Université M. Khider de Biskra - Algérie
Faculté des Sciences Exactes, Sciences de la Nature et de la Vie
Département des sciences de la matière

Module : Méthodes numériques et programmation

Niveau 2ème année - 1er semestre

SAMIR KENOUCHE
polycopié de cours

Visiter ma page personnelle
http://sites.univ-biskra.dz/kenouche

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Sommaire

Liste des Figures 3

1 Intégration numérique : intégrales simples 8
- 1.1 Méthode du point milieu . . . 16
- 1.2 Méthode du trapèze . . . 17
- 1.3 Méthode de Simpson . . . 20
- 1.4 Au moyen de routines Matlab . . . 28

2 Intégration numérique : intégrales double et triple 33
- 2.1 Intégrale double . . . 33
- 2.2 Intégrale triple . . . 40

3 Résolution d'équations non-linéaires 47
- 3.1 Méthode du point fixe . . . 47
- 3.2 Méthode de dichotomie . . . 54
- 3.3 Méthode de fausse position (ou de Lagrange) . . . 57
- 3.4 Méthode de Newton . . . 59
- 3.5 Méthode de la sécante . . . 63
- 3.6 Au moyen de routines Matlab . . . 66

4 Résolution numérique des équations différentielles 71
- 4.1 Méthodes à un pas . . . 72
  - 4.1.1 Méthode d'Euler . . . 72
  - 4.1.2 Méthode de Heun . . . 73
  - 4.1.3 Méthode de Runge–Kutta, d'ordre 3 . . . 73
  - 4.1.4 Méthode de Runge–Kutta, d'ordre 4 . . . 73
  - 4.1.5 Équations différentielles d'ordre n . . . 79
- 4.2 Au moyen de routines Matlab . . . 83

5 Calcul formel 88
- 5.1 Dérivée d'une Fonction . . . 88
- 5.2 Point d'inflexion d'une fonction . . . 93
- 5.3 Extremums d'une fonction . . . 96
- 5.4 Dérivées partielles . . . 98
- 5.5 Résolution formelle des équations et système d'équations différentielles 102
- 5.6 Résolution formelle d'équations et de système d'équations . . . 107
- 5.7 Résolution formelle des intégrales simples et multiples . . . 113

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6 Méthodes d'interpolation 117
6.1 Méthode de Lagrange 117
6.2 Méthode de Hermite 121
6.3 Interpolation aux nœuds de Tchebychev 124
6.4 Interpolation par spline linéaire 129
6.5 Interpolation par spline cubique 131
6.6 Au moyen de routines Matlab 133

Bibliographie 137

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Liste des Figures

1.1 Interface Matlab . . . 9
1.2 Formule du point milieu composite représentée sur 4 sous-intervalles . 17
1.3 Formule du Trapèze composite représentée sur 4 sous-intervalles . . . 18
1.4 Formule de Simpson composite représentée sur 4 sous-intervalles . . . 21
1.5 Aire de l'intégrale . . . 24
1.6 Influence du nombre de sous-intervalle sur l'erreur d'intégration . . . 28
1.7 Figure générée par le code Matlab ci-dessus . . . 31

2.1 Discrétisation du domaine Ω . . . 34
2.2 Figure générée par le code Matlab ci-dessus . . . 38

3.1 Racine de la fonction f obtenue par la méthode du *point fixe* . . . 51
3.2 Racine de la fonction f obtenue par la méthode de *dichotomie* . . . 56
3.3 Principe de la méthode de *Newton* . . . 59
3.4 Racine de la fonction f obtenue par la méthode de *Newton* . . . 62
3.5 Racine de la fonction f obtenue par la méthode de la *sécante* . . . 65

4.1 Solutions numériques obtenues par les méthodes de `Euler`, de `Heun` et de `Runge-Kutta d'odre 4` . . . 76
4.2 Évolution de l'erreur relative en fonction du pas de discrétisation . . . 79
4.3 Solution exacte et solution numérique obtenue par méthode `Euler` . . 81
4.4 Équation différentielle du troisième ordre résolue par la méthode de `Euler` . . . 83
4.5 Comparaison entre la solution analytique et la solution numérique générée par le solveur `ode23` . . . 85
4.6 Solution numérique $y(t)$ pour différentes valeurs de α . . . 87
4.7 Dérivée première de la solution numérique $y(t)$ pour différentes valeurs de α . . . 87

5.1 Figure générée par le code Matlab ci-dessus . . . 90
5.2 Figure générée par le code Matlab ci-dessus . . . 95
5.3 Figure générée par le code Matlab ci-dessus . . . 97
5.4 Figures générées par le code Matlab ci-dessus . . . 101
5.5 Graphe de la solution $f(t)$ . . . 105
5.6 Graphe des solutions $x(t)$ et $y(t)$ . . . 106
5.7 Figure générée par le code Matlab ci-dessus . . . 111

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6.1 Interpolation de *Lagrange* 120
6.2 Interpolation de *Hermite* 122
6.3 Illustration du phénomène de `Runge` 125
6.4 Atténuation du phénomène de `Runge` en adoptant les nœuds de `Tchebychev` 125
6.5 Effet du nombre de points d'interpolation selon `Tchebychev` 126
6.6 Figures générées par le code Matlab ci-dessous pour $n = 10$ et $n = 20$ 128
6.7 Interpolation par splines linéaires 130
6.8 Interpolation par spline cubique 133

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Liste des Exercices

Exercice ❶ ☞ Ⓡ, *page 9*
Exercice ❶ ☞ Ⓢ, *page 15*
Introduction, *page 16*
Exercice ❶ ☞ Ⓡ, *page 19*
Exercice ❶ ☞ Ⓢ, *page 20*
Exercice ❷ ☞ Ⓡ, *page 22*
Exercice ❸ ☞ Ⓡ, *page 29*
Exercice ❸ ☞ Ⓢ, *page 32*
Introduction, *page 33*
Exercice ❶ ☞ Ⓡ, *page 35*
Exercice ❶ ☞ Ⓢ, *page 40*
Exercice ❷ ☞ Ⓡ, *page 42*
Exercice ❷ ☞ Ⓢ, *page 46*
Introduction, *page 47*
Exercice ❶ ☞ Ⓡ, *page 48*
Exercice ❶ ☞ Ⓢ, *page 53*
Exercice ❷ ☞ Ⓡ, *page 54*
Exercice ❷ ☞ Ⓢ, *page 58*
Exercice ❸ ☞ Ⓡ, *page 60*
Exercice ❸ ☞ Ⓢ, *page 62*
Exercice ❹ ☞ Ⓡ, *page 63*
Exercice ❹ ☞ Ⓢ, *page 65*
Exercice ❺ ☞ Ⓡ, *page 67*
Exercice ❻ ☞ Ⓢ, *page 70*
Introduction, *page 71*
, *page 72*
Exercice ❶ ☞ Ⓡ, *page 74*
Exercice ❷ ☞ Ⓡ, *page 80*
Exercice ❸ ☞ Ⓡ, *page 84*
Exercice ❹ ☞ Ⓡ, *page 86*
Exercice ❶ ☞ Ⓢ, *page 87*
Introduction, *page 88*
Exercice ❶ ☞ Ⓡ, *page 89*
Exercice ❶ ☞ Ⓢ, *page 93*
Exercice ❷ ☞ Ⓡ, *page 94*
Exercice ❷ ☞ Ⓢ, *page 95*
Exercice ❸ ☞ Ⓡ, *page 96*
Exercice ❸ ☞ Ⓢ, *page 98*
Exercice ❹ ☞ Ⓡ, *page 99*
Exercice ❹ ☞ Ⓢ, *page 102*
Exercice ❺ ☞ Ⓡ, *page 104*
Exercice ❺ ☞ Ⓢ, *page 107*
Exercice ❻ ☞ Ⓡ, *page 108*
Exercice ❼ ☞ Ⓡ, *page 110*
Exercice ❼ ☞ Ⓢ, *page 112*
Exercice ❽ ☞ Ⓡ, *page 113*
Exercice ❾ ☞ Ⓡ, *page 115*
Exercice ❾ ☞ Ⓢ, *page 116*
Introduction, *page 117*
Exercice ❶ ☞ Ⓡ, *page 118*
Exercice ❶ ☞ Ⓢ, *page 120*
Exercice ❷ ☞ Ⓡ, *page 121*
Exercice ❷ ☞ Ⓢ, *page 123*
Exercice ❸ ☞ Ⓡ, *page 126*
Exercice ❸ ☞ Ⓢ, *page 129*
Exercice ❹ ☞ Ⓡ, *page 129*
Exercice ❹ ☞ Ⓢ, *page 130*
Exercice ❺ ☞ Ⓡ, *page 132*
Exercice ❺ ☞ Ⓢ, *page 133*
Exercice ❻ ☞ Ⓡ, *page 134*
Exercice ❻ ☞ Ⓢ, *page 136*

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Les étudiants(es) en science possèdent souvent des connaissances mathématiques très développées, néanmoins il a été constaté qu'ils trouvent des difficultés à concrétiser ces connaissances sur un ordinateur. La rédaction de ce polycopié de cours s'inscrit dans cette optique, afin de mettre à la disposition des étudiants(es), d'outils pratiques aidant à la stimulation de leurs connaissances opérationnelles. Ce polycopié s'adresse à tous les étudiants(es) suivant un cursus universitaire de type scientifique, à l'instar de la *physique, la chimie, la biologie, filières technologiques ... etc*. Les prérequis exigés sont relatifs aux notions élémentaires en mathématique appliquée, abordées durant les premières années du cycle universitaire. Bien évidemment, la liste des méthodes numériques présentées ici est strictement conformes au programme officiel.

Toutes les méthodes numériques sont programmées par le biais du "langage" Matlab. Ce dernier est commercialisé par la société *MathWorks* (http://www.mathworks.com/). Le choix de ce logiciel tient aussi, à sa simplicité d'utilisation, car il ne nécessite pas de déclaration explicite de types de variables (entiers, réels, complexes, les chaînes de caractères) manipulées. Matlab est particulièrement efficient pour le calcul matriciel car sa structure de données interne est fondée sur des matrices. De plus, il dispose d'une multitude de boites à outils *toolboxes* dédiées à différents domaines scientifiques (*statistique, traitement du signal, traitement d'images, ... etc*). Il existe des logiciels ayant des syntaxes comparables à celle de Matlab, comme *Scilab* (http://www.scilab.org/), *Octave* (http://www.gnu.org/software/octave/), *FreeMat* (http://freemat.sourceforge.net/), *Maple* (http://www.maplesoft.com/products/maple/) et *Sage* (http://www.sagemath.org/).

Matlab est un langage interprété, son fonctionnement est différent des langages classiques (Fortran, Pascal, ...), dits langages compilés. Un algorithme écrit en langage interprété nécessite pour fonctionner un interprète. Ce dernier est un programme traduisant directement les instructions, en langage machine, au fur et à mesure de leurs exécutions. L'interprète analyse séquentiellement la syntaxe de l'algorithme avant de le dérouler dynamiquement. En revanche, dans un langage compilé, le code source est lu dans un premier temps puis compilé par un compilateur qui le convertit en langage machine directement compréhensible par l'ordinateur. Il en résulte ainsi, qu'un langage interprété sera plus lent qu'un langage compilé à cause de la conversion dynamique de l'algorithme, alors que cette opération est réalisée préalablement pour un langage compilé. Néanmoins, l'un des avantages majeur d'un langage interprété, tient à la facilité de détection d'éventuelles erreurs de programmation. Le programme interprète indiquera rapidement, au cours de l'exécution, l'emplacement de l'erreur de syntaxe et proposera éventuellement une aide supplémentaire. Dans le langage compilé, les erreurs apparaissent au cours de la compilation, qui est souvent longue, et de plus il est difficile d'appréhender l'origine de l'erreur.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Dans ce polycopié de cours, chaque section est suivie d'exercices corrigés de façon détaillée. Les étudiants (es) sont invités à résoudre les exercices supplémentaires, donnés dans chaque fin de section. Cela permettra de tester et de consolider leur compréhension. Par ailleurs, il est vivement conseillé, notamment pour les débutants, d'implémenter "manuellement" les algorithmes avant de recourir aux multiples fonctions et commandes prédéfinies du logiciel. L'apprentissage de ce dernier peut se faire en consultant régulièrement son *help* (aide). Étant donné le nombre très important de *fonction* et de *commande* disponibles, il est impossible de mémoriser chacune d'elles. Notons que ce *help* est disponible en langue anglaise, ce qui nécessite donc un apprentissage des rudiments de cette langue. Les notions abordées dans ce polycopié de cours sont organisées en six chapitres.

Le premier chapitre est consacré à l'intégration numériques (méthode du point milieu, du trapèze et celle de Simpson). Dans le second chapitre, il sera question de la résolution numérique des intégrales double et triple. Le troisième chapitre traite la recherche de racines d'une fonction réelle de variable réelle (méthode de point fixe, dichotomie, Newton, sécante). Le quatrième chapitre mis en lumière les diverses techniques de résolution numériques d'équations différentielles (méthode de *Runge-Kutta*, *Euler* et celle de *Heun*). Le cinquième chapitre mis en exergue les potentialités du logiciel relatives au calcul symbolique. Enfin, le dernier chapitre est dédié aux méthodes d'interpolation (méthode de Lagrange, de Hermite, de Tchebychev et interpolation par spline). Par ailleurs, on notera que l'utilité d'un algorithme se mesure au moins suivant deux critères, qui sont la rapidité de convergence vers la solution approchée et la précision par rapport aux erreurs (erreurs d'arrondi et de troncature) inhérentes au calcul numérique.

La composition typographique est réalisée au moyen du logiciel LaTeX, sous un environnement *Linux*. J'invite les lecteurs à signaler d'éventuelles erreurs et imperfections en envoyant un mail à l'adresse.

✉ kennouchesamir@gmail.com
✉ s.kenouche@univ-biskra.dz
☎ xx xx xx xx

Tous les scripts Matlab, présentés dans ce document, sont écrits avec la version :

```
< M A T L A B (R) >
(c) Copyright 1984-2008 The MathWorks, Inc.
All Rights Reserved
Version 7.6.0.324 (R2008a)
```

Notons au passage, que la société *MathWorks* commercialise deux versions de MATLAB annuellement. Les lettres a et b désignent respectivement les versions sorties en Mars et en Septembre.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

1 Intégration numérique : intégrales simples

Méthodes du point milieu, du trapèze et de Simpson

Sommaire

5.1 Méthode du point milieu 115
5.2 Méthode du trapèze 116
5.3 Méthode de Simpson 119
5.4 Au moyen de routines Matlab 126

L'origine du nom Matlab vient de la combinaison de deux mots qui sont **Mat**rix (Matrice en français) et **lab**oratory (Laboratoire en français). Ce logiciel est utilisé dans les domaines de l'enseignement, de la recherche scientifique et de l'industrie pour le calcul numérique. Matlab est pourvu d'une interface interactive et conviviale, et permet avec une grande flexibilité d'effectuer des calculs numériques, symboliques et des visualisations graphiques de très haute qualité.

La fenêtre principale Matlab Fig. (1.1) regroupe quatre sous-fenêtres qui sont : `Fenêtre de commande` (command window), `Espace de travail` (workspace), `Répertoire de travail` (current folder) et Historique des commandes (command history).

1. La sous-fenêtre centrale `command windows` permet d'introduire séquentiellement les différentes commandes matlab et d'en afficher le résultat. L'invité $\gg$ indique que Matlab est prêt à recevoir les commandes pour réaliser des calculs.
2. Le `Workspace` affiche le nom, le type ainsi que la taille des variables exécutées.
3. Le `Current Directory` affiche le répertoire de travail courant avec son chemin (path en anglais).
4. La sous-fenêtre `Command History` quant à elle énumère toutes les commandes ayant été saisies.

Néanmoins, pour plus de flexibilité il serait recommandé d'écrire les instructions directement dans l'éditeur de texte intégré du logiciel. L'éditeur de texte en question se lance en tapant la commande `edit` dans la fenêtre des commandes. Une deuxième possibilité de lancement de cet éditeur est de cliquer directement sur l'icône *new M-file*. Ainsi, on utilisera l'expression `script Matlab` pour désigner les algorithmes écrits via

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

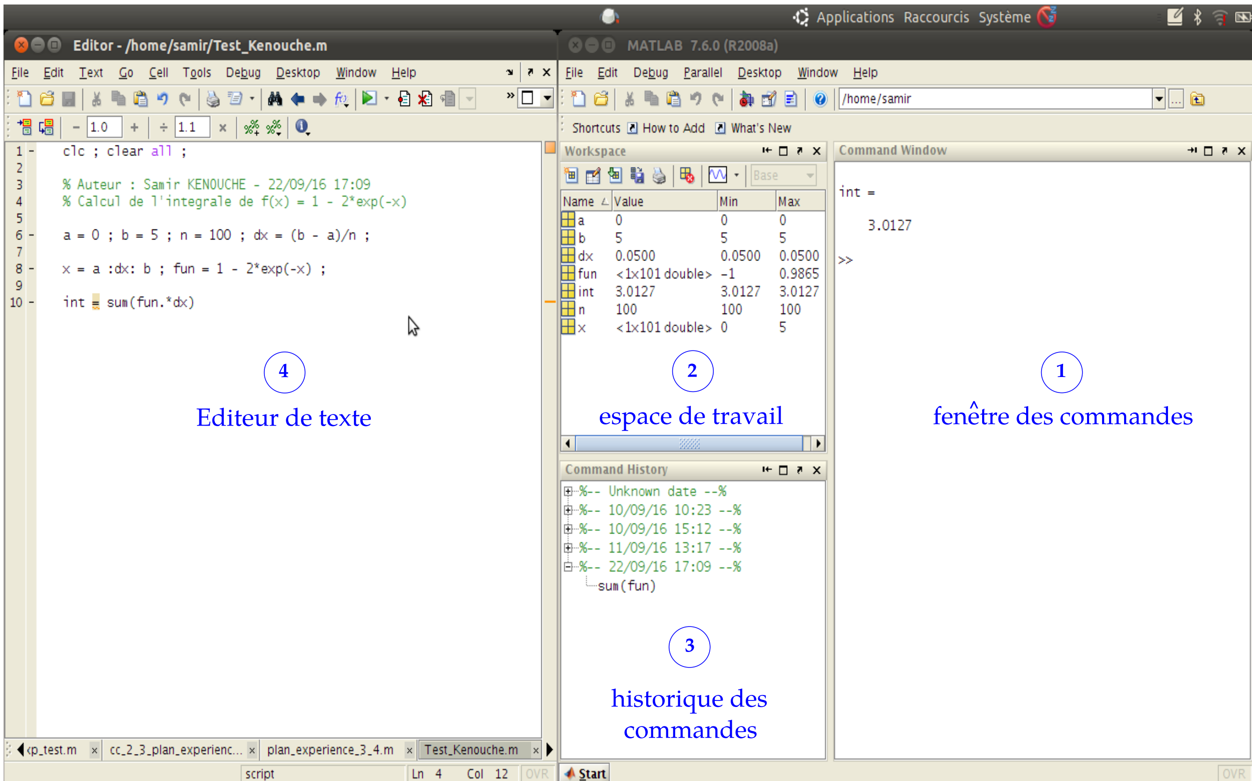

FIGURE 1.1: Interface Matlab

l'éditeur de texte. Dans ce cas le fichier portera l'extension `monFichier.m`. Tout ce qui est écrit après le signe pourcentage (%) est un `commentaire`. Matlab ne tiendra pas compte de ces lignes de commentaires lors de l'exécution du programme.

Avant d'entreprendre l'étude sur les différentes méthodes d'intégration numériques, on présentera dans un premier temps les multiples manières de déclarer et d'évaluer des fonctions mathématiques.

Exercice ❶ ☞ ®

1. Déclarer les fonctions suivantes :

$$\begin{cases} f_1(x) = \dfrac{1+x^2}{\sqrt{x^3+3}}, \text{Avec} \quad x \in [-1, 1] \\ f_2(x) = a + \log(b + x^2) \\ f_3(x,y) = \dfrac{1}{\sqrt{x^2+y^2}} \\ f_4(x,y) = \sin(a + x^2) + \cos(b + y^2) \end{cases} \tag{1.1}$$

2. Évaluer ces fonctions pour $x = 1/2$ et $y = 5$, $a = 0.1, b = 2$ le cas échéant.

Voici le script Matlab :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;
%%%%%%%%%%%%%%%%%%%%%% PREMIERE POSSIBILITE %%%%%%%%%%%%%%%%%%%%%%%%
lowerBound = -1 ; upperBound = 1 ; n = 300 ;
step =(upperBound - lowerBound)/n; x = lowerBound :step: upperBound ;
f1x = (1 + x.^2)./sqrt(x.^3 + 3) ; f1xEval1 = eval('f1x', x) ;

%%%%%%%%%%%%%%%%%%%%%% DEUXIEME POSSIBILITE %%%%%%%%%%%%%%%%%%%%%%%%%
lowerBound = -1 ; upperBound = 1 ; n = 300 ;
step =(upperBound - lowerBound)/n; x = lowerBound :step: upperBound ;

f2x = fittype('a + log(b + x^2)') ; f1xEval2 = f2x(3,1/2,x(18));

%%%%%%%%%%%%%%%%%%%%%% TROISIEME POSSIBILITE %%%%%%%%%%%%%%%%%%%%%%%%

lowerBound = -1 ; upperBound = 1 ; n = 300 ;
step =(upperBound - lowerBound)/n; x = lowerBound :step: upperBound ;
f1x = inline('(1 + x.^2)./sqrt(x.^3 + 3)'); f1xEval3 = feval(f1x, x);

%%%%%%%%%%%%%%%%%%%%%% QUATRIEME POSSIBILITE %%%%%%%%%%%%%%%%%%%%%%%%

lowerBound = -1 ; upperBound = 1 ; n = 300 ;
step =(upperBound - lowerBound)/n; x = lowerBound :step: upperBound ;
f1x = @(x) (1 + x.^2)./sqrt(x.^3 + 3) ; f1xEval4 = f1x(x) ;
```

Les deux dernières méthodes sont plus flexibles dans la mesure où elles permettent une évaluation directe de la fonction. Le résultat affiché par ces deux méthodes, pour $x = 1/2$ est : `f1x(0.5) = 0.7071`. Pour les fonctions avec paramètres ($f_2(x)$ et $f_4(x, y)$), on utilisera la troisième possibilité selon :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f2x = @(x,a,b) a + log(b + x.^2) ;
f4x = @(x,y,a,b) sin(a + x.^2) + cos(b + y.^2) ;
f2xEval = f2x(0.5,0.1,2) ; f4xEval = f4x(0.5,5,0.1,2) ;
```

Les évaluations renvoyées sont : `f2xEval = 0.9109` et `f4xEval = 0.0508`. On peut, en outre, afficher les informations, par exemple, de la fonction `f2x` au moyen de la commande `functions` :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
>> functions(fx2)
>> ans =
        function: [1x21 char]
            type: 'anonymous'
            file: ''
       workspace: [1x1 struct]
```

Il existe une autre possibilité pour déclarer une fonction, qui se fait à travers la création d'un fichier `M-file`.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
function funEval = func(x,y,a,b)
funEval = sin(a + x.^2) + cos(b + y.^2) ;
return
```

Pour écrire un fichier `M-file`, il faut débuter la ligne du programme par le mot-clef `function`. De cette façon Matlab saura que vous êtes entrain d'écrire un fichier `M-file`. Ce dernier doit être sauvegardé, dans le répertoire de travail, sous le nom de `func.m`. Il faut absolument que le nom du fichier soit le même que celui de la fonction définie. Ensuite on appelle cette fonction à partir du programme principal :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;
x = 0.5 ; y = 5 ; a = 0.1 ; b = 2 ;
funEval = func(x,y,a,b)

>> funEval =
            0.0508
```

On peut aussi accroitre la robustesse de cette fonction, en imposant, par exemple, que le nombre d'arguments soit égale à quatre et de type réels. Voici la réécriture du fichier `M-file` :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
function funEval = func(x,y,a,b)

if nargin ~= 4

    errordlg('La fonction admet quatre arguments en input')

end
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
8
9  A = [x, y, a, b] ;
10
11 if isreal(A) ~= 1
12
13        error('Cette fonction accepte des arguments reels')
14 else
15
16     funEval = sin(a + x.^2) + cos(b+y.^2);
17 end
18
19 return
```

De cette façon, si vous tentez d'évaluer cette fonction avec trois arguments, par exemple avec `func(5,3,9)`, Matlab affiche une boite de dialogue contenant le message d'erreur indiqué dans `errordlg('La fonction admet quatre arguments en input')`. D'un autre côté, si l'on essaye d'évaluer cette fonction avec un argument de type *complexe*, par exemple avec `func(5*i,3,9,6)`, Matlab affiche le message d'erreur suivant :

```
??? Error using ==> func at 13
Cette fonction accepte des
arguments reels
```

Il est possible de tester le nombre d'arguments en entrée et en sortie de la fonction `func`. Il suffit d'écrire dans la fenêtre des commandes les instructions suivantes :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clc ; clear all ;
2 input_arg = nargin(@func) % arguments en entree
3 output_arg = nargout(@func) % arguments en sortie
4
5 >> input_arg =
6              4
7 >> output_arg =
8              1
```

Les fonctions `sqrt` et `log` donnent respectivement la racine carrée et le logarithme népérien de l'argument en entrée. Les fonctions `sin` et `cos` sont respectivement les fonctions sinus et cosinus. La syntaxe usuelle de la commande `inline` est `fx = inline(expr, arg1, arg2, ..., argn)`, définissant une fonction `fx` dépendant des arguments en entrée `arg1, arg2, ..., argn`. La chaîne de caractères `expr` traduit l'expression mathématique de `fx`. La syntaxe `fx = @(arg1, arg2,...,argn) expr`

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

définit la *fonction anonyme* (Anonymous Function) dont `@` est le `handle` (identifiant de la fonction). Les commandes `eval` et `feval` ont pour syntaxes respectives `eval('expr', x)` et `feval(inline('expr'), x)`. Elles évaluent la fonction définit par `expr` au point `x`. Ce dernier peut être un scalaire ou un vecteur. La sortie renvoyée par la commande `fittype` est :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;
f2x = fittype('a + log(b + x.^2)')

>> f2x =
     General model:
       f2x(a,b,x) = a + log(b + x.^2)
```

La variable x est prise, par défaut, comme variable indépendante. Si l'on souhaite changer cette variable, il faudra mettre :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;
f2x = fittype('a + log(b + t.^2)','independent','t')

>> f2x =
        General model:
        f2x(a,b,t) = a + log(b + t.^2)
```

Une autre syntaxe possible de la commande `fittype` est d'écrire :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;
fx = fittype({'cos(x)','1'},'coefficients',{'a','b'})

>> fx =
       Linear model:
       fx(a,b,x) = a*cos(x) + b
```

Pour les *fonctions anonymes*, il existe une autre manière de récupérer l'identifiant, en utilisant la commande `str2func`, selon la syntaxe :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
close all ; clc ; clear all ;
a = - pi ; b = pi ; n = 100 ; h = (b-a)/n ; t = a :h: b ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
3 fun =  str2func('cos') ; plot(t, fun(t))
```

En outre, la commande `str2func` peut être combinée avec `cellfun` afin de définir un tableau de fonction. Voici un exemple qui illustre sa syntaxe :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  close all ; clc ; clear all ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  a = 0 ; b = 3*pi ; n = 100 ; h = (b-a)/n ; t = a :h: b ;
4
5  multi_fun = cellfun(@str2func, {'sin' 'cos' 'log' 'sqrt'}, ...
6                     'UniformOutput', false) ;
7  col = {'o-','o-r','o-k','o-g'} ;
8
9  for ik = 1 : size(multi_fun, 2)
10
11         fun = multi_fun{ik}(t) ;
12         plot(t, fun, col{ik}) ; hold on
13
14 end
```

La commande `cellfun` évalue les fonctions à partir de chaque cellule. Ainsi, la syntaxe `multi_fun{1}(t)` fait appel à la fonction `sin` et `multi_fun{2}(t)` fait appel à la fonction `cos` ... etc. L'argument `(...,'UniformOutput', false,...)` renvoie une sortie pour chaque cellule du tableau de cellules `multi_fun`. D'autres commandes intéressantes sont disponibles, à l'instar de `structfun`, `arrayfun` et `spfun`. En cas d'erreur dans un script, Matlab affiche en rouge, un message d'erreur dans la fenêtre des commandes. De plus, Matlab détecte la ligne où s'est produite l'erreur et ajoute un curseur localisant l'erreur en question. Voici un exemple :

```
» fun = @(t,a) sin(a*cos(t)

??? fun = @(t,a) sin(a*cos(t)
                            |
Error: Expression or statement is incorrect-possibly unbalanced (, or
[.
```

Dans le cas où l'erreur s'est produite dans un fichier `M-file`, Matlab affiche en plus le nom du fichier et le numéro de la ligne concernée.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❶ ☞ Ⓢ

1. Déclarer les fonctions suivantes :

$$\begin{cases} f_1(x) = \sqrt{a + x^3} \\ f_2(x) = \dfrac{x^3 + 2\,x + 1}{1 - x^2} \\ f_3(x, y) = a\,\exp(x^2) + b\,\exp(y^2) \end{cases} \tag{1.2}$$

2. Tester les commandes `inline`, `fonction anonyme`, `feval`, `eval` et les fonctions définis à partir d'un fichier `m-file`.
3. Évaluer ces fonctions pour $x = 2$, $y = 5/2$, $a = 3$ et $b = 5$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Introduction

Très souvent le calcul explicite de l'intégrale, d'une fonction f continue sur $[a, b]$ dans $\mathbb{R}$, définie par $I(f) = \int_a^b f(x)\,dx$ peut se révéler très laborieux, ou tout simplement impossible à atteindre. Par conséquent, on fait appel à des méthodes numériques, afin de calculer une approximation de $I(f)$. Dans ces méthodes numériques, la fonction f, est remplacée par une somme finie constituée de n sous-intervalles selon :

$$\int_a^b f(x)\,dx = \frac{(b-a)}{n} \times \sum_{k=1}^{n} f(x_k) \tag{1.3}$$

Dans ce type d'évaluation, on calcul forcément une approximation (passage d'une intégrale à une somme) de la vraie valeur. La méthode d'intégration mise en œuvre est dite d'ordre p si l'erreur d'intégration :

$$Err(f) = \left| \int_a^b f(x)\,dx - \frac{(b-a)}{n} \times \sum_{k=1}^{n} f(x_k) \right| \tag{1.4}$$

est nulle quand f est un polynôme de degré inférieur ou égal à p et non nulle pour au moins un polynôme de degré supérieur ou égal à $p+1$, soit $f \in \mathbb{C}^{p+1}([a, b])$. Dans ce chapitre, nous allons étudier et construire quelques méthodes d'intégration usuelles dites *composites* dans lesquelles la fonction à intégrer est substituée par un polynôme d'interpolation sur chaque intervalle élémentaire $\Delta x = \dfrac{(b-a)}{n}$.

1.1 Méthode du point milieu

Soit f une fonction continue sur l'intervalle [a, b]. On se propose dans cette section d'évaluer l'intégrale $I(f)$ par la méthode du point milieu.

$$I(f) = \int_a^b f(x)\,dx \tag{1.5}$$

Notons que l'expression analytique de $f(x)$ peut être connue comme elle peut être inconnue.

L'idée de base de cette méthode, est de subdiviser l'intervalle [a, b] en n sous-intervalles $[x_k, x_{k+1}]$. Dans le cas où les sous-intervalles sont équidistants, on écrira $\Delta x = (b-a)/n$. Ainsi, le schéma numérique de cette méthode s'écrira comme :

$$I(f) = \sum_{k=1}^{n} \int_{I_k} f(x)\,dx \tag{1.6}$$

$$I(f) = \Delta x \times \sum_{k=1}^{n} f(\bar{x}_k) \qquad \text{avec} \qquad \bar{x} = \frac{x_{k+1} - x_k}{2} \tag{1.7}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

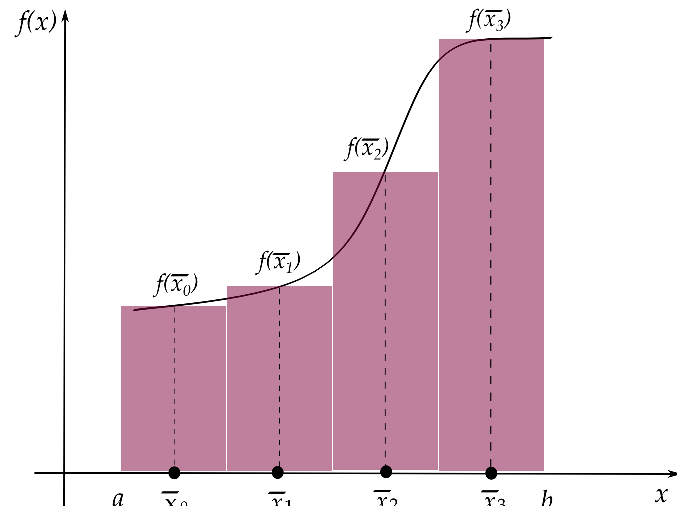

FIGURE 1.2: Formule du point milieu composite représentée sur 4 sous-intervalles

Comme illustré par l'Éq. (1.7), pour chaque $\Delta x = [x_k, x_{k+1}]$ on prend le point central de l'ordonnée $f(\bar{x})$, valeur médiane entre $f(x_k)$ et $f(x_{k+1})$. En effet, $I(f)$, comme montré sur la figure ci-dessus, représente l'aire comprise entre la courbe $y = f(x)$ et l'axe des abscisses entre les droites $x = a$ et $x = b$. Par exemple pour les quatre premiers sous-intervalles, on obtient :

$$I_1(f) \approx \Delta x\, f(\bar{x}_0)$$
$$I_2(f) \approx \Delta x\, f(\bar{x}_1)$$
$$I_3(f) \approx \Delta x\, f(\bar{x}_2)$$
$$I_4(f) \approx \Delta x\, f(\bar{x}_3)$$

Par ailleurs, notons qu'il existe plusieurs façons de mettre en œuvre la méthode des rectangles. Ainsi, on peut prendre la borne inférieure ou la borne supérieure sur chaque intervalle $[x_k, x_{k+1}]$. La méthode du point milieu est d'ordre zéro et son erreur d'intégration est majorée par :

$$\left| \int_a^b f(x)\, dx - \Delta x \times \sum_{k=1}^{n} f(\bar{x}_k) \right| \leq \frac{1}{2} \frac{(b-a)^2}{n} \max_{x\in[a,b]} \left| f^{(1)}(x) \right| \qquad (1.8)$$

1.2 Méthode du trapèze

Cette méthode est basée sur l'interpolation, de chaque intervalle $[x_k, x_{k+1}]$, par un polynôme de degré un. En d'autres mots, sur chaque intervalle $[x_k, x_{k+1}]$, la fonction f continue et dérivable sur $[a, b]$, est substituée par la droite joignant les points $(x_k, f(x_k))$ et $(x_{k+1}, f(x_{k+1}))$. Le schéma numérique de la méthode du trapèze est donné par :

$$I(f) \approx \frac{\Delta x}{2} [f(a) + f(b)] + \Delta x \times \sum_{k=1}^{n-1} f(x_k) \qquad (1.9)$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Sur la figure ci-dessous, nous avons écrit la formule du trapèze sur quatre sous-intervalles. Chaque trapèze est obtenu en remplaçant la fonction à intégrer par son polynôme de *Lagrange* de degré un.

