الجمهورية الجزائرية الديمقراطية الشعبيـة
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعليم العالـي و البحث العلمـي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

MOHAMED BOUDIAF UNIVERSITY OF M'SILA
FACULTY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

جامعة محمد بوضياف – المسيلة
كلية التكنولوجيا
قسم الإلكترونيك

الرقم : 2.1.3/ق.إ/2024

المسيلة في : 2024/11/18

شهادة إدارية

بعد الإطلاع على التقارير الايجابية الواردة من السادة الخبراء أعضاء لجنة دراسة المطبوعة الجامعية والاتية أسماؤهم:

- خلفة علي — أستاذ محاضر "أ" — جامعة محمد بوضياف - المسيلة
- زريق عبد المالك — أستاذ محاضر "أ" — جامعة محمد بوضياف - المسيلة
- بلقاسم سبتي — أستاذ — جامعة باتنة 2

صادق أعضاء اللجنـة العلميـة على قبول المطبوعة البيداغوجية مع إمكانية إتخاذها سندا في تدريس طلبة السنة الثانية ليسانس الكترونيك واتصالات، في ميدان علوم و تكنولوجيا و أن تعتمد في أي تقييم المسار العلمي للأستاذ المعني **جغدالي لخضر** (أستاذ محاضر قسم "ب"– جامعة محمد بوضياف - المسيلة) تحت عنوان :

Travaux Pratiques Méthodes Numérique

رئيس القسم

طباخ مصطفى

رئيس اللجنة العلمية

مزعاش عمار

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET

DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOHAMED BOUDIAF - M'SILA

FACULTE DE TECHNOLOGIE

Département d'Electronique

Polycopié Pédagogique

Licence électronique et Télécommunication

L2 –S4

Travaux Pratiques

Méthodes Numériques

Dr : DJAGHDALI Lakhdar

Maitre de Conférences « B »

Année universitaire : 2023 /2024

Avant-propos

Ce polycopié se trouve être un manuel pédagogique de travaux pratiques de la matière : Méthodes numériques, destinés aux étudiants de la deuxième année licence LMD en électronique et télécommunication. Il donne une idée sur l'implémentation en MATLAB de quelques méthodes étudiées dans les différents chapitres du cours de méthodes numériques.

Les objectifs de ce TP sont :

1. Présenter les fondements mathématiques du calcul scientifique en analysant les propriétés théoriques des méthodes numériques, tout en illustrant leurs avantages et inconvénients à l'aide d'exemples.

2. Utiliser un logiciel de calcul scientifique MATLAB pour implémenter, visualiser et comparer les résultats.

Table des Matières

Chapitre 2 : Interpolation et approximation polynômiale de fonctions

Chapitre 3 : Intégrations Numériques

Chapitre 4 : Résolution Numérique Des Equations Différentielles

Chapitre 5 : Résolution Numérique Des Systèmes D'équations Linéaires

Introduction générale

Le calcul scientifique implique la création, l'analyse et l'application de méthodes appartenant à des domaines mathématiques tels que l'analyse, l'algèbre linéaire, la géométrie, la théorie de l'approximation, les équations fonctionnelles, l'optimisation ou le calcul différentiel.

La physique, les sciences biologiques, les sciences de l'ingénieur, l'économie et la finance sont des domaines où les méthodes numériques sont naturellement utilisées. L'évolution constante des ordinateurs et des algorithmes renforce son rôle : la taille des problèmes que l'on sait résoudre aujourd'hui est telle qu'il est envisageable de rendre possible la simulation de phénomènes réels.

Ce polycopié est organisé en 5 chapitres, chaque chapitre comporte un rappel théorique de différentes méthodes, un programme de test et un énoncé des travaux pratiques. Toutefois, en exposant les résultats de test afin de caractériser les performances de chaque méthode.

Dans le premier chapitre, nous avons exposé les différentes méthodes de résolution numérique des équations non linéaires à savoir, la méthode de dichotomie, la méthode de point fixe, la méthode de Newton et la méthode de Lagrange.

Le problème de l'interpolation et l'approximation polynomiale est traité dans le chapitre deux, nous allons étudier la méthode de Lagrange et la méthode de Newton.

Le troisième chapitre est consacré à l'intégration numérique de fonctions. Deux approches sont développées : la méthode des trapèzes et la méthode de Simpson.

On traite la résolution numérique des équations différentielles ordinaires dans le chapitre quatre. Deux méthodes sont présentées à savoir, la méthode d'Euler et la méthode de Runge-Kutta.

Dans le chapitre cinq, on aborde la résolution numérique des systèmes d'équations linéaires. Deux approches sont traitées ; la méthode de Gauss, et la méthode de Gauss-Seidel.

CHAPITRE I

Résolution numérique des équations non linéaires

Chapitre 1 : Résolution numérique des équations non linéaires

1. Introduction

L'objet essentiel de ce chapitre est l'approximation des racines d'une fonction réelle d'une variable réelle, c'est-à-dire la résolution approchée d'une équation non linéaire. Étant donné une équation non linéaire, à une seul variable, est définie par :

$$f(x) = 0 \qquad \text{(I-1)}$$

La valeur de la variable qui vérifie cette égalité est appelée solution (ou racine) de l'équation, elle est notée x_0. Dans beaucoup des cas, on doit recourir aux méthodes numériques car la solution ne peut pas être déterminée analytiquement. Ainsi, résoudre cette équation revient à définir une succession d'éléments (solutions intermédiaires) qui convergent vers la solution désirée lorsque tant vers l'infini. On ne parle pas de la solution exacte, car souvent elle n'existe pas, cependant on cherche une solution qui soit approchée de la solution exacte avec une certaine précision.

2. But du TP :

Il existe plusieurs techniques permettant de résoudre l'équation non linéaire, ces méthodes se distinguent par leurs principes et leurs vitesses de convergence.

Le but de ce TP c'est de mettre en œuvre les algorithmes des méthodes de résolution des équations non linéaires étudiées pendant le cours: la méthode de dichotomie (ou de bissection), point fixe , Newton et Lagrange . Nous les présentons dans l'ordre de complexité croissante des algorithmes.

2.1 La méthode de dichotomie

a. Principe

Considérons une fonction f continue sur un intervalle [a, b]. On suppose que f admet une et une seule racine α dans]a, b[et que f(a)*f(b)<0. On note

c=(a+b)/2 le milieu de l'intervalle.

1. Si *f(c)=0*, c'est la racine de f et le problème est résolue.
2. Si *f(c)≠0*, nous regardons le signe de *f(a)*f(c).*

1. Si $f(a)*f(c)<0$, alors $\alpha \in]a, c[$
2. Si $f(c)*f(b)<0$, alors $\alpha \in]c, b[$

La figure ci-dessous illustré le principe de la méthode de Dichotomie :

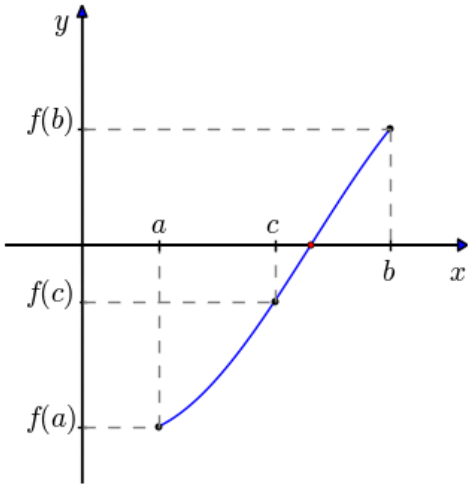


Figure I.1 : Principe de la méthode de Dichotomie

On recommence le processus en prenant l'intervalle $[a, c]$ au lieu de $[a, b]$ dans le premier cas, et l'intervalle $[c, b]$ au lieu de $[a, b]$ dans le second cas. De cette manière, on construit par récurrence sur $\boldsymbol{n}$ trois suites (a_n), (b_n) et (c_n) telles que $a_0=a$, $b_0=b$ et telles que pour tout $n\geq 0$,

1. $c_n=(a_n+b_n)/2$
2. Si $f(c_n)*f(b_n)<0$ alors $a_n+1=c_n$ et $b_n+1=b_n$.
3. Si $f(c_n)*f(a_n)<0$ alors $a_n+1=a_n$ et $b_n+1=c_n$.

b. Etude de la convergence

Théorème

Soit f une fonction continue sur $[a, b]$, vérifiant $f(a)*f(b)<0$ et soit $\alpha\in [a, b]$ l'unique solution de l'équation $f(x)=0$. Si l'algorithme de dichotomie arrive jusqu'à l'étape $\boldsymbol{n}$ alors on a l'estimation:

$$|\alpha - c_n| \leq \frac{b-a}{2^{n+1}} \qquad \text{(I-2)}$$

Par conséquent, la suite (c_n) converge vers $\boldsymbol{\alpha}$. C'est aussi vrai si $(c_n)=\alpha$.

