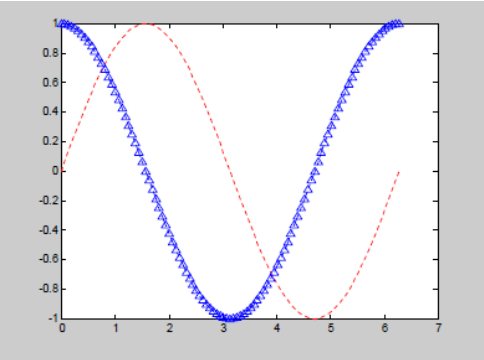
grid : Superposer une grille sur la figure.

legend : Donner une légende à chaque tracée dans la figure.

Remarquons que Matlab attribue des couleurs par défaut aux courbes. Il est possible de modifier la couleur, le style du trait et celui des marqueurs de points, en spécifiant après

chaque couple (abscisse, ordonnée) une chaîne de caractères (entre guillemets).

Exemple 5.3 *x = 0 :2*pi/100 :2*pi ; plot (x,sin(x),' :r',x,cos(x),'b-.^') ;*



Courbe sin(x) et cos(x)

Exemple 5.4 *Soit un vecteur y contenant des valeurs comprises entre* -6π *et* 2π *avec un pas de* 0.001. *Soit deux fonctions f et g définie par :*

$$f(x) = \sin(\frac{\pi}{6}x) + 2x, \quad g(x) = \sin(\frac{\pi}{12}x) - x^2,$$

Ecrire un script Matlab représentant f et g en fonction de y sur le même graphe.

Solution

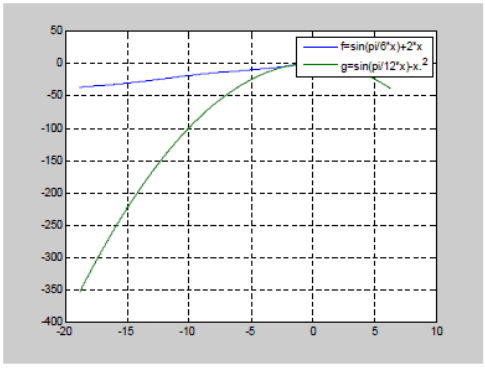
*x=-6*pi :0.001 :2*pi ;*

*f=sin(pi/6*x)+2*x ;*

*g=sin(pi/12*x)-x.^2 ;*

plot(x, f, x, g) ; grid on ;

*legend('f=sin(pi/6*x)+2*x', 'g=sin(pi/12*x)-x.^2')*



Courbes de f et g

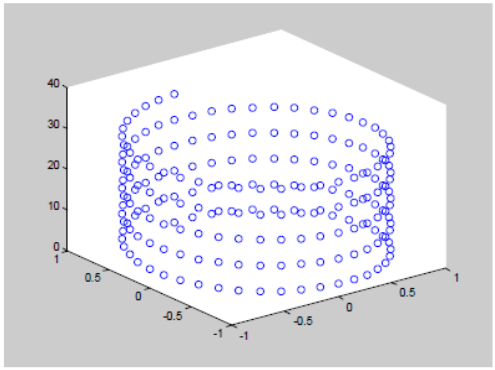
5.1.2 Courbes dans l'espace

La fonction étend les fonctionnalités de la fonction *plot* aux courbes de l'espace. Les possibilités de

personnalisation des axes, ajouter des légendes ou des titres sont les mêmes.

Exemple 5.5 *t = 0 :pi/20 :10*pi ;*

xt = sin(t) ; yt = cos(t) ; plot3 (xt, yt, t,'o')



Courbe dans l'espace

5.1.3 Autres fonctions de tracé de données en 2D

Nous allons présenter ici d'autres fonctions permettant la représentation en 2D de données.

Plot avec deux axes d'ordonnées

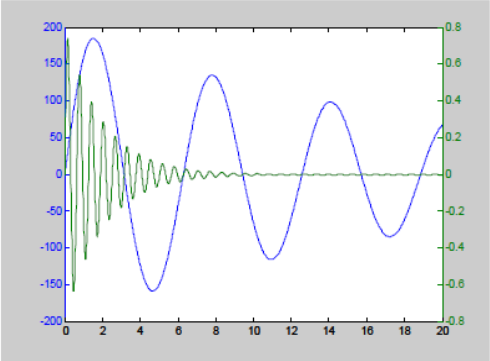
Lorsque l'on veut représenter deux courbes ayant des ordonnées très différentes (i.e. avec des ordres de grandeurs très différents), il peut être nécessaire d'utiliser la fonction *plotyy* qui gère de manière automatique et indépendante les échelles des ordonnées des deux tracés.