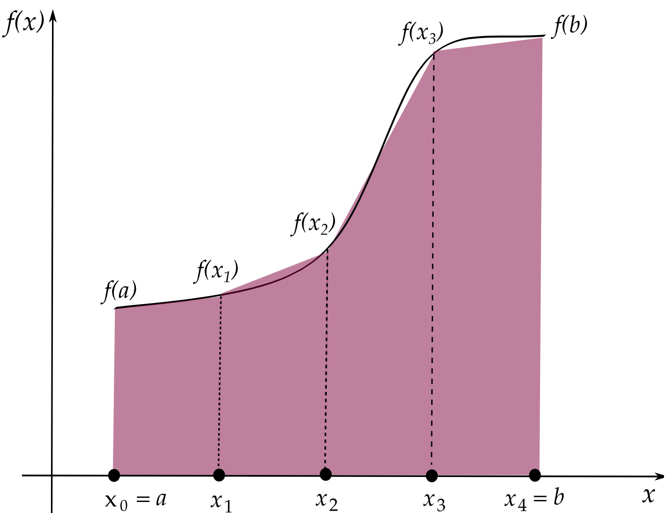

FIGURE 1.3: Formule du Trapèze composite représentée sur 4 sous-intervalles

Par exemple pour ces quatre trapèzes on écrira :

$$I_1(f) \approx \frac{x_1 - x_0}{2}\,[f(x_0) + f(x_1)]$$
$$I_2(f) \approx \frac{x_2 - x_1}{2}\,[f(x_1) + f(x_2)]$$
$$I_3(f) \approx \frac{x_3 - x_2}{2}\,[f(x_2) + f(x_3)]$$
$$I_4(f) \approx \frac{x_4 - x_3}{2}\,[f(x_3) + f(x_4)]$$

Tous les nœuds d'interpolation ont le même poids $\frac{\Delta x}{2}$. La méthode du trapèze est d'ordre un $(O(h^2))$ et fournit une bien meilleure précision que la méthode du point milieu $(O(h))$. Il existe une version améliorée de la méthode de trapèze, dite méthode de *Poncelet* dont le schéma numérique est donné par :

$$I(f) \approx \frac{\Delta x}{4}\left(f(x_0) + f(x_{2n}) + 7 \times (f(x_1) + f(x_{2n-1})) + 8\sum_{k=1}^{n-2} f(x_{2k+1})\right) \quad (1.10)$$

L'erreur d'intégration de la méthode du trapèze est majorée par :

$$\underbrace{\left|\int_a^b f(x)\,dx - I(f)\right|}_{Err(I(f))} \leq \frac{1}{12}\,\frac{(b-a)^3}{n^2}\,\max_{x\in[a,b]}\left|f^{(2)}(x)\right| \quad (1.11)$$

De prime à bord on remarque que l'erreur d'intégration est inversement proportionnelle au carré du nombre de sous-intervalles. Ceci se traduit par l'écriture mathématique

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$\int_n = \lim_{n\to+\infty} \sum_n$ qui est d'autant plus vraie que n tend vers l'infini. On prendra un exemple pour calculer l'erreur d'intégration selon la formule ci-dessus.

Exercice ❶ ☞ ®

Considérant l'intégrale définie, sur $[1, 3]$, par $f(x) = 1 + \log(x)$. Cette fonction est de classe $\mathcal{C}^2(x \in [1, 3])$.

– Déterminer le nombre de sous-intervalles permettant d'atteindre une erreur d'intégration inférieure à 10^{-3}.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ; close all ;
2
3  lowerBound = 1 ; upperBound = 3 ; n = 10 ;
4  dx = (upperBound - lowerBound)/n ; xi = lowerBound :dx: upperBound ;
5
6  syms x real
7
8  fun = 1 + log(x) ; dfun = diff(fun) ;
9
10 pretty(dfun) ; ddfun = diff(dfun) ; pretty(ddfun) ;
11 ddfun_real = subs(ddfun, x, xi) ; % 2eme derivee
12
13 [maxi, idx] = max(ddfun_real) ; % max(|f''(x)|) = maxi ==> maxi =
      0.1111
14 ezplot(ddfun, [1 3]) % graphe de la 2eme derivee
```

Les sorties renvoyées sont

```
1  >> pretty(dfun)  % expression formelle de la 1ere derivee
2
3                                        1
4                                       ----
5                                        x
6  >> pretty(ddfun) % expression formelle de la 2eme derivee
7
8                                        1
9                                    -  ----
10                                        2
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
11                                        x
12 >> [maxi, idx] = max(ddfun_real)
13
14 maxi =     % maximum de la 2eme derivee
15
16    -0.1111 % en prendra la valeur absolue ==> 0.1111
17
18 idx =      % indice du maximum de la 2eme derivee
19
20     11
```

Ainsi, afin d'atteindre une erreur $|Err(I(f))| < 10^{-3} \Leftrightarrow \dfrac{(3-1)}{12\,n^2} \times 0.1111 < 10^{-3}$ $\Longrightarrow n^2 > 18.5 \Longrightarrow n > 4.30$. Il en résulte qu'à partir de cinq sous-intervalles, on atteint une erreur de quadrature inférieure à 10^{-3}.

Exercice ❶ ☞ Ⓢ

Soient les intégrales définies par :

$$\begin{cases} f_1(x) = x - \log(x) \quad \text{avec} \quad x \in [1,2] \\ f_2(x) = \cos(x)\, exp(x) \quad \text{avec} \quad x \in [0,\pi] \\ f_3(x) = \dfrac{1}{1+(x-\pi)^2} \quad \text{avec} \quad x \in [0,5] \end{cases} \tag{1.12}$$

– Déterminer, pour les méthode du point milieu et du trapèze, le nombre de sous-intervalles permettant d'atteindre une erreur d'intégration inférieure à 10^{-5}.

1.3 Méthode de Simpson

Cette méthode est basée sur l'interpolation, de chaque intervalle $[x_k, x_{k+1}]$, par un polynôme de degré deux. Ainsi, la fonction f est substituée par ce polynôme du second degré qui définit donc un arc de parabole passant par les points d'ordonnées $f(x_k)$, $f(x_{k+1})$ et $f(x_{k+2})$. Le schéma numérique de cette méthode est donné par :

$$I(f) \approx \frac{\Delta x}{6}\left(f(a) + f(b) + 2 \times \sum_{k=1}^{n-1} f(x_k) + 4 \times \sum_{k=1}^{n-1} f\left(\frac{x_{k+1}-x_k}{2}\right)\right) \tag{1.13}$$

Sur la figure ci-dessous, nous avons écrit la formule ds *Simpson* sur quatre sous-intervalles. Ainsi, chaque sous-intervalle est interpolé par son polynôme de *Lagrange* de degré deux sur trois nœuds.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

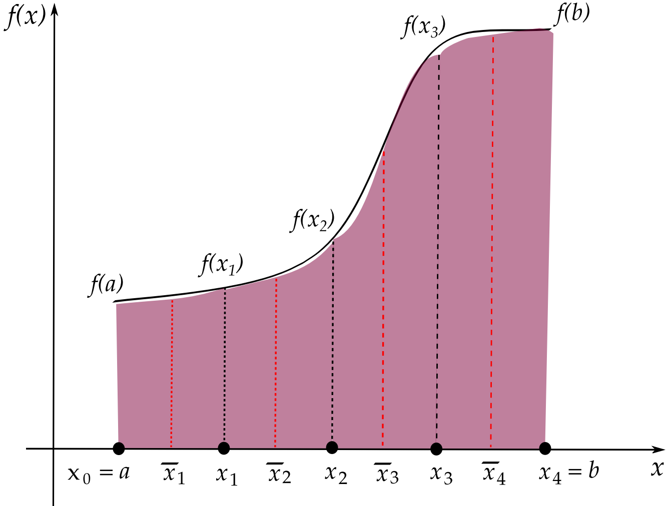

FIGURE 1.4: Formule de Simpson composite représentée sur 4 sous-intervalles

Par exemple pour les quatre premiers sous-intervalles on écrira :

$$I_1(f) \approx \frac{x_1 - x_0}{6}\left[f(x_0) + 4\,f\left(\frac{x_1 + x_0}{2}\right) + f(x_1)\right]$$

$$I_2(f) \approx \frac{x_2 - x_1}{6}\left[f(x_1) + 4\,f\left(\frac{x_2 + x_1}{2}\right) + f(x_2)\right]$$

$$I_3(f) \approx \frac{x_3 - x_2}{6}\left[f(x_2) + 4\,f\left(\frac{x_3 + x_2}{2}\right) + f(x_3)\right]$$

$$I_4(f) \approx \frac{x_4 - x_3}{6}\left[f(x_3) + 4\,f\left(\frac{x_4 + x_3}{2}\right) + f(x_4)\right]$$

Les nœuds d'interpolation situés aux extrémités sont pondérés en $\frac{\Delta x}{6}$. En revanche, les nœuds centraux sont pondérés en $2\,\frac{\Delta x}{3}$. La méthode de Simpson est d'ordre quatre ($O(h^4)$) et fait mieux que celle du trapèze. Ceci provient du fait qu'elle pondère plus le point central. L'erreur d'intégration de la méthode de Simpson est majorée par :

$$\left|\int_a^b f(x)\,dx - I(f)\right| \leq \frac{1}{2880}\,\frac{(b-a)^5}{n^4}\,\max_{x\in[a,b]}\left|f^{(5)}(x)\right| \tag{1.14}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❷ ☞ ®

1. Calculez les approximations de l'intégrale :

$$I(f) = \int_0^{2\pi} x\, exp(-x)\, cos(2\,x)\, dx \simeq -0.122122604618968 \qquad (1.15)$$

en utilisant les méthodes du point milieu, du trapèze et de Simpson. Conclure.

2. Écrire les fonctions `M-files` correspondant à ces trois méthodes d'intégration.
3. Tracer l'aire de l'intégrale , pour `n = 150` sous-intervalles.
4. Étudier l'influence du nombre de sous-intervalles (`n`) sur l'erreur d'intégration.
5. Appliquez les mêmes étapes pour l'intégrale :

$$I(f) = \int_0^{1} exp\left(-\frac{x^2}{2}\right)\, dx \simeq +0.856086378341836 \qquad (1.16)$$

Note : L'algorithme du point milieu doit être écrit de deux façons différentes. D'une part, en utilisant la boucle for et d'autre part, au moyen des fonctions sum et linspace

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 15/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DU POINT MILIEU, du
5  % TRAPEZE ET DE SIMPSON
6
7  %%%%%%%%%%%%%%%%%%%%%% POINT MILIEU : 1ERE APPROCHE %%%%%%%%%%%%%%%%%%
8  a = 0 ; b = 2*pi ; n = 100 ; dx = (b - a)/n ; x = a :dx: b ;
9  fun =  x.*exp(-x).*cos(2.*x) ; int = 0 ;
10
11 for ik = 1:length(x)-1
12
13     xbar = (x(ik) + x(ik+1))/2 ; func = eval('fun', xbar) ;
14     int =  int + dx*func(ik);
15 end
16
17 disp(strcat('L''INTEGRALE, PAR LA METHODE DU POINT MILIEU VAUT :',...
18 num2str(int)))
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
19
20 %%%%%%%%%%%%%%%%%%%% POINT MILEU : 2EME APPROCHE %%%%%%%%%%%%%%%%%%%%
21 a = 0 ; b = 2*pi ; n = 100 ; dx = (b - a)/n ;
22 x = a :dx: b ; fun =  inline('x.*exp(-x).*cos(2.*x)') ;
23 x = linspace(a+dx, b-dx, n) ; fun = feval(fun ,x) ;
24 int = dx.*sum(fun)
25
26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TRAPEZE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27 a = 0 ; b = 2*pi ; n = 1000 ; dx = (b - a)/n ; x = a :dx: b ;
28 f =  x.*exp(-x).*cos(2.*x) ; int = 0 ;
29 init = (f(1) + f(end))*(dx/2) ;
30
31 for ik = 1:length(x)-1
32
33     int = int + dx*f(ik) ;
34 end
35 int = init + int
36 disp(strcat('L''INTEGRALE, PAR LA METHODE DU TRAPEZE VAUT :',...
37 num2str(int)))
38
39 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Simpson %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
40 clear all ; clc ; close all ;
41
42 a = 0 ; b = 2*pi ; n = 1000 ; dx = (b - a)/n ; x = a :dx: b ;
43 fun =  x.*exp(-x).*cos(2.*x) ; som1 = 0 ; som2 = 0 ;
44
45 for ik = 1:n-1
46
47     som1 = som1 + fun(ik) ; xbar = (x(ik+1) - x(ik))/2 ;
48     fun2 = eval('fun',xbar) ; som2 = som2 + fun2(ik) ;
49 end
50
51 int = (dx/6)*(fun(1) + fun(end) + 2*som1 + 4*som2)
52 disp(strcat('L''INTEGRALE, PAR LA METHODE DE SIMPSON VAUT :',...
53 num2str(int)))
54
55 %%%%%%%%%%%%%%%%%%%%%%%%% AIRE DE L'INTEGRALE %%%%%%%%%%%%%%%%%%%%%%%%%
56
57 figure('color',[1 1 1]) ; plot(x, fun,'LineWidth', 1) ; hold on
58
59 for ik = 1 : numel(x)
60
61  plot([x(ik), x(ik)],[0, fun(ik)],'r','LineWidth', 1)
62
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
63 end
64
65 ih = gca ;
66 str(1) = {'Integral area of:'};
67 str(2) = {['$$\int_{0}^{2\pi}x \, \exp(-x) \, \cos(2 \, x) \, dx = $$
   ',num2str(int)]} ; set(gcf,'CurrentAxes', ih) ;
68
69 text('Interpreter','latex', 'String',str,'Position',[3 -0.2],'
   FontSize',12) ;  xlabel('x') ; ylabel('f(x)') ;
```

Les résultats de l'intégrale, calculés par les trois méthodes, sont affichés sous Matlab comme suit :

```
1 L'INTEGRALE, PAR LA METHODE DU POINT MILEU VAUT : -0.1228
2 L'INTEGRALE, PAR LA METHODE DU TRAPEZE VAUT : -0.1222
3 L'INTEGRALE, PAR LA METHODE DE SIMPSON VAUT : -0.1221
```

L'aire de l'intégrale est représentée dans la figure ci-dessous.

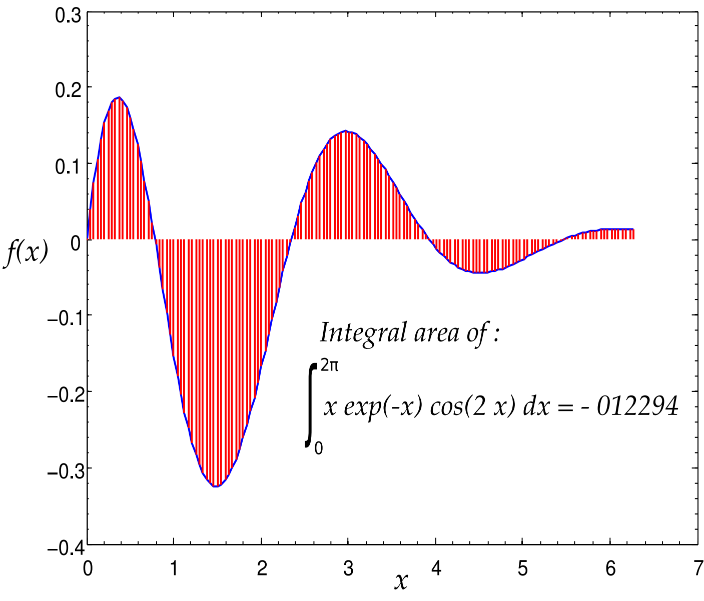

FIGURE 1.5: Aire de l'intégrale

On écrira désormais les trois méthodes d'intégration dans des fichiers de type fonction M-files.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
function int = point_milieu(fun, lowerBound, upperBound, n)
% Integration par la methode du point milieu. Cette fonction admet
% quatre arguments en entree qui sont :

% - fun : fonction a integrer, elle etre un objet inline,
%   une fonction anonyme ou bien une fonction M-file.
% - lowerBound : est la borne inferieure de l'integrale
% - upperBound : est la borne superieure de l'integrale
% - n          : est le nombre de sous-intervalles

if nargout < 4
    n = 100 ; % par defaut
end

dx = (upperBound - lowerBound) / n ; x = lowerBound :dx: upperBound ;
int = 0;

for ik = 1:length(x) - 1

    xbar = (x(ik) + x(ik+1))/2 ;
    int =  int + dx*fun(xbar);
end

return
```

L'appel de cette fonction se fait suivant

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ;

int = point_milieu(@(x) x.*exp(-x).*cos(2.*x), 0, 2*pi, 500)

>> int =

        -0.1220
```

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
function int = methode_trapeze(fun, lowerBound, upperBound, n)

% Integration par la methode du Trapeze. Cette fonction admet
% quatre arguments en entree qui sont :

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
% - fun : fonction a integrer, elle etre un objet inline, une
    fonction
%   anonyme ou bien  une fonction M-file.
% - lowerBound : est la borne inferieure de l'integrale
% - upperBound : est la borne superieure de l'integrale
% - n          : est le nombre de sous-intervalles

if nargout < 4
    n = 100 ; % par defaut
end

dx = (upperBound - lowerBound) / n ; x = lowerBound :dx: upperBound ;
int = 0 ; init = (fun(1) + fun(n))*(dx/2);

for ik = 1:length(x)-1

    int = int + dx*fun(x(ik));
end
int = init + int ;
return
```

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
function int = methode_simpson(fun, lowerBound, upperBound, n)

% Integration par la methode de Simpson. Cette fonction admet
% quatre arguments en entree qui sont :

% - fun : fonction a integrer, elle etre un objet inline, une
    fonction
%   anonyme ou bien  une fonction M-file.
% - lowerBound : est la borne inferieure de l'integrale
% - upperBound : est la borne superieure de l'integrale
% - n          : est le nombre de sous-intervalles

if nargout < 4
    n = 100 ; % par defaut
end

dx = (upperBound - lowerBound) / n ; x = lowerBound :dx: upperBound ;
som1 = 0 ; som2 = 0 ;

for ik = 1:n-1

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
21     som1 = som1 + fun(x(ik)) ;
22     som2 = som2 + (fun(x(ik + 1)) + fun(x(ik)))/2 ;
23 end
24
25 int = (dx/6)*(fun(1) + fun(end) + 2*som1 + 4*som2) ;
26 return
```

L'appel se fait avec :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clc ; clear all ;
2
3 fx = inline('x*exp(-x)*cos(2*x)') ;
4 int = methode_simpson(fx, 0, 2*pi, 10) ;
5
6 >> int =
7
8        -0.1265
```

Le script Matlab calculant l'erreur d'intégration est :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all ; clc ; close all ; format long
2 %%%%%%%%%%%%%%%%%%%%%%%% ERREUR D'INTEGRATION %%%%%%%%%%%%%%%%%%%%%%%
3 a = 0 ; b = 2*pi ; n = 150 ; ih = 0 ; intexact = - 0.12212260461896 ;
4
5 for n = 60 : 20 : 600
6
7       dx = (b - a)/n ; x = a :dx: b ; fun = x.*exp(-x).*cos(2.*x) ;
8       som1 = 0 ; som2 = 0 ; ih = ih + 1 ;
9     for ik = 1:n-1
10
11     som1 = som1 + fun(ik) ; xbar = (x(ik+1) - x(ik))/2;
12     fun2 = eval('fun',xbar) ; som2 = som2 + fun2(ik);
13
14     end
15
16     int(ih) = (dx/6)*(fun(1) + fun(end) + 2*som1 + 4*som2) ;
17     err(ih) = abs((int(ih) - intexact)/intexact) ; % ERREUR RELATIVE
18
19 plot(n, err(ih), '+','MarkerSize',8, 'LineWidth', 1/2) ; hold on ;
20 xlabel('NOMBRE DE SOUS-INTERVALLES') ; ylabel('ERREUR RELATIVE D''
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
    INTEGRATION (x 100 \%)') ;
21 end
```

Le graphe généré par ce script Matlab est :

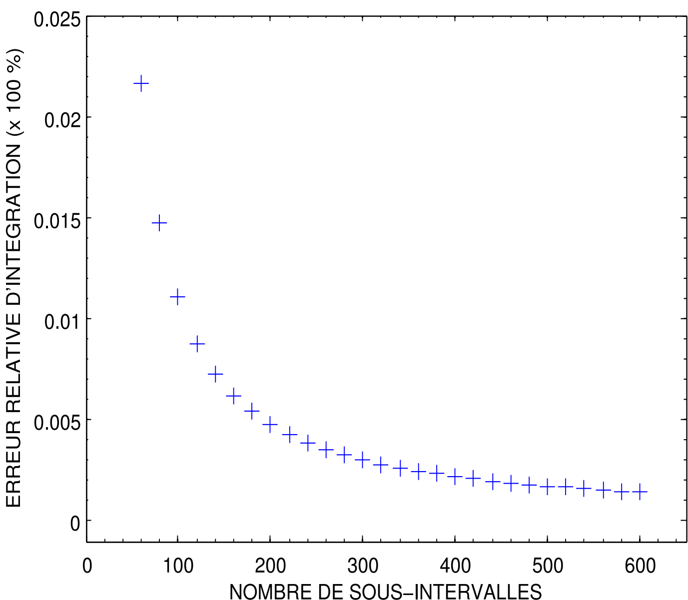

FIGURE 1.6: Influence du nombre de sous-intervalle sur l'erreur d'intégration

À partir de ce graphique, il apparait clairement que plus le nombre de sous-intervalles est élevé, plus l'erreur d'intégration est faible. Ce résultat s'explique par le fait que plus `n` (nombre de sous-intervalles) est grand plus on s'approche de la forme continue de la fonction à intégrer. Toutes les méthodes énumérées dans les sections précédentes, dans lesquelles le pas de discrétisation est régulier sur tout l'intervalle d'intégration font partie de la catégorie des méthodes dites de *Newton-Cotes*.

1.4 Au moyen de routines Matlab

Il existe des routines Matlab prédéfinies qui permettent de résoudre numériquement des intégrales simple, double et triple. Ces commandes sont les suivantes : `quad`, `quadl`, `quadgk` et `quadv`. Les commandes relatives au calcul d'intégrales double et triple seront abordées dans le prochain chapitre. La syntaxe usuelle de la commande `quad` est la suivante :

```
1 [q, fcnt] = quad(fun, lowerBound, upperBound, Tol)
```

Ainsi, `quad` calcul numériquement l'aire (intégrale) sous la courbe de la *fonction anonyme* `fun`. Cette dernière peut également être écrite comme une *fonction M-file*. L'algorithme d'intégration de cette commande est basé sur la méthode de *Simpson adaptative* (adaptive Simpson quadrature). Les arguments en entrée `lowerBound` et

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

upperBound désignent respectivement les bornes inférieure et supérieure de l'intégrale. L'argument Tol désigne la tolérance considérée relative à l'erreur d'intégration. Par défaut, Tol vaut 1e-06. L'argument de sortie q représente la valeur de l'intégrale calculée. La sortie fcnt exprime le nombre d'évaluation de la fonction à intégrer. Les commandes quadl (adaptive Lobatto quadrature) et quadv (Vectorized quadrature) présentent une syntaxe similaire à celle de la commande quad. La commande quadgk (adaptive Gauss-Kronrod quadrature) accepte d'autres arguments optionnels selon la syntaxe :

```
[q, errbnd] = quadgk(fun, lowerBound, upperBound, 'param1', val1, 'param2', val2, ....)
```

La sortie errorbnd exprime l'erreur absolue d'intégration $|Q - I|$, avec Q et I sont respectivement les valeurs approchée et exacte de l'intégrale. Parmi les arguments optionnels, on citera, 'AbsTol' (Absolute error tolerance) dont la valeur par défaut vaut 1e-10 et 'RelTol' (Relative error tolerance) dont la valeur par défaut vaut 1.e-6. Voir aussi 'MaxIntervalCount' (Maximum number of intervals allowed) et 'Waypoints' (Vector of integration waypoints). Nous donnons dans l'exercice ci-dessous, un exemple d'utilisation de ces commandes pour le calcul d'intégrales simples :

Exercice ❸ ☞ ®

1. Au moyen des commandes quad, quadl et quadgk, évaluer l'intégrale :

$$I(f) = \int_0^{2\pi} x\, exp(-x)\, cos(2\,x)\, dx \simeq -0.122122604618968 \qquad (1.17)$$

2. Se servant de la commande quadv, évaluer l'intégrale :

$$\int_0^{2\pi} x\, exp(-x)\, cos(k\,x)\, dx \qquad (1.18)$$

pour différentes valeurs du paramètre k = 2 : 24.

3. Tracer la courbe représentant la valeur de l'intégrale en fonction du paramètre k.

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; clc ;
%%%%%%%%%%%%%%%%%% AU MOYEN DES ROUTINES MATLAB %%%%%%%%%%%%%%%%%
% @copyright 16/11/2015 Samir KENOUCHE

fun = @(x) x.*exp(-x).*cos(2.*x) ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
6
7  lowerBound = 0 ;  upperBound = 2*pi; tol = 1e-07 ;
8
9  [int1, funEval1] = quad(fun, lowerBound, upperBound, tol) % METHODE
10 % ADAPTATIVE DE Simpson. funEval1 : Nbr D'EVALUATIONs DE fun
11
12 [int2, funEval2] = quadl(fun, lowerBound, upperBound, tol) % METHODE
13 %ADAPTATIVE DE Lobatto
14
15 [int3, errorbnd] = quadgk(fun, lowerBound, upperBound,'RelTol',...
16     1e-8,'AbsTol',1e-12) % METHODE ADAPTATIVE DE Gauss-Kronrod
17 % errorbnd : TRADUIT L'ERREUR ABSOLUE D'INTEGRATION |Q - I|.
18
19 n = 1000 ; dx = (upperBound - lowerBound)/n ;
20 x = lowerBound : dx : upperBound ;
21
22 int4 = trapz(x, fun(x)) ; % METHODE DU TRAPEZE
23
24 for k = 2:14  % POUR LES INTEGRALES PARAMERTRIQUE
25
26 [int5, funEval4] = quadv(@(x) x.*exp(-x).*cos(k.*x), lowerBound,...
27  upperBound);
28
29 figure(1) ; hold on
30 plot(k,int5,'+','MarkerSize',10,'LineWidth',1)
31 xlim([1 15]) ; xlabel('VALEUR DU PARAMETRE k','FontSize',12) ;
32 ylabel('VALEUR DE L''INTEGRALE','FontSize',12)
33
34 end
```

Ci-dessous, les résultats renvoyés par le script Matlab.

```
1  >> int1      =
2               -0.1221
3
4  >> funEval1 =
5                125
6
7  >> int2      =
8               -0.1221
9
10 >> funEval2 =
11               48
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
12
13 >> int3     =
14            -0.1221
15
16 >> errorbnd =
17             1.4734e-15
```

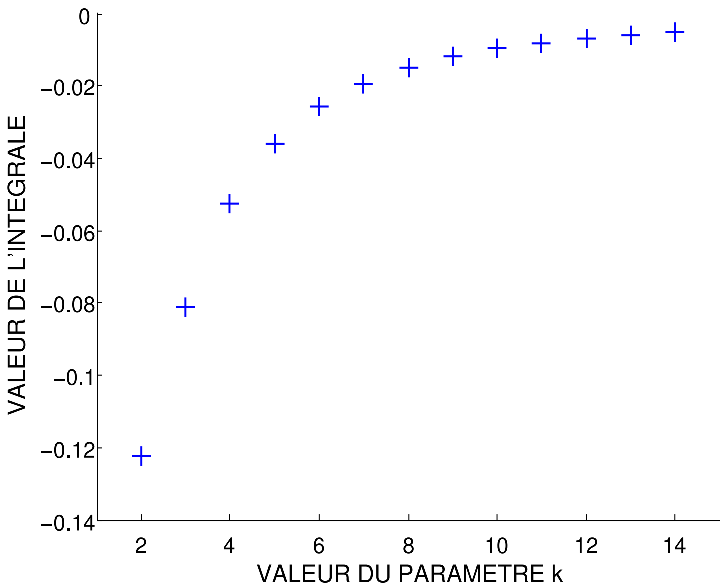

FIGURE 1.7: Figure générée par le code Matlab ci-dessus

À partir de ce graphe, on observe que plus le paramètre k augmente, plus la valeur de l'intégrale est élevée. Dans cette section, nous avons fait appel à des commandes Matlab prédéfinies pour approximer les intégrales. Ces commandes sont basées sur des techniques d'intégration dites *formule de quadrature composite adaptative* (adaptive composite quadrature formula en anglais). Elles sont dites *adaptatives*, car elles adaptent le pas de discrétisation (Δx non uniforme) en fonction des variations de la fonction considérée. Cette technique d'intégration est très efficiente notamment pour les fonctions présentant, par endroits, de brusques variations, et que ces dernières sont contenues dans des portions de sous-intervalles relativement réduites.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

1. Calculer numériquement, avec les méthodes du `point milieu`, du `trapèze` et de `Simpson`, les intégrales :

(a) $\displaystyle\int_0^1 x^2+3x^2+5\,dx$ (b) $\displaystyle\int_1^2 x^{1/3}+5x^{1/2}\,dx$ (c) $\displaystyle\int_0^\pi 1+\cos(x)\,dx$

(d) $\displaystyle\int_1^5 \frac{1}{x^2+1}\,dx$ (e) $\displaystyle\int_0^3 \frac{x}{\sqrt{x^2+1}}\,dx$ (f) $\displaystyle\int_0^{\pi/2} \sin(\cos(x))\,dx$

(g) $\displaystyle\int_1^3 (x+1)\exp(x^2)\,dx$ (h) $\displaystyle\int_1^5 \frac{\log(x)}{x^2+3}\,dx$ (i) $\displaystyle\int_0^{\pi/2} \sin(x^2+1)\,dx$

(j) $\displaystyle\int_{-5}^0 \frac{1}{x^2-3\,x+3}\,dx$ (k) $\displaystyle\int_0^1 \frac{\exp(x)-1}{\exp(x)+1}\,dx$ (l) $\displaystyle\int_1^2 \frac{\exp(-2\,x)}{(1+2\,\exp(-x))^2}\,dx$

(m) $\displaystyle\int_0^{1/2} \frac{1}{(2x-1)^{4/3}}\,dx$ (n) $\displaystyle\int_1^2 \frac{dx}{x^2\sqrt{4-x^2}}$ (o) $\displaystyle\int_0^2 \frac{dx}{x^2+3\,x+7}$

2. Tester les commandes `quad`, `quadl` et `quadgk`
3. Quel nombre de sous-intervalles n faut-il choisir pour atteindre une erreur d'intégration inférieure à 10^{-6}.
4. Tracer l'aire de l'intégrale, pour `n = 100` sous-intervalles.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

2 Intégration numérique : intégrales double et triple

Sommaire

6.1 Intégrale double 131
6.2 Intégrale triple 138

Introduction

Dans ce chapitre, il sera question de la résolution numérique des intégrales double et triple. On donnera dans un premier temps, un aperçu théorique inhérent au calcul des intégrales multiples avant de commencer l'écriture des scripts Matlab. La mise en pratique avec Matlab, sera réalisée en adoptant deux approches différentes. Une première approche basée sur une résolution "intuitive" (manuellement) et une deuxième approche basée sur des commandes Matlab préprogrammées.

2.1 Intégrale double

Formellement, une intégrale multiple, par exemple double, d'une fonction $f(x,y)$ est définie par $I(f) = \int\int_{(x,y)\in\Omega} f(x,y)\,dx\,dy$, où Ω est le domaine suivant $\Omega = \{(x,y) \in \mathbb{R}^2,\ a \leq x \leq b, c \leq y \leq d\}$. Les bornes $(a < b, c < d)$ forment un rectangle ou un carré fermé du plan $\mathbb{R}^2$ dont les côtés sont parallèles aux axes de coordonnées.

Soit une fonction f : $\mathbb{R}^2 \longrightarrow \mathbb{R}$ une fonction continue par morceaux sur $[a,b] \times [c,d]$, nous avons :

$$\int_a^b \int_c^d f(x,y)\,dx\,dy = \int_a^b \left(\int_c^d f(x,y)\,dy \right) dx = \int_c^d \left(\int_a^b f(x,y)\,dx \right) dy \quad (2.1)$$

C'est l'un des corolaires du *théorème de Fubini*. Ce théorème illustre la possibilité d'intégrer la fonction séparément sur l'intervalle horizontal et vertical. On peut commencer

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

l'intégration soit avec la variable x ou bien la variable y, le résultat est le même. Toutefois, le niveau de difficulté du calcul de l'intégrale peut être sensiblement différent. Rappelons aussi que l'ordinateur ne sait résoudre que des problèmes discrets. Ainsi, l'idée de base de la résolutions numérique de ces intégrales, consiste à convertir un système initialement continu (intégrale) en un système discret (somme). Dans ce cas, on subdivise alors le domaine Ω au moyen de rectangles élémentaires de côtés $\Delta x_i = x_{i+1} - x_i$ et $\Delta y_i = y_{i+1} - y_i$ ensuite on prend dans chacun de ces rectangles le point moyen $(\bar{x}_i, \bar{y}_i)$ pour lequel on évalue la fonction $f(\bar{x}_i, \bar{y}_i)$. Où, $i = a_1 + \Delta x, a_1 + 2\Delta x, ..., a_1 + n\Delta x = n_x = a_2$ et $j = b_1 + \Delta y, b_1 + 2\Delta y, ..., b_1 + n\Delta y = n_y = b_2$. Avec, a_1, a_2, b_1 et b_2 sont respectivement les bornes d'intégration suivant les dimensions x et y.

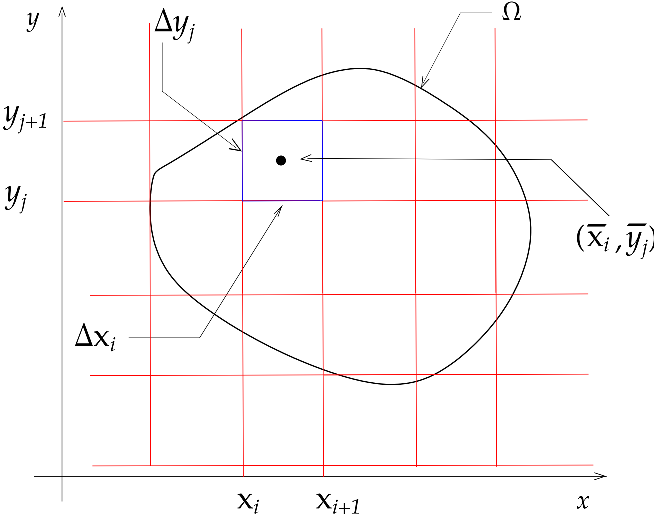

FIGURE 2.1: Discrétisation du domaine Ω

On aura ainsi la somme :

$$S_{n_x,n_y} = \sum_{i=1}^{n_x-1} \sum_{j=1}^{n_y-1} f(\bar{x}_i, \bar{y}_j)\, \Delta x_i\, \Delta y_i \tag{2.2}$$

Par définition

$$\int\int_{(x,y)\in\Omega} f(x,y)\, dx\, dy = \lim_{\substack{n_x\to\infty \\ n_y\to\infty}} S_{n_x,n_y} \tag{2.3}$$

De façon à ce que les surfaces définies par les rectangles $\Delta x_i\, \Delta y_j$ tendent vers zéro. La surface du domaine Ω s'obtient avec :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$$\int\int_{(x,y)\in\Omega} dx\, dy \tag{2.4}$$

On pourra définir, entre autre, la valeur moyenne de la fonction f, notée $< f(x,y) >$, dans le domaine Ω par :

$$< f(x,y) > = \frac{\displaystyle\int\int_{(x,y)\in\Omega} f(x,y)\, dx\, dy}{\displaystyle\int\int_{(x,y)\in\Omega} dx\, dy} \tag{2.5}$$

Notons par ailleurs, que souvent un changement de variable s'impose. Il est d'usage de procéder à un cangement de variable par le biais des coordonnées polaires. Dans ce cas, l'élément de surface $dx\, dy$ se transforme en $r\; dr d\theta$. La formule de changement de variable s'écrit alors :

$$\int\int_{(x,y)\in\Omega} f(x,y)\, dx\, dy = \int\int_{(r,\theta)\in\Omega} f(r\,\cos(\theta), r\,\sin(\theta))\, r\, dr\, d\theta \tag{2.6}$$

Exercice ❶ ☞ ®

1. Soient les intégrales :

$$I_1 = \int_0^1 \int_0^1 (1 + x\,y + x^2 + y^2)\, dx\, dy = 1.916666666666667 \tag{2.7}$$

$$I_2 = \int_{\pi}^{2\,\pi} \int_0^{\pi} y^2\,\sin(x) + x^2\,\cos(y)\, dx\, dy = -20.670851083718247 \tag{2.8}$$

$$I_3 = \int_0^1 \int_0^2 (4 - x^2 - y^2)\, dx\, dy = 4.666666666666667 \tag{2.9}$$

2. Calculer analytiquement et numériquement l'intégrale (2.7).
3. Tracer le graphe de la fonction $f_2(x,y)$, $x \times y \in [\pi, 2\pi] \times [0, \pi]$.
4. Approcher numériquement, par deux approches différentes (manuellement et par des commandes Matlab), les intégrales (2.8) et (2.9).
5. Calculer numériquement la valeur moyenne de la fonction $f(x,y) = \exp(x-y)$ sur le domaine Ω défini par $\Omega = \{(x,y) \in \mathbb{R}^2,\, 0 \leq x \leq 2, 0 \leq y \leq 1\}$

On commencera, dans un premier temps, par résoudre analytiquement l'intégrale I_1. D'après le *théorème de Fubini*, on peut écrire :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$$I_1 = \int_0^1 \int_0^1 (1 + x\,y + x^2 + y^2)\,dx\,dy = \int_0^1 \left(\int_0^1 (1 + x\,y + x^2 + y^2)\,dy \right) dx$$

$$= \int_0^1 \left[y + x\,\frac{y^2}{2} + x^2\,y + \frac{y^3}{3} \right]_0^1 dx = \int_0^1 (x^2 + \frac{x}{2} + \frac{4}{3})\,dx$$

$$\left[\frac{x^3}{3} + \frac{x^2}{4} + \frac{4\,x}{3} \right]_0^1 = \frac{1}{3} + \frac{1}{4} + \frac{4}{3} = 1.9167$$

De façon algorithmique :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; close all ; clear all ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%% INTEGRALES DOUBLES %%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 16/11/2015 Samir KENOUCHE
4  
5  lowerBound1 = 0 ; upperBound1 = 1 ; lowerBound2 = 0 ;
6  upperBound2 = 1 ; n = 800 ; dx = (upperBound1 - lowerBound1)/n ;
7  
8  dy = (upperBound2 - lowerBound2)/n ; xi = lowerBound1:dx:upperBound1;
9  yi = lowerBound2 :dy: upperBound2 ;
10 
11 fun = @(x,y) (1 + x*y + x.^2 + y.^2);
12 
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% INTEGRALE I1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 
15 nx = length(xi) ; ny = length(yi) ; int = 0;
16 
17 for i = 1:nx-1
18     for j = 1:ny-1
19 
20         xbar = (xi(i+1) + xi(i))/2 ;
21         ybar = (yi(j+1) + yi(j))/2 ;
22 
23         int = int + fun(xbar,ybar)*dx*dy ;
24     end
25 end
26 
27 disp(strcat('LA VALEUR DE L''INTEGRALE int Vaut : ', num2str(int)))
```