c. Test d'arrêt

Pour que la valeur de c_n de la suite à la n-ième itération soit une valeur approchée $\boldsymbol{\alpha}$ de à $\varepsilon>0$ près, il suffit que $\boldsymbol{n}$ vérifie:

$$\frac{b-a}{2^{n+1}} \leq \varepsilon \qquad \text{(I-3)}$$

On a alors:

$$|\alpha - c_n| \leq \frac{b-a}{2^{n+1}} \leq \varepsilon \qquad \text{(I-4)}$$

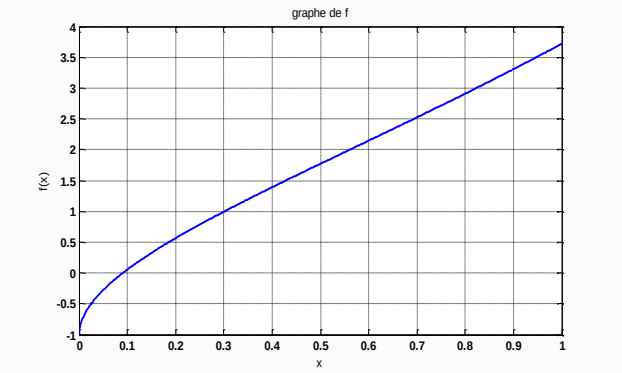
ce qui permet de calculer à l'avance le nombre maximal $n_0 \in \mathbb{N}$ d'itérations assurant la précision ε.

$$\frac{b-a}{2^{n+1}} \leq \varepsilon \Leftrightarrow \frac{b-a}{\varepsilon} \leq 2^{n+1} \Leftrightarrow n \geq \frac{\log\frac{b-a}{\varepsilon}}{\log(2)} - 1 \qquad \text{(I-5)}$$

Exemple

On considère la fonction $f(x) = \exp(x) + 3\sqrt{x} - 2$ sur l'intervalle [0, 1]. Le code Matlab suivant nous permet de tracer le graphe de f.

```
x = 0:0.001:1;
f = inline('exp(x)+3*sqrt(x)-2');
plot(x,f(x))
grid on;
ylabel('f(x)');
xlabel('x');
title('graphe de f');
```



La figure montre que f admet un unique raçine $\alpha \in [0, 1]$. Si on veut utiliser la méthode de dichotomie pour estimer α à une tolérance $\varepsilon = 10^{-10}$ près, il nous faut au plus 33 itérations. En effet, la suite (x_n) qui approche α vérifie

$$|x_n - \alpha| \leq \frac{1}{2^{n+1}}$$

$$\frac{1}{2^{n+1}} \leq 10^{-10} \Rightarrow n \geq 10\frac{\log(10)}{\log(2)} - 1 \approx 33$$

d. Mise en œuvre de la méthode dichotomie sous Matlab :

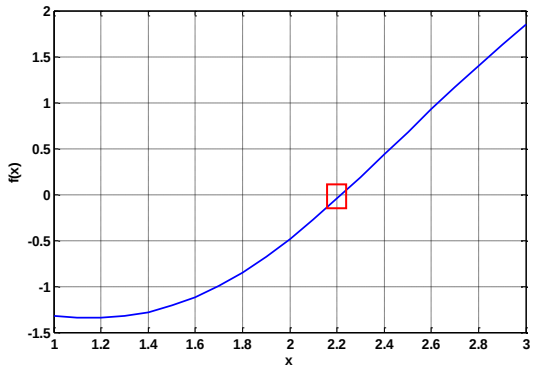
Dans ce test, nous allons trouver la racine de la fonction $f(x) = x - e^{\sin(x)}$ sur l'intervalle [1, 3] en utilisant la méthode de dichotomie, jusqu'à la convergence avec une précision de 10^{-3} .

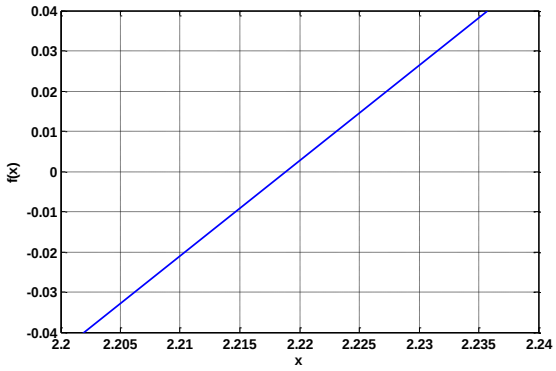
```
clc;
clear all;
x=1:0.1:3;
f=inline('x-exp(sin(x))');
plot(x,f(x)), grid
a=input('Donner la valeur de a=');
b=input('Donner la valeur de b=');
Nmax=input('Donner la valeur de Nmax=');
fa=f(a);
fb=f(b);
```

```
eps=1.0e-3; err=b-a;
if ((fa*fb)<0)
    for i=1:Nmax
        x=(a+b)/2;
        err=abs(b-a);
        if (f(x)*(fa)<0)
            b=x;
            fb=f(x);
            if err<eps
                break;
            end
        else
            a=x;
            fa=f(x);
            if err<eps
                break;
            end
        end
 fprintf('Dans l''iteration i=%d\t la solution est x0=%f\t f(x0)=
%f\n',i,x,f(x));
    end
    fprintf('La solution finale est x0 = %f\n',x);
else
    disp('On ne peut pas faire de Dichotomie dans cet intervalle\n');
end
```

Exécution :

Localisation de la racine géométriquement





Donner la valeur de a=1
Donner la valeur de b=6
Donner la valeur de Nmax=40

Dans l'iteration i=1	la solution est x0=3.500000	f(x0)= 2.795864
Dans l'iteration i=2	la solution est x0=2.250000	f(x0)= 0.072727
Dans l'iteration i=3	la solution est x0=1.625000	f(x0)= -1.089293
Dans l'iteration i=4	la solution est x0=1.937500	f(x0)= -0.605932
Dans l'iteration i=5	la solution est x0=2.093750	f(x0)= -0.284459
Dans l'iteration i=6	la solution est x0=2.171875	f(x0)= -0.109381
Dans l'iteration i=7	la solution est x0=2.210938	f(x0)= -0.019084
Dans l'iteration i=8	la solution est x0=2.230469	f(x0)= 0.026647
Dans l'iteration i=9	la solution est x0=2.220703	f(x0)= 0.003736
Dans l'iteration i=10	la solution est x0=2.215820	f(x0)= -0.007686
Dans l'iteration i=11	la solution est x0=2.218262	f(x0)= -0.001978
Dans l'iteration i=12	la solution est x0=2.219482	f(x0)= 0.000878
Dans l'iteration i=13	la solution est x0=2.218872	f(x0)= -0.000550

La solution finale est x0 = 2.219177
>>

Énoncé du TP N°1

Dans ce TP, il est demandé de trouver la racine de $f(x) = x+exp(x)+\frac{10}{1+x^2} - 5$, 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.

On donne : tolérance $= 10^{-4}$, $a_0 = -1.3$ et $b_0 = 3/2$.

3. La méthode de point fixe

Le principe de cette méthode consiste à transformer l'équation $f(x)=0$ en une équation équivalente $g(x)=x$ où g est une fonction auxiliaire "bien" choisie. Le point $\boldsymbol{\alpha}$ est alors un *point fixe* de g. Approcher les zéros (les raçines) de f revient à approcher les points fixes de g. Le choix de la fonction g est motivé par les exigences du théorème de point fixe. En effet, elle doit être contractante dans un voisinage I de α, ce qui revient à vérifier que $|g'(x)|<1$ sur ce voisinage. Dans ce cas, on construit une suite (x_n) $n\in\mathbb{N}$ définie par:

$$\begin{cases} x_0 \text{ dans un voisinage } \mathbf{I} \text{ de } \alpha \\ \forall\ n \geq 0,\quad x_{n+1} = g(x_n) \end{cases} \qquad \text{(I-6)}$$

Il ne reste plus qu'à appliquer *localement* le *théorème de point fixe* pour démontrer que $\alpha = \lim_{n\to+\infty} x_n$.

La figure ci-dessous illustre le principe de la méthode de point fixe :

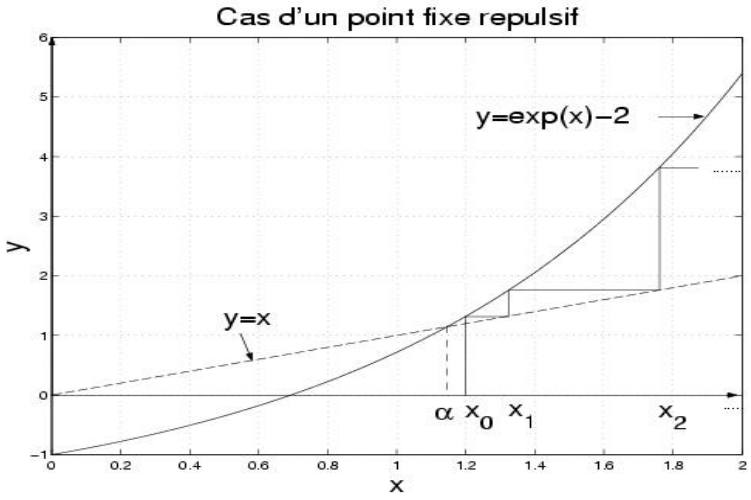


Figure I.2 : Principe de la méthode de point fixe

Le principe de la méthode du point fixe correspond à la recherche du point d'intersection entre les deux fonctions :

- la première fonction est la droite y = x
- la deuxième fonction est y = g(x)
- On choisit initialement un point x_0, qui sera la première estimée de la solution.