Exemple 5.6 *x = 0 :0.01 :20 ; % Représentation graphique ci dessous*

*y1 = 200*exp(-0.05*x).*sin(x) ;*

*y2 = 0.8*exp(-0.5*x).*sin(10*x) ;*

plotyy(x,y1,x,y2) ;



Courbe avec deux axes

la fonction *plotyy* permet de représenter de manière lisible ces deux courbes sur une même figure, contrairement à ce que l'on aurait obtenu avec *plot* .

Plot sous forme de barres

Pour représenter certaines fonctions, notamment dans le cas de fonctions de distribution par exemple, il est d'usage d'utiliser une représentation sous forme de barres plutôt que de points ou de traits. La fonction *bar* permet ce genre de tracé.

Exemple 5.7 *Si l'on veut représenter une loi gaussienne sous forme de barres, on peut écrire :*

```
>> x=-pi :2*pi/100 :pi ;
   bar(x,exp(-x.^2)) ;
```

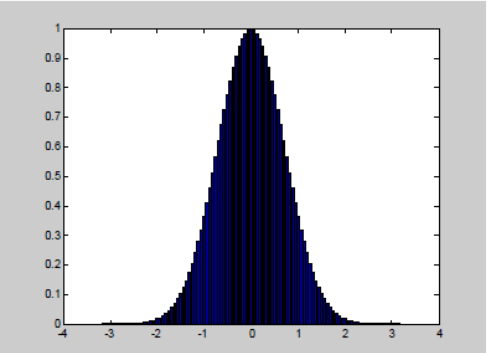
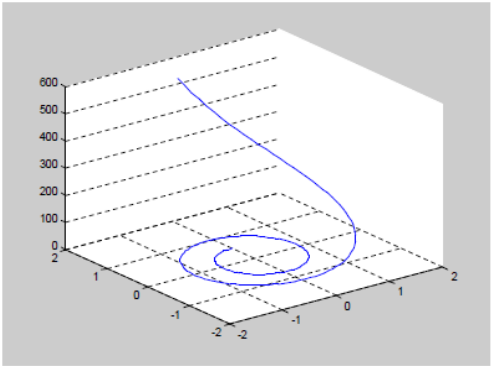


Figure : loi gaussienne

Voici un exemple de courbe paramétrée tracée dans un espace à trois dimensions :

Exemple 5.8 *t = -2*pi :2*pi/100 :2*pi ;*

```
plot3(exp(-t/10).*sin(t), exp(-t/10).*cos(t), exp(-t)) ;
grid
```



Courbe 3D

5.1.4 Tracé de surfaces

On peut utiliser différentes fonctions de tracé de surface, par exemple *mesh*, selon la syntaxe générale suivante :

[X,Y] = meshgrid(x,y) ;

Exemple 5.9 *>> x = -1 :0.2 :1 ;*

y = -2 :0.2 :2 ;

[X, Y] = meshgrid(x,y) ;

Z = Y .^2 - X.^2 ;

mesh(X, Y,Z) ;

En reprenant l'exemple précédemment tracé en surfaces, on peut par exemple tracer 10 lignes de niveau :

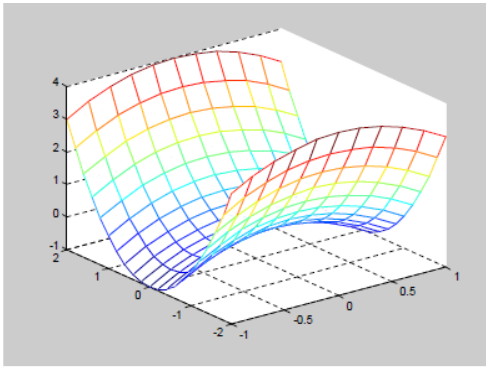
Exemple 5.10 *x = -1 :0.05 :1 ;*

y = -2 :0.05 :2 ;

[X, Y] = meshgrid(x,y) ;

Z = Y .^2 - X.^2 ;

contour(X,Y,Z, 10) ;



Courbe en surface

Les fonctions les plus courantes sont

mesh, qui trace une série de lignes entre les points de la surface en mode «lignes cachées» ;

meshc, qui fonctionne comme mesh mais en ajoutant les courbes de niveau dans le plan (x,y) ;

surf, qui «peint» la surface avec une couleur variant selon la cote ;

surfl, qui «peint» la surface comme si elle était éclairée ;

surfc, qui fonctionne comme mesh mais en ajoutant les courbes de niveau dans le plan (x,y) ;

Exemple 5.11 *Reprenons l'exemple précédent pour comparer les quatre premières fonctions :*

x = -1 :0.2 :1 ;

y = -2 :0.2 :2 ;

[X,Y] = meshgrid(x,y) ;