La valeur de l'intégrale I_1 renvoyée est :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
LA VALEUR DE L'INTEGRALE int Vaut :1.9167
```

Le script Matlab ci-dessous, expose l'affichage de la fonction $f_2(x, y)$, le calcul de son intégrale I_2 manuellement et au moyen de la commande Matlab prédéfinie `dblquad`.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; close all ; clear all ;
%%%%%%%%%%%%%%%%%%%%%%%%% INTEGRALES DOUBLES %%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 17/11/2015 Samir KENOUCHE

lowerBound1 = pi ; upperBound1 = 2*pi ; lowerBound2 = 0 ;
upperBound2 = pi ; n = 800 ; dx = (upperBound1 - lowerBound1)/n ;
dy = (upperBound2 - lowerBound2)/n ; xi = lowerBound1:dx:upperBound1;
yi = lowerBound2 :dy: upperBound2 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%% GRAPHE DE f2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[u, v] = meshgrid(xi,yi) ; w = v.^2.*sin(u)+ u.^2.*cos(u) ;

figure('color',[1 1 1]) ;
surf(u,v,w,'MarkerFaceColor','none') ; colormap(jet) ; shading interp
xlabel('x','FontSize',12) ; ylabel('y','FontSize',12) ;
zlabel('y^2 sin(x) + x^2 cos(y)','FontSize',12) ; grid off ;
axis auto ; view(-24,28)

%%%%%%%%%%%%%%%%%%%%%% INTEGRALE I2 : 1ERE APPROCHE %%%%%%%%%%%%%%%%%%

fun = @(x,y) y.^2*sin(x)+x.^2*cos(y) ; tol = 1e-08 ;
int1 = dblquad(fun, lowerBound1, upperBound1,lowerBound2, upperBound2
    , tol);

%%%%%%%%%%%%%%%%%%%%%% INTEGRALE I2 : 2EMME APPROCHE %%%%%%%%%%%%%%%%%

nx = length(xi) ; ny = length(yi) ; int2 = 0;

for i = 1:nx-1
    for j = 1:ny-1

        xbar = (xi(i+1) + xi(i))/2 ;
        ybar = (yi(j+1) + yi(j))/2 ;

        int2 = int2 + fun(xbar,ybar)*dx*dy ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
36      end
37  end
38
39
40  disp(strcat('LA VALEUR DE L''INTEGRALE int1 Vaut : '...
41            , num2str(int1)))
42  disp(strcat('LA VALEUR DE L''INTEGRALE int2 Vaut : '...
43            , num2str(int2)))
```

Les résultats sont affichés somme suit :

```
1  LA VALEUR DE L'INTEGRALE int1 Vaut :-20.6709
2  LA VALEUR DE L'INTEGRALE int2 Vaut :-20.6709
```

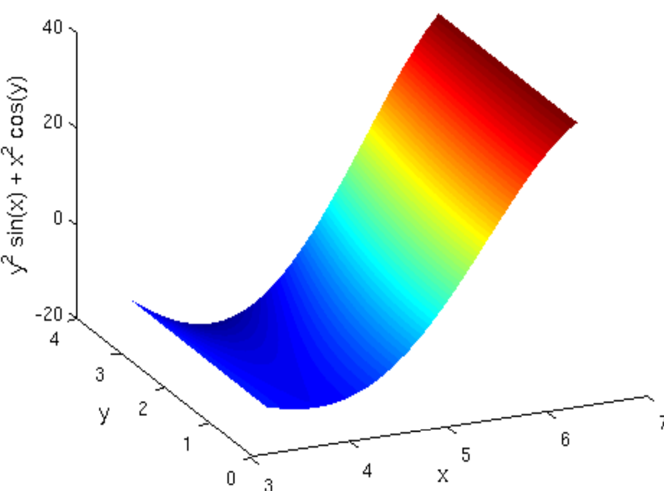

FIGURE 2.2: Figure générée par le code Matlab ci-dessus

Les deux approches affichent strictement le même résultat de l'intégrale. La commande `dblquad` permet de calculer numériquement une intégrale double. Sa syntaxe usuelle est donnée par :

```
[int, evalFun] = dblquad(fun, xmin, xmax, ymin, ymax, tol, method)
```

L'argument de sortie `int` constitue la valeur de l'intégrale calculée. Le deuxième argument de sortie `evalFun` désigne le nombre d'évaluation de la fonction. La commande `dblquad` évalue l'intégrale double `fun(x,y)` sur le rectangle définit par `xmin <= x <= xmax` et `ymin <= y <= ymax` avec la tolérance considérée `tol`. Par défaut, la valeur

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

de cette tolérance vaut : 10^{-6}. L'argument `fun` est une *fonction handle*. L'argument `method` indique la méthode utilisée pour résoudre l'intégrale. Il admet deux valeurs, à savoir : `method = @quad` (*adaptive Simpson quadrature*) prise par défaut ou bien `method = @quadl` (*adaptive Lobatto quadrature*).

Pour le calcul de la valeur moyenne, voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; close all ; clear all ; format short
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 17/11/2015 Samir KENOUCHE

lowerBound1 = 0 ; upperBound1 = 2 ; lowerBound2 = 0 ;
upperBound2 = 1 ; n = 800 ; dx = (upperBound1 - lowerBound1)/n ;

dy = (upperBound2 - lowerBound2)/n ; xi = lowerBound1:dx:upperBound1;
yi = lowerBound2 :dy: upperBound2 ; fun = @(x,y) exp(x - y);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MOYENNE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nx = length(xi) ; ny = length(yi) ; numer = 0 ; denum = 0 ;

for i = 1:nx
    for j = 1:ny

        numer = numer + fun(xi(i),yi(j))*dx*dy ;
        denum = denum + dx*dy ;
    end
end

moyVal = numer/denum ;

disp(strcat('LA VALEUR MOYENNE DE LA FONCTION Vaut :',...
    num2str(moyVal)))
```

L'affichage renvoyé est :

```
LA VALEUR MOYENNE DE LA FONCTION Vaut : 2.0203
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

1. Calculer l'intégrale double :

$$I(f) = \int\int_{(x,y)\in\Omega} f(x,y)\,dx\,dy \tag{2.10}$$

$$(a) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 0 \leq x \leq 1, 1 \leq y \leq 2\}.$$
$$f(x,y) = x\,y\ \exp(-x-y) \tag{2.11}$$

$$(b) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ |x| < 1, |y| < 1\}.$$
$$f(x,y) = (x+y)\ \exp(x+y) \tag{2.12}$$

$$(c) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 0 \leq x \leq y, 0 \leq y \leq 2\}.$$
$$f(x,y) = (x^2+y^2) \tag{2.13}$$

$$(d) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 0 \leq x \leq 1, 0 \leq y \leq 1, x^2+y^2 \geq 1\}.$$
$$f(x,y) = \frac{x\,y}{1+x^2+y^2} \tag{2.14}$$

$$(e) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 1 \leq x \leq 2, 0 \leq x\,y \leq \pi/2\}.$$
$$f(x,y) = \cos(x\,y) \tag{2.15}$$

$$(f) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 0 \leq x \leq 2, 0 \leq y \leq 2\}.$$
$$f(x,y) = (x-y)\ \exp(x+5\,y) \tag{2.16}$$

$$(i) \quad \Omega = \{(x,y) \in \mathbb{R}^2,\ 0 \leq x \leq 1, 0 \leq y \leq 2\}.$$
$$f(x,y) = 1 - x^2\,y \tag{2.17}$$

2.2 Intégrale triple

Le principe de résolution des intégrales triples est analogue à celui des intégrales doubles. Il s'agit de déterminer sur quelle domaine les variables varient et d'intégrer successivement par rapport aux trois variables d'intégration.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Soit une fonction f : $\mathbb{R}^3 \longrightarrow \mathbb{R}$ une fonction continue par morceaux sur $[a,b] \times [c,d] \times [e,f]$, de façon similaire que précédemment, le *théorème de Fubini* nous autorise d'écrire :

$$\int_a^b \int_c^d \int_a^b f(x,y,z)\,dx\,dy\,dz = \int_a^b \left(\int_c^d \left(\int_e^f f(x,y,z)\,dz \right) dy \right) dx \qquad (2.18)$$

La séquence d'intégration choisie est susceptible d'affecter le degré de difficulté du calcul. De la même manière que dans le cas des intégrales doubles, la discrétisation d'une intégrale triple se fait selon :

$$S_{n_x,n_y,n_z} = \sum_{i=1}^{n_x-1} \sum_{j=1}^{n_y-1} \sum_{k=1}^{n_z-1} f(\bar{x}_i, \bar{y}_j, \bar{z}_k)\,\Delta x_i\,\Delta y_i\,\Delta z_i \qquad (2.19)$$

Par définition

$$\int\int\int_{(x,y,z)\in\Omega} f(x,y,z)\,dx\,dy\,dz = \lim_{\substack{n_x\to\infty \\ n_y\to\infty \\ n_z\to\infty}} S_{n_x,n_y,n_z} \qquad (2.20)$$

De façon à ce que les volumes élémentaires $\Delta x_i\,\Delta y_j\,\Delta z_k$ de chaque parallélépipède tendent vers zéro. Le volume du domaine Ω s'obtient avec :

$$\int\int\int_{(x,y,z)\in\Omega} f(x,y,z)\,dx\,dy\,dz \qquad (2.21)$$

D'autres systèmes de coordonnées, à l'instar des coordonnées cylindrique et sphérique, sont également utilisées pour résoudre ce type d'intégrale. En coordonnées cylindrique le volume élémentaire devient $dv = \rho d\rho d\theta dz$, ce qui donne :

$$\int\int\int_{(x,y,z)\in\Omega} f(x,y,z)\,dx\,dy\,dz = \int\int\int_{(\rho,\theta,z)\in\Omega} f(\rho\,\cos(\theta), \rho\,\sin(\theta), z)\,\rho\,d\rho\,d\theta\,dz \qquad (2.22)$$

En coordonnées sphérique le volume élémentaire devient $dv = r^2\,\sin(\theta)\,dr\,d\phi\,d\theta$, ce qui donne :

$$\int\int\int_{(x,y,z)\in\Omega} f(x,y,z)\,dx\,dy\,dz = \int\int\int_{(r,\phi,\theta)\in\Omega} f(r\cos(\phi)\,\sin(\theta), r\sin(\phi)\,\sin(\theta), r\,\cos(\theta))$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$$r^2 \, \sin(\theta) \, dr \, d\phi \, d\theta \tag{2.23}$$

Exercice ❷ ☞ ®

1. Soient les intégrales :

$$I_1 = \int_0^1 \int_1^2 \int_0^2 y^2 \, x^2 \, \sin(z) \, dx \, dy \, dz = 1.296296296296296 \tag{2.24}$$

$$I_2 = \int_0^{\pi} \int_0^{\pi} \int_{-1}^{1} z^2 \, \sin(x) + z^2 \, \cos(y) \, dx \, dy \, dz = 4.188790204291397 \tag{2.25}$$

$$I_3 = \int_0^1 \int_0^1 \int_0^1 x^2 \, y \, \exp(x \, y \, z) \, dx \, dy \, dz = 0.210000000000000 \tag{2.26}$$

2. Calculer analytiquement et numériquement l'intégrale I_1.
3. Déterminer l'erreur d'intégration.
4. Approcher numériquement, par deux approches différentes (manuellement et avec la commande `triplequad`), les intégrales I_1, I_2 et I_3.

On commencera par résoudre analytiquement l'intégrale I_1 :

$$\int_0^1 \int_1^2 \int_0^2 y^2 \, x^2 \, \sin(z) \, dx \, dy \, dz = \int_0^1 \left(\int_1^2 \left(\int_0^2 y^2 \, x^2 \, \sin(z) \, dz \right) dy \right) dx \tag{2.27}$$

$$\int_0^1 \left(\int_1^2 \left(\left[-y^2 \, x^2 cos(z) \right]_0^2 \right) dy \right) dx = \int_0^1 \left(\int_1^2 0.6 \, y^2 \, x^2 dy \right) dx$$

$$\frac{1}{1.8} \int_0^1 8 \, x^2 - x^2 dx = \frac{1}{1.8} \left[\frac{8 \, x^3}{3} - \frac{x^3}{3} \right]_0^1 = 1.2962$$

Pour la résolution numérique, voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; close all ; clear all ;
%%%%%%%%%%%%%%%%%%%%%% INTEGRALES TRIPLES %%%%%%%%%%%%%%%%%%%%%%
% @copyright 18/11/2015 Samir KENOUCHE

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
lowerBound1 = 0 ; upperBound1 = 1 ; lowerBound2 = 1 ;
upperBound2 = 2 ; lowerBound3 = 0 ; upperBound3 = 2 ;

n = 200 ; dx = (upperBound1 - lowerBound1)/n ;
dy = (upperBound2 - lowerBound2)/n;dz=(upperBound3 - lowerBound3)/n ;

xi = lowerBound1 :dx: upperBound1 ;yi =lowerBound2 :dy: upperBound2 ;
zi = lowerBound3 :dz: upperBound3 ; fun = @(x,y,z) y.^2*x.^2*sin(z) ;

%%%%%%%%%%%%%%%%%%%%%%%%% INTEGRALE I1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

nx = length(xi) ; ny = length(yi) ; nz = length(zi) ; int1 = 0;

for i = 1:nx - 1
    for j = 1:ny - 1
        for k = 1 : nz - 1

          xbar = (xi(i+1) + xi(i))/2 ;
          ybar = (yi(j+1) + yi(j))/2 ;
          zbar = (zi(k+1) + zi(k))/2 ;

        int1 = int1 + fun(xbar,ybar,zbar)*dx*dy*dz ;

        end
    end
end

disp(strcat('LA VALEUR DE L''INTEGRALE int1 Vaut : '...
         , num2str(int1)))
  intexact = 1.2962 ; err = abs(intexact - int1)/ intexact ;
```

Les sorties renvoyées par ce script sont :

```
LA VALEUR DE L'INTEGRALE int1 Vaut : 1.1014
>> err =
        0.1503
```

L'erreur d'intégration est de 15%, on peut bien entendu diminuer cette erreur en augmentant le nombre de points de discrétisation, mais dans ce cas on va accroitre le temps de calcul. Ainsi, un compromis doit être trouvé entre un temps de calcul raisonnable et une erreur d'intégration acceptable. Le calcul de l'intégrale I_2 par les deux approches est présenté dans le script ci-dessous :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; close all ; clear all ;
%%%%%%%%%%%%%%%%%%%%%%% INTEGRALES TRIPLES %%%%%%%%%%%%%%%%%%%%%%%
% @copyright 17/11/2015 Samir KENOUCHE

lowerBound1 = 0 ; upperBound1 = pi ; lowerBound2 = 0 ;
upperBound2 = pi ; lowerBound3 = -1 ; upperBound3 = 1 ;

n = 200 ; dx = (upperBound1 - lowerBound1)/n ;
dy = (upperBound2 - lowerBound2)/n ; dz = (upperBound3 - lowerBound3)
   /n ;
xi = lowerBound1 :dx: upperBound1 ;  yi = lowerBound2 :dy:
   upperBound2 ; zi = lowerBound3 :dz: upperBound3 ;

%%%%%%%%%%%%%%%%%%% INTEGRALE I2 : 1ERE APPROCHE %%%%%%%%%%%%%%%%%%%

fun = @(x,y,z) z.^2.*sin(x) + z.^2.*cos(y) ; tol = 1e-08 ;
int1 = triplequad(fun, lowerBound1, upperBound1,lowerBound2,
   upperBound2, lowerBound3, upperBound3, tol);

%%%%%%%%%%%%%%%%%%% INTEGRALE I2 : 2EMME APPROCHE %%%%%%%%%%%%%%%%%%

nx = length(xi) ; ny = length(yi) ; nz = length(zi) ; int2 = 0;

for i = 1:nx - 1
    for j = 1:ny - 1
        for k = 1 : nz - 1

          xbar = (xi(i+1) + xi(i))/2 ;
          ybar = (yi(j+1) + yi(j))/2 ;
          zbar = (zi(k+1) + zi(k))/2 ;

        int2 = int2 + fun(xbar,ybar,zbar)*dx*dy*dz ;

        end
    end
end

disp(strcat('LA VALEUR DE L''INTEGRALE int1 Vaut : '...
         , num2str(int1)))

disp(strcat('LA VALEUR DE L''INTEGRALE int2 Vaut : '...
         , num2str(int2)))
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Les résultats sont affichés somme suit :

```
LA VALEUR DE L'INTEGRALE int1 Vaut :4.1888
LA VALEUR DE L'INTEGRALE int2 Vaut :4.1887
```

Il est très recommandé d'indenter son script Matlab, afin d'améliorer sa lisibilité et ainsi détecter facilement d'éventuelles erreurs, comme par exemple, l'oubli d'un `end` dans une boucle. Ceci peut se révéler très utile notamment pour des scripts impliquant de nombreuses boucles. La syntaxe usuelle de la commande `triplequad` est :

```
int = triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax,tol,method)
```

Elle évalue numériquement l'intégrale triple `fun(x,y,z)` sur un rectangle tridimensionnel définit par `xmin <= x <= xmax`, `ymin <= y <= ymax` et `zmin <= z <= zmay` avec la tolérance considérée `tol`. L'argument `fun` est une *fonction handle*. Comme pour les intégrales doubles, l'entrée `method` admet comme valeur : `method = @quad` prise par défaut ou bien `method = @quadl`. Signalons par ailleurs, que ces intégrales peuvent également être résolues en se servant de la boite à outil `Symbolic Math Toolbox`, suivant les instructions :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc ; clear all ;
syms x y z real

fun1 = y^2*sin(x) + x^2*cos(y) ; % INTEGRALE DOUBLE
int2Val = int(int(fun1, x,pi,2*pi), y, 0, pi) ;
int2Val = double(int2Val)

fun2 = z^2*sin(x) + z^2*cos(y) ; % INTEGRALE TRIPLE
int3Val = int(int(int(fun2, x, 0, pi), y, 0, pi), z, -1, 1) ;
int3Val = double(int3Val)
```

Ce qui donne :

```
>> int2Val =
            -20.6709 % integrale double
>> int3Val =
              4.1888 % integrale triple
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❷ ☞ Ⓢ

1. Calculer numériquement l'intégrale triple :

$$I(f) = \int\int\int_{(x,y,z)\in\Omega} f(x,y,z)\,dx\,dy\,dz$$

$$(a) \quad \Omega = \{(r,y,z) \in \mathbb{R}^3,\ 0 \le x \le 1, 0 \le y \le 1, 0 \le z \le 1\}.$$
$$f(x,y,z) = x^2 + y^2 + z^2$$

$$(b) \quad \Omega = \{(r,\theta,\phi) \in \mathbb{R}^3,\ 0 \le r \le R, 0 \le \theta \le 2\pi, -\pi/2 \le \phi \le \pi/2\}.$$
$$f(r,\theta,\phi) = r^2\,\cos(\phi)\,dr\,d\theta\,d\phi$$

$$(c) \quad \Omega = \{(r,y,z) \in \mathbb{R}^3,\ x \ge 0, y \ge 0, z \ge 0, x+y+z \le 1\}.$$
$$f(x,y,z) = 1 - x\,y\,z$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3 Résolution d'équations non-linéaires

Méthodes du point fixes, dichotomie, fausse position, Newton et la sécante

Sommaire

8.1 Méthode du point fixe 173
8.2 Méthode de dichotomie 180
8.3 Méthode de fausse position (ou de Lagrange) 183
8.4 Méthode de Newton 185
8.5 Méthode de la sécante 189
8.6 Au moyen de routines Matlab 192

Introduction

Il existe toute une panoplie de méthodes numériques (dichotomie, point fixe, `Newton`, `Lagrange`) conduisant à chercher numériquement les zéros de fonction $f(x) = 0$ d'une variable réelle. La majorité de ces méthodes sont itératives. En d'autres mots, elles calculent des approximations successives $x_1, x_2, x_3, \ldots$ de la véritable racine x^* de l'équation $f(x) = 0$, à partir d'une valeur initiale x_0 plus au moins bien choisie. Ce qui les distingue, entre autre, c'est leur vitesse de convergence et leur robustesse. Dans certaines applications, cette vitesse de convergence devient un facteur déterminant notamment quand il s'agit de calculer les racines d'une succession de fonctions.

3.1 Méthode du point fixe

Le principe de la méthode du point fixe consiste à transformer, la fonction $f(x) = 0$, $f : [a \quad b] \longrightarrow R$, en une fonction $\varphi(x) = x$. La fonction $\varphi : [a \quad b] \longrightarrow R$, est construite de façon à ce que $\varphi(\alpha) = \alpha$ quand $f(\alpha) = 0$. Trouver la racine de $f(x)$, se résume donc à déterminer un $\alpha \in [a \quad b]$ tel que :

$$\alpha = \varphi(\alpha) \tag{3.1}$$

Dans le cas où un tel point existe, il sera qualifié de `point fixe` de φ et cette dernière est dite `fonction d'itération`. Le schéma numérique de cette méthode est donné par :

$$x^{(k+1)} = \varphi(x^{(k)}) \quad pour \quad k \geq 0 \tag{3.2}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

On rappelle que le vecteur erreur e_n est calculé à partir de : $e_n = |x_n - x_{app}|$. Avec, x_{app} est la solution approchée, de la valeur exacte, déterminée avec une tolérance fixée préalablement. n, étant le nombre d'itérations. Par ailleurs, l'estimation de l'erreur servira, entre autre, à comparer la vitesse de convergence pour des méthodes numériques différentes. Sur le plan pratique, l'erreur est représentée graphiquement en traçant e_{n+1} en fonction de e_n avec une échelle logarithmique. Ainsi, `l'ordre` noté p, d'une méthode numérique s'obtient à partir de :

$$|e_{n+1}| \approx A\,|e_n|^p \Longrightarrow \log|e_{n+1}| \approx p\,\log|e_n| + \log A \tag{3.3}$$

Ainsi l'ordre, p, est quantifié via la pente de l'équation ci-dessus. On en déduira que :

1. Si $p = 1 \Longrightarrow x_n$ converge `linéairement` vers la solution approchée. Dans ce cas on gagne la même quantité de précision à chaque itération.
2. Si $p = 2 \Longrightarrow x_n$ converge `quadratiquement` vers la solution approchée. Dans ce cas on gagne le double de précision à chaque itération.
3. Si $p = 3 \Longrightarrow x_n$ converge `cubiquement` vers la solution approchée. Dans ce cas on gagne le triple de précision à chaque itération.

D'un point de vue pratique, et pour un n suffisamment élevé, la vitesse de convergence d'une méthode itérative est évaluée au moyen de la relation :

$$K_p(x,n) = \frac{x_{n+2} - x_{n+1}}{(x_{n+1} - x_n)^p} \tag{3.4}$$

Tenant compte de cette équation, il vient que plus $K_p(x,n)$ tend vers zéro plus la vitesse de convergence de la méthode est élevée.

Exercice ❶ ☞ ®

Dans cet exercice, il est demandé de trouver la racine de la fonction $f_1(x) = x - cos(x)$, en utilisant la méthode du `point fixe`

1. Tracer la fonction $f_1(x)$ sur l'intervalle $[-1/2 \quad 3]$.
2. Écrire un programme Matlab permettant l'implémentation du schéma numérique de cette méthode.
3. Afficher, sur le même graphe, la fonction $f_1(x)$ et la solution approchée.
4. Afficher le graphe représentant le nombre d'approximations successives $x^{(k+1)}$ en fonction du nombre d'itérations.
5. Tracer l'évolution de l'erreur en fonction du nombre d'itérations.
6. Tracer l'erreur ln$|e_{n+1}|$ en fonction de ln$|e_n|$ et déterminer l'ordre de la méthode numérique. Calculer la vitesse de convergence.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Appliquez le même algorithme pour résoudre l'équation : $f_2(x) = x + exp(x) + 1$ avec $x \in [-2 \quad 3/2]$.

On donne : tolérance $= 10^{-6}$, *les valeurs initiales sont* $x_0 = 0.8$ *pour* $f_1(x)$ *et* $x_0 = -1/5$ *pour* $f_2(x)$.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 15/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DU POINT FIXE
5  x0 = 0.8; it = 0; tol = 1e-05; Nbrit = 30;
6
7  while it < Nbrit
8
9      it = it +1;
10     x = cos(x0) ;
11     x0 = x ;
12
13     xn(it) = x;
14
15       if abs(x - cos(x)) < tol
16
17         sol = x;
18         break
19     end
20 end
21
22 a = 0; b = 3; n = 1000; dx = (b - a)/n; x1 = a:dx:b;
23 y = inline('cos(x)');
24
25 figure('color', [1 1 1])
26 plot(x1, x1,'k') ; hold on ; plot(x1, y(x1),'LineWidth',2) ; hold on
27     plot(sol,y(sol),'ro','MarkerSize',12,'LineWidth',2)
28      hold on ; plot(sol,y(sol) ,'rx','MarkerSize',12,'LineWidth',2)
29
30 hold on
31     plot(sol,0,'ko','MarkerSize',12,'LineWidth',2) ; hold on
32  plot(sol,0 ,'kx','MarkerSize',12,'LineWidth',2) ; hold on
33  line(x1, zeros(1, length(x1)),'LineStyle','-.','Color','k','
     LineWidth',1)
34 xlabel('x') ; ylabel('f(x)')
35
36 hold on ; line(sol.*ones(1, length(x1)), y(x1),'LineStyle','-.','
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
   Color','k','LineWidth',1) ; axis([0 3 -1 1.5])
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 text('Interpreter','latex',...
40  'String','$ x = cos(x) $','Position',[2 -1/3],'FontSize',15)
41 text('Interpreter','latex','String','$ y = x $',...
42  'Position',[1.2 1],'FontSize',15) ; text(sol,2*sol,['\alpha = ',
   num2str(sol)],...
43  'Position',[0.8 -1/4],'BackgroundColor',[1 1 1]);
44
45 %%%%%%%%%%%%%%%%%%%%%%%%%%% CALCUL D'ERRRUR %%%%%%%%%%%%%%%%%%%%%%%%%%%
46 err = abs(xn - sol); error_1 = err(1:end-1) ; error_2 = err(2: end);
47
48 figure('color', [1 1 1]) ; loglog(error_1(1:end-4),error_2(1:end-4),'
   +')
49 ylabel('Ln |e_{n+1}|') ;  xlabel('Ln |e_{n}|')
50
51 cutoff = 7;  % ATTENTION AU CHOIX DE LA NIEME ITERATION
52
53 pente = (log(error_2(end-cutoff)) - log(error_2(end-(cutoff+1))))/(
   log(error_1(end-cutoff)) - log(error_1(end-(cutoff+1)))); %
   VITESSE DE CONVERGENCE log|en+1| = log|en|
54 pente = round(pente);
55
56 %%%%%%%%%%%%%%%%%%%%% VITESSE DE CONVERGENCE (CV) %%%%%%%%%%%%%%%%%%%%%
57 n = 20;  % ATTENTION AU CHOIX DE LA NIEME ITERATION
58 vitesse_CV = (xn(n+2) - xn(n+1))/(xn(n+1) - xn(n));
59
60 msg1 = gtext(strcat('CETTE METHODE EST D''ORDRE : ', num2str(pente)))
   ;% CLIQUER SUR LA FIGURE POUR AFFICHER msg1
61
62 msg2 = gtext(strcat('LA VITESSE DE CONVERGENCE VAUT : ', num2str(
   vitesse_CV)));
63 % CLIQUER SUR LA FIGURE POUR AFFICHER msg2
```

Les résultats des différents calculs sont portés sur les figures suivantes

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

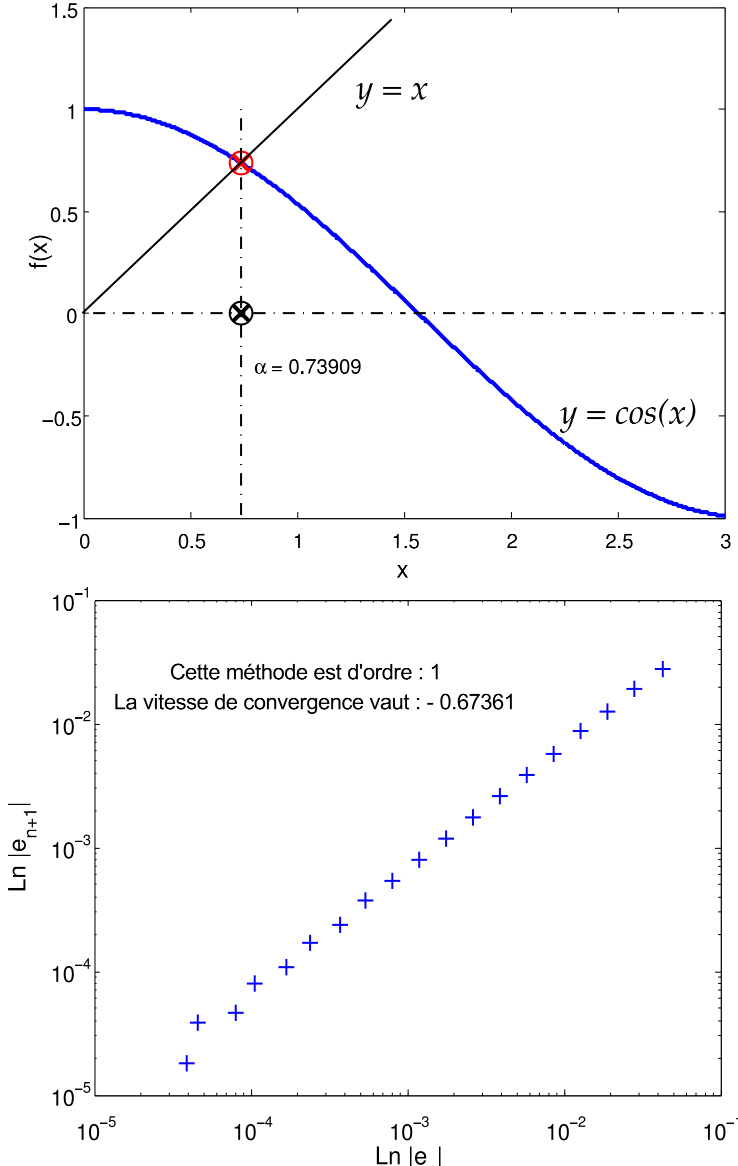

FIGURE 3.1: Racine de la fonction f obtenue par la méthode du *point fixe*

L'inconvénient de la méthode du point fixe, est qu'elle converge trop lentement. Afin d'améliorer sa convergence on peut la transformer en une nouvelle suite par le biais de l'algorithme d'accélération de convergence. Cette algorithme est connue sous le nom de `d'Aitken-Shanks`.

$$x^{(k+1)} = x^{(k)} - \frac{(\varphi(x^{(k)}) - x^{(k)})^2}{\varphi(\varphi(x^{(k)})) - 2\,\varphi(x^{(k)}) + x^{(k)}} \tag{3.5}$$