On calcule $g(x_0)$, cette dernière valeur sera considérée comme étant la deuxième estimée de la solution soit $x_1 = g(x_0)$ son tour, cette dernière valeur sera injectée dans la fonction $g(x)$ et ainsi de suite jusqu'à la satisfaction d'un critère d'arrêt.

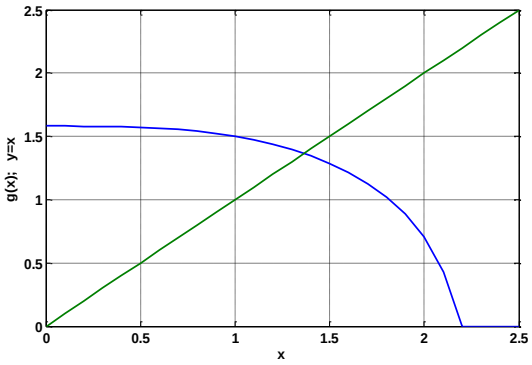
3.1 La condition d'arrêt

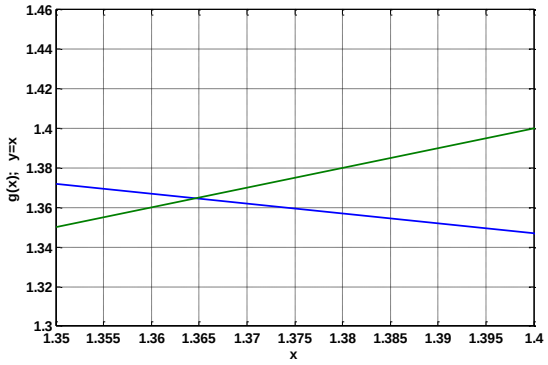
Le critère d'arrêt soit $|x_{n+1} - x_n| < \varepsilon$

3.2 Mise en œuvre de la méthode point fixe sous Matlab :

Dans ce test, nous allons trouver Le point fixe de la fonction $g(x) = \frac{\sqrt{10-x^3}}{2}$ sur l'intervalle [0, 2.5] en utilisant la méthode de point fixe, jusqu'à la convergence avec une précision de 10^{-4} .

```
% La méthode du point fixe ;
clear all,
clc;
x = 0:0.1:2.5;
g=inline('0.5*sqrt(10-x.^3)');
%g=inline('exp(sin(x))');
y=inline('x');
plot(x,g(x),x,y(x),'linewidth',1.5); grid ;
x0 = input('La solution initiale est x0 = ');
eps = input('La tolerence est eps = ');
Nmax = input('Le nmbre maximal d''iterations Nmax = ');
i=1;
err=0.01;
while (i<=Nmax  && err>eps)
    x = x0;
    x0 = g(x);
    err = abs(x0-x);
    if err <=eps
        break;
    end
fprintf('Pour i=%d\t la solution est x0=%f\t avec erreur=%f\n',i,x0,err);
i=i+1;
end
```






```
La solution initiale est x0 = 1
La tolerence est eps = 0.0001
Le nmbre maximal d'iterations Nmax = 40
Pour i=1   la solution est x0=1.500000   avec erreur=0.500000
Pour i=2   la solution est x0=1.286954   avec erreur=0.213046
Pour i=3   la solution est x0=1.402541   avec erreur=0.115587
Pour i=4   la solution est x0=1.345458   avec erreur=0.057082
Pour i=5   la solution est x0=1.375170   avec erreur=0.029712
Pour i=6   la solution est x0=1.360094   avec erreur=0.015076
Pour i=7   la solution est x0=1.367847   avec erreur=0.007753
Pour i=8   la solution est x0=1.363887   avec erreur=0.003960
Pour i=9   la solution est x0=1.365917   avec erreur=0.002030
Pour i=10  la solution est x0=1.364878   avec erreur=0.001039
Pour i=11  la solution est x0=1.365410   avec erreur=0.000532
Pour i=12  la solution est x0=1.365138   avec erreur=0.000272
Pour i=13  la solution est x0=1.365277   avec erreur=0.000139
>>
```

Énoncé du TP N°2

Dans ce TP, 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 $[\frac{-1}{2}, \ 3]$.
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.

Appliquez le même algorithme pour résoudre l'équation : $f_2(x) = x + exp(x) + 1$ avec $x \in [-2, \frac{3}{2}]$. On donne : tolérance $= 10^{-6}$, les valeurs initiales sont $x_0 = 0.8$ pour $f_1(x)$ et $x0 = -1/5$ pour $f_2(x)$.

4 . La méthode de Newton

4.1 Principe de la méthode de Newton

La méthode consiste à introduire une suite (x_n) d'approximation successives de l'équation *f(x) = 0*.

• On part d'un x_0 proche de la solution.

• À partir de x_0, on calcule un nouveau terme x_1 de la manière suivante : on trace la tangente de la courbe f en x_0. Cette tangente coupe l'axe des abscisses en x_1 comme indiqué sur le figure ci-dessous.

• On réitère ce procédé en calculant x_2 en remplaçant x_0 par x_1, puis x_3 en remplaçant x_1 par x_2 et ainsi de suite . . ..

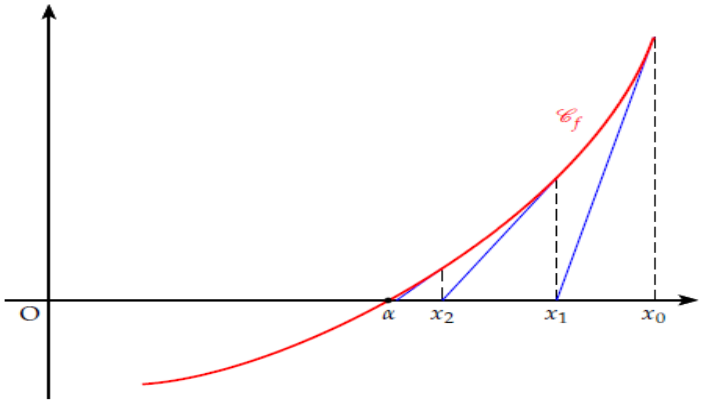


Figure I.3 : Principe de la méthode de Newton

4.2 Formule de recurrence

x_{n+1} est l'abscisse du point d'intersection de la tangente à la courbe de fonction f(x) $\wp_f$ en x_n avec l'axe des abscisses.

L'équation de la tangente en x_n est :

$$y = \acute{f}(x_n)(x - x_n) + f(x_n) \qquad \text{(I-7)}$$

Cette tangente coupe l'axe des abscisses quand y = 0 :

$$\acute{f}(x_n)(x - x_n) + f(x_n) = 0 \Leftrightarrow \acute{f}(x_n)(x - x_n) = -f(x_n) \qquad \text{(I-8)}$$

$$(x - x_n) = -\frac{f(x_n)}{\acute{f}(x_n)} \Leftrightarrow x = x_n - \frac{f(x_n)}{\acute{f}(x_n)} \qquad \text{(I-9)}$$

On a donc la relation de récurrence suivante :

$$x_{n+1} = x_n - \frac{f(x_n)}{\acute{f}(x_n)} \qquad \text{(I-10)}$$

4.3 Conditions d'application:

Pour que la suite (x_n) existe :

• La fonction f doit être dérivable en chacun des points considérés. En pratique la fonction doit être dérivable dans un intervalle centré en $\boldsymbol{\alpha}$ contenant x_0.

• La dérivée ne doit pas s'annuler sur cet intervalle. Pour que la suite (x_n) soit convergente, il faut prendre un x_0 assez proche de la valeur $\boldsymbol{\alpha}$ qui annule la fonction. On le détermine à l'aide du théorème des valeurs intermédiaires.

4.4 La condition d'arrêt :

Lorsque la suite converge, elle converge de façon quadratique c'est à dire que le nombre de chiffres significatifs double à chaque itération. Si l'on s'en tient à une précision inférieure à 10^{-15}, la suite doit alors converger en moins de 10 itérations. On pourra mettre une condition d'arrêt de l'algorithme lorsque le nombre de boucle dépassera 10 car alors la suite ne converge pas. Il faudra alors prendre un x_0 plus proche de $\boldsymbol{\alpha}$.

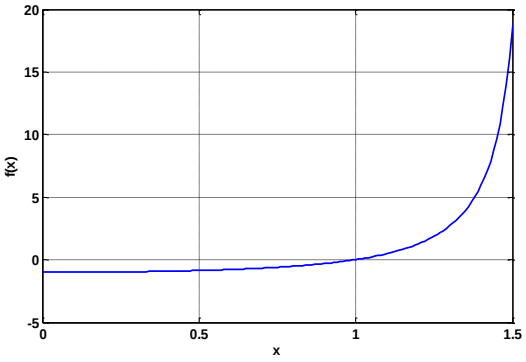
On prendra comme critère d'arrêt pour une précision de ε : $\left|\frac{f(x_n)}{\acute{f}(x_n)}\right| < \varepsilon$.