Z = Y .^2 - X.^2 ;

subplot(221)

mesh(X, Y, Z) ;

xlabel('x') ; ylabel('y') ; zlabel('z') ; title('mesh') ;

subplot(222)

meshc(X, Y, Z) ;

xlabel('x') ; ylabel('y') ;zlabel('z') ; title('meshc') ;

subplot(223)

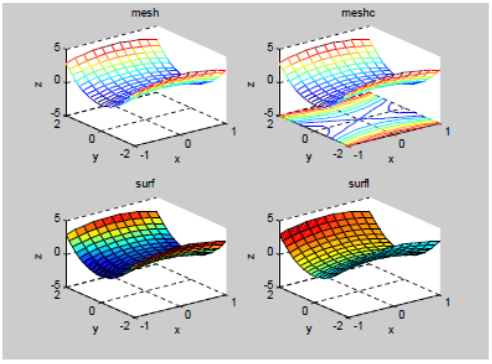
surf(X, Y, Z) ;

xlabel('x') ; ylabel('y') ;zlabel('z') ; title('surf') ;

subplot(224)

surfl(X, Y, Z) ;

xlabel('x') ; ylabel('y') ;zlabel('z') ; title('surfl') ;



Comparaison des quatre fonctions

Bibliographie

[1] F. Z. Baba Hamed. Le calcule scientifique appliqué au Génie Civil sous Matlab, Université des Sciences et de la Technologie Mohamed Boudiaf d'Oran.

[2] J. Baranger. Introduction à l'analyse numérique. Ed. Hermann, 1977.

[3] S. Benkouda. Méthodes numériques (cours et TD). Université Frères Mentouri de Constantine.

[4] Radi Bouchaïd, Mathématiques numériques pour l'ingénieur - Utilisation de l'outil Matlab. Cours, exercices et problèmes de synthèse corrigés - Calcul Scientifique, Ellipses.

[5] Chemseddine Chohra : Polycopié de cours et exercices : Outils d'informatique. Dépt. Info/Math, Année universitaire 2018-2019.

[6] J. Chaussard : Introduction à Matlab, Ecole Sup Galilée - Cursus Ingénieur - 1ère année 2016-2017

[7] J.Dellis . Introduction à Matlab. Université de Picardie

[8] A. Fortin, Analyse numérique pour ingénieurs, Presses Internationales Polytechnique, 2008

[9] J. P. Grivet. Méthodes numériques Appliquées. Collection Grenoble sciences dirigée par Jean Bornarel, © EDP Sciences, 2009. ISBN 978-2-7598-0386-6.

[10] A. Hanachi. Méthodes Numériques pour les ODEs et EDP. Support de cours pour 3ème année Licence Mathématique Académique, Université de Batna 02, Année universitaire 2019-2020.

[11] L. Jolivet et R. Labbas. Analyse et analyse numérique : rappel de cours et exercices corrigés. Applications mathématiques avec Matlab, 2005.

[12] M. Kadja. Résolution numérique des équations aux dérivées partielles : Méthodes des différences finies, Cours et Exercices. Les éditions de l'université Frères Mentouri de Constantine, Année universitaire 2000-2003.

[13] S. Kenouche. Polycopié de cours : Méthodes numériques et programmation. Université M. Khider de Biskra, 2016.

[14] K. Khettab. Polycopié du TP : Méthodes numériques, L2– ST, Méthodes d'approximation numérique pour la résolution des équations non linéaires, Programmation avec Matlab, Université Mohamed Boudiaf - M'sila, Année universitaire 2017-2018.

[15] P. Lascaux.& Téodor R. Analyse numérique matricielle appliquée à l'art de l'ingénieur, volume 2. Masson, Paris, 1994.

[16] M. R. Laydi, Introduction à la méthode des différences finies, Rédaction provisoire 2004-ENS2M.

[17] Th. Lapresté, Introduction à Matlab, ellipses 2009.

[18] A. Mameri. Premier cours en : Méthodes Numériques. Université Larbi Ben Mhidi Oum El Bouaghi.

[19] J-L Merrien, Analyse numérique avec Matlab : Indications, corrigés détaillés, méthodes, Dunod

[20] Y. Morel, Initiation et travaux pratiques Matlab, Master 1.

[21] Alfio. Quarteroni, Riccardo. Sacco, Fausto. Saleri, Méthodes Numériques Algorithmes,analyse et applications, Springer-Verlag Italia, Milano 2004.

[22] A.Quarteroni, Sacco R. and Saleri F. Méthodes numériques pour le calcul scientifique. Springer, 2000.

[23] M. Saad. Analyse numérique. Ecole centrale de Nantes, Dépt. Info/Math, Année universitaire 2011-2012.

[24] Yang, Won Y., et al. Applied numerical methods using Matlab. John Wiley & Sons, 2020.

Pour les programmes veuillez me contacter à cet e-mail friouias@gmail.com