L'algorithme `d'Aitken-Shanks` peut être appliqué à toute méthode de point fixe. Très souvent la convergence de cette méthode est très rapide.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; close all ; clc ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 28/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE Aitken-Shanks

xinit = 0.8 ; it = 0 ; tol = 1e-05 ; Nbrit = 30 ;

while it < Nbrit
    it = it +1;

    phi = cos(xinit);

    phiphi = cos(phi);

    x = xinit - ((phi - xinit).^2)/(phiphi - 2*phi + xinit);

    xinit = x ;

    xn(it) = x;

      if abs(x - cos(x)) < tol

        sol = x;
        break
    end

end
```

À titre comparatif et tenant compte des deux codes Matlab présentés ci-dessus, la méthode du point fixe converge vers la racine approchée `sol = 7.390850858357935e-01`, au bout de 24 itérations. Alors que celle d'Aitken-Shanks converge vers la même solution au bout de 2 itérations seulement.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❶ ☞ Ⓢ

1. En utilisant les méthodes du *point fixe* et de *Aitken-Shanks*, trouver la racine des fonctions :

$$\begin{cases} f_1(x) = x - x^3, \quad x \in [0, 1] \\ f_2(x) = x - \exp(-x), \quad x \in [-1, 1] \\ f_3(x) = x - x^{4/5} + 2, \quad x \in [-2, 2] \\ f_2(x) = x - \exp(-x) - 4, \quad x \in [0, 5] \end{cases} \tag{3.6}$$

2. Tracer, sur la même figure, la fonction et sa racine calculée.
3. Afficher le graphe représentant le nombre d'approximations successives $x(k+1)$ en fonction du nombre d'itérations.
4. Tracer l'erreur $ln|e_{n+1}|$ en fonction de $ln|e_n|$ et déterminer l'ordre de la méthode numérique.
5. Calculer la vitesse de convergence des deux méthodes

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3.2 Méthode de dichotomie

Le principe de la méthode de `dichotomie`, encore appelée méthode de `bissection`, est basé sur le théorème de la valeur intermédiaire. La méthode est décrite comme suit : soit, $f : [a \quad b] \longrightarrow R$, une fonction continue sur l'intervalle $[a \quad b]$. Si $f(a) \times f(b) < 0 \longrightarrow$ il existe donc au moins une racine de $f(x)$ appartenant à l'intervalle $[a \quad b]$. On prend $c = \dfrac{a+b}{2}$ la moitié de l'intervalle $[a \quad b]$ tel que :

1. Si $f(c) = 0 \longrightarrow c$ est la racine de $f(x)$.
2. Sinon, nous testons le signe de $f(a) \times f(c)$ (et de $f(c) \times f(b)$).
3. Si $f(a) \times f(c) < 0 \longrightarrow$ la racine se trouve dans l'intervalle $[a \quad c]$ qui est la moitié de $[a \quad b]$.
4. Si $f(c) \times f(b) < 0 \longrightarrow$ la racine se trouve dans l'intervalle $[c \quad b]$ qui est la moitié de $[a \quad b]$.

Ce processus de division, par deux, de l'intervalle (à chaque itération on divise l'intervalle par deux) de la fonction est réitéré jusqu'à la convergence pour la tolérance considérée. Ainsi, pour la nième itération, on divise : $[a_n \quad b_n]$ en $[a_n \quad c_n]$ et $[c_n \quad b_n]$, avec à chaque fois $c_n = \dfrac{a_n + b_n}{2}$.

Exercice ❷ ☞ Ⓡ

Dans cet exercice, il est demandé de trouver la racine de $f(x) = x + exp(x) + \dfrac{10}{1+x^2} - 5$, en utilisant la méthode de `dichotomie`

1. Écrire un programme Matlab permettant l'implémentation du schéma numérique de cette méthode.
2. Afficher, sur le même graphe, la fonction $f(x)$, la solution approchée et les approximations successives.
3. Calculer l'ordre et la vitesse de convergence de la méthode numérique.

On donne : tolérance $= 10^{-8}$, $a_0 = -1.30$ *et* $b_0 = 3/2$

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 14/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE DICHOTOMIE
5  
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
a = -1.3 ;   b = 3/2 ; itmax = 100 ; tol = 1e-6 ;
it = 0 ; center = (b + a)/2 ;  x = [a, center, b] ;
fun = @(x) x + exp(x) + 10./(1 + x.^2) - 5 ;

while it < itmax

    if fun(a)*fun(center) < 0

     b = x(2) ; a = x(1) ; center = (a + b)/2 ; x = [a, center, b] ;

    elseif fun(center)*fun(b) < 0

     b = x(3) ; a = x(2) ; center = (a + b)/2 ; x = [a, center, b] ;

    end

    if abs(fun(center)) < tol

     sol = center ;
       break
    end

    if it == itmax & abs(fun(center)) > tol

     disp('PAS DE CONVERGENCE POUR LA TOLERANCE CONSIDEREE')

    end

     it = it + 1;
     xit(it) = center ;
     itNumber = it ;
end

disp(strcat('LA RACINE APPROCHEE VAUT :', num2str(sol)))
%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%%
 ainit = -1.3 ; binit = 3/2 ; n = 500 ; dx = (binit - ainit)/n ;
 xn = ainit :dx: binit ;

figure('color', [1 1 1])  ; plot(xn,fun(xn),'LineWidth',1) ; hold on
plot(sol,fun(sol) ,'ro','MarkerSize',12,'LineWidth',2)
plot(sol,fun(sol) ,'rx','MarkerSize',12,'LineWidth',2)
plot(xit,fun(xit),'kx','MarkerSize',10,'LineWidth',1.5)
plot(xit,fun(xit),'ko','MarkerSize',10,'LineWidth',1.5)

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
50 line(xn, zeros(1, length(xn)),'LineStyle','-.','Color','k',...
51 'LineWidth',1) ; xlabel('x') ; ylabel('f(x)')
52 title('RACINE DE f(x) = 0 PAR LA METHODE DE DICHOTOMIE')
```

Les sorties renvoyées par ce script sont :

```
1 LA RACINE APPROCHEE VAUT :-0.92045
2 >> itNumber =
3               21
```

Sur ce graphique, les étiquettes noires représentent les approximations successives `xit` et l'étiquette rouge c'est la racine calculée `sol` par cette méthode numérique.

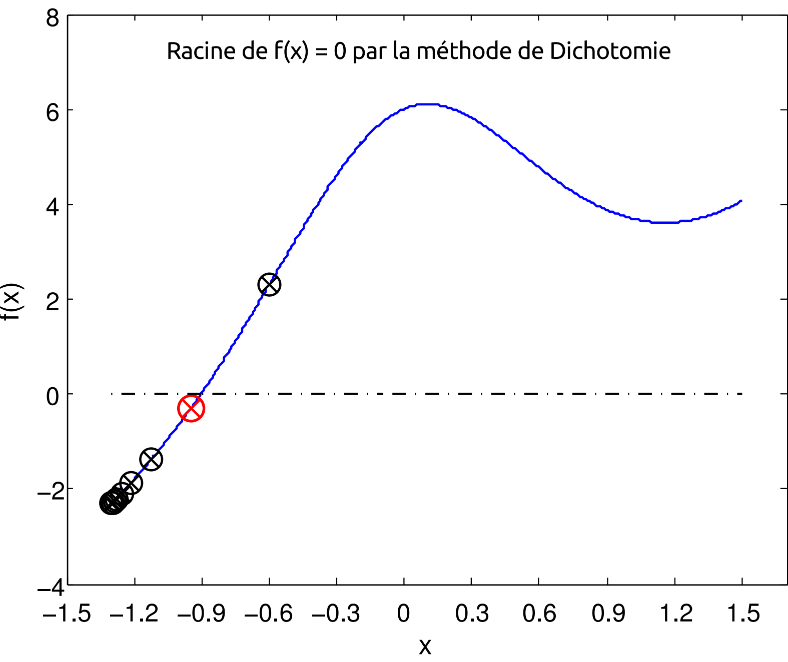

FIGURE 3.2: Racine de la fonction f obtenue par la méthode de *dichotomie*

La méthode de dichotomie est simple à mettre en œuvre. Néanmoins, elle souffre d'un inconvénient du fait que son seule critère d'arrêt consiste à contrôler la longueur de l'intervalle I_n à chaque itération. Ceci risque de rejeter la racine recherchée, car elle ne tient pas suffisamment en compte du comportement effectif de f. Ceci est parfaitement illustré par l'example ci-dessus.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3.3 Méthode de fausse position (ou de Lagrange)

Cette méthode découle explicitement de celle de `dichotomie`. Elle est plus efficace, dans la mesure qu'elle tient compte non seulement du signe de la fonction aux extrémités mais aussi de sa valeur aux bornes de l'intervalle considéré. Dans cette méthode, on divise l'intervalle $[a_n, b_n]$ en $[a_n, w_n]$ et $[w_n, b_n]$, la borne w_n représente l'abscisse du point d'intersection de la droite passant par $(a_n, f(a_n))$, $(b_n, f(b_n))$ et l'axe des abscisses. La borne w_n coupe l'axe des abscisses au point :

$$w_n = b_n - f(b_n) \times \frac{b_n - a_n}{f(b_n) - f(a_n)} = a_n - f(a_n) \times \frac{b_n - a_n}{f(b_n) - f(a_n)} \tag{3.7}$$

L'algorithme est construit de façon analogue à celui de `dichotomie`, en omettant de prendre la moitié de l'intervalle de travail après chaque itération.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 15/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE fausse position
5  a = -1.3 ;   b = 3/2 ; tol =  1e-6 ; itmax = 100 ;
6  fx = @(x) x + exp(x) + 10./(1 + x.^2) - 5 ;
7  w = a - fx(a)*((b - a)/(fx(b) - fx(a))) ; it = 0 ;
8
9  while it < itmax
10
11         if fx(a)*fx(w) < 0
12
13         w = a - fx(a) *((w - a)/(fx(w) - fx(a))) ;
14
15         elseif fx(w)*fx(b) < 0
16
17         w = w - fx(w)*((b - w)/(fx(b) - fx(w))) ;
18
19         end
20
21         if abs(fx(w)) < tol
22             sol = w ;
23             break
24         end
25
26      it = it + 1 ;
27      itNumber = it ;
28 end
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
29
30 disp(strcat('LA RACINE APPROCHEE VAUT :', num2str(sol)))
```

La solution approchée renvoyée par ce script est :

```
1 LA RACINE APPROCHEE VAUT :-0.90456
2 >> itNumber =
3              42
```

Ci-dessous, la fonction évaluée aux solutions trouvées respectivement par la méthode de `dichotomie` et de `fausse position` :

```
1 >> fx(-0.92045) =
2                  -0.1086     % dichotomie
3 >> fx(-0.90456) =
4                   1.7826e-05 % fausse position
```

À partir de ces résultats, on constate clairement la précision de la méthode de `fausse position` par rapport à celle de `dichotomie`. Néanmoins, la convergence de la méthode de `fausse position` est deux fois plus lente que celle de `dichotomie`

Exercice ❷ ☞ Ⓢ

1. Soient les équations non-linéaires suivantes :

$$\begin{cases} f_1(x) = \exp(-x) - \cos(x) - 2, \quad x \in [-2, 2] \\ f_2(x) = x^4 - 3\,x^2, \quad x \in [-2, 0] \\ f_3(x, y) = \cos(x) - x\, \exp(x), \quad x \in [0, 2\,\pi] \\ f_4(x) = 4\,\dfrac{\sin(x)}{x}, \quad x \in [1, 4] \end{cases} \tag{3.8}$$

2. Utiliser les méthodes de `dichotomie` et de `fausse position` pour déterminer les racines de ces équations. Prenez comme valeur initiale $x_0 = 1/2$, le nombre d'itération maximal `itmax = 100` et une tolérance `tol = 1e-5`. Commenter le résultat renvoyé par les deux méthodes.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3.4 Méthode de Newton

Comme il a été montré précédemment, la méthode de dichotomie exploite uniquement le signe de la fonction f aux extrémités des sous-intervalles. Lorsque cette fonction est différentiable, on peut établir une méthode plus efficiente en exploitant les valeurs de la fonction f et de ses dérivées. Le but dans cette section, est la programmation, sous Matlab, de la méthode itérative de `Newton`. Afin d'appréhender cette dernière, soit la figure ci-dessous :

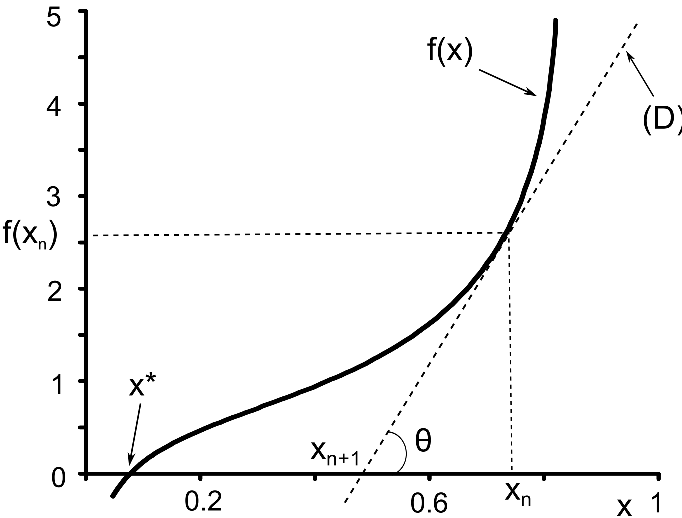

FIGURE 3.3: Principe de la méthode de *Newton*

Géométriquement, la solution approchée x_{n+1} n'est autre que le point d'intersection de l'axe des abscisses et la tangente, au point $(x_n,\ f(x_n))$, d'équation D : $y = f'(x_n) \times (x_n - x_{n+1}) + f(x_n)$. Notons que x^* est la véritable racine de l'équation $f(x) = 0$, dont on cherche à approcher. À partir de la figure ci-dessus, on a :

$$\tan\theta = \frac{f(x_n)}{x_n - x_{n+1}} \tag{3.9}$$

Or on sait que :

$$f'(x_n) = \lim_{(x_n - x_{n+1})\to 0} \frac{f(x_n) - 0}{x_n - x_{n+1}} = \tan\theta \tag{3.10}$$

À partir des Eqs. (3.9) et (3.10) on obtient ainsi le schéma numérique de la méthode de `Newton`, soit :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.11}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Tenant compte de toutes les méthodes vues jusqu'à présent, on constate que la méthode de Newton nécessite à chaque itération l'évaluation de deux fonctions, à savoir f et de sa dérivée. Néanmoins, cet effort est compensé par une vitesse de convergence accrue, puisque cette méthode est d'ordre deux. Cet accroissement de la vitesse de convergence est conditionné par le choix de la valeur initiale qui doit être la proche possible du zéro recherché.

Exercice ❸ ☞ ®

Nous allons résoudre l'équation : $f(x) = x + exp(x) + 1$. Nous choisissons $x_0 = -1/2$ comme valeur initiale. Écrire un code matlab, portant sur l'implémentation de la méthode de Newton, en suivant les étapes suivantes :

1. Faire un test si $f'(x) = 0 \Longrightarrow$ arrêt du programme.
2. Le critère d'arrêt est : $|x_{n+1} - x_n| < \varepsilon$, x_n étant la solution approchée et ε, la tolérance considérée.
3. Afficher la solution approchée x_n.
4. Afficher le nombre d'itérations conduisant à la solution approchée.
5. Afficher sur le même graphe, la fonction $f(x)$, la solution approchée x_n et la droite tangente au point $(x_n, f(x_n))$.

Appliquez le même algorithme pour résoudre l'équation : $f(x) = 8x^3 - 12x^2 + 1$

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; close all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 09/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE NEWTON

Nmax = 100; x = -1/2; it = 0; tol = 1e-04;
verif = tol + 1/2;

 while (it < Nmax & verif >= tol)

 fx   = inline('x +exp(x) + 1');
 dfx = inline('1 + exp(x)');
 fx = feval(fx, x);
 dfx = feval(dfx,x);
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
15
16 if dfx ~= 0
17 xn = x - (fx/dfx);
18 verif = abs(fx/dfx);
19 x = xn;
20 elseif dfx == 0
21     disp('PAS DE SOLUTION, LA DERIVEE EST NULLE')
22 end
23
24 if (it == Nmax & verif > tol)
25   disp('LA METHODE DE NEWTON NE CONVERGE PAS POUR LA TOLERANCE
     CONSIDEREE')
26 end
27
28 it = it +1;
29  end
30
31  disp(strcat('CONVERGENCE, LA SOLUTION APPROCHEE EST xn = ', num2str(
     xn)))
32  disp(strcat('LE NOMBRE D''ITERATION EST = ', num2str(it)))
33
34  xi = linspace(-12/2,6/2,1000);
35  fx   = inline('xi +exp(xi) + 1');
36  fxi = feval(fx, xi);
37  dfx = inline('1 + exp(x)'); droite = dfx(xn)*(xi - xn) + fx(xn);
38
39  figure('color',[1 1 1]) ; plot(xi,fxi,'LineWidth',2)
40  hold on ; plot(xi,droite,'r','LineWidth',1)
41      hold on
42      plot(x,fx(x),'kx','MarkerSize',12,'LineWidth',2)
43  hold on
44  plot(x,fx(x),'ko','MarkerSize',12,'LineWidth',2)
45
46  xlabel('x','fontweight','b','fontsize',12,'LineWidth',1)
47  ylabel('f(x)','fontweight','b','fontsize',12,'LineWidth',1)
48
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50
51  text('Interpreter','latex','String','$f(x) = x + exp(x) + 1 $',...
52  'Position',[-10/2 20/2],'FontSize',15)
53
54  text(xn,2*xn,['x_n = ',num2str(xn)],'Position',[-3 2],...
55  'BackgroundColor',[1 1 1]);
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Les résultats des différents calculs sont portés sur la figure suivante

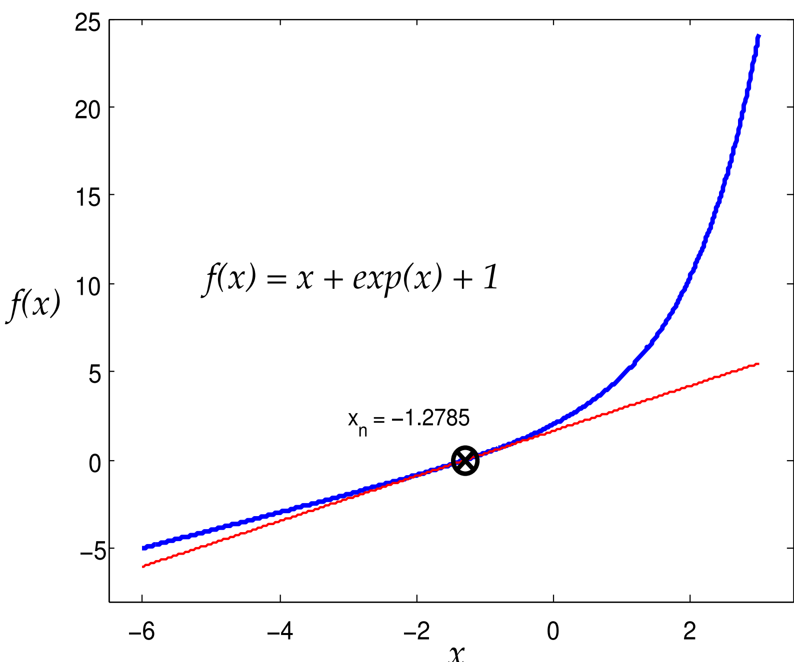

FIGURE 3.4: Racine de la fonction f obtenue par la méthode de *Newton*

Notons que la méthode de `Newton` converge de façon quadratique uniquement dans le cas où la racine recherchée de f est simple. Dans le cas contraire, elle converge de façon linéaire. Par ailleurs, cette méthode, peut être utilisée afin de résoudre des systèmes d'équations non linéaires.

Exercice ❸ ☞ Ⓢ

1. En utilisant la méthode de *Newton*, trouver les racines des fonctions :

$$\begin{cases} f_1(x) = \cos(x + 3\pi/8), & x \in [-1, 1] \\ f_2(x) = (x - \frac{1}{2})^2, & x \in [-1, 1] \\ f_3(x) = \cos(1/x^2), & x \in [-1, 1] \\ f_4(x) = \dfrac{x^2 - 3}{2}, & x \in [-5, 5] \end{cases} \tag{3.12}$$

2. Tracer, sur la même figure, la fonction et sa racine approchée.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3.5 Méthode de la sécante

La méthode de Newton est rapide et très utilisée, quand sa dérivée existe. Malheureusement dans certaines situations nous n'avons pas accès à cette dérivée. C'est pourquoi on s'est proposé d'étudier une autre méthode, dite de la sécante. Cette dernière s'affranchit de la dérivée de la fonction $f(x)$, en l'approchant par l'accroissement :

$$f'(x^k) \approx \frac{f^{(k)} - f^{(k-1)}}{x^{(k)} - x^{(k-1)}} \tag{3.13}$$

Le schéma numérique de cette méthode est donné par :

$$x^{(k+1)} = x^{(k)} - f^{(k)} \times \frac{x^{(k)} - x^{(k-1)}}{f^{(k)} - f^{(k-1)}} \tag{3.14}$$

En observant l'équation (3.14), on constate des similitudes avec celle de Newton. Par ailleurs, la méthode de la sécante nécessite l'initialisation de deux valeurs approchées x_0 et x_1 de la racine exacte de l'équation $f(x) = 0$.

Exercice ❹ ☞ ®

Trouver la racine de l'équation : $f(x) = x - 0.2 \times sin(4\,x) - 1/2$. On prend $x_0 = -1$ et $x_1 = 2$ comme valeurs initiales. Écrire un code matlab, portant sur l'implémentation de la méthode de la sécante en considérant une tolérance : $tol = 10^{-10}$.

1. Afficher la solution approchée x_n.
2. Afficher le nombre d'itérations conduisant à la solution approchée.
3. Afficher sur le même graphe, la fonction $f(x)$, la solution approchée et les approximations successives $x^{(k+1)}$.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 10/11/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE LA SECANTE
5
6  x(1) = -1 ; x(2) = 2; fun = x - 0.2.*sin(4.*x) - 1/2;
7  it = 0 ; tol = 1e-10 ; Nit = 20 ;
8
9  while it <  Nit
10
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
11      it = it + 1;
12
13      fun = eval('fun',x);
14      var = x(2) - fun(2)*((x(2) - x(1))/(fun(2) - fun(1)));
15
16      xn = var;
17      x = [x(2), xn];
18
19      if abs(xn -var) < tol
20
21          sol = xn;
22          approx(it) = xn;
23
24          break
25      end
26
27  end
28
29  fun = inline('x - 0.2.*sin(4.*x) - 1/2'); outPut = [sol, fun(sol)];
30  format long
31  %%%%%%%%%%%%%%%%%%%%%%%% affichage graphique %%%%%%%%%%%%%%%%%%%%%%
32
33  b = 4 ; a = -2 ; n = 1000 ; dx = (b - a)/n ; x = a:dx:b ;
34
35  figure('color', [1 1 1])
36  plot(x,fun(x),'LineWidth',1) ; hold on
37      plot(outPut(1),outPut(2),'ro','MarkerSize',12,'LineWidth',2)
38       hold on ; plot(outPut(1),outPut(2) ,'rx','MarkerSize',12,'
       LineWidth',2)
39
40  hold on
41      plot(approx,fun(approx),'kx','MarkerSize',10,'LineWidth',1.5)
42  hold on ; plot(approx,fun(approx),'ko','MarkerSize',10,'LineWidth'
      ,1.5) ; hold on
43
44  line(x, zeros(1, length(x)),'LineStyle','-.','Color','k','LineWidth'
      ,1)
45  xlabel('x') ; ylabel('f(x)') ;
46  title('RACINE DE f(x) PAR LA METHODE DE LA SECANTE')
47
48  %%%%%%%%%%%%%%%%%%%%%% AFFICHAGE SUR LA FIGURE %%%%%%%%%%%%%%%%%%%%%
49  text('Interpreter','latex','String','$x - \: 0.2.*sin(4.*x) - \: 1/2
       $','Position',[-3/2 2],'FontSize',15)
50
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
51 text(sol,2*sol,['sol = ',num2str(sol)],'Position',[-1/2 1/2],...
52  'BackgroundColor',[1 1 1]);
```

Les résultats des différents calculs sont portés sur la figure suivante

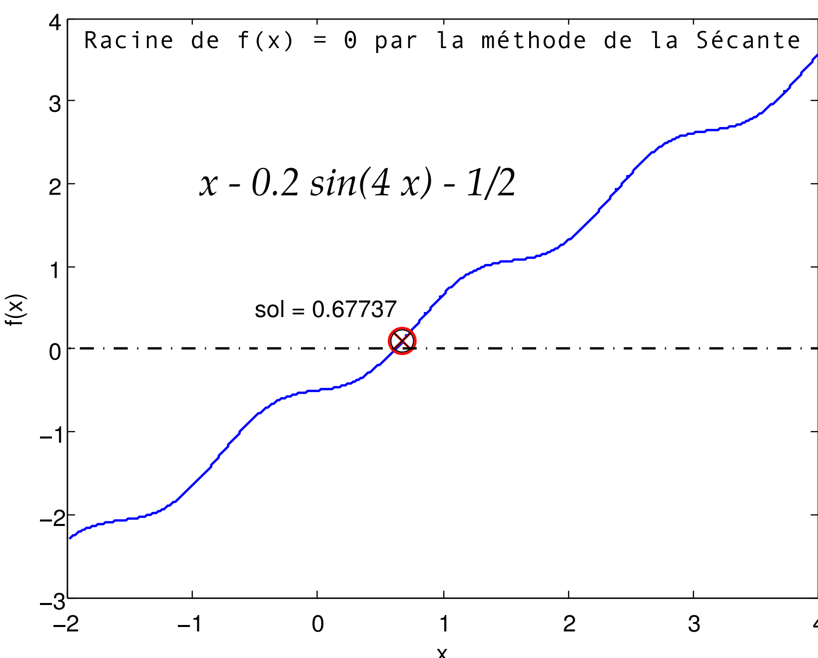

FIGURE 3.5: Racine de la fonction f obtenue par la méthode de la *sécante*

Signalons aussi que la divergence d'un algorithme n'implique pas forcément l'inexistante de la solution. Parfois, il faudra juste retenter avec une autre valeur initiale x_0 ou changer d'algorithme.

Exercice ❹ ☞ Ⓢ

1. Trouver, à l'aide de la méthode de la `secante`, la racine approchée des équations :

$$\begin{cases} f_1(x) = \cos(x^x) - \sin(\exp(x)), \quad x \in [1/2, 3] \\ f_2(x) = x\,\log(x) - \log(x), \quad x \in [-2, 2] \\ f_3(x) = x\,\exp(x) - \exp(x), \quad x \in [-2, 2] \\ f_4(x) = x^3 - 3\,x^2 2^{-x} + 3x4^{-x} - 8^{-x}, \quad x \in [0, 1] \\ f_5(x) = x^3 + 4x^2 - 10, \quad x \in [-3, 3] \end{cases} \tag{3.15}$$

2. Tracer, sur un graphe approprié, la fonction et la racine trouvée.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

3.6 Au moyen de routines Matlab

Matlab dispose de commandes prédéfinies destinées à chercher les zéros d'une fonction. On testera dans un premier temps la commande `fzero`. Elle pour syntaxe :

```
[x , fval, exitflag, output] = fzero(fun, x0, options).
```

Cette commande cherche un zéro autour de la valeur initiale $x0$ indiquée. La racine approchée x renvoyée est prés d'un point ou `fun` change de signe. La commande renvoie NaN (Not a Number) dans le cas où la recherche de la racine a échoué. On peut aussi spécifier un intervalle dans le quelle `fzero` va chercher la racine approchée. Cependant, il faut s'assurer que la fonction change de signe dans cet intervalle. L'argument `options` est de type `srtucture` qui recèle les options d'optimisation indiquées dans `optimset`. Voici un exemple :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  fun = @(x) x - 0.2.*sin(4.*x) - 1/2 ; lB = -2 ;  uB = 4;
4
5  opts = optimset('Display','iter','FunValCheck','on','TolX',1e-6) ;
6  [racine,funEval,exitTest,output] = fzero(fun, [lB uB], opts); % find
      the zero of fun between lowerBound and upperBound
```

Voici les résultats affichés par ce script Matlab :

```
1  %%%%%%%%%%%%%%%%%%%%%%% affichage par defaut %%%%%%%%%%%%%%%%%%%%%%
2   Func-count     x          f(x)
3      2             -2       -2.30213
4      3        0.357245     -0.340747
5      4        0.730969      0.187769
6      5        0.598194    -0.0379606
7      6        0.620523   -0.00202224
8      7        0.621766   8.20089e-06
9      8        0.621761  -6.06744e-09
10     9        0.621761  -6.06744e-09
11
12 Zero found in the interval [-2, 4]
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 >> racine    =
15             0.6218
16 >> funEval   =
17            -6.0674e-09
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
18 >> exitTest =
19             1
20 >> output =
21
22             intervaliterations: 0
23                     iterations: 7
24                      funcCount: 9
25                      algorithm: 'bisection, interpolation'
26                        message: [1x34 char]
```

La sortie `exitTest = 1` signifie que l'algorithme a convergé vers la solution approchée. Une valeur de `exitTest < 0` indiquerai une divergence de l'algorithme. L'argument `funEval = -6.0674e-09` donne l'évaluation de la fonction à la dernière itération, c'est-à-dire à la racine approchée. Les sorties renvoyées par `output` sont `funcCount = 9` représentant le nombre d'évaluations de la fonction et `intervaliterations` est le nombre d'itérations pour trouver un intervalle contenant la racine approchée. Le champ `iterations = 7` exprime le nombre d'itérations pour trouver la racine approchée et `message` renvoie la chaîne de caractères : `'Zero found in the interval [-2, 4]'`. La sortie `algorithm` renvoie l'algorithme utilisé pour résoudre cette équation qui est `'bisection'`. Le champ de la structure `opts` indiqué par `optimset('Display','iter', ...)` affiche des détails pour chaque itération. Le champ indiqué par `optimset(...., 'FunValCheck','on',...)` contrôle si les valeurs de la fonction sont réelles et affiche dans le cas contraire un avertissement quand `fzero` renvoie une valeur complexe ou NaN. Le champ indiqué par `optimset(..., 'TolX', 1e-6)` désigne la tolérance avec laquelle sera calculée la racine approchée.

Exercice ❺ ☞ ®

1. Trouver la racine des fonctions suivantes :

$$\begin{cases} f_1(x) = \exp(x) - 2\cos(x), & x \in [-1, 1] \\ f_2(x) = \dfrac{x^3 + 2x - 5}{x^2 + 1}, & x \in [-2, 2] \end{cases} \tag{3.16}$$

2. Tracer, dans un graphe approprié, les fonctions et leurs racines. Annoter les graphes.

Prenez comme valeurs initiales $x_0 = 1/2$ pour $f_1(x)$ et $x_0 = 2$ pour $f_2(x)$.

Nous allons désormais voir la commande `fsolve`, appliquée pour la résolution de systèmes d'équations non-linéaires. Soit à résoudre le système d'équations suivant :

$$\begin{cases} 2x_1 - x_2 = \exp(-x_1) \\ -x_1 + 2x_2 = \exp(-x_2) \end{cases} \tag{3.17}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Pour résoudre ce système d'équations, nous commencerons d'abord par écrire un programme *M-file*, selon :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 function fun = myfunction(x)
2
3 fun = [2*x(1) - x(2) - exp(-x(1));
4       -x(1) + 2*x(2) - exp(-x(2))];
5 return
```

Bien évidemment, comme il a été déjà signalé au troisième chapitre, ce fichier doit être absolument sauvegardé, dans le répertoire courant, sous le nom de `myfunction.m`. Sinon Matlab ne reconnaitra pas la fonction. Une fois cette étape terminée, on écrira le script Matlab comme suit :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all ; clc ;
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3
4 xinit = [-5 ; -3];
5 opts = optimset('NonlEqnAlgorithm','lm','LargeScale','off','Display',
     'iter','TolX',1e-6,'TolFun',1e-6);
6 [x, fval, exitflag, output, jacobian] = fsolve(@myfunction, xinit,
     opts) ;
```

Voici les résultats affichés par ce script Matlab :

```
1  %%%%%%%%%%%%%%%%%%%%%%% Affichage par defaut %%%%%%%%%%%%%%%%%%%%%%
2   Iteration  Func-count    Residual      derivative     Lambda
3       0           3          24597.8
4       1          11          38.3734      -40.9          8.07794e-28
5       2          18         0.520524      -0.136         9.41035e-29
6       3          25      0.000166037      -0.000798      0.421728
7       4          31       3.0741e-08      -2.42e-11      0.191427
8       5          37      1.38904e-12       3.13e-17      0.0948403
9
10 Optimization terminated: directional derivative along
11  search direction less than TolFun and infinity-Norm of
12  Gradient less than 10*(TolFun+TolX).
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Le champ spécifié par `optimset('NonlEqnAlgorithm','lm', ....)` impose

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

l'utilisation de l'algorithme de Levenberg-Marquardt. Si par exemple la chaîne de caractères 'lm' est remplacée par 'gn', cela signifie qu'on utilisera l'algorithme de Gauss-Newton. Ce champ optimset(....,'LargeScale','off', ...) stipule l'utilisation de méthodes moyenne dimension. L'argument (...,'TolFun',1e-6) est la tolérance finale sur la valeur de la fonction. La sortie jacobian renvoie le Jacobien de la fonction :

```
>> jacobian =
             2.5671   -1.0000
            -1.0000    2.5671
```

La sortie output, de type *structure*, renvoie les champs suivants :

```
>> output =

        iterations: 6
         funcCount: 37
          stepsize: 1
         algorithm: [1x46 char] % medium-scale: Levenberg-Marquardt
           message: [1x147 char] % Optimization terminated ...
```

Les valeurs numériques renvoyées sont iterations: 6, funcCount: 37 et stepsize: 1. La fonction a été évaluée 37 fois et l'algorithme converge vers la solution approchée au bout de la $6^{\text{ième}}$ itération. La sortie stepsize: 1 indique le pas final de la méthode moyenne dimension.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❻ ☞ Ⓢ

1. Trouver les racines fonctions non-linéaires suivantes :

$$\begin{cases} f_1(x) = -x^2 + \cos^2(2x), \quad x \in [0,2] \\ f_2(x) = x - \exp(-(1+x)), \quad x \in [-1,1] \\ f_3(x) = x(1+\exp(x)) - \exp(x), \quad x \in [-1,1] \\ f_4(x) = x^3 - 4x - 9, \quad x \in [1,3] \\ f_5(x) = x - \exp(1/x), \quad x \in [1,4] \\ f_6(x) = -5x^3 + 39x^2 - 43x - 39, \quad x \in [1,5] \\ f_7(x) = x^3 - 6x + 1, \quad x \in [-2,2] \\ f_8(x) = \frac{5}{2} - \sin(x) + \frac{\pi}{6} - \frac{\sqrt{3}}{2}, \quad x \in [0,\pi] \\ f_9(x) = x^2 - x - 2, \quad x \in [-3,3] \end{cases}$$

2. Trouver les racines des systèmes d'équations suivants :

$$\begin{cases} x_1 + x_2 - 3 = 0 \\ x_1^2 + x_2^2 - 9 = 0 \end{cases}$$

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ \sin(\pi\, x_1) + x_2^3 = 0 \end{cases}$$

Prenez comme valeurs initiales $x_0 = [\frac{1}{2}, \frac{1}{2}]$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

4 Résolution numérique des équations différentielles

Euler, Heun et Runge-Kutta d'ordre 3 et 4

Sommaire

10.1 Méthodes à un pas . . . 218
10.1.1 Méthode d'Euler . . . 218
10.1.2 Méthode de Heun . . . 219
10.1.3 Méthode de Runge–Kutta, d'ordre 3 . . . 219
10.1.4 Méthode de Runge–Kutta, d'ordre 4 . . . 219
10.1.5 Équations différentielles d'ordre n . . . 225
10.2 Au moyen de commandes Matlab . . . 229
10.3 Méthode des différences finies . . . 234

Introduction

On désire calculer la solution sur l'intervalle $I = [a, b]$ du problème de Cauchy

$$\dot{y}(t) = f(t, y(t)) \quad \text{avec} \quad y(t_0) = y_0 \tag{4.1}$$

On comprend, ainsi, qu'une équation différentielle est une équation dépendant d'une variable t et d'une fonction $y(t)$ et qui contient des dérivées de $y(t)$. Elle peut s'écrire comme :

$$F\left(t, y(t), y^{(1)}(t), y^{(2)}(t), ..., y^{(k)}(t)\right) = 0 \iff y^{(k)}(t) = \frac{d^k y(t)}{dt^k} \tag{4.2}$$

Pour $k = 2$, l'équation ci-dessus peut se mettre sous la forme différentielle :

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f(t, y(t)) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{4.3}$$

L'existence d'une solution unique de l'équation différentielle est tributaire de l'imposition de certaines conditions limites sur $y(t)$ et ses dérivées.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Dans l'équation (6.10), les conditions initiales sont les valeurs de $y(a), y^{(1)}(a), y^{(2)}(a), ..., y^{(k-1)}(a)$. D'un point de vue pratique, l'inconnue $y(t)$ est un vecteur ayant k fonctions inconnues avec pour conditions initiales le vecteur $y_k(a)$ (ou $y_k(t_0)$). Cependant, il faut noter que très souvent la solution analytique n'existe pas, et on doit par conséquent approcher la solution exacte $y(t)$ par des méthodes numériques.