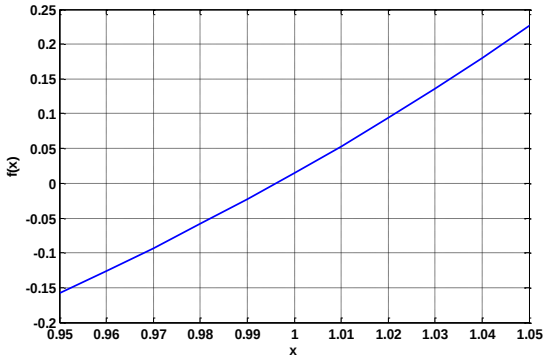
4.5 Mise en œuvre de la méthode Newton sous Matlab :

Dans ce test, nous allons trouver la racine de la fonction $f(x) = x * \tan(x) - \operatorname{csch}(x)$ sur l'intervalle [0, 1.5] en utilisant la méthode de Newton, jusqu'à la convergence avec une précision de 10^{-4} .

```
% La méthode de Newton
clear all; close all; clc;
a=input('Donner a= ');
b=input('Donner b= ');
eps=input('Donner eps= ');
Nmax=input('Pour la boucle for, donner Nmax= ');
x=0:0.01:1.5;
f=inline('x.*tan(x)-cosh(x)');
plot(x,f(x));grid
df=inline('tan(x) - sinh(x) + x.*(tan(x)^2 + 1)');
ddf=inline('2*tan(x)^2 - cosh(x) + 2*x.*tan(x).*(tan(x)^2 + 1) + 2');
if (df(a)== 0 && df(b)== 0 || f(a)*f(b)>0)
    disp('On ne peut pas calculer la racine');
end
if (f(a)*ddf(a)>0)
    x0=a;
else
    x0=b;
```

```
end
for i=1:Nmax
    x0 = x0 - f(x0)/df(x0);
    err=abs(f(x0)/df(x0));
   if err<=eps
      fprintf('i=%d\t racine=%2.6f\t f(x0)=%2.6f\t err=%2.6f\n',i,x0,f(x0),err);
        break;
   end
end
```






```
Donner a= 0.5
Donner b= 3
Donner eps= 0.0001
Pour la boucle for, donner Nmax= 40
On ne peut pas calculer la racine
i=8    racine=0.996253     f(x0)=0.000172      err=0.000046
```

Énoncé du TP N°3

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 $f'(x) = 0 \Rightarrow$ 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 .

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

5 Méthode de Lagrange

5.1 Description de la méthode

Cette méthode est également appelée méthode de la sécante, méthode des parties proportionnelles ou encore regula falsi. On considère un intervalle [a, b] et une fonction f de

classe C^2 de [a, b] dans $\mathbb{R}$. On suppose que $f(a)*f(b) < 0$ et que $\acute{f}$ ne s'annule pas sur [a, b], alors l'équation $f(x) = 0$ admet une unique solution sur l'intervalle [a, b].

La méthode de la Lagrange consiste à construire une suite (x_n) qui converge vers α de la manière suivante : soit D_0 la droite passant par $(a, f(a))$ et $(b, f(b))$, elle coupe l'axe Ox en un point d'abscisse $x_0 \in]a, b[$. On approche donc la fonction f par un polynôme P de degré 1 et on résout $P(x) = 0$.

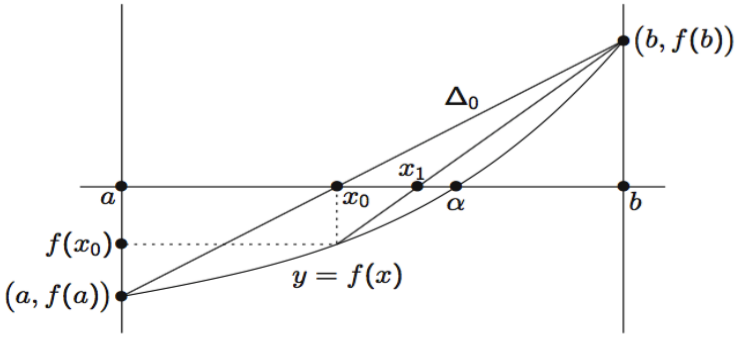


Figure I.4 : Principe de la méthode de Lagrange

Ensuite, suivant la position de α par rapport à x_0, on considère la droite passant par $(a, f(a))$ et $(x_0, f(x_0))$ si $f(x_0)*f(\alpha) < 0$ ou celle passant par $(x_0, f(x_0))$ et $(b, f(b))$ si $f(x_0)*f(b) < 0$.

On appelle x_1 l'abscisse du point d'intersection de cette droite avec l'axe Ox. On réitère ensuite le procédé. Plaçons-nous dans le cas où $\acute{f} > 0$ est dérivable et f est convexe ($\acute{f} \geq 0$), c'est-à-dire que sa représentation graphique est au dessus des tangentes et en dessous des cordes. Alors la suite (x_n) est définie par :

$$\begin{cases} x_0 = \alpha \\ x_{n+1} = \frac{bf(x_n) - x_n f(b)}{f(x_n) - f(b)} \end{cases} \qquad \text{(I-11)}$$

En effet, si f est convexe, on remplace l'intervalle $[x_n, b]$ par l'intervalle $[x_{n+1}, b]$. L'équation d'une droit passant par $(c, f(c))$ et $(b, f(b))$ avec c ≠b est :

$$y - f(b) = \frac{f(b) - f(c)}{b - c}(x - b) \qquad \text{(I-12)}$$

On cherche son intersection avec l'axe Ox donc on prend $y = 0$ et on obtient la formule donnée plus haut. On pose

$$g(x) = \frac{bf(x) - xf(b)}{f(x) - f(b)} = x - \frac{b - x}{f(b) - f(x)} f(x) \qquad \text{(I-13)}$$

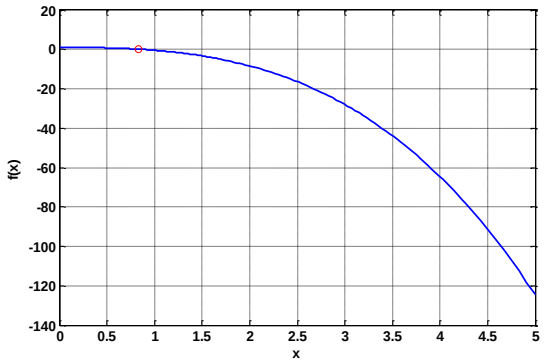
5.2 La condition d'arrêt

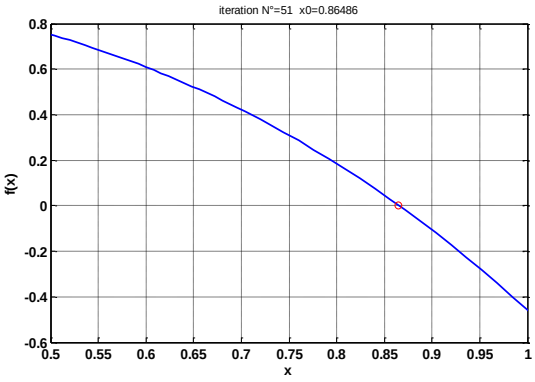
Le critère d'arrêt soit $|x_{n+1} - x_n| < \varepsilon$

5.3 Mise en œuvre de la méthode de Lagrange sous Matlab :

Dans ce test, nous allons trouver la racine de la fonction $f(x) = \cos(x) - x^3$ sur l'intervalle $[0, 5]$ en utilisant la méthode de Lagrange, jusqu'à la convergence avec une précision de 10^{-4} .

```
% la méthode de lagrange
clear all; close all; clc;
% les parametres d'entrée
a=input('donner a= ');
b=input('donner b= ');
eps=input('donner eps= ');

f=inline('cos(x)-x^3');
df=inline('-sin(x)-3*x^2');
ddf=inline('-cos(x)-6*x');

err=1; i=0;
% le programme principal
if (df(a)== 0) && (df(b)== 0)
disp('On peut pas calculer la racine');
end
% initialisation de x0
if (f(a)*ddf(a)<0)
x0=a;
c=b;
else x0=b;
    c=a;
end
    while (err >= eps)

       % x1 = x0 - (f(x0)/df(x0));
       % err=abs(f(x0)/df(x0));

        x0=(f(x0)*c-x0*f(c))/(f(x0)-f(c));
        err=abs(x0-(f(x0)*c-x0*f(c))/(f(x0)-f(c)));

           % affichage des resultats
fprintf('i=%d\t racine=%2.8f\t f(x0)=%2.8f\t err=%2.8f\t \n',i,x0,f(x0),err);
           % x0 prend la valeur de x1
        %x0 = x1;
           % affichage de la fonction
        fplot(f,[a b]);
        hold all;
        plot(x0,f(x0),'ro');
                   xlabel('x'),ylabel('f');
                   title(['iteration N°=',int2str(i),'  x0=',num2str(x0),]);
        hold off; grid on;
        pause(1)
         i=i+1;
    end
```





donner a= 0
donner b= 4
donner eps= 0.0001

i=0 racine=0.06092579 f(x0)=0.99791844 err=0.05987481
i=1 racine=0.12080060 f(x0)=0.99094966 err=0.05855915
i=2 racine=0.17935974 f(x0)=0.97818813 err=0.05694348
i=3 racine=0.23630323 f(x0)=0.95901508 err=0.05501137
i=4 racine=0.29131459 f(x0)=0.93314496 err=0.05276583
. . . .
. . . .
. . .
racine=0.86475821 f(x0)=0.00215208 err=0.00010436
i=51 racine=0.86486257 f(x0)=0.00183852 err=0.00008915

Énoncé du TP N°4

Il vous est demandé de trouver la racine de l'équation : $f(x) = x - 0.2\sin(4x) - \frac{1}{2}$. Prenez $x_0 = -1$ et $x_1 = 2$ comme valeurs initiales. Écrire un code matlab, portant sur l'implémentation de la méthode de Lagrange en considérant une tolérance :

$tol = 10^{-4}$.

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 $x(k+1)$.

Chapitre II

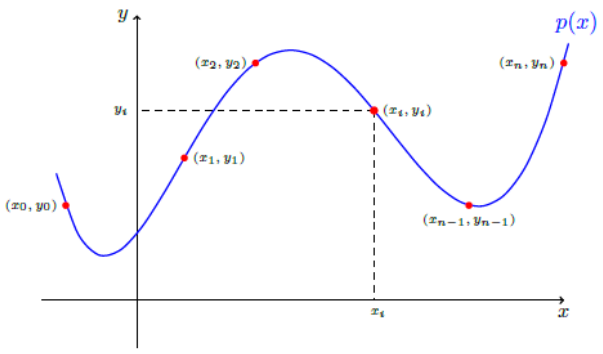
Interpolation et approximation polynômiale de fonctions

Chapitre 2 : Interpolation et approximation polynômiale de fonctions

II. 1 Introduction

Les fonctions les plus faciles à évaluer numériquement sont les fonctions polynômes. Il est donc important de savoir approximer une fonction arbitraire par des polynômes.

L'interpolation polynomiale est l'une des méthodes d'approximation d'une fonction ou d'un ensemble de données par un polynôme. En d'autres termes, étant donné un ensemble de n+1 points $\{(xi; yi)\ ;\ i = 0; \dots ; n\}$ (obtenu, par exemple, à la suite d'une expérience), on cherche un polynôme p qui passe par tous ces points, c'est-à-dire $p\ (x_i) = y_i;\ \ i = 0; \dots ; n$, et éventuellement vérifie d'autres conditions, de degré si possible le plus bas. (Fig II.1).



Si un tel polynôme existe, il est appelé polynôme d'interpolation ou polynôme interpolant.

Les abscisses x_i sont appelles nœuds d'interpolation tandis que les couples *(xi; yi)* sont appelés points d'interpolation, ou points de collocation.

II.2 Interpolation de Lagrange

L'approche de Lagrange pour construire le polynôme d'interpolation est simple, systématique, on se donne *n+1* points $(x_0; y_0)\ ;\ \dots\ ;\ (x_n; y_n)$, avec les x_i distincts deux à deux. Le polynôme d'interpolation sous la forme de Lagrange est une combinaison linéaire :

$$L(x) = y_0 l_0(x) + y_1 l_1(x) + \cdots + y_n l_n(x) = \sum_{i=0}^{n} y_i l_i(x) \qquad \text{(II-1)}$$

de polynômes de base de Lagrange

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j} = \frac{x-x_0}{x_i-x_0} \dots \frac{x-x_{i-1}}{x_i-x_{i-1}}\ \frac{x-x_{i+1}}{x_i-x_{i+1}}\ \dots\ \frac{x-x_n}{x_i-x_n} \qquad \text{(II-2)}$$

Où $i = 0; \ldots ; n$. Noter comment, étant donné l'hypothèse initiale qu'il n'y a pas de deux x_j identiques, alors lorsque $i \neq j$, $(x_i - x_j) \neq 0$, cette expression est toujours bien définie. Pour tout $j \neq i$, $l_i(x)$ inclut le terme (x - x_j) dans le numérateur, donc le produit entier sera nul à $x = x_j$ c'est-à-dire

$$l_i(x) = 0 \text{ pour tout } j \neq i$$

D'autre part :

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x_i - x_j}{x_i - x_j} = 1 \qquad \text{(II-3)}$$

Il s'ensuit que $y_i l_i(x) = y_i$ donc à chaque point x_i, $L(x_i) = 0 + \ldots + 0 + y_i + 0 + \ldots + 0 = y_i$, ceci montrant que L interpole exactement les points $(x_0; y_0) ; \ldots ; (x_n; y_n)$. Il est donc important de remarquer que le problème d'interpolation à une unique solution explicite L.

De tout ce qui précède, nous avons le théorème suivant.

Théorème (Interpolation de Lagrange)

Etant donné n+1 points $(x_0; y_0) ; \ldots ; (x_n; y_n)$. Si les x_i sont distincts deux à deux, alors il existe un unique polynôme p_n de degré inferieur ou égal à n tel que $p_n(x_i) = y_i; i = 0; \ldots ; n$ qu'on peut écrire sous la forme dite de Lagrange :

$$p_n(x) = \sum_{i=0}^{n} y_i l_i(x) \quad \text{où} \quad l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \qquad \text{(II-4)}$$

II.3 Mise en œuvre sous Matlab: la méthode d'interpolation de Lagrange

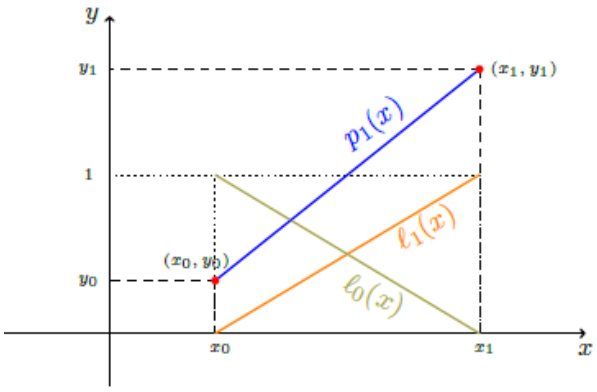
```
function p = polynome_lagrange(xi, fxi, x)
% Polynôme d'interpolation de Lagrange
% Entré:
%   1)xi:  Un vecteur ligne contenant les abscisses xi.
%   2)fxi: Un vecteur ligne contenant les valeurs f(xi)
%          de la fonction A  interpolé pour les xi correspondants.
%   3)x :  Un point en lequel Ã  évaluer le polynôme d'interpolation.
% Sortie:
%   1) p : La valeur du polynôme de Lagrange en x.
% Exemple d'appel:
   xi=[-1 0 1 2 3] ;
   fxi=[-2 -1 0 3 2] ;
   p = polynome_lagrange(xi, fxi, 2.5) ;
n = length(xi);
    for i = 1 : n
        if x==xi(i)  %Vérifier si x est égal Ã  l'un des xi.
            p=fxi(i)
            return
        end
    end
d = zeros(1,n);
    for i=1:n
```

```
      d(i)=1;
       for j=1:n
         if i~=j
          d(i) = d(i)*(xi(i)-xi(j));  %Calcule des dominateurs de Li.
         end
       end
    end
num=1;
   for i=1:n
    num = num*(x - xi(i));   %Calcul du produit de (x-xi).
   end
p=0;
  for i=1:n
    q=num/(x-xi(i));
    p = p+fxi(i)*q/d(i);   %Evaluation de pn(x).
  end
end.
```

Exemple 1

Pour *n = 1* le polynôme de Lagrange s'écrit :

$$p_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}$$

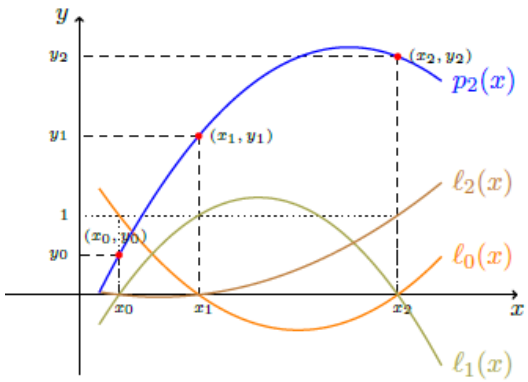


C'est l'équation de la droite qui passe par les points *(x0; y0) et (x1; y1).*

Exemple 2

Pour *n = 2* le polynôme de Lagrange s'écrit :

$$p_2(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$



C'est l'équation de la droite qui passe par les points *(x0; y0), (x1; y1) et (x2; y2).*

Soit $f: \mathbb{R} \Rightarrow \mathbb{R}$ une fonction continue et soit $x_0; x_1; \ldots ; x_n$, $(n + 1)$ points distincts deux à deux. Interpoler la fonction f aux points $x_i; i = 0; \ldots ; n$ signifie chercher un polynôme p_n de degré $\boldsymbol{n}$ tel que :

$$p_n(x_i) = f(x_i) \quad pour \quad i = 0, \ldots, n$$

La solution de ce problème est donc donnée par :

$$p_n(x) = \sum_{i=0}^{n} f(x_i) l_i(x) \quad où \quad l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

et le polynôme $p_n(x)$ est appelle interpolant de f de degré n aux points $x_0; x_1; \ldots ; x_n$.

II.4 Enoncé du TP N°1

Le but de ce TP est l'implémentation des algorithmes d'interpolation étudies au cours de méthodes numériques, sous Matlab, il sera ensuite question d'étudier un phénomène qui se produit lorsque l'on augmente le nombre de points de collocation.