4.1 Méthodes à un pas

Ces méthodes sont basées sur un développement en série de `Taylor` suivant un ordre plus au moins élevé. Elles sont qualifiées `à un pas`, car le calcul de y_{k+1} ne réclame que la valeur de y_k à l'instant précédent. Une méthode `à deux pas` utilisera à la fois y_k et y_{k-1}. Les schémas numériques des méthodes d'`Euler` explicite, de `Heun` (ou de `Runge-Kutta` d'ordre 2) et de `Runge-Kutta` classique d'ordre 4 sont donnés dans ce qui suit

4.1.1 Méthode d'Euler

Afin d'atteindre la solution $y(t)$, sur l'intervalle $t \in [a, b]$, on choisit $n+1$ points dissemblables $t_0, t_1, t_2, ..., t_n$, avec $t_0 = a$ et $t_n = b$ et le pas de discrétisation est défini par $h = (b-a)/n$. La solution à estimer peut être approchée par un développement limité de Taylor

$$y(t_k + h) = y(t_k) + \frac{dy(t_k)}{dt}(t_{k+1} - t_k) + ... \tag{4.4}$$

Puisque $\frac{dy(t_k)}{dt} = f(t_k, y(t_k))$ et $h = t_{k+1} - t_k$, on obtient ainsi le schéma numérique d'`Euler` :

$$\begin{cases} y_0 = \text{valeurs initiales} \\ y_{k+1} = y_k + h\, f(t_k, y_k), \quad \text{avec} \quad k = 0, 1, ..., n-1. \end{cases} \tag{4.5}$$

Cette méthode est d'ordre 1, cela veut dire que l'erreur est proportionnelle au carré du pas (h) de discrétisation. Intuitivement, on comprend que pour améliorer la précision cette méthode, il suffira de réduire h. Cette réduction du pas de discrétisation aura pour incidence l'accroissement du temps de calcul ($\sim 1/h$). Par ailleurs, l'avantage de la méthode d'`Euler`, tire son origine du fait qu'elle réclame uniquement l'évaluation de la fonction f pour chaque pas d'intégration.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

4.1.2 Méthode de Heun

La Méthode de `Heun` est une version améliorée de celle d'`Euler`. L'erreur sur le résultat générée par cette méthode est proportionnelle à h^3, meilleur que celle de la méthode d'`Euler`. Néanmoins, la méthode de `Heun` réclame une double évaluation de la fonction f.

$$\begin{cases} y_0 = \text{valeurs initiales} \\ y_{k+1} = y_k + \frac{h}{2}\left(f(t_k, y_k) + f(t_{k+1}, y_k + h\, f(t_k, y_k))\right), \quad \text{avec } k = 0, 1, \ldots, n-1. \end{cases} \tag{4.6}$$

Le schéma numérique de cette méthode résulte de l'application de la formule de quadrature du trapèze. Notons également que la méthode de `Heun` fait partie des méthodes de Runge-Kutta explicites d'ordre deux.

4.1.3 Méthode de Runge–Kutta, d'ordre 3

Les méthodes de type `Runge-Kutta` permettent d'obtenir une plus grande précision (elles génèrent des solutions numériques plus proches des solutions analytiques) que les deux méthodes précédentes.

$$\begin{cases} y_0 = \text{valeurs initiales} \\ y_{k+1} = y_k + \left(f(t_k, y_k) + 4\, f(t_k + \dfrac{h}{2}, y_k + y_k\, \dfrac{h}{2}) + f(t_k + h, y_k + (2\, y_{2k} - y_{1k})\, h\right) \\ \text{avec } k = 0, 1, \ldots, n-1 \text{ et} \end{cases} \tag{4.7}$$

$$\begin{cases} y_{1k} = f(t_k, y_k) \\ y_{2k} = f(t_k + \frac{h}{2}, y_k + y_{1k}\, \frac{h}{2}) \end{cases} \tag{4.8}$$

C'est la méthode de Runge-Kutta explicite à trois niveaux.

4.1.4 Méthode de Runge–Kutta, d'ordre 4

La méthode de `Runge-Kutta` (classique) d'ordre 4, est une méthode explicite très populaire. Elle calcule la valeur de la fonction en quatre points intermédiaires selon :

$$\begin{cases} y_0 = \text{valeurs initiales} \\ y_{k+1} = y_k + \dfrac{h}{6}\left(f(t_k, y_{1k}) + 2\, f(t_k + \dfrac{h}{2}, y_{2k}) + 2\, f(t_k + \dfrac{h}{2}, y_{3k}) + f(t_{k+1}, y_{4k})\right) \\ \text{avec } k = 0, 1, \ldots, n-1 \text{ et} \end{cases} \tag{4.9}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$$\begin{cases} y_{1k} = y_k \\ y_{2k} = y_k + \frac{h}{2}\, f(t_k, y_{1k}) \\ y_{3k} = y_k + \frac{h}{2}\, f(t_k + \frac{h}{2}, y_{2k}) \\ y_{4k} = y_k + h\, f(t_k + \frac{h}{2}, y_{3k}) \end{cases} \tag{4.10}$$

Notons que le nombre de termes retenus dans la série de `Taylor` définit l'ordre de la méthode de `Runge-Kutta`. Il vient que la méthode `Runge-Kutta` d'ordre 4, s'arrête au terme $O(h^4)$ de la série de `Taylor`.

1. Résoudre numériquement, par le biais des méthodes de `Euler`, de `Heun` et de `Runge-Kutta` d'ordre 4, l'équation différentielle du premier ordre suivante :

$$\begin{cases} y' = \dfrac{-y\,t^2 - y^2 + 2\,t}{1 - t^3} \\ y(0) = 1 \end{cases} \tag{4.11}$$

2. Afficher sur la même figure, la solution des trois méthodes.
3. Analyser l'erreur en fonction du pas de discrétisation pour la méthode de `Runge-Kutta` d'ordre 4.
4. Connaissant la solution exacte de l'équation différentielle ci-dessous. Proposer une démarche permettant la détermination de l'ordre de convergence de la méthode de `Runge-Kutta`. On donne :

$$\begin{cases} y' = \dfrac{t - y}{2} \\ y(0) = 1 \end{cases} \tag{4.12}$$

   La solution exacte est : $3\,exp(-t/2) + t - 2$
5. Tracer le graphique donnant l'erreur relative, pour chaque pas de discrétisation, en fonction du nombre d'itérations.
6. Tracer le graphique donnant le maximum de l'erreur relative en fonction du pas de discrétisation.

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all ;
2 close all ;
3 clc ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 13/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE METHODES NUMERIQUES (Euler, Heun,
% Runge-Kutta) POUR LA RESOLUTION D'EQUATIONS DIFFERENTIELLES

%%%%%%%%%%%%%%%%%%%% METHODE DE Runge-Kutta d'ordre 4 %%%%%%%%%%%%%%%%

dydt = inline('(-y.*t.^2 - y.^2 + 2.*t)./(1 - t.^3)','t','y');
a = 0 ; b = 1 ; n = 70 ; h = (b-a)/n ; t = a :h: b;

epsilon = 0.0001 ; u(1) = 1 + epsilon ;

for i = 1:n-1

   u1(i) = u(i)  ; u2(i) = u(i) + h/2*dydt(t(i),u1(i)) ;

   u3(i) = u(i) + h/2*dydt(t(i) + h/2, u2(i)) ; u4(i) = u(i) + h*dydt
   (t(i) + h/2, u3(i)) ;

   u(i+1) =u(i) + h/6*(dydt(t(i),u1(i)) + 2*dydt(t(i) + h/2,...

   u2(i)) + 2*dydt(t(i) + h/2,u3(i)) + dydt(t(i+1),u4(i))) ;

end

 figure('color',[1 1 1])
 plot(t(1:end-1),u,'o-g')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% METHODE DE Heun %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all ;
clc ;

dydt = inline('(-y.*t.^2 - y.^2 + 2.*t)./(1 - t.^3)','t','y');
a = 0 ; b = 1 ; n = 70 ; h = (b-a)/n ; t = a:h:b;

epsilon = 0.0001 ; u(1) = 1 + epsilon;

for i = 1:n - 1

  u(i+1) =u (i) + h/2*(dydt(t(i),u(i)) + dydt(t(i + 1),...
      u(i) + h*dydt(t(i),u(i))));
end
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
47
48 tn = t(1:end-1) ;
49 hold on ; plot(tn,u,'o-r') ;
50
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%% METHODE D'Euler %%%%%%%%%%%%%%%%%%%%%%%%%
52 clear all ;
53 clc ;
54
55 dydt = inline('(-y.*t.^2 - y.^2 + 2.*t)./(1 - t.^3)','t','y');
56
57 a = 0 ; b = 1 ; n = 70 ; h = (b-a)/n ; t = a:h:b;
58 epsilon = 0.0001 ; u(1) = 1 + epsilon;
59
60 for i = 1:n - 1
61
62   u(i+1) =u (i) + h/2*(dydt(t(i),u(i))) ;
63
64 end
65
66 hold on
67 plot(t(1:end-1),u,'o-b')
68
69 legend('Runge-Kutta','Heun','Euler')
```

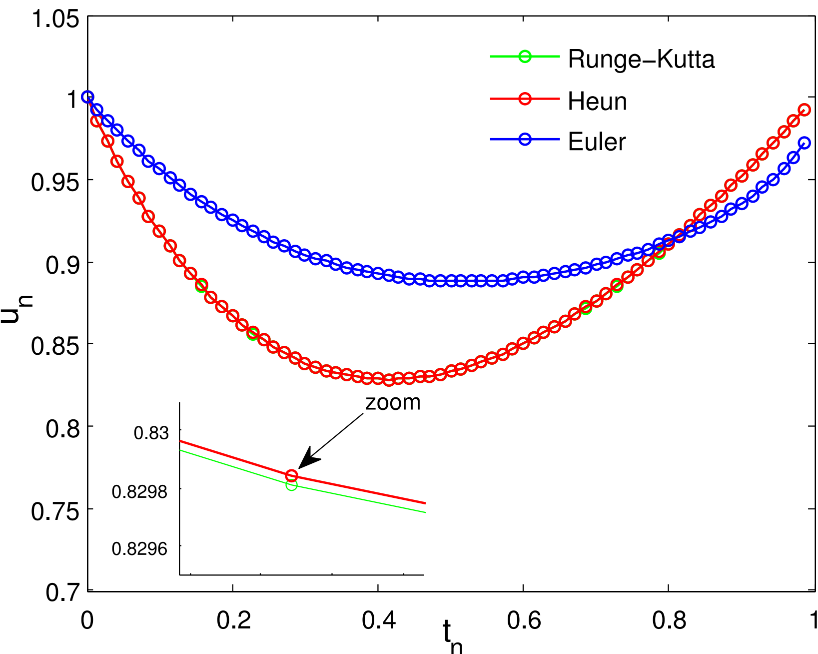

FIGURE 4.1: Solutions numériques obtenues par les méthodes de Euler, de Heun et de Runge-Kutta d'odre 4

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1
2  clear all ; close all ; clc ;
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  % @copyright 14/12/2015 Samir KENOUCHE : ANALYSE DE L'ERREUR EN
5  % FONCTION DU PAS DE DISCRITISATION DANS LE CADRE DE LA METHODE DE
6  % Runge-Kutta D'ORDRE 4
7
8  dydt = inline('(t - y)/2','t','y');  % EQUATION DIFFERENTIELLE
9  funex = inline('3*exp(-t/2)- 2 + t','t') ; % SA SOLUTION EXACTE
10
11 a = 0 ; b = 6 ; ik = 0 ; epsilon = 0.0001 ; u(1) = 1  ;
12
13 col = {'k','b','r','m','c','g'};
14
15 for n = 60:10:100
16
17     h = (b-a)/n ; t = a :h: b;
18
19   for i = 1: n -1
20
21    u1(i) = dydt(t(i),u(i)) + 2*dydt(t(i) + h/2,...
22    u(i) + h/2*dydt(t(i),u(i))) ;
23
24    u2(i) = dydt(t(i) + h/2,u(i)+h/2*dydt(t(i)+h/2,...
25    u(i)+h/2*dydt(t(i),u(i))));
26
27    u3(i) = dydt(t(i+1), u(i) + h*dydt(t(i)+h/2,...
28    u(i)+h/2*dydt(t(i)+h/2,u(i) + h/2*dydt(t(i), u(i))))) ;
29
30    u(i+1) = u(i) + h/6*(u1(i) + 2*u2(i) + u3(i)) ;
31
32   end
33
34 ik = ik + 1;
35
36 t = t(1:end-1) ;
37
38 err  =  abs((funex(t) - u)./funex(t));
39
40 max_err(ik) = max(err) ;  pas(ik) = h ;
41
42 figure(1) ; hold on ; plot(err,col{ik}, 'LineWidth',1) ;
43 xlabel('NOMBRE D''ITERATIONS') ; ylabel('ERREUR RELATIVE')
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
44
45 end
46
47 figure(3) ; plot(t,u,'o-') ; hold on
48 plot(t, funex(t),'o-r')
49
50 for p = 1:6
51
52 [coeff, s] = polyfit(pas,max_err,p) ; % s ETANT L'ERREUR
53 % D'INTERPOLATION, POUR UN DEGRE p, GENEREE PAR LA FONCTION : polyfit
54 evalp = polyval(coeff, pas) ;
55
56 err_iterpolation(p) = max(abs(evalp - max_err));
57
58 end
59
60 [min_err_interp, degre_interp] = min(err_iterpolation);
61
62 coeff_opt = polyfit(pas,max_err,degre_interp);
63
64 eval_opt = polyval(coeff_opt, pas) ;
65
66 figure('color', [1 1 1])
67 plot(pas, max_err,'r+','MarkerSize',10,'LineWidth',1)
68
69 hold on ; plot(pas,eval_opt) ; axis([0.05 0.11 0 7e-08])
70 xlabel('PAS DE DISCRETISATION') ; ylabel('MAXIMUM DE L''ERREUR
     RELATIVE')
71
72 str1 = {'CETTE METHODE EST D''ORDRE :', num2str(degre_interp)}
73 uicontrol('Style','text','Position',[260 80 150 60],...
74           'BackgroundColor',[1 1 1],'String',str1);
```

s(p = 1) : normr = 9.3851e-09 ; s(p = 2) : normr = 7.4149e-10 ; s(p = 3) : normr = 1.6060e-11 ; s(p = 4) : normr = 9.9262e-24 ; s(p = 5) : normr = 1.9921e-23 ; s(p = 6) : normr = 2.8892e-23.

D'où le choix p = 4, qu'on retrouve également avec une démarche similaire en posant err_iterpolation = max(abs(evalp - max_err)) et [min_err_interp, degre_interp] = min(err_iterpolation). Il en ressort que degre_interp = 4. Avec un raisonnement analogue on peut aisément obtenir l'ordre des méthodes d'Euler et de Heun.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

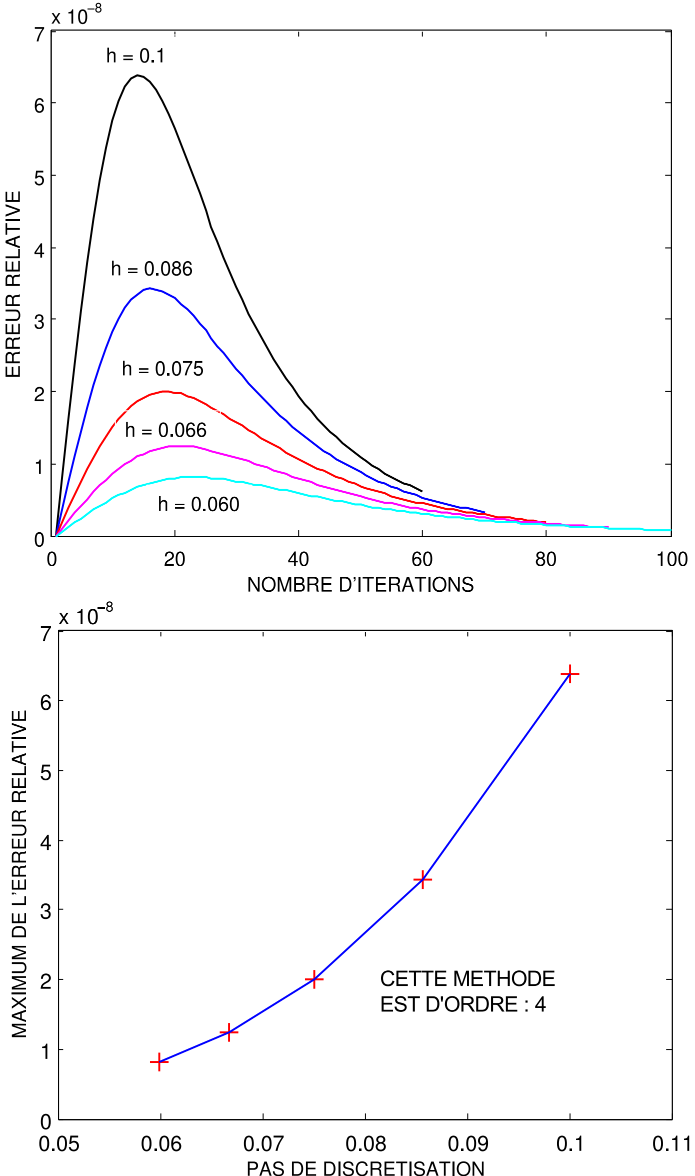

FIGURE 4.2: Évolution de l'erreur relative en fonction du pas de discrétisation

4.1.5 Équations différentielles d'ordre n

N'importe quelle équation différentielle d'ordre n peut être ramenée à un systèmes de n équations du premier ordre. Nous allons illustrer ceci par un exemple. Soit l'equation différentielle du second ordre suivante :

$$\begin{cases} y'' = \dfrac{t\,y'}{2} - y + 3 \\ y(0) = 1 \quad \text{et} \quad y'(0) = 0 \end{cases} \tag{4.13}$$

Posons $u_1(t) = y(t)$ et $u_2(t) = y'(t)$ il vient $u_2'(t) = \dfrac{t\,u_2(t)}{2} - u_1(t) + 3$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❷ ☞ ®

1. Résoudre numériquement, au moyen de la méthode de Euler, l'équation différentielle du second ordre (4.13).
2. Afficher sur la même figure, la solution numérique et la solution exacte, donnée par : $t^2 + 1$.
3. Afficher le graphe de la dérivée de la fonction $y(t)$.
4. Appliquer le même code Matlab pour résoudre l'équation différentielle du troisième ordre suivante :

$$\begin{cases} y^{'''}(t) = 0.001\,\left(y^{''}(t) + (1 - y(t)^2)\right) \times y^{'}(t) + sin(t) \\ y(0) = 1, \quad y^{'}(0) = 5, \quad \text{et} \quad y^{''}(0) = 0 \end{cases} \tag{4.14}$$

5. Afficher sur la même figure, la solution $y(t)$ ainsi que ses première et deuxième dérivée.

La résolution numérique de ce système d'équations, par la méthode d'Euler, est donnée par le code Matlab ci-dessous.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; clc ;
close all ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 15/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT LA
% RESOLUTION D'EQUATIONS DIFFERENTIELLES DU SECOND ORDRE

a = 0 ; b = 1 ; n = 64 ; h = (b-a)/n ; t = a :h: b ;

fun = inline('[u(2), (t*u(2))/2 - u(1) + 3]','t','u');
% ou bien fun = @(t,u)[u(2), (t*u(2))/2 - u(1) + 3];
u = [1 0] ;

for i=1:n

  u(i+1,:) = u(i,:) + h*feval(fun,t(i),u(i,:));

end

funex = t.^2 + 1 ; figure('color',[1 1 1])
plot(t, u(:,1),'o','MarkerSize',8) ; hold on
plot(t,funex,'r','LineWidth',1.5) ; hold on
axis([-0.05 1.1 0.8 2.1])
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
23 ih1 = legend('SOLUTION NUMERIQUE','SOLUTION EXACTE');
24 set(ih1,'Interpreter','none','Location','NorthWest','Box','on',...
25     'Color','none') ; xlabel('t') ; ylabel('y(t)')
26
27 figure('color',[1 1 1]) ; plot(t,u(:,2),'or','MarkerSize',8)
28 axis([-0.05 1.1 0 2.2]) ; ih2 = legend('FONCTION DERIVEE');
29 set(ih2,'Interpreter','none','Location','SouthEast','Box','on',...
30     'Color','none') ; xlabel('t') ; ylabel('dy/dt')
```

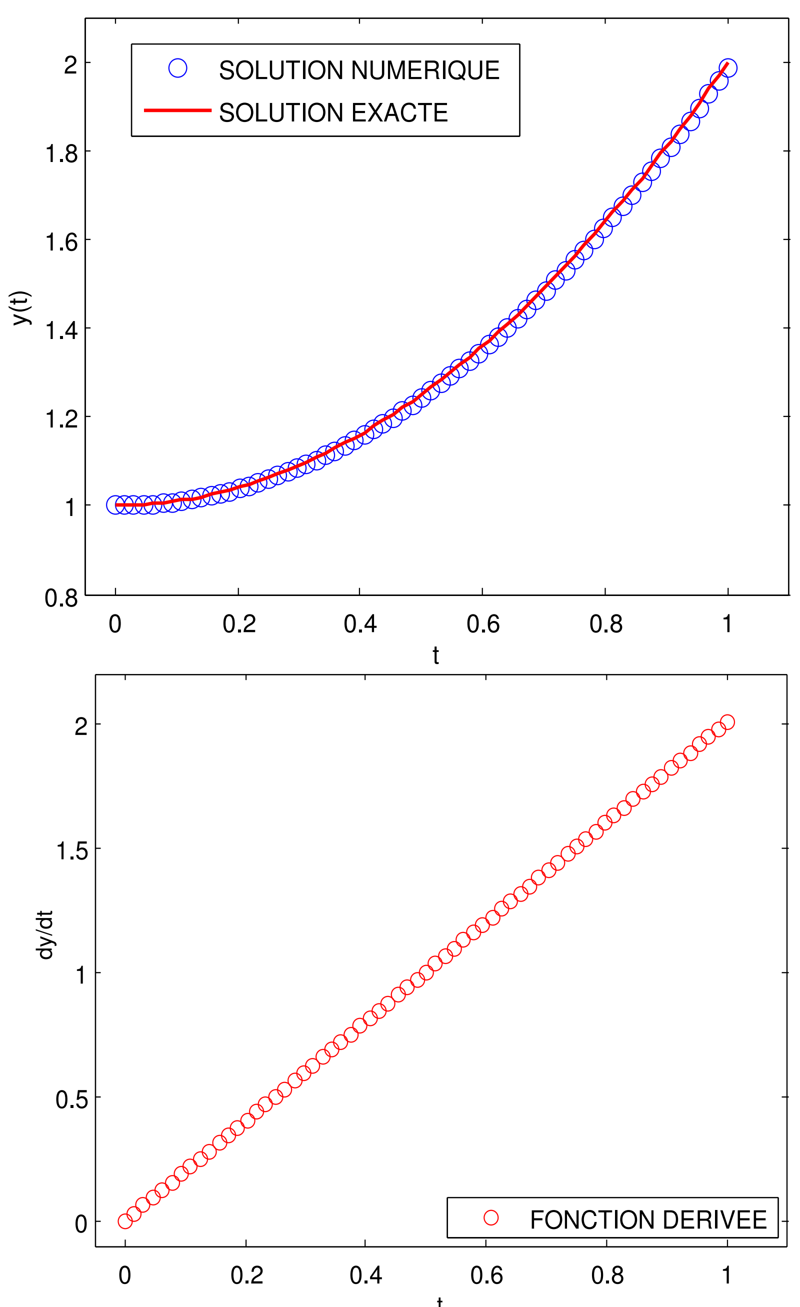

FIGURE 4.3: Solution exacte et solution numérique obtenue par méthode `Euler`

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Nous allons appliquer le même code pour résoudre l'équation différentielle du troisième ordre suivante :

$$\begin{cases} y^{'''}(t) = 0.001\,\left(y^{''}(t) + (1 - y(t)^2)\right) \times y^{'}(t) + sin(t) \\ y(0) = 1, \quad y^{'}(0) = 5, \quad \text{et} \quad y^{''}(0) = 0 \end{cases} \tag{4.15}$$

De la même façon que précédemment, posons $u_1(t) = y(t)$, $u_2(t) = y^{'}(t)$ et $u_3(t) = y^{''}(t) \Rightarrow y^{'''}(t) = u_3^{'}(t)$. Il en ressort que :

$$u_3^{'}(t) = 1e - 03\;(u_3(t) + (1 - u_1(t)^2)) \times u_2(t) + \sin(t)$$

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ;
clc ;
close all ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 15/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT LA
% RESOLUTION D'EQUATIONS DIFFERENTIELLES DU 3EMME ORDRE

a = 0 ; b = 10 ; n = 64 ; h = (b-a)/n ; t = a :h: b ;
fun = @(t,u)[u(2), u(3), 0.001*(u(3) + (1 - u(1).^2))*u(2) + sin(t)];
u = [1 5 0] ;

for i=1:n

  u(i+1,:) = u(i,:) + h*feval(fun,t(i),u(i,:));

end

figure('color',[1 1 1])

plot(t, u(:,1),'-o','MarkerSize',6,'LineWidth',1) ; hold on ;

plot(t, u(:,2),'r-o','MarkerSize',6,'LineWidth',1) ; hold on

plot(t, u(:,3),'k-o','MarkerSize',6,'LineWidth',1) ; hold on

axis([-1/2 10.5 -12 32])
ih1 = legend('y(t)','y_prime','dy_2primes');

set(ih1,'Interpreter','none','Location','NorthWest','Box','off',...
    'Color','none') ; xlabel('TEMPS') ; ylabel('SOLUTION NUMERIQUE')
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

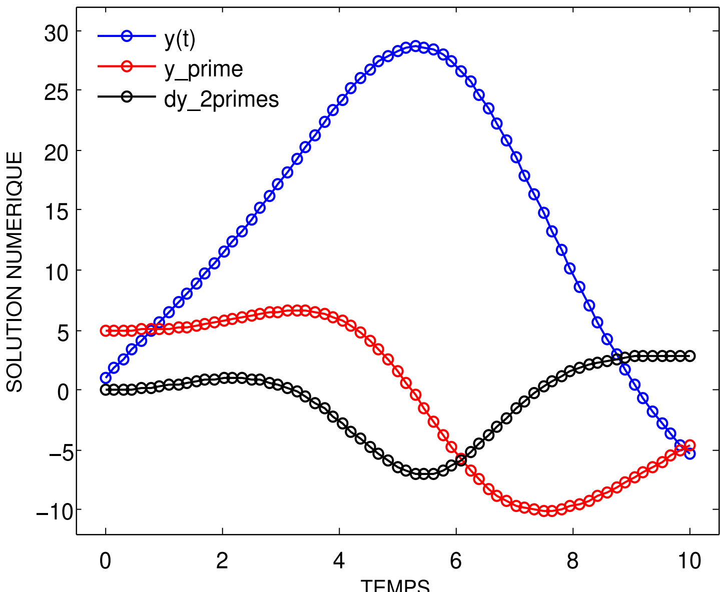

FIGURE 4.4: Équation différentielle du troisième ordre résolue par la méthode de `Euler`

4.2 Au moyen de routines Matlab

Matlab comprend un certain nombre de *solveurs* destinés à la résolution d'équations différentielles. Les plus utilisés sont `ode23` et `ode45`, qui sont basés sur la méthode de *Runge-Kutta* explicite à un pas. Le *solveur* `ode113`, utilise la méthode de *Adams-Bashforth-Moulton* multi-pas. Les autres *solveurs* sont `ode15s, ode23s, ode23t, ode23tb`. Ils ont tous la même syntaxe :

```
[t, y] = solveur(eqs, [ti ; tf], yinit, opts).
```

Cette syntaxe renvoie la solution `y` au temps `t`. L'argument `eqs` est le système d'équations différentielles. Ce système peut être défini de plusieurs manières. Soit à travers un fichier *M-file*, dans ce cas on doit rajouter l'identifiant `@eqs`. Soit à travers une commande `inline` ou bien au moyen de la *fonction anonyme*. Ce système d'équations différentielles est résolu sur l'intervalle `[ti ; tf]`, avec les conditions initiales `yinit = [y(ti) ; y(tf)]`. L'argument d'entrée `opts`, de type *srtucture*, compte les options d'optimisation indiquées dans `odeset`, sa syntaxe est donnée par :

```
opts = odeset('Property1', value1, 'Property2', value2, ...).
```

Ainsi, chaque propriété est suivie de sa valeur. À titre illustratif, la propriété `('Stats','on', ...)` affiche, à la fin de l'exécution, des statistiques relatives au calcul effectué. Pour plus d'informations sur les paramètres d'optimisation, taper `help odeset`.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❸ ☞ ®

1. Résoudre symboliquement et numériquement au moyen du *solveur* ode23, l'équation différentielle du second ordre suivante :

$$\begin{cases} y''(t) + 3\,y = 4\,\sin(t) + \exp(-t) \\ y(0) = 1, \quad y'(0) = 1 \end{cases} \tag{4.16}$$

2. Afficher sur la même figure, la solution numérique et la solution analytique.

Voici le script Matlab

Script Matlab

```
clear all ;
clc ;
close all ;

%%%%%%%%%%%%%%%%%%%%%%%%%% SOLUTION SYMBOLIQUE %%%%%%%%%%%%%%%%%%%%%%%%%%
syms t

ySym = dsolve('D2y + 3*y = 4*sin(t) + exp(-t)','y(0) = 1','Dy(0) = 1'
    ,'t') ;

%%%%%%%%%%%%%%%%%%%%%%%%%% SOLUTION NUMERIQUE %%%%%%%%%%%%%%%%%%%%%%%%%%
a = 0 ; b = 10 ; n = 200 ; h = (b-a)/n ; tn = a :h: b ;
y = [1 ; 1] ;

eqs = @(tn,y) [y(2); - 3*y(1) + 4*sin(tn) + exp(-tn)] ;

opts = odeset('Stats','on','RelTol', 1e-3) ;

[tn, ysol] = ode23(eqs, [tn(1) ; tn(end)], y, opts) ; % SOLVEUR ode23

figure('color', [1 1 1]) ;

plot(tn, ysol(:,1),'o-','LineWidth',1) ; hold  on ;

plot(tn, double(subs(ySym, t, tn)),'o-r','LineWidth',1)

xlabel('t','FontSize',12) ; ylabel('y(t)','FontSize',12) ;

ih1 = legend('SOLUTION NUMERIQUE','SOLUTION ANALYTIQUE') ;
set(ih1,'Interpreter','none','Location','NorthWest','Box','off','
    Color','none')
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

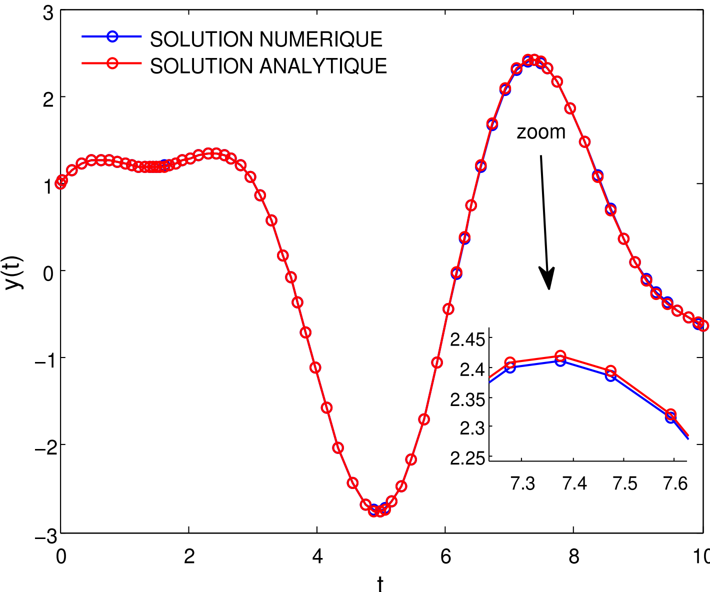

FIGURE 4.5: Comparaison entre la solution analytique et la solution numérique générée par le solveur `ode23`

Un autre façon de faire est de créer des fonctions imbriquées dans un fichier *M-file*, selon la syntaxe suivante :

```
function [] = eqs_sol(t,y)

tsol = [t(1) ; t(end)] ;

opts = odeset('Stats','on','RelTol', 1e-3) ;

[t , ysol] = ode23(@eqs, tsol, y, opts)  ;

plot(t,ysol(:,1))

function dydt = eqs(t, y)

dydt = [y(2) ; - 3*y(1) + 4*sin(t) + exp(-t)] ;

end
end
```

Qu'il faudra ensuite appeler, en tapant dans la fenêtre des commandes : `eqs_sol(t, [1 ; 1])`. Cette fonction accepte deux arguments en entrée, le vecteur `t` et les conditions initiales `[1 ; 1]`. Le fichier doit être sauvegardé sous le nom `eqs_sol.m`. Nous allons désormais résoudre une équation différentielle du second ordre avec paramètre variable, noté α :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❹ ☞ ®

1. Résoudre numériquement au moyen du *solveur* ode45, l'équation différentielle du second ordre suivante :

$$\begin{cases} y''(t) - \alpha\,(1 - y^2)\,y' = -y \\ y(0) = 1, \quad y'(0) = 0 \end{cases} \tag{4.17}$$

2. Afficher sur la même figure, la solution numérique pour différentes valeurs de α. Il en est de même pour les fonctions dérivées. Avec, le paramètre α qui prend ses valeurs de 1.5 à 4 par pas de 0.5.