❖ Déterminer le polynôme d'interpolation $\boldsymbol{p_3}$ qui interpole les points $(x_i; y_i)$ suivantes : (0; 1) ; (1; 3) ; (3; 0) et (4; 5) :

➢ Le polynôme de Lagrange s'écrit sous la forme

$$p_3(x) = \sum_{i=0}^{3} y_i \, l_i(x) = l_0(x) + 3l_1(x) + 5l_3(x)$$

Où

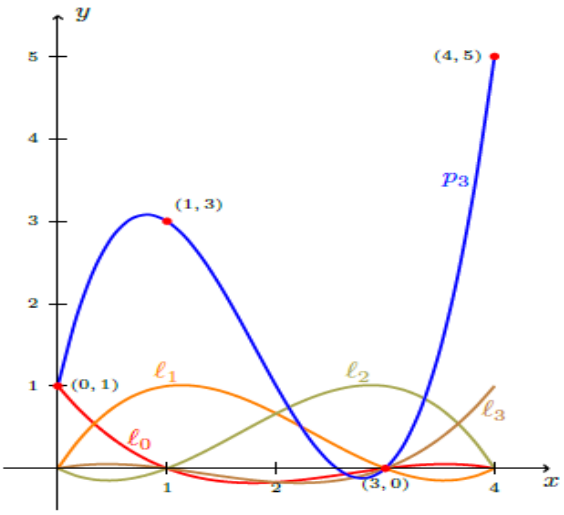
$$l_0(x) = \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} = \frac{-1}{12}(x - 1)(x - 3)(x - 4)$$

$$l_1(x) = \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} = \frac{1}{6}x(x - 3)(x - 4)$$

$$l_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = \frac{1}{12}x(x-1)(x-3)$$

$$p_3(x) = \frac{-1}{12}(x-1)(x-3)(x-4) + \frac{1}{2}x(x-3)(x-4) + \frac{5}{12}x(x-1)(x-3)$$

$$p_3(x) = \frac{5}{6}x^3 - \frac{9}{2}x^2 + \frac{17}{3}x + 1$$



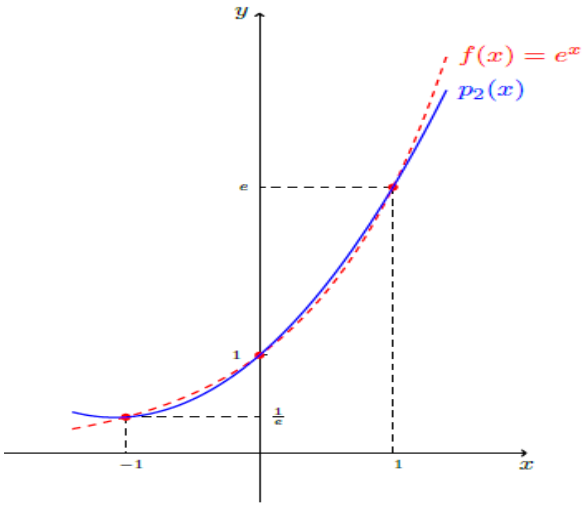
❖ Soit $f: \mathbb{R} \Rightarrow \mathbb{R}$ la fonction définie par $f(x) = e^x$.

Cherchons l'interpolant de f aux points : -1; 0; 1.

$$p_2(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

$$= e^{-1}\frac{x(x-1)}{2} + \frac{(x+1)(x-1)}{-1} + e\frac{(x+1)x}{2}$$

$$= \left(\frac{e}{2} - 1 + \frac{1}{2e}\right)x^2 + \left(\frac{e}{2} - \frac{1}{2e}\right)x + 1$$



❖ 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.

La méthode d'interpolation de Lagrange présente un inconvénient majeur, elle n'est pas récursive. En effet, si l'on souhaite passer d'un polynôme p_n de degré n à un polynôme p_{n+1} de degré (n + 1), en ajoutant un point de collocation, on doit reprendre pratiquement tout le processus à zéro. Il est intéressant donc de mettre p_n sous une forme récurrente qui permet de compléter les valeurs déjà obtenues sans refaire tous les calculs. On arrive donc au polynôme d'interpolation de Newton.

II. 5 Interpolation de Newton

L'interpolation de Newton est une méthode ne diffère de l'interpolation lagrangienne que par la façon dont le polynôme est calcule, le polynôme d'interpolation qui en résulte est le même. Pour cette raison on parle aussi plutôt de la forme de Newton du polynôme de Lagrange.
Soit $f : \mathbb{R} \Rightarrow \mathbb{R}$ une fonction continue et soit $x_0; x_1; \ldots; x_n$, (n + 1) points distincts deux à deux.
Posons $y_i = f(x_i)$; $i = 0; \ldots; n$.
L'interpolation de Newton est un procède itératif qui permet de calculer le polynôme d'interpolation de Lagrange p_n de degré n basé sur (n + 1) points $\{(x_i; y_i)$; $i = 0; \ldots; n\}$ à partir du polynôme d'interpolation p_{n-1} de degré n-1 basé sur n $\{(x_i; y_i)$; $i = 0; \ldots; n-1\}$, en posant

$$p_n(x) = p_{n-1}(x) + C_n(x) \qquad \text{(II-5)}$$

Ce procède permet de construire le polynôme d'interpolation sous la forme de Newton comme une combinaison linéaire :

$$N(x) = a_0N_0(x) + a_1N_1(x) + \cdots + a_nN_n(x) = \sum_{i=0}^{n} a_iN_i(x) \qquad \text{(II-6)}$$

Le polynôme de base de Newton

$$N_i(x) = (x - x_0)(x - x_1) \dots (x - x_{i-1}) = \prod_{0\le j<i}(x - x_j) \qquad \text{(II-7)}$$

pour i=0, … , n

Il est facile de vérifier que la famille $\{N_i;\ i = 0;\ \_\ \_\ \_\ ;\ n\}$ forme une base de P_n l'espace des polynômes de degré inferieur ou égal à n. Le problème du calcul du polynôme d'interpolation p_n est alors ramenée au calcul des coefficients $\{a_i;\ i = 0;\ \dots\ ;\ n\}$ tels que

$$p_n(x) = \sum_{i=0}^{n} a_iN_i(x) \qquad \text{(II-8)}$$

Noter que si on calcule les n + 1 coefficients $\{a_i;\ i = 0;\ \dots\ ;\ n\}$ et après on ajoute un point d'interpolation, il n'y a plus à calculer que le coefficient a_{n+1} car la nouvelle base est déduite de l'autre base en ajoutant le polynôme N_{n+1}.

Pour calculer les coefficients $\{a_i;\ i = 0;\ \dots\ ;\ n\}$ on doit donc s'assurer que :

$$p_n(x_i) = f(x_i) \quad \text{pour i=0, … , n}$$

Le polynôme d'interpolation dans la base de Newton évalue en x_0 donne :

$$p_n(x_0) = \sum_{i=0}^{n} a_iN_i(x_0) = a_0 = f(x_0) \qquad \text{(II-9)}$$

Le premier coefficient est donc

$$a_0 = f(x_0)$$

On doit ensuite s'assurer que $p_n(x_1) = f(x_1)$, c'est-à-dire :

$$a_0 = a_1(x_1 - x_0) = f(x_1)$$

En résumé la formule du polynôme d'interpolation de Newton d'ordre *n-1* s'écrit :

$$f(x) = a_1 + a_2(x - x_1) + \cdots + a_n(x - x_1)(x - x_2) \dots (x - x_{n-1})$$

Ce polynôme est le plus populaire car il présente de nombreux avantages :

- Les coefficients $a_1; \dots; a_n$ sont déterminés par une procédure simple,
- Les points peuvent être dans n'importe quel ordre (pas obligatoirement ascendant ou descendant)

- Si on ajoute un point, on n'a pas besoin de recalculer tous les coefficients, on calcul un seul coefficient uniquement.

Le calcul d'un coefficient du polynôme de Newton se donne par :

$$a_n = \frac{f(x_n,x_{n-1},\dots,x_2)-f(x_k,x_{n-1},\dots,x_1)}{x_n-x_1} \qquad \text{(II-10)}$$

II. 6 Mise en œuvre sous Matlab: la méthode d'interpolation de Newton

```
function [d, a, p] = polynome_newton(xi, fxi, x)
% Polynôme d'interpolation de Newton
% Entré:
%   xi: Un vecteur ligne contenant les abscisses xi.
%   fxi: Un vecteur ligne contenant les valeurs f(xi)
%        de la fonction à interpoler pour les xi correspondants.
%   x: Un point en lequel à évaluer le polynôme d'interpolation.
%      Sortie:
%   d : Matrice contenant la table de différences divisées.
%   a : Vecteur contenant les coefficients du polynôme de Newton.
%   p : La valeur du polynôme de Newton en x.
%   Exemple d'appel:
 xi=[-1 0 1 2 3]
 fxi=[-2 -1 0 3 2]
 [d, a, p] = polynome_newton(xi, fxi, 2.5)
n = length(xi);
d = zeros(n,n);
a = zeros(1,n);
d(:,1) = fxi;
a(1) = d(1,1);
  for i=2:n
       for j=2:i
         d(i,j) = (d(i,j-1)-d(i-1,j-1))/(xi(i)-xi(i-j+1));
       end
       a(i) = d(i,i);
  end
p=a(n);
  for i=n-1:-1:1
    p = p*(x - xi(i))+a(i);
  end
```

II.7 Enoncé du TP N°2

Construire, selon la méthode de Newton, 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 Newton. Déterminer ce polynôme.
2. Tracer, sur la même figure, le polynôme et les points d'interpolation.

Chapitre III

Intégrations numériques

Chapitre 3 : Intégrations Numériques

III. 1 Introduction

L'intégration numérique est le calcul par des méthodes numériques l'intégrale :

$$I(f) = \int f(x)\,dx \qquad \text{(III-1)}$$

où $f(x)$ est une fonction connue seulement en quelques points ou encore une fonction n'ayant pas de primitive.

D'après l'approche de l'interpolation polynomiale, toute fonction $f(x)$ peut s'écrire :

$$f(x) = P_n(x) + E_n(x) \qquad \text{(III-2)}$$

où $Pn(x)$ est un polynôme d'interpolation de degré n, sa formule générale est donnée par :

$$P_n(x) = a_0 + a_1x + a_2x^2 + +a_3x^3 + \cdots + +a_nx^n \quad (a_n \neq 0) \qquad \text{(III-3)}$$

$E_n(x)$ est l'erreur d'interpolation d'ordre $(n+1)$ définie par :

$$E_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n) \qquad \text{(III-4)}$$

Où $x_1, x_2, x_3, \dots, xn$ sont les points d'interpolation et $\xi \in [x_0, x_n]$.

En effet, l'intégration numérique est basée principalement sur la relation :

$$I(f) = \int_{x_0}^{x_n} f(x)dx = \int_{x_0}^{x_n} p_n(x)dx + \int_{x_0}^{x_n} E_n(x)dx \qquad \text{(III-5)}$$

En faisant varier la valeur de n, on obtient les *formules de Newton-Cotes*. La précision sur la valeur de l'intégrale dépend alors de la valeur de n, en effet plus n est grande plus la précision est élevée. Dans ce chapitre nous allons présenter deux méthodes numériques : la *méthode de trapèzes* et la *méthode de Simpson*.

III. 2 Méthode de trapèzes

Formule de la méthode de trapèzes

Considérons que la fonction $f(x)$ est connue seulement en deux points $(a, f(a))$ et $(b, f(b))$, elle peut être alors écrit par :

$$f(x) = P_1(x) + E_1(x) \qquad \text{(III-6)}$$

où $P_1(x)$ est un polynôme de degré 1 qui est représenté par une droite, $E_1(x)$ est l'erreur d'interpolation d'ordre 2.

On peut approximativement écrire :

$$\int_a^b f(x)dx \approx \int_a^b p_1(x)dx \qquad \text{(III-7)}$$

Graphiquement $\int_a^b f(x)\,dx$ est l'aire délimité par les axes $x=a$, $x=b$, $y=0$ et la courbe de $f(x)$, alors que $\int_a^b p(x)$ est l'aire délimité par les mêmes axes et la courbe de la droite $P_1(x)$, qui a la forme d'un trapèze (voir Figure III.1).

On utilise le polynôme de Newton de degré 1 définie par :

$$p_1(x) = \alpha_0 + \alpha_1(x-a) \quad \text{avec} \quad \alpha_0 = f(a) \;,\; \alpha_1 = \frac{f(b)-f(a)}{b-a} \qquad \text{)}$$

Il vient alors :

$$\int_a^b f(x)dx \approx \int_a^b p_1(x)dx = \int_a^b [\alpha_0 + \alpha_1(x-a)]dx \qquad \text{(III-9)}$$

Par un calcul on déduit que :

$$\int_a^b f(x)dx \approx \frac{h}{2}\big(f(a)+f(b)\big) \text{ avec } h = b-a \qquad \text{(III-10)}$$

Remarquons que le résultat de cette intégrale est bien une surface d'un trapèze.

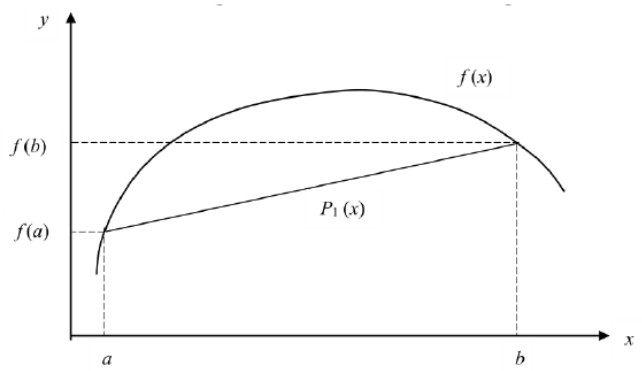


Figure III.1 : Illustration graphique d'une intégrale approximée par un seul trapèze

Pour augmenter la précision de cette approximation on divise l'intervalle $[a, b]$ en n sous-intervalles de même longueur h, et on calcul la somme de toutes les intégrales correspondent aux sous-intervalles. On note les abscisses obtenues par : x_0 ,x_1 ,x_2 ,...,x_l ,x_{l+1} ,...,xn où $a=x_0$ et $b=xn$. Cette nouvelle approximation est illustrée sur la Figure III.2. Il vient alors :

$$\int_a^b f(x)dx = \sum_{l=0}^{n-1}\int_{x_l}^{x_{l+1}} f(x)dx \approx \sum_{l=0}^{n-1}\frac{h}{2}[f(x_l)+f(x_{l+1})] \qquad \text{(III-11)}$$

Après un calcul on déduit la formule de la méthode de Trapèzes suivante :

$$\int_a^b f(x)dx \approx \frac{h}{2}\left(f(x_0)+f(x_n)+2\sum_{i=1}^{n-1} f(x_i)\right) \qquad \text{(III-12)}$$

Avec : $h=\frac{b-a}{n}$, $x_i = a+ih$, $x_0 = a$, $x_n = b$, $i = 0,1,2,\dots,n$

Graphiquement, on remarque que plus le nombre n de sous-intervalles est grande, plus la précision sur la valeur approchée de l'intégrale est grande.

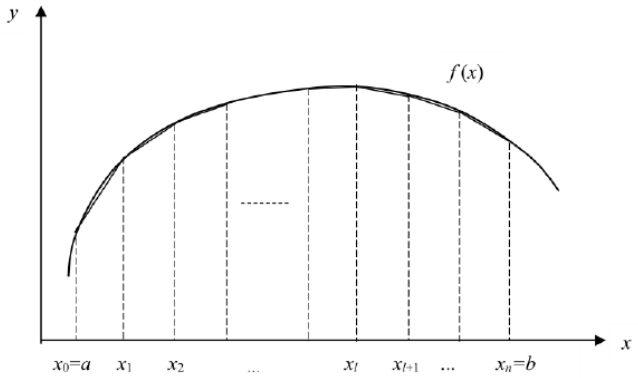


Figure III.2 : Illustration graphique d'une intégrale approximée par plusieurs trapèzes

III.3 Mise en œuvre par MATLAB la méthode des trapèzes

1) En utilisant l'algorithme de la méthode des trapèzes, écrire un programme Matlab qui permet de calculer l'intégrale $I(f) = \int_0^5 e^{\sin(x)}\,dx$ en utilisant la méthode de trapèzes pour n=5.

2) Exécuter ce programme pour n=20 ,40 ,80 ,200 et donner les valeurs approchées de $I(f)$, Conclure.

3) Matlab dispose d'une commande prédéfinie permettant le calcul numérique de l'intégrale d'une fonction $f(x)$ dans un intervalle $[a\ ,b]$ par la méthode des trapèzes, sa syntaxe est :

I=trapz(x, f), où x est un vecteur ligne dont les valeurs sont comprises entre a et b avec un pas h.

En utilisant cette commande calculer une valeur approchée de $I(f)$.

- 1) Le programme MATLAB de la méthode des trapèzes :

```
clear all;close all;clc
a=0;
b=5;
n=5;
h=(b-a)/n;
f=@(x)exp(sin(x));
s=0;
```

```
for i=1:n-1
s=s+f(a+i*h);
end
I=(h/2)*(f(a)+f(b))+h*s
```

Après exécution du programme on obtient : n=10 ; I= 7.1705

- 2) En utilisant le programme réalisé, on obtient :

n	5	10	20	40	80	200	500
I	7.1147	7.1705	7.1845	7.1880	7.1888	7.1891	7.1891

On conclut que la valeur approchée de l'intégrale converge vers la valeur I=7.1891 lorsque n augmente plus en plus.

- 3) On utilise la fonction prédéfinie de la méthode de trapèzes :

```
a=0;
b=5;
n=5
h=(b-a)/n;
x=a:h:b;
y=exp(sin(x));
I=trapz(x,y)
```

Après exécution on trouve les mêmes résultats précédents :

n	5	10	20	40	80	200	500
I	7.1147	7.1705	7.1845	7.1880	7.1888	7.1891	7.1891

On remarque que cette commande donne les mêmes résultats obtenus par le programme précédent.