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```matlab
clc ;
clear all ;
close all ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 1 ; b = 25 ; n = 100 ; h = (b-a)/n ; t = a :h: b ;

y = [1 ; 0] ; alpha = [1.5 ; 2.0 ; 2.50 ; 3.0 ; 3.5 ; 4.0] ;
col = {'o-k','o-m','-co','o-g','o-','o-r'} ;

for ik = 1 : numel(alpha)

eqs = @(t,y) [y(2); alpha(ik)*(1 - y(1).^2)*y(2) - y(1)] ;

opts = odeset('Stats','on','RelTol', 1e-3) ;

[t, ySol] = ode45(eqs, [t(1) ; t(end)], y, opts) ;

ysol = squeeze(ySol(:,1)) ; ysolDer = squeeze(ySol(:,2)) ;

figure(1) ; plot(t, ysol, col{ik}) ; hold on ; % SOLUTIONS

xlabel('t','FontSize',12) ; ylabel('y(t)','FontSize',12)

figure(2) ; plot(t, ysolDer, col{ik}) ; hold on ; % DERIVEES
xlabel('t','FontSize',12) ; ylabel('y^{''}(t)','FontSize',12)

end
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

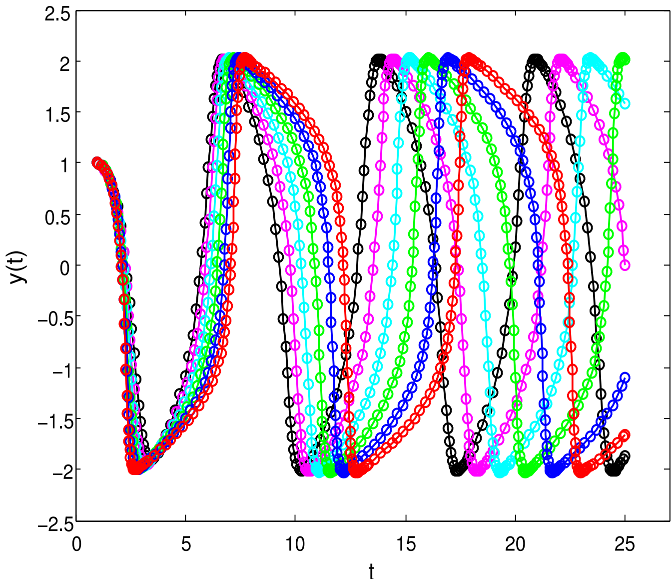

FIGURE 4.6: Solution numérique $y(t)$ pour différentes valeurs de α

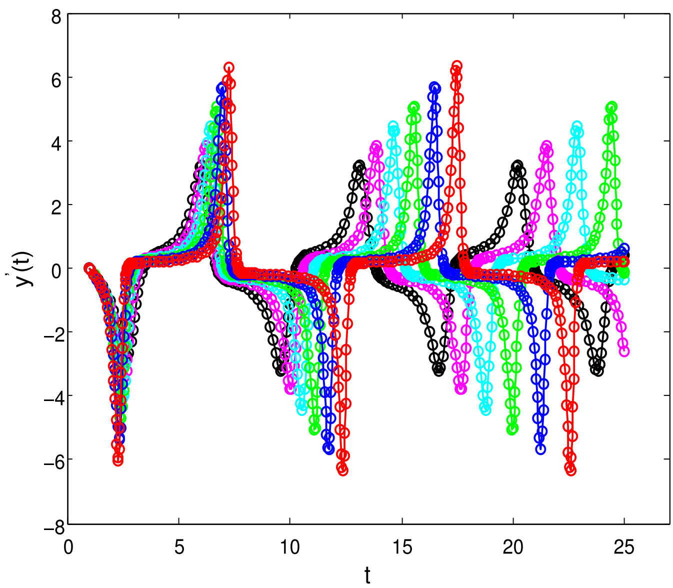

FIGURE 4.7: Dérivée première de la solution numérique $y(t)$ pour différentes valeurs de α

Exercice ❶ ☞ Ⓢ

1. Résoudre numériquement les équations différentielles suivantes :

$$\begin{cases} y' + y\,\cos(x) = \cos(x), \quad y(0) = 1 \\ y' = x + \cos(y), \quad y(0) = 0 \\ y'' - y' - 2y = 2\,x\exp(-x) + x^2, \quad y(0) = 0, y'(0) = 1 \\ y'' + y = 3\,x^2 - 4\sin(x), \quad y(0) = 0, y'(0) = 1 \\ y''' + y' = x, \quad y(0) = 0, y'(0) = 1, y''(0) = 1 \\ y'''' - y = 8\,\exp(x), \quad y(0) = 0, y'(0) = 2, y''(0) = 4, y'''(0) = 6 \end{cases}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

5 Calcul formel

Dérivées, extremums, dérivées partielles, systèmes d'équations et intégrales

Sommaire

4.1 Dérivée d'une Fonction 85
4.2 Point d'inflexion d'une fonction 90
4.3 Extremums d'une fonction 93
4.4 Dérivées partielles 95
4.5 Résolution formelle des équations et système d'équations différentielles 99
4.6 Résolution formelle d'équations et de système d'équations 104
4.7 Résolution formelle des intégrales simples et multiples 110

Introduction

Dans ce chapitre, nous allons réaliser sous Matlab, des calculs symboliques (intégration, dérivation, résolution d'équations ou un système d'équations, recherche des racines, ... etc) afin d'obtenir des expressions mathématiques comme résultat. La collection de fonctions Matlab permettant la réalisation de ce type de calcul se trouve dans la boite à outil `Symbolic Math Toolbox`.

5.1 Dérivée d'une Fonction

Soit la fonction d'une variable $f : \mathbb{I} \longrightarrow \mathbb{R}$. On définit sa fonction dérivée par l'expression :

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{5.1}$$

Pour un point x_0 apparentant à l'intervalle de définition de f, le nombre $f^{'}(x_0)$ exprime la pente, au point $(x_0, f(x_0))$, de la droite tangente à la courbe $y = f(x)$. Le rapport :

$$f^{'}(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \tag{5.2}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

vaut la pente de la droite passant par les points $(x_0, f(x_0))$ et $(x_0+\Delta x, f(x_0+\Delta x))$. Il en ressort que l'équation de la droite tangente à la courbe $y = f(x)$ au point (x_0, y_0) est :

$$y - y_0 = f'(x_0)(x - x_0) \Longrightarrow y = y_0 + f'(x_0)(x - x_0) \tag{5.3}$$

Cette approximation est d'autant plus vrai que $\Delta x \longrightarrow 0$. Lorsque la limite de (5.2) existe, on dira que f est dérivable en x_0. Dans le cas où la fonction f est dérivable pour tout $x \in \mathbb{I}$, on dira qu'elle est dérivable sur $\mathbb{I}$ et on aura par conséquent la fonction $f'(x)$. Cette dernière peut également être dérivable et on aura $f''(x)$ ...

Exercice ❶ ☞ ®

1. Déterminer l'expression symbolique de la dérivée de la fonction :

$$\begin{cases} f(x) = \cos(x) + \sqrt{x^2+1} \\ \text{Avec,} \quad x \in [-3, 3] \end{cases} \tag{5.4}$$

2. Tracer, sur la même figure, la fonction $f(x)$, sa dérivée ainsi que la droite tangente au point $(-1, f(-1))$. Annoter le graphe.
3. Tester les fonctions `diff`, `subs`, `pretty` et `plotyy`.

Voici le script Matlab :

Script Matlab

```
clc ; clear all ; close all ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
syms x real
N = 1000 ; lB = -3 ; uB = 3 ; h = (uB - lB)/N ;
funsym = cos(x) + sqrt(x.^2 + 1) ;
dfunsym = diff(funsym, x) ; pretty(dfunsym) ; xvar = lB :h: uB ;
dfun = subs(dfunsym, x, xvar) ; fun = subs(funsym, x, xvar) ;

delta = subs(funsym, x, -1) + subs(dfunsym, x, -1).*(xvar + 1) ;

figure('Color',[1 1 1]) ;
[ax, ih1, ih2] = plotyy(xvar, fun, xvar, dfun,'plot') ;
set(get(ax(1),'Ylabel'),'interprete','latex','String', ['$$f(x)$$'],'
   FontSize',12);
set(get(ax(2),'Ylabel'),'interprete','latex','String', ['$$\frac{df(x
   )}{dx}$$'],'FontSize',12);
xlabel('\fontsize{12}\fontname{Tex} x ') ; hold on ;
plot(xvar, delta,'k--') ;
legend('f(x)','df(x)/dx','tangente au point (-1, f(-1))')
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Ainsi, l'expression symbolique de la dérivée, générée par la fonction `pretty` est :

```
>> pretty(dfunsym)
                                              x
                          - sin(x) + ------------
                                           2    1/2
                                         (x  + 1)
```

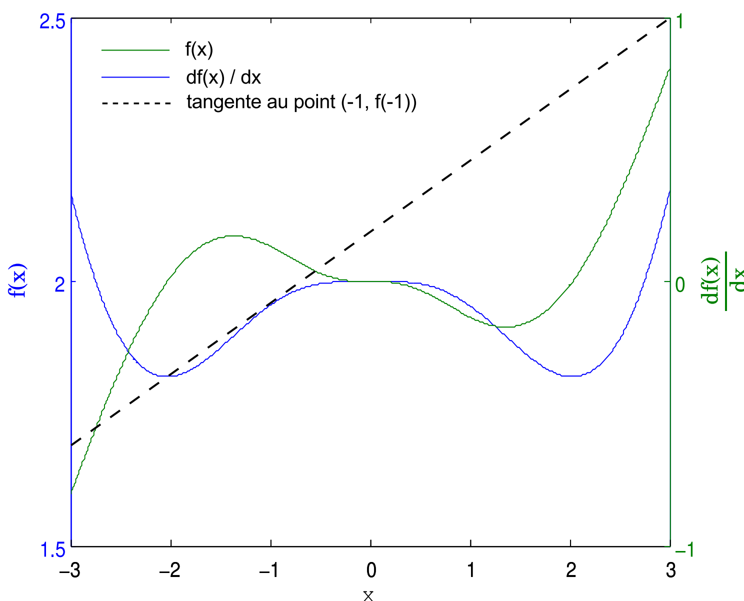

FIGURE 5.1: Figure générée par le code Matlab ci-dessus

Il est intéressant de remarquer qu'à chaque fois que la dérivée est négative, la fonction est décroissante et inversement, dans les régions où la dérivée est positive, la fonction est croissante. Les expressions et variables formelles sous Matlab sont créées comme des chaînes de caractères, par le biais de la commande `syms`. Comme vous pouvez le voir, on peut écrire plusieurs instructions sur une même ligne, en les séparant d'un point-virgule ou d'une virgule. Les commandes `clear all` et `clc` permettent d'effacer respectivement les variables existantes en mémoire et celles présentes dans la fenêtre des commandes. La fonction `diff` détermine l'expression symbolique de la fonction dérivée. Sa syntaxe usuelle est `diff(expr, n)`, qui détermine la `n`ième dérivées de l'expression formelle `expr`. La fonction `subs` réalise des substitutions dans des expressions symboliques. Dans le cas où l'on désire obtenir des informations sur une fonction Matlab donnée, taper son nom sur la fenêtre des commandes. Par exemple, pour avoir des détails sur la fonction `pretty`, on écrira :

```
>> help pretty
 --- help for sym/pretty ---
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
3
4  PRETTY Pretty print a symbolic expression.
5     PRETTY(S) prints the symbolic expression S in a format that
6     resembles type-set mathematics.
7
8     See also sym/subexpr, sym/latex, sym/ccode.
```

Pour avoir l'aide en mode graphique, taper

```
1  >> doc pretty
2  Symbolic Math Toolbox
3
4  pretty
5  Pretty-print symbolic expressions
6  Syntax
7
8  pretty(S)
9  Description
10
11 The pretty function prints symbolic output in a format that resembles
       typeset mathematics.
12
13 pretty(S) prettyprints the symbolic matrix S using the default line
      width of 79.
14 Examples
15
16 The following statements
17 A = sym(pascal(2))
18 B = eig(A)
19 pretty(B)
20
21 return
22 A =
23 [1, 1]
24 [1, 2]
25
26 B =
27 [3/2+1/2*5^(1/2)]
28 [3/2-1/2*5^(1/2)]
29
30                    [            1/2 ]
31                    [ 3/2 + 1/2 5    ]
32                    [                ]
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
33                  [          1/2 ]
34                  [ 3/2 - 1/2 5    ]
```

Par ailleurs, la fonction `syms` ne permet pas de créer des constantes symboliques. On utilisera à cet effet la fonction `sym`. Par exemple pour définir $\sqrt{(\pi)}$, on pourra écrire :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ; close all ;
2  th = sym('sqrt(pi)') ; res = 2*th ; pretty(res)
3
4  >> ans =
5             1/2
6          2 pi
```

La boite à outil `Symbolic Math Toolbox`, dispose de toute une panoplie de fonctions permettant la manipulation des expressions symboliques. On citera à titre illustratif :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2
3  syms x y real
4
5  fun = exp((x + y)^2) ; exp_fun = expand(fun)
6
7  fun_xy = [x^2 + y^2, x^3+y^3] ; fact_fun_xy = factor(fun_xy)
8
9  fun_x = (x^2 + exp(x))*(x+1) ; col_fun_x = collect(fun_x)
10
11 >> exp_fun =
12
13            exp(x^2)*exp(x*y)^2*exp(y^2)
14
15 >> fact_fun_xy =
16
17            [x^2+y^2, (x+y)*(x^2-x*y+y^2)]
18
19 >> col_fun_x =
20
21            x^3+x^2+exp(x)*x+exp(x)
22
23 % en utilisant la commande pretty on aura :
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
24
25 >> pretty(exp_fun)
26
27                 2          2         2
28           exp(x) exp(x y)   exp(y)
29
30 >> pretty(fun_xy)
31
32           [ 2     2      3     3]
33           [x   + y      x   + y ]
34
35 >> pretty(fun_x)
36
37               2
38           (x + exp(x)) (x + 1)
```

La fonction expand réalise un développement en puissance de l'expression en input. Les fonctions `factor` et `collect` permettent respectivement la factorisation et le réarrangement selon une structure polynomiales des expressions mathématiques données en input.

1. Déterminer les expressions symboliques des dérivées des fonctions suivantes :

$$\begin{cases} f_1(x) = 2 - x^3 + \sqrt{x^2+1}, & x \in [-3, 3] \\ f_2(x) = x \times cos(x), & x \in [-\pi, \pi] \\ f_3(x) = \sin(x) + x^2, & x \in [-4, 4] \end{cases} \tag{5.5}$$

2. Pour chaque fonction, Tracer son graphe, celui de sa dérivée et la droite tangente au point $(-1, f(-1))$. Annoter les graphes.

5.2 Point d'inflexion d'une fonction

On rappelle que le point d'inflexion d'une fonction est déterminé en annulant sa deuxième dérivée. De plus, il faut que cette dernière change de signe de part et d'autre de ce point.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❷ ☞ ®

1. Trouver le point d'inflexion de la fonction :

$$f(x) = \frac{1}{\frac{1}{3} + \exp(x)}, \quad x \in [-4, 4] \tag{5.6}$$

2. Tracer, sur une même figure, le graphe de la fonction, celui de sa dérivée second, son point d'inflexion et la droite tangente au point d'inflexion.

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ; close all ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @ 20/10/2015 Samir KENOUCHE, PROGRAMMATION Matlab DEDIEE AU
% CALCUL FORMEL AVEC LA BOITE A OUTILS Symbolic Math Toolbox
syms x real

n = 500 ; lB = -4 ; uB = 4 ; step = (uB - lB)/n ;
xvar = lB :step: uB ;

fx = 1/(1/3 + exp(x)) ; dfx = diff(fx, x) ; % PREMIERE DERIVEE
valdfx = subs(dfx, x, 1/2) ; % SUBSTITUTION DE x PAR 1/2
ddfx = diff(fx, x, 2) ; valddfx = subs(ddfx, x, xvar)  ; % 2EMME
   DERIVEE
pt_infl = solve(ddfx) ; % CALCUL ddfx = 0
ddfxVal = [pt_infl subs(ddfx, x, pt_infl)] ;

drt = double(subs(fx, x, pt_infl)) + double(subs(dfx, x, pt_infl)).*(
   xvar - double(pt_infl)) ;
% TANGENTE AU POINT D'INFLEXION
%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%
figure('color',[1 1 1]) ; ezplot(fx, [-4 4]) ; hold on ; plot(xvar,
   valddfx,'r') ; hold on ;
plot(xvar, drt,'g','LineWidth',1) ; hold on ;
plot(double(pt_infl), double(subs(fx,pt_infl)),'ro','MarkerSize',10,
   'LineWidth',1) ; hold on
plot(double(pt_infl), double(subs(fx,pt_infl)),'r+','MarkerSize',10,
   'LineWidth',1) ; text(-3.5,1.5,'Point d''inflexion') ;
xlabel('x','FontSize',12) ; ylabel('f(x)','FontSize',12) ; hold on ;
line(double(pt_infl).*ones(1, length(xvar)), double(subs(fx, x, xvar)
   ),'LineStyle','-.','Color','k','LineWidth',1) ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
25 legend('f(x)','f^{''''}(x)','Tangente') ; axis([-4.5 4.5 -1/2 3])
```

La variable de sortie `ddfxVal = [-log(2), 0]` renvoie l'abscisse du point d'inflexion ainsi que la valeur de la second dérivée évaluée en ce point.

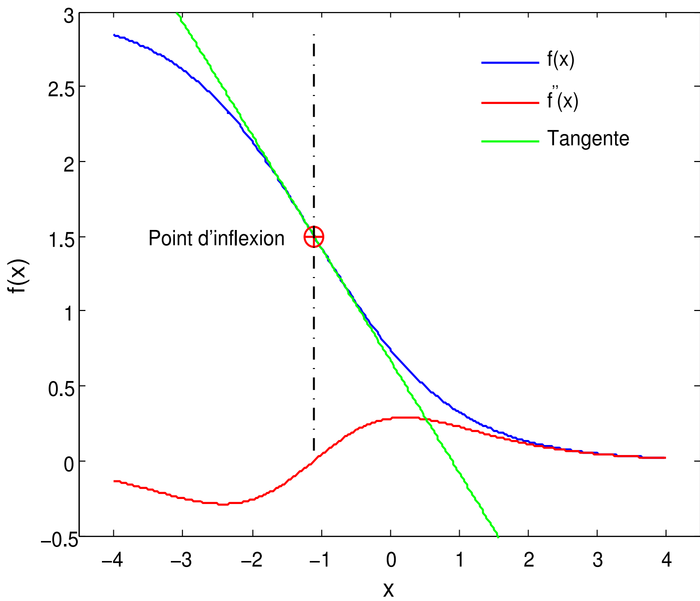

FIGURE 5.2: Figure générée par le code Matlab ci-dessus

La commande `ezplot` trace le graphe des expressions symboliques sur un intervalle $[-2\pi, 2\pi]$ par défaut. Néanmoins, on peut spécifier un autre intervalle avec la syntaxe usuelle `ezplot(fun,[xmin, xmax, ymin, ymax]`. À partir de la figure ci-dessus, on voit clairement le changement de signe de $f''(x)$ de part et d'autre du point d'inflexion. Le changement de signe de la dérivée seconde correspond à un changement de convexité de la fonction. Il est intéressant de voir qu'au point d'inflexion, la courbe de $f(x)$ traverse sa tangente.

Exercice ❷ ☞ Ⓢ

1. Trouver le point d'inflexion des fonctions suivantes :

$$\begin{cases} f_1(x) = \dfrac{4}{x-1}, \quad x \in [-6, 6] \\ f_2(x) = (\exp(x) - 5) \times (\exp(x) + 1), \quad x \in [-5, 5] \\ f_3(x) = \dfrac{\log(x^2) + 1}{2\,x}, \quad x \in [-4, 4] \\ f_4(x) = (2\,x^2 + 2\,x - 1) \times \exp(-2\,x), \quad x \in [-3, 3] \end{cases} \tag{5.7}$$

2. Tracer, sur une même figure, le graphe de la fonction, celui de sa dérivée second, son point d'inflexion et la droite tangente au point d'inflexion.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

5.3 Extremums d'une fonction

Nous rappelons que la dérivée traduit les variations, croissance ($f'(x) > 0$) et décroissance ($f'(x) < 0$), d'une fonction. Une dérivée nulle, en un point x_0, est caractérisée par une tangente horizontale et donc la fonction présente un extrémum en ce point. Ainsi, les conditions $f'(x_0) = 0$ et $f''(x_0) > 0$ sont suffisantes pour que la fonction $f(x)$ admette un minimum. Les conditions $f'(x_0) = 0$ et $f''(x_0) < 0$ prouvent l'existence d'un maximum.

Exercice ❸ ☞ ®

1. Trouver, suivants deux approches différentes, les extremums de la fonction suivante :

$$\begin{cases} f(x) = (x-1) \times \exp(-x^2 + 2\,x + 1) \\ \text{Avec}, \quad x \in [-2.5, 5] \end{cases} \tag{5.8}$$

2. Tracer, sur la même figure, la fonction $f(x)$ et les extremums trouvés.
3. Tester les fonctions `max`, `min` et `find`.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ; close all ;
%%%%%%%%%%%%%% RECHERCHE DES EXTREMUMS D'UNE FONCTION %%%%%%%%%%%%
syms x y real
fun = (x-1)*exp(-x^2 + 2*x + 1) ;  dfun = diff(fun, x) ;
rootn = solve(dfun) ; root1 = double(rootn(1)) ;
root2 = double(rootn(2)) ;

val1 = subs(diff(dfun),x,root1) ; val2 = subs(diff(dfun),x,root2) ;
abscisses = [root1 root2] ; val = [val1 val2] ;

for n = 1:length(val)

    if val(n) < 0

        maximum = abscisses(n)
    else
        minimum = abscisses(n)
    end
end

figure('color',[1 1 1]) ; ezplot(fun, [-5/2, 4]) ; hold on ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
22 plot(minimum, double(subs(fun,minimum)),'ro','MarkerSize',10, '
   LineWidth',1) ; hold on
23 plot(minimum, double(subs(fun,minimum)),'r+','MarkerSize',10, '
   LineWidth',1) ; hold on
24 plot(maximum, double(subs(fun,maximum)),'ro','MarkerSize',10, '
   LineWidth',1) ;
25 plot(maximum, double(subs(fun,maximum)),'r+','MarkerSize',10, '
   LineWidth',1) ;
26 text(-1.0,-3.2,'Minimum') ; text(1/2,3,'Maximum') ;
27 xlabel('x','Fontsize',12) ; ylabel('f(x)','Fontsize',12)
```

Les extremums trouvés sont :

```
1 >> maximum =
2              1.7071
3 >> minimum =
4              0.2929
```

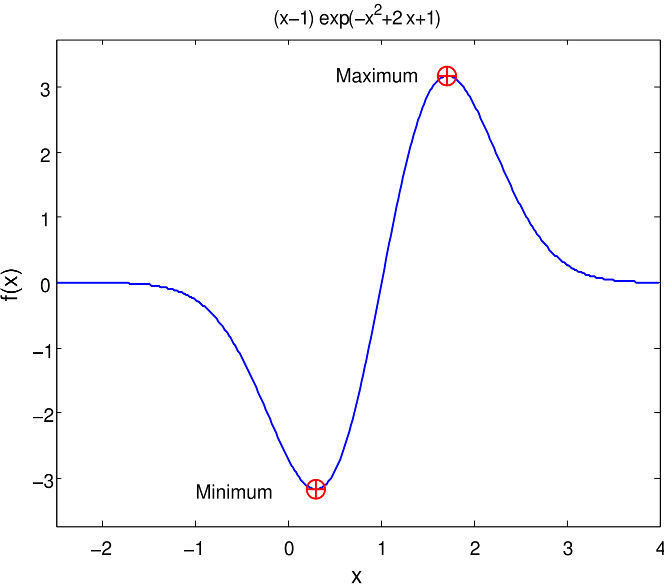

FIGURE 5.3: Figure générée par le code Matlab ci-dessus

Ces extremums peuvent également être obtenus par les instructions suivantes :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all; clc ;
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
3  lowerBound = -4 ; upperBound = 4 ; n = 1000 ;
4  fx = @(x) (x-1).*exp(- x.^2 + 2.*x + 1) ;
5  step=(upperBound - lowerBound)/n ; x = lowerBound :step: upperBound ;
6
7  idxMin = find(fx(x) == min(fx(x))) ; xMin2 = x(idxMin) ;
8  idxMax = find(fx(x) == max(fx(x))) ; xMax2 = x(idxMax) ;
```

La fonction `find` renvoie l'indice pour lequel la fonction `fx` est minimum pour `idxMin` et l'indice correspondant au maximum pour `idxMax`. Les fonctions `min` et `max` renvoient respectivement le minimum et le maximum des valeurs de la fonction `fx`. Les arguments de sortie générés par ce script sont :

```
1  >> xMin2 =
2            0.2960
3  >> xMax2 =
4            1.7040
```

Exercice ❸ ☞ Ⓢ

1. Trouver les extremums des fonctions suivantes :

$$\begin{cases} f_1(x) = \exp(-\frac{x}{10}) \times \cos(x), \quad x \in [-5,5] \\ f_2(x) = x^3 - 2\,\sin(x) + 2, \quad x \in [-1,1] \\ f_3(x) = (x+1) \times (x-1) \times (x-3), \quad x \in [-1,3] \\ f_4(x) = \sin(x) + x^2 + 2, \quad x \in [-2,2] \\ f_5(x) = \dfrac{(x^3 + x^2 + 1)}{(x^2 - 3)}, \quad x \in [0,3] \end{cases} \tag{5.9}$$

2. Tester les commandes `fminsearch`, `fmincon` et les fonctions `max`, `min`, `find`.

5.4 Dérivées partielles

Soit la fonction à deux variables $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$. Les dérivées partielles $f(x,y)$, par rapport à x et à y, sont les fonctions $\dfrac{\partial f(x,y)}{\partial x}$ et $\dfrac{\partial f(x,y)}{\partial y}$ définies par :

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x} \tag{5.10}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{\Delta y \to 0} \frac{f(x, y+\Delta y) - f(x,y)}{\Delta y} \tag{5.11}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Ainsi, on dérive par rapport à une variable en considérant l'autre comme constante. Très souvent cette dernière, est mise en indice. Les conditions d'existence des extremums sont analogues à celles d'une fonction à une variable.

Exercice ❹ ☞ ®

1. Déterminer les expressions symboliques des seconds dérivées partielles de la fonction :

$$f(x,y) = (x^2+y^2)\times\sin(x) + (x^2+y^2)\times\cos(y) \qquad (5.12)$$

2. Tracer le graphe de $f(x,y)$, de $\dfrac{\delta^2 f(x,y)}{\delta x^2}$ et de $\dfrac{\delta^2 f(x,y)}{\delta y^2}$.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ; close all ;
%%%%%%%%%%%%%%%%%%%%%%% DERIVEES PARTIELLES %%%%%%%%%%%%%%%%%%%%%
syms x y real

fxy = (x^2 + y^2)*sin(x) + (x^2 + y^2)*cos(y) ;

dfx = diff(fxy, x, 2) ; % 2EME DERIVEE PARTIELLE PAR RAPPORT A x
dfy = diff(fxy, y, 2) ; % 2EME DERIVEE PARTIELLE PAR RAPPORT A y

pretty(dfx)
pretty(dfy)

%%%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%

fi1 = figure('color', [1 1 1]) ; ezsurf(fxy) ; colormap(jet) ;
xlabel('\fontsize{12}\fontname{Tex} x');
ylabel('\fontsize{12}\fontname{Tex} y') ;
title('\fontname{Tex}\fontsize{14}','interpreter','latex','String',['
   $$f(x,y)$$']) ;

fi2 = figure('color', [1 1 1]) ; ezsurf(dfx) ; colormap(jet) ;
xlabel('\fontsize{12}\fontname{Tex} x');
ylabel('\fontsize{12}\fontname{Tex} y') ;
title('\fontname{Tex}\fontsize{14}','interpreter','latex','String',['
   $$\frac{d^2f(x,y)}{dx^2}$$']) ;

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
25 fi3 = figure('color', [1 1 1]) ; ezsurf(dfy) ; colormap(jet) ;
26 xlabel('\fontsize{12}\fontname{Tex} x');
27 ylabel('\fontsize{12}\fontname{Tex} y') ;
28 title('\fontname{Tex}\fontsize{1
```

Ainsi, les expressions formelles des deuxièmes dérivées partielles par rapport à x et y sont affichées suivant :

```
1 >> pretty(dfx)
2
3                                      2    2
4           2 sin(x) + 4 x cos(x) - (x   + y) sin(x) + 2 cos(y)
5
6 >> pretty(dfy)
7                                                 2   2
8           2 sin(x) + 2 cos(y) - 4 y sin(y) - (x + y ) cos(y)
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

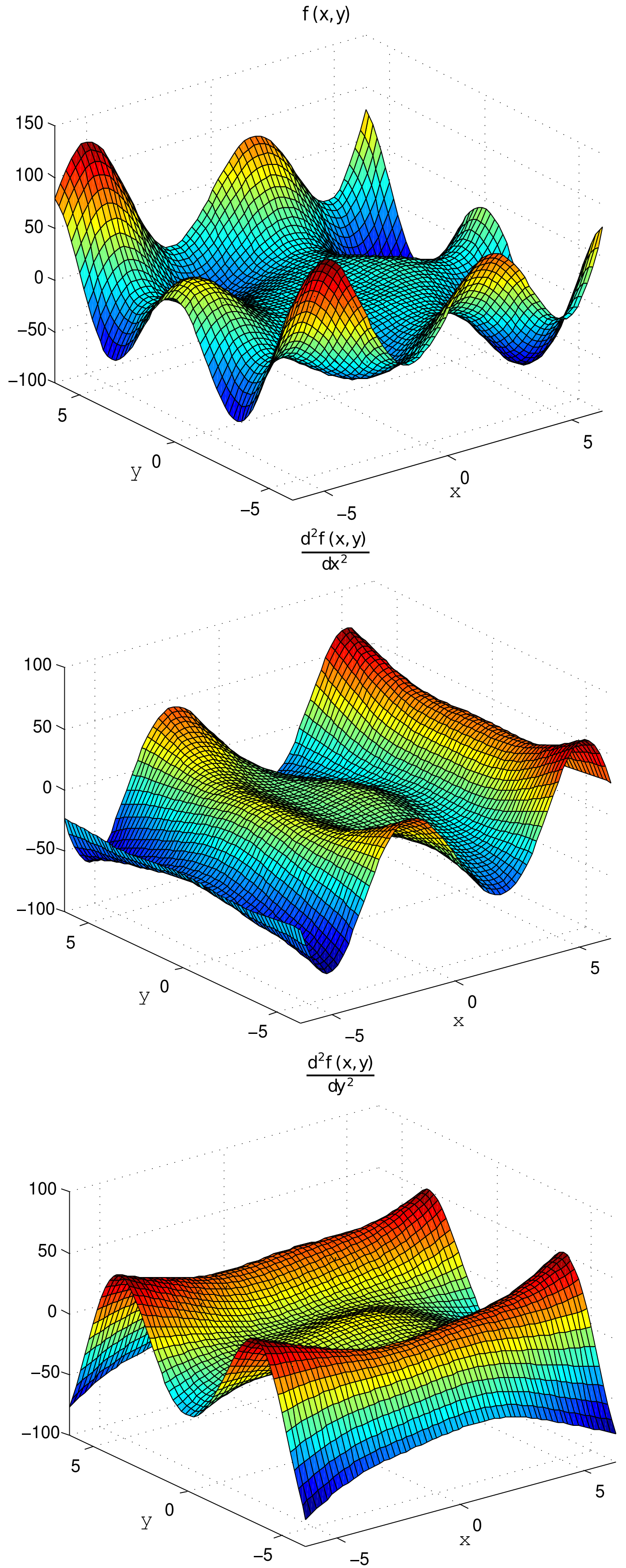

FIGURE 5.4: Figures générées par le code Matlab ci-dessus

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❹ ☞ Ⓢ

1. Déterminer les expressions formelles des dérivées partielles, des fonctions suivantes :

$$\begin{cases} f_1(x,y) = (x^2+y^2)\times\log(x+\exp(y)) \\ f_2(x,y) = \sin(x^2+y^2) \\ f_3(x,y) = \sqrt{x^2+y^2} \\ f_4(x,y,z) = (5\,x^2\,y - 3\,y + y\,z^3)\times(x^2+y^2+z^2) \\ f_5(x,y,z) = \dfrac{x\,y\,\exp(z)}{x^2+y^2+z^2} \\ f_6(x,y,z) = x\,y\,\log(z^2+1) \end{cases} \tag{5.13}$$

2. Tracer, au moyen de la commande `ezsurf`, le graphe des dérivées partielles de $f_1(x,y)$, $f_2(x,y)$ et $f_3(x,y)$. Annoter les graphes. Tester les commandes `ezmesh`, `ezsurfc`, `ezcontour` et `ezcontourf`.

5.5 Résolution formelle des équations et système d'équations différentielles

Matlab dispose de la commande `dsolve` permettant la résolution formelle d'équations différentielles ordinaires et des systèmes d'équations également. Avec cette commande, la connaissance des conditions initiales n'est pas nécessaire afin d'obtenir les solutions de ces équations. Sa syntaxe usuelle est :

```
sol = dsolve('eq1, eq2,...', 'cond1, cond2,...', 'v')
```

Ainsi, `dsolve` résout symboliquement les équations différentielles ordinaires `eq1`, `eq2` suivant la variable indépendante `v` et tenant compte des conditions initiales indiquées par `cond1`, `cond2`. Nous allons illustrer ce propos par l'exemple suivant :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2
3  expr = dsolve('Dy = 3*y + 5', 'x') ; % equation du premier ordre
4  pretty(expr)
5
6  >> pretty(expr)                     % solution renvoyee
7               - 5/3 + exp(3 x) C1
```

La lettre `D` devant une variable signifie qu'on définit la dérivée de cette variable, c'est l'opérateur *différenciation*. De la même façon, écrire `D2` se traduit par $\dfrac{d^2}{dx^2}$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Les lettres placées juste après cet opérateur, sont considérées comme les variables dépendantes. Ces variables ne sont pas obligatoirement déclarées comme symboliques. Par défaut la variable indépendante est `t`. Dans l'exemple ci-dessus, nous avons remplacé cette dernière par la variable `x`. Dans le cas où le nombre de conditions initiales est inférieur au nombre de variables dépendantes, la solutions est renvoyée avec une constante arbitraire, notée `C1, C2, ...`, etc. Pour un système d'équations, il vient :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2  [expr1, expr2] = dsolve('Dx = -x + 2', 'Dy = - 5*x + 2') ;
3  % systeme d'equations
4
5  >> pretty(expr1)
6
7                2 + exp(-t) C2   % solution 1
8
9  >> pretty(expr2)
10
11              -8 t + 5 exp(-t) C2 + C1 % solution 2
```

En intégrant des conditions initiales, on utilisera la syntaxe :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2
3  expr = dsolve('D2y = y', 'y(0) = 1','Dy(0) = 0')
4  % equation du second ordre
5
6  >> pretty(expr)
7              1/2 exp(-t) + 1/2 exp(t)
```

Signalons que les conditions aux limites ne sont pas forcément imposées à l'instant initial, ils peuvent exister pour différentes valeurs temporelles. La fonction graphique `ezplot` peut servir à tracer les expressions formelles des solutions des équations différentielles dépourvues de constantes arbitraires `C1, C2,..., etc.`

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❺ ☞ ®

1. Déterminer les expressions formelles des solutions, de l'équation différentielle suivante :

$$f^{'}(t) = \frac{1}{4}\, f(t) + \sin(t-1), \qquad f(0) = 1/2 \tag{5.14}$$

2. meme question pour le système d'équations différentielles :

$$\begin{cases} x^{'}(t) = \dfrac{1}{2} + 4\, y(t) + 1 \\ y^{'}(t) = -2\, x(t) + 3\, y(t) + 2 \\ \text{Avec,} \quad x(0) = 0 \quad \text{et} \quad y(0) = 1 \end{cases} \tag{5.15}$$

3. Tracer les solutions trouvées dans les deux cas

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ; close all
%%%%%%%%% EQUATIONS ET SYST. D'EQUATIONS DIFFERENTIELLES %%%%%%%%%
syms t

eq = dsolve('Dfx = 1/4*fx + sin(t - 1)','fx(0) = 1/2') ;
figure('color', [1 1 1]) ; ezplot(eq) ; xlim([-7 7]) ;
xlabel('t','FontSize',12) ; ylabel('f(t)','FontSize',12)
fx = simple(eq) ; pretty(fx) ;

[eq1, eq2] = dsolve('Dx = 1/2*x + 4*y + 1, Dy = -2*x + 3*y + 2','x(0)
    = 0, y(0) = 1'); % SYSTEME D'EQUATION DIFFERENTIELLE AVEC
   CONDITIONS AUX LIMITES
 pretty(eq1)
 pretty(eq2)

 borneinf = -1/2;  bornesup = 6 ;  dx = 1/500 ;
 ti =  borneinf :dx: bornesup ; xt = subs(eq1,t, ti) ; yt = subs(eq2,
   t, ti) ;

%%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%

 figure('color', [1 1 1])
 plot(ti, xt,'LineWidth',1) ; hold on ;
 plot(ti, yt,'r','LineWidth',1)
 xlabel('t','FontSize',12) ; ylabel('Solutions','FontSize',12)
 ih = legend('x(t)','y(t)') ;  axis([1 13/2 -30000 65000]) ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
24 set(ih,'Interpreter','none','Location','North','Box','off',...
25 'Color','none')
```

Parfois la fonction ezplot utilise peu de valeurs en abscisse pour tracer la fonction en question. Par conséquent, cette dernière paraitra très peu lissée. Afin de remédier à cela, on peut utiliser la fonction Subs pour substituer la variable sur l'axe des abscisses. En outre, la fonction ezplot accepte très peu d'arguments en input contrairement à la fonction plot. Si l'on souhaite établir une substitution multiple au moyen de la fonction subs, il faudra procéder comme suit :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clc ; clear all ;
2 syms x y real
3
4 exp = sin(x^2 + y^2) + cos(x^2 + y^2) ;
5 exp_evaluation = subs(exp1, {x,y}, {pi/3, 2*pi})
6
7 >> exp_evaluation =
8                   -0.7024
```

Ci-dessous, l'affichage Matlab généré par le script Matlab.

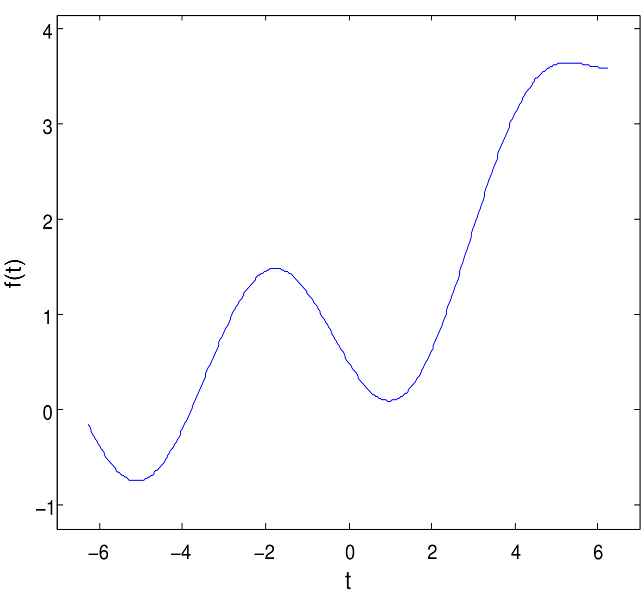

FIGURE 5.5: Graphe de la solution $f(t)$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

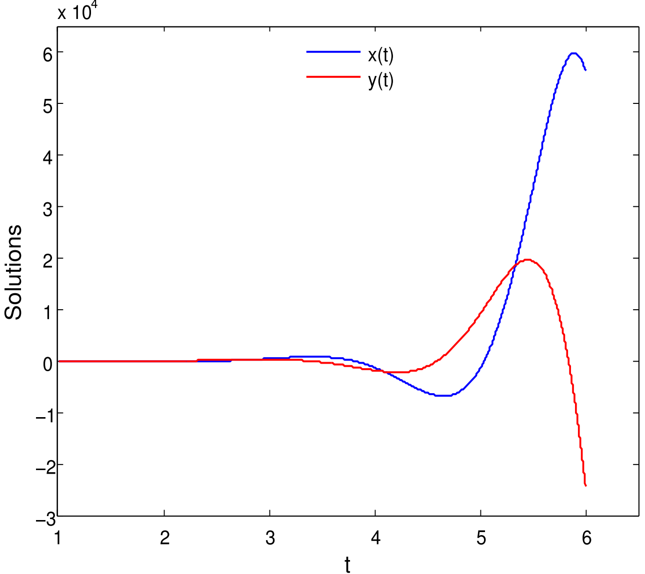

FIGURE 5.6: Graphe des solutions $x(t)$ et $y(t)$

Les expressions formelles renvoyée par le script Matlab sont :

```
1
2  >> pretty(fx)
3
4      16
5    - -- cos(t - 1) - 4/17 sin(t - 1)
6      17
7
8                        /      16                      \
9           + exp(1/4 t) |1/2 + -- cos(1) - 4/17 sin(1)|
10                       \      17                      /
11
12 >> pretty(eq1)
13
14
15     10               /450            1/2     1/2  10            1/2  \
16     -- + exp(7/4 t) |--- sin(1/4 103  t) 103     --- cos(1/4 103  t)|
17     19               \1957                        19                 /
18
19 >> pretty(eq2)
20
21     - 6/19 - 1/16
22
23                 /  3280            1/2     1/2  400            1/2  \
24       exp(7/4 t) |- ---- sin(1/4 103  t) 103  - --- cos(1/4 103  t)|
25                 \  1957                         19                 /
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❺ ☞ Ⓢ

1. Déterminer les expressions formelles des solutions, des équation différentielles suivantes :

$$(a) y'(t) = -y(t)^2 + cos(t), y(0) = 0, \forall t \in \mathbb{R}^+ \qquad (b) y'(t) + 5y(t) = 3, \forall t \in \mathbb{R}$$

$$(c)\ y'(t) + y(t)^3 = 0,\ \forall t \in \mathbb{R}^+ \qquad (d)\ y' = -\sin(y(t)),\ \forall t \in \mathbb{R}$$

$$(e)\ y''(t) = -y(t),\ \forall t \in \mathbb{R} \qquad (f)\ 2\,y'' + y' = -2\,t,\ \forall t \in \mathbb{R}$$

$$(i)\ y''(t) - (t^2 + 1) = \exp(y(t)),\ \forall t \in \mathbb{R} \qquad (j)\ y''(t) + t^2 = -y(t),\ \forall t \in \mathbb{R}$$

2. Tracer, à l'aide de la commande `ezplot`, $y(t)$ et $y'(t)$ de l'équation différentielle en (a).

5.6 Résolution formelle d'équations et de système d'équations

La commande `solve` résout de façon formelle des équations algébriques mais aussi des systèmes d'équations. Sa syntaxe usuelle est :

```
expr = solve(eq, var)
```

La commande `solve` résout l'équation `eq = 0`. L'entrée `var` représente la variable selon laquelle l'équation est résolue. Ces arguments peuvent être de type symbolique ou chaîne de caractères. Pour un système d'équations algébriques, on aura la syntaxe :

```
expr = solve(eq1, eq2, ..., eqn, var1, var2, ..., varn)
```

Les arguments d'entrée `eq1, eq2, ..., eqn` sont les expressions symboliques des équations à résoudre. Les entrées `var1, var2, ..., varn` sont les variables selon lesquelles ces équations sont résolues. Soit à résoudre ce système à deux équations :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2
3  syms x y real ;  % unreal dans le cas contraire
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
4 exprs = solve('2*x - y = 3','3*x - 12*y + 1 = 7') ;
```

La sortie renvoyée est de type *structure*, elle contient les deux racines. Pour avoir accès à ces dernières, il faudra écrire :

```
1  >> exprs =
2
3           x: [1x1 sym]
4           y: [1x1 sym]
5
6  >> exprs.x =
7
8           10/7
9
10 >> exprs.y
11
12         -1/7
```

Les sorties `exprs.x` et `exprs.y` sont de type *symbolique*. On peut avoir accès au valeurs numériques correspondant en se servant de la fonction `double` (conversion en réel double précision).

Exercice ❻ ☞ ®

1. Résoudre l'équation et le système d'équations suivants :

$$x + \exp(x) = -1 \quad \text{et} \quad a\,x^2 + (a+b)\,x + b$$

$$\begin{cases} 3\,x + y - 2 = 5 \\ x - 3\,y + 1 = 12 \end{cases}$$

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 %%%%%%%%%%%% RESOLUTION D'EQ. ET DE SYST. D'EQUATIONS %%%%%%%%%%%%
2 clc ; clear all ; close all ;
3 syms x y real ;  % unreal DANS LE CAS CONTRAIRE
4
5 eq = 'x + exp(x) = - 1' ; solexpr = solve(eq) ;
6 soleq = double(solexpr) ; % CONVERSION EN REEL DOUBLE PRECISION
7
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
8  expr = solve('a *x^2 + (a + b)*x + b', 'x') % RESOLUTIUON SELON LA
      VARIABLE x
9  sols = subs(expr, {'a','b'}, {2 5})
10
11 %%%%%%%%%%%%%%%%%%%%%%%%% SYSTEME D'EQUATIONS %%%%%%%%%%%%%%%%%%%%%%%%
12
13 sol = solve('3*x + y - 2 = 5','x - 3*y + 1 = 12') ;
14 soln = [sol.x sol.y] ;
15
16 solx = double(soln(1)) ; soly = double(soln(2)) ;
17
18 digits(8) ; solx = vpa(solx) ; soly = vpa(soly) ;
```

Les résultats renvoyés par ce script Matlab sont :

```
1  >> soleq =
2            -1.2785
3  >> sols(1) =
4            -1.0000
5  >> sols(2) =
6            -2.5000
7  >> solx     =
8             3.2000000
9  >> soly     =
10           -2.6000000
```

À défaut de l'existence d'une solution analytique (exacte), la fonction `solve` calcule une solution approchée. C'est le cas de la solution `soleq` dans le script Matlab précédent. Voici un autre exemple du calcul de la solution approchée :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ;
2
3  syms x real
4  fun = x - cos(x) ; xn = solve(fun) ; solx = double(xn)
5
6  >> solx =
7          0.7391
```

Par ailleurs, Matlab renvoie parfois un résultat nul, alors qu'il se peut que ce résultat en question ne le soit pas. Ceci provient du fait que le résultat est tellement petit qu'il faudra accroitre la précision des calculs. En outre, à chaque arrondi, on perd a priori un

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

peu de précision. Dans ce code, nous avons utilisé la fonction `digits` qui permet un changement de précision selon le chiffre spécifié en argument d'entrée. La fonction `vpa` (`Variable precision arithmetic`) calcul, dans cet exemple, les solutions `solx` et `soly` selon la précision spécifiée dans `digits`. Par défaut, les résultats sont affichés avec quatre chiffres après la virgule. Cependant, Matlab dispose également d'autres fonctions pour contrôler le format numérique. En faisant appel à la commande `format`, elle affiche les nombres en virgules fixes ou flottantes. Voici un exemple :

```
format short ; pi ; ans = 3.1416
format short e ; pi ; ans = 3.1416e+00
format long ; pi ; ans = 3.141592653589793
format long e ; pi ; ans = 3.141592653589793e+00
```

La fonction `solve` de Matlab est incapable de résoudre des inéquations. Pour ce faire, il faudra recourir à celle de Maple, en utilisant la syntaxe suivante :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clc ; clear all ;
2
3 syms x real
4 interv = maple('solve((x + 1)^3 > (2*x-1)^2)')
5
6
7 >> interv =
8           RealRange(Open(0),Inf)
```

L'ensemble solution est $]0, +\inf[$.

Exercice ❼ ☞ ®

1. Trouver les racines du polynôme :

$$p(x) = x^2 + 2\,x - 1 \tag{5.16}$$

2. Tracer, sur la même figure, le polynôme et les racines trouvées.

Le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clc ; clear all ; close all ;
2 %%%%%%%%%%%%%%%%%% RACINES DES POLYNOMES %%%%%%%%%%%%%%%%%%%%%
3 syms x real
4 polyx = x^2 + 2*x - 1 ; racines = solve(polyx) ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
5  racine1 = double(racines(1)) ; racine2 = double(racines(2)) ;
6  polyVal1 = subs(polyx, x, racine1) ; polyVal2 = subs(polyx, x,
     racine2) ;
7  %%%%%%%%%%%%%%%%%%%%%%%% AFFICHAGE GRAPHIQUE %%%%%%%%%%%%%%%%%%%%%%%
8  figure('color',[1 1 1]) ; ezplot(polyx, [-3 3]) ; hold on ;
9  plot(racine1, polyVal1,'ro','MarkerSize',10, 'LineWidth',1) ; hold on
     ;
10 plot(racine1, polyVal1,'r+','MarkerSize',10, 'LineWidth',1) ;
11 plot(racine2, polyVal2,'ro','MarkerSize',10, 'LineWidth',1) ; hold on
     ;
12 plot(racine2, polyVal2,'r+','MarkerSize',10, 'LineWidth',1) ;
13 xlabel('x','FontSize',12) ; ylabel('f(x)','FontSize',12) ;
14 text(-2.7,1.5,'Racine 1') ; text(-0.1,1.5,'Racine 2')
```

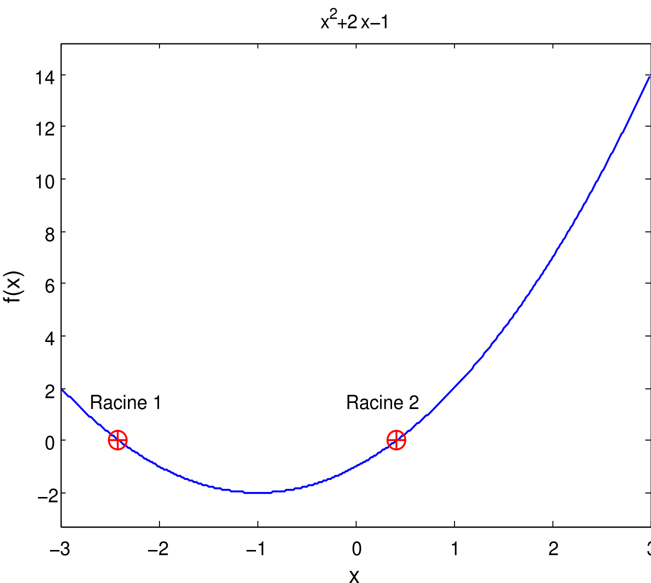

FIGURE 5.7: Figure générée par le code Matlab ci-dessus

Noter qu'un polynôme de degré n en MATLAB est représenté par un vecteur de taille $n+1$ contenant les coefficients dans un ordre décroissant. Dans le code ci-dessus, le polynôme `polyx = x^2 + 2*x - 1` est définit par `polyx = [1 2 -1]`. Il est possible aussi de faire la conversion d'un polynôme formel en un polynôme numérique et inversement. Ces possibilités sont offertes par les fonctions `sym2poly` et `poly2sym`.

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  clc ; clear all ;
3
4  syms x real
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
5
6  polyx = x^3 + 2*x^2 + x - 3 ;
7  polyn = sym2poly(polyx) ; polysy = poly2sym(polyn, x) ;
8
9  >> polyn =
10           1   2   1 -3
11
12 >> pretty(polysy) =
13                     3      2
14                    x  + 2 x  + x - 3
```

Exercice ❼ ☞ Ⓢ

1. Résoudre les équations suivantes :

$$(a) \quad (a\,x+b)\,\exp(5\,x) \quad (b) \quad \sin(x)-x/2 \quad (c) \quad x-\exp(x/y) \tag{5.17}$$

2. Trouver les racines des polynômes suivants :

$$\begin{cases} p_1(x) = x^3 - 5\,x^2 + 7, & x \in [-3,3] \\ p_2(x) = x^2 - 3\,x - 5, & x \in [-3,3] \end{cases} \tag{5.18}$$

3. Tracer, sur la même figure, les polynômes $p_1(x)$, $p_2(x)$ et les racines trouvées.
4. Résoudre les systèmes d'équations algébriques suivants :

$$\begin{cases} y - sin(x) = 0 \\ y - x^2 + 1 = 0 \end{cases} \tag{5.19}$$

$$\begin{cases} y - 4\,x^2 + 3 = 0 \\ x^2 + y^2 - 1 = 0 \end{cases} \tag{5.20}$$

$$\begin{cases} x^2 + x\,y + y = 3 \\ x^2 - 4\,x + 3 = 0 \end{cases} \tag{5.21}$$

$$\begin{cases} \sin(x+y) - \exp(x)\,y = 0 \\ x^2 - y = 2 \end{cases} \tag{5.22}$$

$$\begin{cases} x^2 + y^2 - 3 = 0 \\ y = (5 + 4\,\cos(x))^{-1} \end{cases} \tag{5.23}$$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

5.7 Résolution formelle des intégrales simples et multiples

La boite à outil `Symbolic Math Toolbox` dispose de la fonction `int` permettant de calculer de façon formelle les intégrales simple et multiple. Sa syntaxe est :

```
expr = int(fun, var, a, b)
```

L'argument d'entrée `fun` est de type *symbolique*. Les autres entrées sont respectivement, la variable d'intégration ainsi que les bornes inférieure et supérieure de l'intégrale. La sortie `expr` renvoyée constitue la primitive de `fun`.

Exercice ❽ ☞ ®

1. Déterminer l'expression symbolique des intégrales :

$$\begin{cases} I_1 = \displaystyle\int_{-1}^{1} \frac{1}{\frac{1}{2} + \exp(x+1)}\, dx \\ I_2 = \displaystyle\int_{0}^{\infty} \exp(-x^2)\, dx \end{cases} \tag{5.24}$$

2. Par le biais de la fonction `double`, déterminer la valeur numérique des deux intégrales

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clc ; clear all ; close all ;
2  %%%%%%%%%%%%%%%%%%%%%%% INTEGRALES SIMPLES %%%%%%%%%%%%%%%%%%%%%%%%
3  syms x real
4
5  fun1 = 1/(1/2 + exp(x + 1)) ; intfun1 = int(fun1, x) ;
6  expr1 = simple(intfun1) ; valint1 = int(fun1, -1, 1) ;
7
8  valint1 = double(valint1) ; % CONVERSION EN REEL DOUBLE PRECISION
9
10 fun2 = exp(-x^2) ; intfun2 = int(fun2, x) ;
11 expr2 = simple(intfun2) ; valint2 = int(fun2, 0, inf) ;
12 valint2 = double(valint2) ;
13
14 >> pretty(expr1) =         % primitive de l'integrale I1
15                   2 x + 2 - 2 log(1 + 2 exp(x + 1))
16
17 >> valint1 =
18
19                   0.6800
20
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
>> pretty(expr2) =         % primitive de l'integrale I2
                       1/2
                  1/2 pi   erf(x)

>> valint2 =

                  0.8862
```

Avec `erf(x)` est la fonction erreur :

```
>> help erf
 ERF Error function.
    Y = ERF(X) is the error function For each element of X.  X must
   be
    real. The error function is defined as:

      erf(x) = 2/sqrt(pi) * integral from 0 to x of exp(-t^2) dt.

    Class support for input X:
       float: double, single

    See also erfc, erfcx, erfinv, erfcinv.

    Overloaded methods:
       sym/erf

    Reference page in Help browser
       doc erf
```

La même commande est utilisée pour déterminer la primitive des intégrales double et triple. Dans ce cas, la commande `int` est exploitée selon :

```
int2Val = int(int(fun, x, xmin, xmax), y, ymin, ymax) ;
% integrale double
int3Val = int(int(int(fun, x, xmin, xmax), y, ymin, ymax), z, zmin,
   zmax) ; % integrale triple
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❾ ☞ ®

1. Déterminer la primitive des intégrales :

$$\begin{cases} \int\int_{\mathbb{R}^2} y^2 \sin(x) + x^2 \cos(y)\, dx\, dy \\ \int\int\int_{\mathbb{R}^3} z^2 \sin(x) + z^2 \cos(y)\, dx\, dy\, dz \end{cases} \tag{5.25}$$

Voici le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clc ; clear all ; close all ;
%%%%%%%%%%%%%%%%%%%%%%%% INTEGRALES MULTIPLES %%%%%%%%%%%%%%%%%%%%%%%%
syms x y z

fun1 = y^2*sin(x) + x^2*cos(y) ;
exprDoule = int(int(fun1, x), y) ; % INTEGRALE DOUBLE
pretty(exprDoule) ;

fun2 = z^2*sin(x) + z^2*cos(y) ;
exprTriple = int(int(int(fun2, x), y), z) ; % INTEGRALE TRIPLE
pretty(exprTriple) ;

% Les expressions symboliques respectivement des integrales double et
    triple sont affichees comme suit :

>> pretty(exprDoule) =

                      3                3
          - 1/3 y  cos(x) + 1/3 x  sin(y)

>> pretty(exprTriple) =

                      3                  3
          - 1/3 z  cos(x) y + 1/3 z  sin(y) x
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❾ ☞ Ⓢ

1. Déterminer l'expression symbolique (la primitive) des intégrales :

$(a)\ \int_{\mathbb{R}} x^3 + \cos(2\,x)\,dx \qquad (b)\ \int_{\mathbb{R}} x^2\,\exp(x^2)\,dx \qquad (c)\ \int_{\mathbb{R}} \sin^2(x)\,dx$

$(d)\ \int_{\mathbb{R}} \exp(x)\,\sqrt{1+\exp(x)}\,dx \qquad (e)\ \int_{\mathbb{R}} \frac{\sin^2(x)}{x}\,dx \qquad (f)\ \int_{\mathbb{R}} \frac{\exp(x)}{x^3+1}$

$(g)\ \int_{\mathbb{R}} x\,\exp(-x^2)\,dx \qquad (h)\ \int_{\mathbb{R}} \frac{3\,x^2+1}{x^5}\,dx \qquad (i)\ \int_{\mathbb{R}} \frac{x^2-7}{x^3-x+1}\,dx$

$(j)\ \int\int_{\mathbb{R}^2} x^2+y^2-x\,y+y\,dx\,dy \qquad (k)\ \int\int_{\mathbb{R}^2} \frac{x\,y}{x^2+y^2}\,dx\,dy \qquad (l)\ \int\int_{\mathbb{R}^2} \frac{x\,\log(1+x^3)}{y(x^2+y^2}\,dx\,dy$

$(m)\ \int\int_{\mathbb{R}^2} x\,\exp(x/y)\,dx\,dy \qquad (n)\ \int\int_{\mathbb{R}^2} \frac{x^3+y^3}{x^2+y^2}\,dx\,dy \qquad (o)\ \int\int_{\mathbb{R}^2} \frac{\sin(x^2)-\sin(y^2)}{(x^2+y^2}\,dx\,dy$

$(p)\ \int\int\int_{\mathbb{R}^3} -x^2\,\exp(x\,y)\,\sin(z)\,dx\,dy\,dz \qquad (q)\ \int\int\int_{\mathbb{R}^3} \cos(x\,z)-x\,z\,\sin(x\,z)\,dx\,dy\,dz$

$(r)\ \int\int\int_{\mathbb{R}^3} x^3+4x\,y+z^2-3yz^2-3\,dx\,dy\,dz \qquad (s)\ \int\int\int_{\mathbb{R}^3} \exp(z)\,x\,y\,z\,dx\,dy\,dz$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6 Méthodes d'interpolation

Lagrange, Hermite, nœuds de Tchebychev, spline linéaire et cubique

Sommaire

9.1 Méthode de Lagrange 197
9.2 Méthode de Hermite 201
9.3 Interpolation aux nœuds de Tchebychev 204
9.4 Interpolation par spline linéaire 209
9.5 Interpolation par spline cubique 211
9.6 Au moyen de commandes Matlab 213

Introduction

L'interpolation a pour fonction l'estimation de données en des points intermédiaires non évalués. L'illustration la plus triviale est l'interpolation linéaire où les points de mesure sont reliés par des segments de droites.

6.1 Méthode de Lagrange

Soient $n+1$ couples de données $\{x_i, y_i\}$, avec des nœuds différents x_i. On peut associer (relier) à ces données un seul et unique polynôme d'interpolation des y_i aux nœuds x_i, ayant un degré inférieur ou égal à n. Dans le cas général, ce polynôme est donné par :

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\, L_i(x), \quad \text{avec} \quad L_i(x) = \prod_{\substack{i=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \tag{6.1}$$

Cette relation est appelée formule d'interpolation de `Lagrange` et les polynômes L_i sont ses polynômes caractéristiques. Le but de l'interpolation consiste, entre autre, à substituer une fonction $f(x)$ (connue analytiquement ou non) par une fonction plus simple afin de procéder à une intégration numérique ou à un calcul de la dérivée. L'interpolation sert aussi à construire une représentation synthétique de données expérimentales quand leurs nombre devient très élevé.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❶ ☞ ®

Construire, selon la méthode de Lagrange, le polynôme d'interpolation $P_2(x)$ de degré deux qui interpole les points : $(x_0, y_0) = (0, 1)$; $(x_1, y_1) = (1, 2)$ et $(x_2, y_2) = (2, 5)$.

1. Déterminer d'abord ce polynôme de façon analytique.
2. Écrire un algorithme Matlab permettant l'implémentation de la méthode de Lagrange. Déterminer ce polynôme.
3. Tracer, sur la même figure, le polynôme et les points d'interpolation.

Dans un premier temps nous déterminerons, analytiquement, le polynôme $P_2(x)$. Ainsi, tenant compte de l'équation (6.9) et pour $n = 0, 1, 2$, il vient :

$$P_2(x) = y_0 \times L_0(x) + y_1 \times L_1(x) + y_2 \times L_2(x) \tag{6.2}$$

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \times \frac{x - x_2}{x_0 - x_2} \tag{6.3}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0} \times \frac{x - x_2}{x_1 - x_2} \tag{6.4}$$

$$L_2(x) = \frac{x - x_0}{x_2 - x_0} \times \frac{x - x_1}{x_2 - x_1} \tag{6.5}$$

En substituant les équations (6.11), (6.4) et (6.5) dans (6.10), il en résulte :

$$P_2(x) = y_0 \times \left(\frac{x - x_1}{x_0 - x_1} \times \frac{x - x_2}{x_0 - x_2}\right) + y_1 \times \left(\frac{x - x_0}{x_1 - x_0} \times \frac{x - x_2}{x_1 - x_2}\right) + y_2 \times \left(\frac{x - x_0}{x_2 - x_0} \times \frac{x - x_1}{x_2 - x_1}\right)$$

$$\Longrightarrow P_2(x) = 1 \times \left(\frac{(x-1)(x-2)}{(-1)(-2)}\right) + 2 \times \left(\frac{(x)(x-2)}{(1)(-1)}\right) + 5 \times \left(\frac{(x)(x-1)}{(2)(1)}\right)$$

$$\Longrightarrow P_2(x) = \left(\frac{(x^2 - 3\,x + 2)}{2}\right) - 2 \times \left(\frac{(x^2 - 2\,x)}{1}\right) + 5 \times \left(\frac{(x^2 - x)}{2}\right)$$

$$\Longrightarrow P_2(x) = \left(\frac{x^2 - \cancel{3x} + 2 - 4\,x^2 + \cancel{8x} + 5\,x^2 - \cancel{5x}}{2}\right)$$

$$\Longrightarrow P_2(x) = x^2 + 1 \tag{6.6}$$

À partir de ce calcul, on comprend que le calcul analytique montre ses limites pour des degrés n élevés. Par conséquent, il est nécessaire de développer un algorithme permettant

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

l'implémentation de la méthode de Lagrange afin de déterminer les polynômes quel que soit le degré n.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 02/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE LAGRANGE POUR
5  % L'INTERPOLATION POLYNOMIALE
6
7  x(1) = 0 ; x(2) = 1 ; x(3) = 2; n = 3 ; % POLYNOME DE DEGRE 2
8  y(1) = 1 ; y(2) = 2 ; y(3) = 5; interv = 1000; dx = (x(3) - x(1))/
      interv ;
9  xvar = x(1) :dx: x(3) ; polyn = 0 ;
10
11 col = {'+k','+r','+m'};
12
13 for i = 1:n
14
15     lag = 1;
16
17     for j = 1 : n
18
19         if (i~=j)
20
21             lag = (xvar - x(j))./(x(i) - x(j)).*lag;
22
23         end
24     end
25
26  figure(1) ; plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
      hold on ;
27  polyn = polyn + lag.*y(i);
28 end
29
30 hold on
31 plot(xvar, polyn,'LineWidth',1) ; hold on ; xlabel('x') ; ylabel('y')
32 axis([-0.5 2.5 -0.5 5.5])
33
34 title('Interpolation : selon Lagrange')
35
36 p = 2 ; coeff = polyfit(xvar,polyn,p) % COEFFICIENTS DU POLYNOME D'
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
INTERPOLATION. AINSI, % coeff(1) CONTIENT LE COEFFICIENT
xpuissance(p) , coeff(2) CELUI DE xpuissance(p-1), ... % C'EST-A-
DIRE, DANS NOTRE CAS coeff(1)*xpuissance(2) + coeff(2)*x + coeff
(1) = xpuissance(2) + 1.
```

Graphique représentant le polynôme et les points d'interpolation.

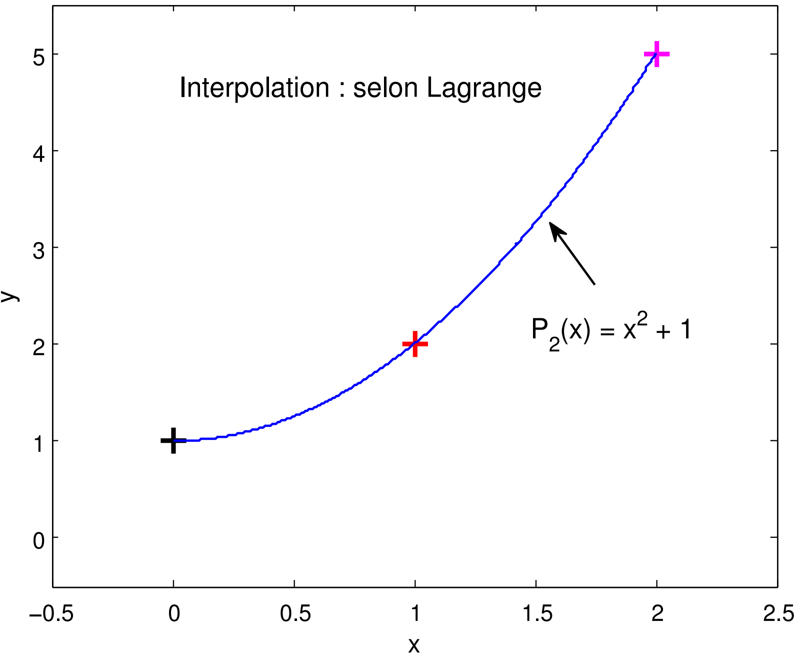

FIGURE 6.1: Interpolation de *Lagrange*

Exercice ❶ ☞ Ⓢ

1. Construire, analytiquement puis numériquement, selon la méthode de *Lagrange*, le polynôme d'interpolation de la fonction $f(x) = \exp(x)$ aux abscisses $-1, 0, 1$.

2. Tracer, sur la même figure, la fonction et le polynôme d'interpolation.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6.2 Méthode de Hermite

L'interpolation de Hermite est une généralisation de celle de Lagrange en faisant coïncider non seulement $f(x)$ et P_n aux nœuds x_i, mais également leurs dérivées d'ordre k_i aux nœuds x_i. Soient $x_0, x_1, ..., x_n$, (n + 1) points distincts de l'intervalle $[a\ b]$ et $y = f(x)$ une fonction définie sur le même intervalle admettant les dérivées $(y_0, y_0^{'})$, $(y_1, y_1^{'})$, ..., $(y_k, y_k^{'})$. Dans ce cas, il existe un seul et unique polynôme tel que $\prod_N(x_i) = y_i$ et $\prod_N^{'}(x_i) = y_i^{'}$, avec $N = 2n + 1$ et $i = 0, 1, 2, ..., n$. Le polynôme de Hermite est donné par :

$$P_n\,(x) = \sum_{i=0}^{n} \Big(y_i\, D_i(x)\ +\ y_i^{'}\,(x - x_i)\Big) \times (L_i(x))^2 \qquad (6.7)$$

Avec,

$$L_i(x) = \prod_{\substack{i=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad c_i = \sum_{\substack{i=0\\ j\neq i}}^{n} \frac{1}{x_i - x_j} \qquad D_i(x) = 1 - 2(x - x_i) \times c_i \qquad (6.8)$$

Par ailleurs, on remarque que l'interpolation de Lagrange (voir la section précédente) est un cas particulier de celle de Hermite ($k_0 = k_1 = ... = k_n = 0$).

Exercice ❷ ☞ ®

Construire, selon la méthode de *Hermite*, le polynôme d'interpolation $P_2(x)$ de degré deux qui interpole les points : $(x_0, y_0) = (0, 1)$; $(x_1, y_1) = (1, 2)$ et $(x_2, y_2) = (2, 5)$.

1. Écrire un algorithme Matlab permettant l'implémentation de la méthode de Hermite. Déterminer ce polynôme.
2. Tracer, sur la même figure, le polynôme et les points d'interpolation.

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % @copyright 03/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
4  % L'IMPLEMENTATION DE LA METHODE DE HERMITE POUR % L'INTERPOLATION
5  x = [0, 1 , 2] ; n = 3 ; % POLYNOME DE DEGRE n - 1.
6  y = [1 , 2, 5] ; interv = 1000 ; dx = (x(3) - x(1))/interv ;
7  yder = [y(1), (y(2)-y(1))/(x(2)-x(1)), (y(3)-y(2))/(x(3)-x(2))];
8
9  xvar = x(1) : dx : x(3) ; polyn = 0 ; col = {'+k','+r','+m'};
10
```


Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
11 for i = 1:n
12     lag = 1 ; ci = 0;
13     for j = 1 : n
14           if (i~=j)
15             lag = (xvar - x(j))./(x(i) - x(j)).*lag;
16             ci = ci + (1/(x(i) - x(j)));
17         end
18     end
19
20 Di = 1 - 2.*(xvar - x(i)).*ci;
21 polyn = polyn + (y(i).*Di + yder(i) .*(xvar - x(i))).*(lag).^2;
22 figure(2) ; plot(x(i),y(i),col{i},'MarkerSize',12,'LineWidth',2);
23 hold on ;
24 end
25
26 hold on
27 plot(xvar, polyn,'b','LineWidth',1) ; hold on ; xlabel('x') ; ylabel(
      'y')
28 axis([-0.5 2.5 -0.5 5.5]) ; title('Interpolation : selon Hermite')
29 p = 2 ;  coeff = polyfit(xvar,polyn,p) ; % EVALUATION DES
      COEFFICIENTS
30 evalp = polyval(coeff, xvar) ; % EVALUATION DU POLYNOME
```

Graphique représentant le polynôme et les points d'interpolation.

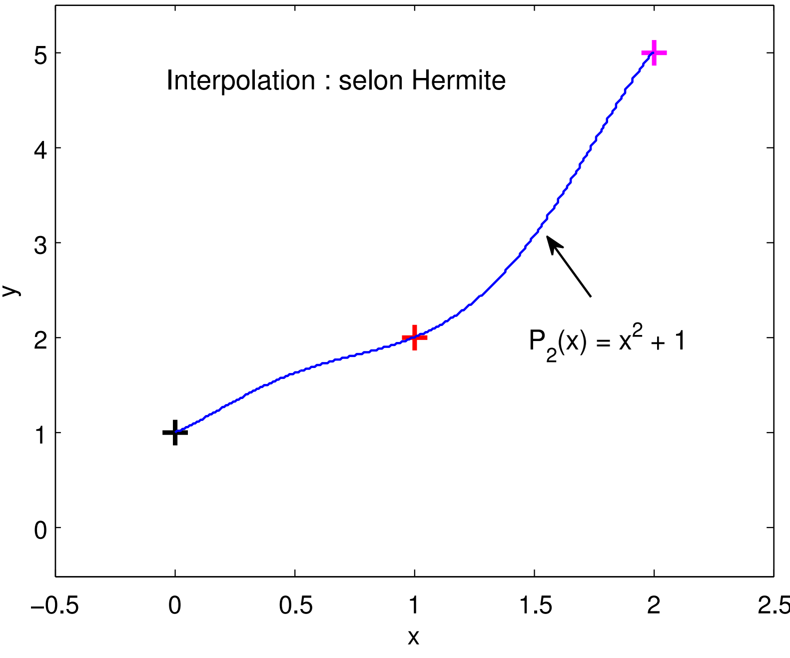

FIGURE 6.2: Interpolation de *Hermite*

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❷ ☞ Ⓢ

1. Construire, selon la méthode de *Hermite*, le polynôme d'interpolation de la fonction $f(x) = 1 - \cos(x^2)$ aux abscisses $-2, -1, 0, 1, 2$.

2. Tracer, sur la même figure, la fonction et le polynôme d'interpolation.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6.3 Interpolation aux nœuds de Tchebychev

Nous avons vu précédemment, dans le cas de l'interpolation de *Lagrange* et celle de *Hermite*, que l'utilisateur a la liberté de choisir les nœuds d'interpolation $\{x_i\}$. En revanche, l'interpolation de *Pafnouti Tchebychev*, impose un choix des nœuds (voir, Eqs. (6.10) ou (6.11)) dans l'intervalle $[a, b]$, appelés points de *Tchebychev*. Il s'agit donc d'une interpolation de *Lagrange* menée en des points particuliers. Tenant compte de cette répartition particulière de nœuds, on peut montrer que, si $f(x)$ est dérivable sur $[a, b]$, alors $P_n(x)$ converge vers $f(x)$ quand $n \longrightarrow \infty$ pour tout $x \in [a, b]$. Ainsi, les nœuds de *Tchebychev* sont déterminés par minimisation de la quantité :

$$max_{x\in[a,b]} \left| \prod_{i=0}^{n} (x - x_i) \right| \tag{6.9}$$

En effet, l'équation (6.9) est minimale pour les nœuds qui annulent les polynômes de *Tchebychev*, soit :

$$x_i = \left(\frac{b+a}{2}\right) + \left(\frac{b-a}{2}\right) \times \cos\left(\frac{(2\,(n-i)+1)\,\pi}{2\,n}\right) \quad \text{avec} \quad i = 0, 1, ..., n \tag{6.10}$$

Il existe également une autre répartition non uniforme des nœuds d'interpolation, ayant les mêmes caractéristiques de convergence que les nœuds de *Tchebychev*, définie par les nœuds de *Tchebychev-Gauss*, soit :

$$x_i = \left(\frac{b+a}{2}\right) - \left(\frac{b-a}{2}\right) \times \cos\left(\frac{(2\,i+1)\,\pi}{(n+1)\,2}\right) \quad \text{avec} \quad i = 0, 1, ..., n \tag{6.11}$$

L'attrait de choisir les nœuds de `Tchebychev`, est de se prémunir (lutter) contre le *phénomène de Runge*. Dans ce qui suit nous nous attellerons de rappeler ce phénomène : intuitivement on est tenté de croire que plus l'écart entre les points d'interpolation est réduit meilleure sera la convergence du polynôme de *Lagrange*. Les choses ne se présentent pas sur cet aspect, ainsi, *Carl Runge* a mis en évidence le fait que plus $n \longrightarrow \infty$, plus le polynôme de *Lagrange* diverge aux bords de l'intervalle $[a, b]$. On comprend dans ce cas, que la convergence n'est pas régulière. Cette irrégularité de convergence est résolue en procédant à l'interpolation aux points de *Tchebychev*. Nous allons illustrer ce *phénomène de Runge* par l'example suivant :

Soit la fonction $f(x) = \dfrac{1}{1+x^2}$, avec $x \in [-5, 5]$. Nous allons considérer les nœuds d'interpolation suivants : $x_0 = -5$, $x_1 = -4$, $x_2 = -3$, $x_3 = -2$, $x_4 = -1$, $x_5 = 0$, $x_6 = 1$, $x_7 = 2$, $x_8 = 3$, $x_9 = 4$, $x_{10} = 5$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

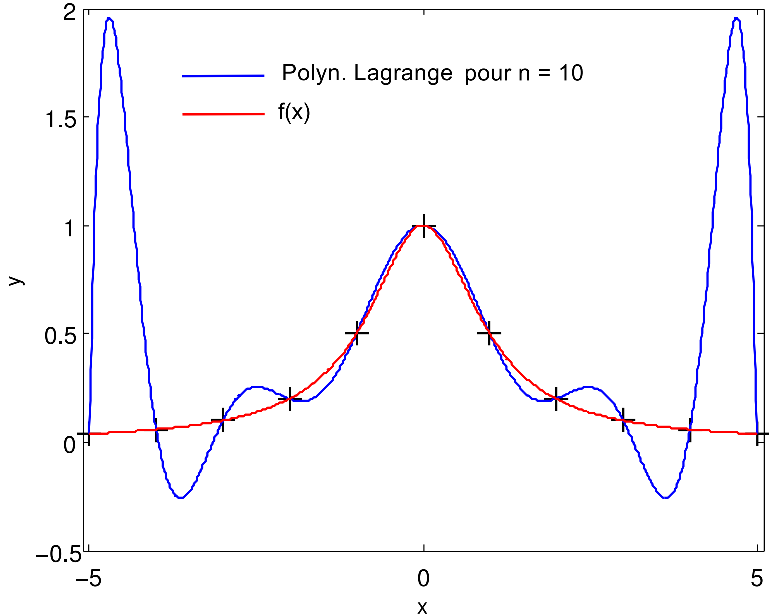

FIGURE 6.3: Illustration du phénomène de `Runge`

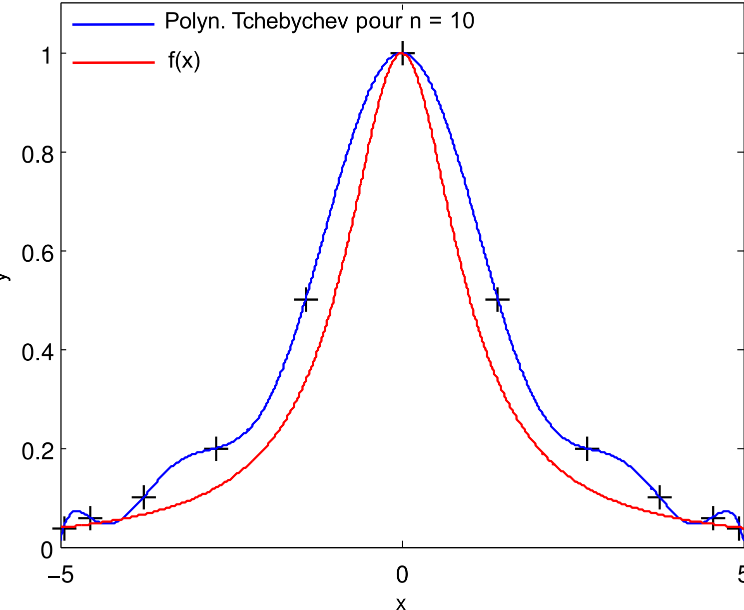

FIGURE 6.4: Atténuation du phénomène de `Runge` en adoptant les nœuds de `Tchebychev`

À partir de la fig. (6.3), on constate une divergence, au voisinage des bornes de l'intervalle, entre le polynôme interpolateur de *Lagrange* et la courbe $f(x)$. Par ailleurs, plus le degré n de ce polynôme est élevé plus la divergence (oscillations sur les bords de $[a, b]$) aux bords de l'intervalle est grande. Ceci est typique d'une convergence irrégulière. Cet exemple illustre très bien le *phénomène de Runge*. Ceci peut être démontré en calculant le maximum de $f(x)$ et de ses dérivées jusqu'à l'ordre 25 avec le code Matlab suivant :

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all ; clc ; format long ;
2 xvar = -5:0.01:5 ; syms x ; n = 24 ; fun1 = 1/(1+ x.^2 ) ;
3 for ik = 1:n + 1
4 dfun1 = diff(fun1, ik) ; fun = subs(dfun1, xvar) ;
5 maxdfun1(ik) = max(abs(fun)) ; end
```

Après exécution, les maximums des valeurs absolues de $f(x)^{(k)}$ pour k = 22 à 25 sont :

```
maxdfun1([22, 23, 24, 25]) = 1.0e+25 *
                0.000112400072778 0.002454380576469 0.062044840173324
1.480206702961776
```

En revanche, dans l'interpolation de *Tchebychev* (fig. (6.4)), la convergence est uniforme. De plus, lorsque le nombre de nœuds d'interpolation augmente, la courbe polynomiale se confond avec la fonction (fig. (6.5)).

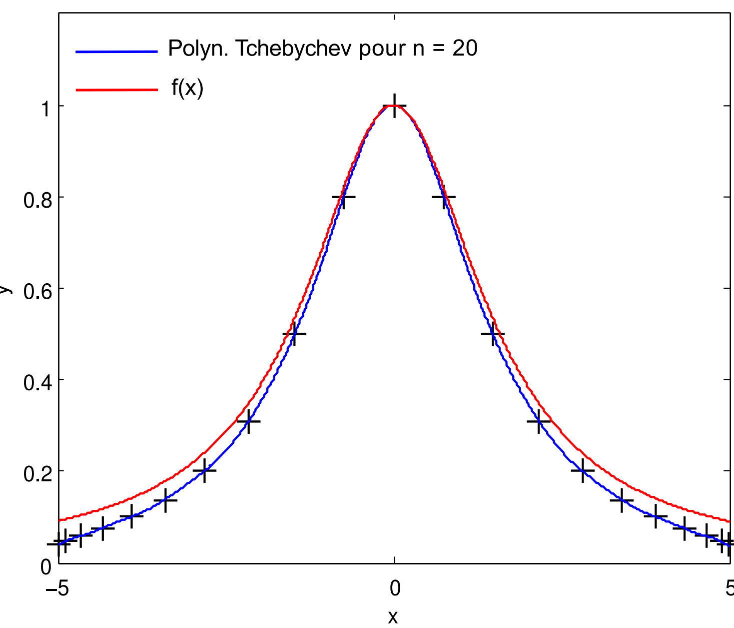

FIGURE 6.5: Effet du nombre de points d'interpolation selon `Tchebychev`

Exercice ❸ ☞ ®

Construire, sous Matlab, le polynôme d'interpolation de *Tchebychev* de la fonction $f(x) = \dfrac{1}{1+x^2}$. Les nœuds d'interpolation sont les suivants : $x_0 = -5$, $x_1 = -4$, $x_2 = -3$, $x_3 = -2$, $x_4 = -1$, $x_5 = 0$, $x_6 = 1$, $x_7 = 2$, $x_8 = 3$, $x_9 = 4$, $x_{10} = 5$.

1. Déterminer ce polynôme pour $n = 10$ et $n = 20$. Conclure
2. Afficher les nœuds de *Tchebychev*
3. Tracer, sur la même figure, le polynôme de *Tchebychev*, les points d'interpolation et la fonction $f(x)$.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Voici le script Matlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; close all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @copyright 03/12/2015 Samir KENOUCHE : ALGORITHME PERMETTANT
% L'IMPLEMENTATION, SOUS MATLAB, DE LA METHODE DE TCHEBYCHEV
format long

x(1) = -5 ; x(2) = -4 ; x(3) = -3; n = 11 ; %POLYNOME DE DCEGRE n - 1
x(4) = -2 ; x(5) = - 1 ; x(6) = 0 ; x(7) = 1 ; x(8) = 2 ; x(9) = 3 ;
   x(10) = 4 ;
x(11) = 5 ;  y = 1./(1 + x.^2);

interv = 1000; dx = (x(11) - x(1))/interv ;
xvar = x(1) :dx: x(11) ; polyn = 0 ;

x=(x(11)+x(1))/2+((x(11)-x(1))/2)*cos((2*(n-(1:n))+1)*pi/(2*n));
xChe =  x; % LES NOEUDS xi ANNULANT LES POLYNOMES DE TCHEBYCHEV :
   CECI AFIN DE MINIMISER L'ECART ENTRE Pn(x) ET LA FONCTION f(x) A
   INTERPOLER

for i = 1:n

    lag = 1;

    for j = 1 : n

        if (i~=j)

          lag = (xvar - x(j))./(x(i) - x(j)).*lag;

        end
    end

 figure(1) ; plot(x(i),y(i),'+k','MarkerSize',12,'LineWidth',1); hold
    on ;
 polyn = polyn + lag.*y(i);
end

hold on

plot(xvar, polyn,'b','LineWidth',1) ; hold on ; xlabel('x') ; ylabel(
   'y')
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
37 title('Interpolation : selon Lagrange')
38 p = 10 ; coeff = polyfit(xvar,polyn,p) ;
39
40  evalp = polyval(coeff, xvar) ; hold on ;
41 plot(xvar, 1./(1 +xvar.^2),'r')
```

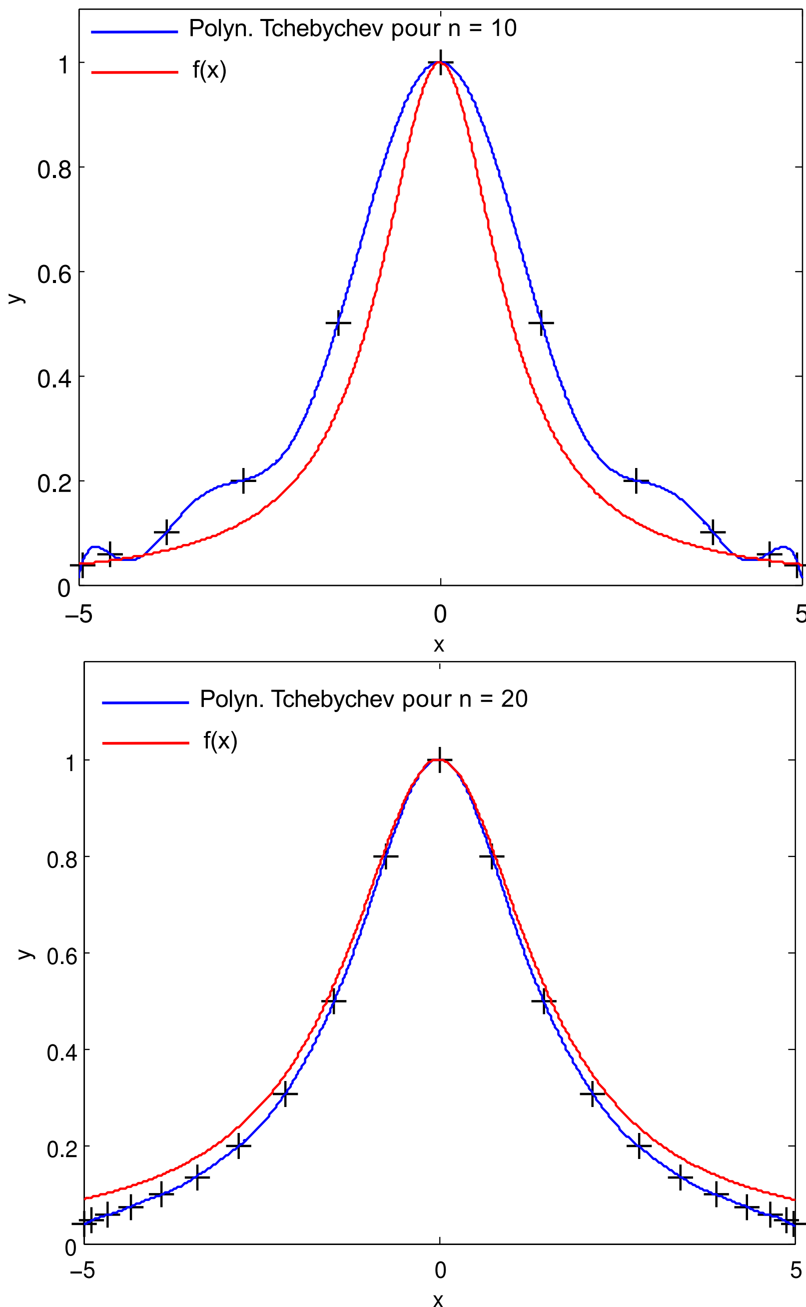

FIGURE 6.6: Figures générées par le code Matlab ci-dessous pour $n = 10$ et $n = 20$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❸ ☞ Ⓢ

Construire, le polynôme d'interpolation de `Tchebychev` de la fonction $f(x) = \sin(x^2) - \cos(x^2)$. Les nœuds d'interpolation sont : $x_1 = -4$, $x_2 = -3$, $x_3 = -2$, $x_4 = -1$, $x_5 = 0$, $x_6 = 1$, $x_7 = 2$, $x_8 = 3$, $x_9 = 4$.

1. Déterminer ce polynôme pour $n = 12$ et $n = 22$. Conclure
2. Afficher les nœuds de `Tchebychev`
3. Tracer, sur la même figure, le polynôme de `Tchebychev`, les points d'interpolation et la fonction $f(x)$.

6.4 Interpolation par spline linéaire

Pour répartition, non nécessairement uniforme, de points $a = x_0 < x1 < \ldots < x_N = b$, on approxime la fonction f sur chaque sous-intervalle $[x_i, x_{i+1}]$, par un segment de droite connectant les deux points $(x_i, f(x_i))$ et $(x_{i+1}, f(x_{i+1}))$. Sur chaque sous-intervalle, ce segment de droite s'écrit :

$$p_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \times (x - x_i), \text{ pour } \quad x_i \leq x \leq x_{i+1} \tag{6.12}$$

La faiblesse de l'interpolation par spline linéaire provient du fait que la fonction d'interpolation est continue mais n'est pas dérivable. Par ailleurs, notons que la commande `plot`, utilisée pour tracer le graphe d'une fonction f sur un intervalle donné $[a, b]$, substitue en réalité la fonction par une interpolée linéaire par morceaux.

Exercice ❹ ☞ Ⓡ

1. Écrire un script Matlab implémentant la méthode d'interpolation par splines linéaires. Tester cette fonction :

$$\begin{cases} f(x) = \cos(x^x) + \sin(x) + 1 \\ \text{Avec} \quad x \in [\frac{1}{2}, 10] \end{cases} \tag{6.13}$$

2. Tracer le graphe de $f(x)$ et la spline linéaire.

Voici le script Mtlab

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1 clear all ; close all ; clc ;
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 fun = @(x) cos(x.^x) + sin(x) + 1 ; N = 10 ; lB = 1/2 ; uB = 10 ;
```


Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
4  h = (uB - lB)/N ; x =  lB :h:  uB ; fx = fun(x) ;
5
6  figure('color',[1 1 1]) ;
7
8  for i = 1 : N
9
10  step = (x(i+1) - x(i))/ N ; xi = x(i) : step : x(i+1) ;
11
12  polynom = fx(i) + ((fx(i+1) - fx(i))/(x(i+1) - x(i))).*(xi - x(i)) ;
13  plot(xi, polynom, 'LineWidth',1) ; hold on ;
14
15 end
16
17 plot(x, fun(x),'rx','MarkerSize',10,'LineWidth',1.5) ;
18 xlabel('\fontsize{12}\fontname{Tex} x');
19 ylabel('\fontsize{12}\fontname{Tex} f(x) ');
20 title('\fontname{Tex}\fontsize{12} Interpolation par splines
      lineaires')
```

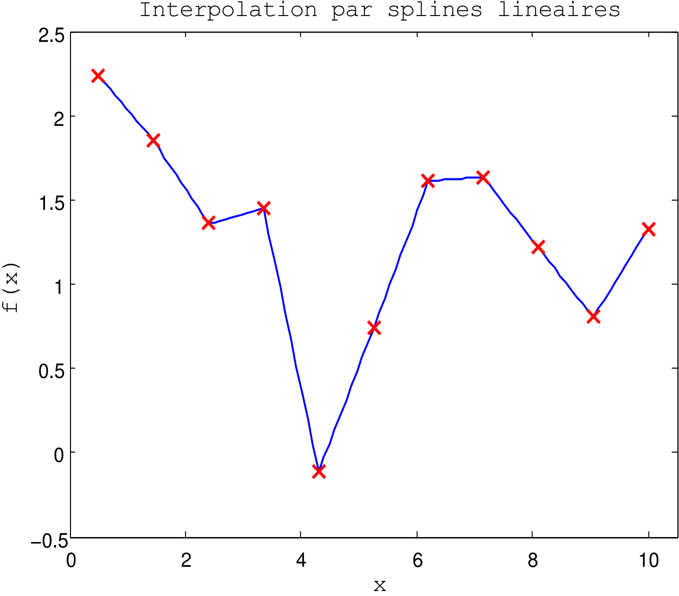

FIGURE 6.7: Interpolation par splines linéaires

Exercice ❹ ☞ Ⓢ

1. Réaliser une interpolation par *spline linéaire* des données $(1, 3)$, $(3, 1)$, $(4, 5)$, $(5, 4)$, $(6, 5)$ et $(7, 10)$.
2. Tracer le graphe représenté par les couples de données et *spline linéaire*.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

6.5 Interpolation par spline cubique

Nous avons déjà, dans les sections précédentes, montré que l'interpolation polynomiale n'est pas bien adaptée à l'approximation de fonctions pour des degrés n élevés. Ceci se manifeste par le `phénomène de Runge`. Une alternative à cette approche est de procéder à une `interpolation polynomiale par morceaux`. L'avantage de ce type d'interpolation est qu'en augmentant le degré n, on accroit simultanément le nombre de morceaux et non le degré du polynôme. Les fonctions splines d'interpolation cubiques sont les fonctions splines de plus petit degré interpolant une fonction f régulière, deux fois dérivable sur $[a, b]$. La régularité d'une fonction peut se mesurer au moyen de ses dérivées. Ainsi, plus une fonction est différentiable, plus la courbe qui lui est associée est lisse et plus la fonction est régulière.

Afin de réaliser cette interpolation, on divise l'intervalle $[a, b]$ en N sous-intervalles de taille $(b - a)/N$. Ensuite, sur chaque sous-intervalle $[x_i, x_{i+1}]$, on construit un polynôme cubique $p_i(x)$ d'interpolation. Ce dernier connecte le point $(x_i, f(x_i))$ au point $(x_{i+1}, f(x_{i+1}))$ et s'écrit :

$$p_i(x) = f_i + f_i^{'}(x - x_i) + \frac{f_i^{''}}{2!}(x - x_i)^2 + \frac{f_i^{'''}}{3!}(x - x_i)^3, \quad i = 0, 1, ..., N - 1 \quad (6.14)$$

Les coefficients à déterminer sont les $f_i, f_i^{'}, f_i^{''}$ et $f_i^{'''}$. Dans le cas de *splines not a knot* (pas constant), $p_i(x)$ est évalué par :

$$p_i(x) = a_i\,(x - x_i)^3 + b_i\,(x - x_i)^2 + c_i\,(x - x_i) + f(x_i) \quad (6.15)$$

Avec,

$$\begin{cases} a_i = \dfrac{\alpha_{i+1} - \alpha_i}{6\,h} \\ b_i = \dfrac{\alpha_i}{2} \\ c_i = \dfrac{f(x_{i+1}) - f(x_i)}{h} - \dfrac{2\alpha_i + \alpha_{i+1}}{6}\,h \end{cases} \quad (6.16)$$

En posant la condition $\alpha_0 = \alpha_N = 0$, le vecteur des α_i est donné par la résolution d'un système linéaire tridiagonale de la forme :

$$A\,\alpha = b_n \quad (6.17)$$

Avec,

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

$$A = h \begin{pmatrix} 4 & 1 & 0 & \dots & 0 \\ 1 & 4 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & 4 & 1 \\ 0 & \dots & 0 & 1 & 4 \end{pmatrix} \quad \text{et} \quad b_n = \frac{6}{h} \begin{pmatrix} f(x_0) - 2\,f(x_1) + f(x_2) \\ f(x_1) - 2\,f(x_2) + f(x_3) \\ f(x_2) - 2\,f(x_3) + f(x_4) \\ \vdots \\ f(x_{N-2}) - 2\,f(x_{N-1}) + f(x_N) \end{pmatrix} \tag{6.18}$$

Avec, α est un vecteur de dimension $N - 1$.

Exercice ❺ ☞ ®

1. Écrire un script Matlab implémentant la méthode d'interpolation par *spline cubique*. Tester cette fonction :

$$\begin{cases} f(x) = \sin(x) - x^3 \\ \text{Avec} \quad x \in [-4, 4] \end{cases} \tag{6.19}$$

2. Tracer le graphe de $f(x)$ et *la spline cubique*.

Voici le script Matlab

Script Matlab

```
clear all ; close all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fun = @(x) sin(x) - x.^3 ; N = 10; lB = -4 ;uB = 4 ;h = (uB - lB)/N ;
 x =  lB :h:  uB ; fx = fun(x) ;

for in = 1: N - 1

bn(in) = (fx(in) - 2*fx(in+1) + fx(in+2))*(6/h) ;

end

sur_diag = diag(ones(N-2 ,1) ,1) ; des_diag = diag(ones(N-2 ,1) ,-1);
in_diag = diag(ones(N - 1 ,1)) ; in_diag(in_diag == 1) = 4;
mtriceA = sur_diag + des_diag + in_diag ; alpha = bn*inv(mtriceA)' ;
alpha = [0 , alpha , 0] ; figure('color',[1 1 1]) ;

for i = 1 : N

 ai = (alpha(i+1) - alpha(i))/(6*h) ; bi = alpha(i)/2 ;
 ci = (fx(i+1) - fx(i))/h - (2*alpha(i) + alpha(i+1))*(h/6) ;

```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
22     step = (x(i+1) - x(i))/ N ;
23     xi = x(i) : step : x(i+1) ;
24
25     polynom = ai*(xi - x(i)).^3 + bi*(xi - x(i)).^2 + ci*(xi - x(i)) +
         fx(i) ;
26     plot(xi, polynom, 'LineWidth',1) ; hold on ;
27
28   end
29
30   plot(x, fun(x),'rx','MarkerSize',10,'LineWidth',1.5) ;
31   xlabel('\fontsize{12}\fontname{Tex} x');
32   ylabel('\fontsize{12}\fontname{Tex} f(x) ');
```

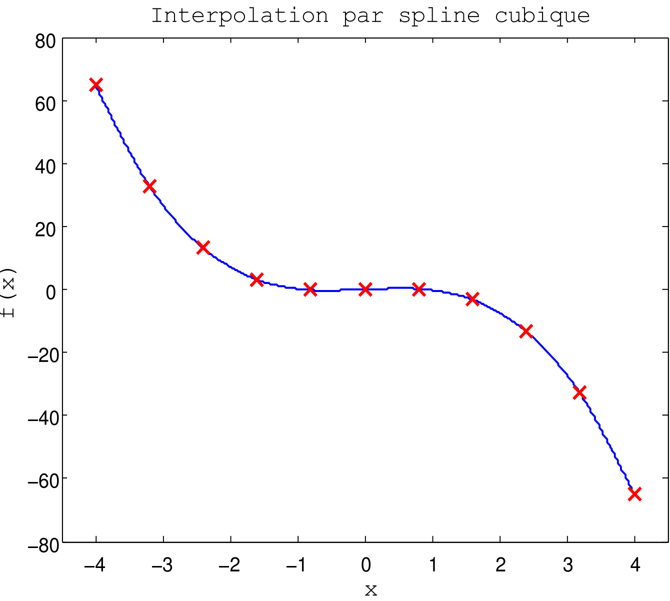

FIGURE 6.8: Interpolation par spline cubique

Exercice ❺ ☞ Ⓢ

1. Réaliser une interpolation par spline cubique des données $(0, 2)$, $(1, 2)$, $(2, 0)$ et $(3, 1/2)$.
2. Tracer le graphe représenté par les couples de données et la spline cubique.

6.6 Au moyen de routines Matlab

Matlab offre également la possibilité de réaliser des interpolations (polynomiales, trigonométriques ...), au moyen de commandes préprogrammées. Parmi ces commandes, on

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

citera : interp1, interp1q, interpft, interp2, interp3, interpn, pchip, spline, griddata ... etc. Nous allons voir l'implémentation de la commande interp1 qui réalise une interpolation en une dimension. Elle peut être utilisée pour plusieurs méthodes d'interpolation. La syntaxe usuelle de cette commande est :

```
yi = interp1(x, y, xi, method).
```

Les arguments en entrée sont x, représentant les nœuds d'interpolation et y est la fonction évaluée au nœuds d'interpolation. Ces deux entrées doivent avoir la même dimension et peuvent êtres des scalaires, des vecteurs ou des matrices. Les xi sont les abscisses de l'interpolation et la sortie yi sont les valeurs d'interpolation aux points xi. L'argument method exprime la méthode d'interpolation considérée qui peuvent êtres : 'linear' (Linear interpolation), 'cubic' (Cubic interpolation), 'spline' (Cubic spline interpolation) et 'nearest' (Nearest neighbor interpolation). Le script Matlab ci-dessous, met en exergue une comparaison entre ces différentes méthodes d'interpolation.

Exercice ❻ ☞ ®

1. Au moyen de la commande interp1, réaliser l'interpolation des points générés par la fonction suivante :

$$\begin{cases} f(x) = \sin(x) - x^3 \\ \text{Avec} \quad x \in [-10, 10] \end{cases} \tag{6.20}$$

2. Prenez $n = 20$, ce qui génère $n+1$ nœuds d'interpolation définis sur l'intervalle $[-10, 10]$

Le script Matlab :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
clear all ; close all ; clc ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lowerBound = -10 ; upperBound = 10 ; n = 20 ;
step = (upperBound-lowerBound)/n ;
xNoeuds =lowerBound :step: upperBound ;
fun = @(xNoeuds) sin(xNoeuds) - xNoeuds.^3;
xInterp = lowerBound : step/2 : upperBound ;
interpfun1 = interp1(xNoeuds, fun(xNoeuds),  xInterp,'linear') ;
interpfun2 = interp1(xNoeuds, fun(xNoeuds),  xInterp,'cubic') ;
interpfun3 = interp1(xNoeuds, fun(xNoeuds),  xInterp,'spline') ;
interpfun4 = interp1(xNoeuds, fun(xNoeuds),  xInterp,'nearest') ;

figure('Color', [1 1 1]) ; hold on ; plot(xInterp, interpfun1,'b') ;
hold on ;  plot(xInterp, interpfun2,'m') ; hold on  ;
```

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

```
15 plot(xInterp, interpfun3,'k') ; hold on ; plot(xInterp, interpfun4,'g
    ') ;
16 hold on ; plot(xNoeuds, fun(xNoeuds),'rx','MarkerSize',8,'LineWidth'
    ,1) ;
17 hold on ; plot(xNoeuds, fun(xNoeuds),'ro','MarkerSize',8,'LineWidth'
    ,1)
18 legend('linear','cubic','spline','nearest') ;
```

Par ailleurs, la commande `polyfit` réalise l'interpolation de `Lagrange`. Son exploitation peut se faire, comme montré sur le script Matlab ci-dessous :

✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎ Script Matlab ✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎✎

```
1  clear all ; close all ; clc ;
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  lowerBound = -10 ; upperBound = 10 ; n = 20 ;
4  step = (upperBound-lowerBound)/n ;
5  xNoeuds =lowerBound :step: upperBound;
6  
7  fun = @(xNoeuds) sin(xNoeuds) - xNoeuds.^3 ;
8  xInterp = lowerBound : step/2 : upperBound ;
9  
10 for nInterp = 1 : 20
11 
12  coeff = polyfit(xNoeuds ,fun(xNoeuds), nInterp) ;% interp lagrange
13  finterp = polyval(coeff, xInterp) ;
14 
15  xvar  = linspace(xNoeuds(1), xNoeuds(end), numel(finterp)) ;
16  errInterp(nInterp)  = abs(sum(fun(xvar) - finterp)) ;
17  npOpti = find(errInterp == min(errInterp)) ;
18 
19 end
20 
21 coeffOpti = polyfit(xNoeuds ,fun(xNoeuds), npOpti) ;
22 finterpOpti = polyval(coeff, xInterp) ; figure('Color', [1 1 1]) ;
23 
24 plot(xNoeuds,fun(xNoeuds),'o') ; hold on ;
25 plot(xInterp, finterpOpti,'-') ;
```

En outre, nous avons optimisé le degré du polynôme interpolateur en calculant l'erreur entre ce dernier et les points à interpoler.

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Exercice ❻ ☞ Ⓢ

1. Au moyen des commandes `interp1q`, `interp1` et `polyfit`, procéder à l'interpolation des points générés par la fonction suivante :

$$\begin{cases} f(x) = x\ \log(x^2+1) \\ \text{Avec} \quad x \in [-4, 4] \end{cases} \tag{6.21}$$

2. Prenez $n = 10$, ce qui génère $n+1$ nœuds d'interpolation définis sur l'intervalle $[-4, 4]$

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Bibliographie

[1] Cooper J., A MATLAB Companion for Multivariable Calculus, Harcourt/Academic Press, 2001. USA.

[2] Bastien J., Introduction à l'analyse numérique : Applications sous Matlab. Dunod, 2003.

[3] Quarteroni A., Saleri F., Gervasio P., Calcul Scientifique, Deuxième edition, Springer-Verlag, 2001. Italia.

[4] Jedrzejewski F., Introduction aux méthodes numériques, Deuxième edition, Springer-Verlag, 2005. France.

[5] FILBET F., Analyse numérique - Algorithme et étude mathématique. Dunod, 2009.

[6] CIARLET P. G., Introduction à l'analyse numérique matricielle et à l'optimisation - cours et exercices corrigés. Mathématiques appliquées pour la maîtrise. Dunod, 1998.

[7] LASCAUX P., THÉODOR R., Analyse numérique matricielle appliquée à l'art de l'ingénieur. Méthodes itératives. Dunod, 2000.

[8] Dion J. G., Gaudet R., Méthodes d'Analyse Numérique : de la théorie à l'application. MODULO, 1996.

[9] Atkinson K., An Introduction to Numerical Analysis. 2nd edition. John Wiley & Sons Inc., New York. 1989.

[10] Axelsson O., Iterative Solution Methods. Cambridge University Press. Cambridge, 1994.

[11] Lambert J., Numerical Methods for Ordinary Differential Systems. John Wiley and Sons, Chichester, 1991.

[12] Langtangen H., Advanced Topics in Computational Partial Differential Equations : Numerical Methods and Diffpack Programming. Springer-Verlag, Berlin. 2003.

[13] Merrien J.L., Analyse numérique avec MATLAB, édition Dunod, Paris. 2007.

[14] Crouzeix M., Mignot A., Analyse numérique des équations différentielles. Masson, 1984

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche

Méthodes numériques et programmation - Copyright © 2016 Samir Kenouche