III.4 Méthode de Simpson

Formule de la méthode de Simpson

Reprenons le raisonnement utilisé dans la méthode des trapèzes, mais cette fois en utilisant un polynôme de degré 2. Considérons maintenant que la fonction $f(x)$ est connue seulement en trois points $(a, f(a))$, $(c, f(c))$ et $(b, f(b))$ où les abscisses a, c et b sont également distancées, elle peut être alors écrit par :

$$f(x)=P_2(x)+E_2(x) \qquad \text{(III-13)}$$

où $P_2(x)$ est un polynôme de degré 2 qui est représenté par un parabole, $E_2(x)$ est l'erreur d'interpolation d'ordre 3.

On peut approximativement écrire :

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx \qquad \text{(III-14)}$$

Graphiquement $\int_a^b f(x)\,dx$ est l'aire délimité par les axes $x=a$, $x=b$, $y=0$ et la courbe de $f(x)$, alors que $\int_a^b p_2(x)\,dx$ est l'aire délimité par les mêmes axes et la courbe parabolique de $P_2(x)$.

On utilise le polynôme de Newton de degré 2 définie par :

$$p_2(x)=\alpha_0+\alpha_1(x-a)+\alpha_2(x-a)(x-c) \qquad \text{(III-15)}$$

Avec :

$$\alpha_0 = f(a)\,,\alpha_1 = \frac{f(c)-f(a)}{h}\,,\alpha_2 = \frac{f(b)-2f(c)+f(a)}{2h^2} \qquad \text{(III-16)}$$

Il vient alors :

$$\int_a^b f(x)dx \approx \int_a^b p_2(x)dx = \int_a^b [\alpha_0 + \alpha_1(x-a) + \alpha_2(x-a)(x-c)]dx \qquad \text{(III-17)}$$

On déduit alors :

$$\int_a^b f(x)dx \approx \frac{h}{3}\big(f(a)+4f(c)+f(b)\big) \text{ avec } h = b-c = c-a = \frac{b-a}{2} \qquad \text{(III-18)}$$

Pour augmenter la précision de cette approximation on divise l'intervalle $[a, b]$ en $2n$ sous-intervalles de même longueur h, et on calcul la somme de toutes les intégrales correspondent aux sous-intervalles. On note les abscisses obtenues par : $x_0\,,x_1\,,x_2\,,\ldots,x_{2l}\,,x_{2l+1}\,,x_{2l+2}\,,\ldots,x_{2n}$ où $a=x_0$ et $b=x_{2n}$. Il vient alors :

$$\int_a^b f(x)dx = \sum_{l=0}^{n-1}\int_{x_{2l}}^{x_{2l+2}} f(x)dx \simeq \sum_{l=0}^{n-1}\frac{h}{3}[f(x_{2l})+4f(x_{2l+1})+f(x_{2l+2})] \qquad \text{(III-19)}$$

On déduit la formule de la méthode de Simpson suivante :

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(a)+f(b)+2[f(x_2)+f(x_4)+\cdots+f(x_{2n-2})] + 4[f(x_1)+f(x_3)+\cdots+f(x_{2n-1})]) \qquad \text{(III-20)}$$

Avec : $h=\dfrac{b-a}{2n}$, $x_i = a+ih\,,x_0 = a\,,x_{2n} = b\,,i = 0,1,2,\ldots,2n-1,2n$

Plus la valeur de sous-intervalles $2n$ est grande, plus la précision sur la valeur approchée de l'intégrale est grande.

III.5 mise en ouvre par MATLAB la méthode de Simpson

1) Ecrire un algorithme de calcul pour la méthode de Simpson.

2) En utilisant l'algorithme de la méthode de Simpson, écrire un programme Matlab qui permet de calculer l'intégrale $I(f) = \int_0^5 e^{\sin(x)}\, dx$ en utilisant la méthode de Simpson pour n=6.

3) Exécuter ce programme pour n=20 ,40 ,80 ,200 et donner les valeurs approchées l'intégrale $I(f)$. Conclure.

4) Matlab dispose d'une commande prédéfinie permettant le calcul numérique de l'intégrale d'une fonction $f(x)$ dans un intervalle $[a\ ,b]$ par la méthode de Simpson, sa syntaxe est : quad(f,a,b)

En utilisant cette commande, calculer une valeur approchée de I.

- 1) le programme MATLAB de la méthode de Simpson :

```
clear all;close all;clc;
% n doit etre pair
a=0;
b=5;
n=10;
h=(b-a)/n;
f=@ (x)exp(sin(x));
s1=0;
for i=1:2:n-1
s1=s1+f(a+i*h);
end
s2=0;
for i=2:2:n-2
s2=s2+f(a+i*h);
end
I=(h/3)*(f(a)+f(b)+4*s1+2*s2)
```

Après exécution on obtient : n=10, I=7.3387

- 2) On change la valeur n on obtient :

n	4	10	20	40	80	200	500
I	7.3387	7.1891	7.1891	7.1891	7.1891	7.1891	7.1891

On conclut que la valeur approchée de l'intégrale converge vers la valeur I=7.1891 lorsque n augmente plus en plus. De plus, la convergence de la méthode de Simpson est rapide comparée à la convergence de la méthode de trapèzes.

- 3) On utilise la fonction prédéfinie de la méthode de Simpson :

>> I=quad(@(x)exp(sin(x)),0,5)

Après exécution on trouve : I=7.1891

III.6 Enoncé du TP :

Soit l'intégrale : $I(f) = \int_0^5 e^{\sin(x)}\, dx$

1) Quant on fait la programmation des méthodes numérique de Trapèzes et de Simpson

2) Ecrire un programme Matlab qui permet de calculer $I(f)$ par la méthode de Trapèzes. En prenant $n = 10$.

3) Exécuter ce programme pour $n = 20$, 40 , 80 , 200 et donner les valeurs approchées de $I(f)$. Conclure.

4) Donner une valeur approchée de $I(f)$ en utilisant la fonction Matlab prédéfinie de la méthode de Trapèzes.

5) Refaire les questions précédentes en utilisant la méthode de Simpson.

Chapitre IV

Résolution Numérique Des Equations Différentielles

Chapitre 4 : Résolution Numérique Des Equations Différentielles

IV. 1 Introduction

La résolution numérique des équations différentielles est probablement le domaine de l'analyse numérique où les applications sont les plus nombreuses. Que ce soit en mécanique des fluides, en transfert de chaleur ou en analyse de structure, on aboutit souvent à la résolution d'équations différentielles, de systèmes d'équations différentielles ou plus généralement d'équations aux dérivées partielles.

Dans ce chapitre nous allons étudier des méthodes approximatives notamment : la *méthode d'Euler* et les *méthodes de Range-Kutta*, qui sont utilisées pour calculer des solutions approchées des équations différentielles de premier ordre.

Le but de ce chapitre est de calculer la solution sur l'intervalle du problème de Cauchy

$$y'(t) = f(t, y(t)) \text{ avec } y(t_0) = y_0 \qquad \text{(IV-1)}$$

Remarques importantes

- Avec les outils numériques de résolution d'équations différentielles, il n'est plus possible d'obtenir une solution pour toutes les valeurs de la variable indépendante t. On obtient plutôt une approximation de la solution analytique seulement à certaines valeurs de t notée ti et distancées d'une valeur $h_i = t_{i+1} - t_i$.
- Dans les méthodes présentées dans ce chapitre, la distance hi est constante pour tout i et est notée h, elle est appelée le pas de t.
- On note $y(t_i)$ la solution analytique de l'équation différentielle (IV-1) en $t = t_i$.
- On note y_i la solution approximative en $t = t_i$ obtenue à l'aide d'une méthode numérique.

IV.2. La méthode d'Euler

Soit un intervalle $[a, b]$ des valeurs de t. On considère $(n+1)$ valeurs équidistantes dans $[a,b]$ notés $t_0, t_1, t_2, \ldots, t_i, t_{i+1}, \ldots, t_n$ (ou n sous intervalles de même longueur h), tel que :

$t_0 = a$, $t_n = b$, $t_{i+1} = t_i + h$, $h = (b - a)/n$, $i = 0, 1, 2, \ldots, n - 1$.

Pour $t = t_0$ on a :

$$y'(t_0) = f(t_0, y(t_0)) = f(t_0, y_0) \qquad \text{(IV-2)}$$

qui est la pente de la tangente à la courbe de $y(t)$ au point (t_0, y_0) (voir Figure IV.1).
L'équation de cette tangente est donc :

$$d_0(t) = y_0 + f(t_0, y_0)(t - t_0) \quad \text{(IV-3)}$$

Pour $t = t_1$ on a :

$$d_0(t_1) = y_0 + f(t_0, y_0)(t_1 - t_0) = y_0 + hf(t_0, y_0) \quad \text{(IV-4)}$$

Dans la méthode d'Euler on considère que la solution approximative en $t = t_1$ est :

$$y_1 = d_0(t_1) = y_0 + hf(t_0, y_0) \quad \text{(IV-5)}$$

Sur la Figure IV.1, on remarque bien qu'il y a une différence entre la solution approximative y_1 et la solution exacte $y(t_1)$ pour l'abscisse t_1.
La pente de la tangente à la courbe de $y(t)$ au point $(t_1, y(t_1))$ (voir Figure IV.1) est :

$$y'(t_1) = f(t_1, y(t_1)) \quad \text{(IV-6)}$$

Puisque la valeur exacte de $y(t_1)$ est inconnue, on la remplace approximativement par la valeur approchée y_1, soit alors :

$$y'(t_1) = f(t_1, y(t_1)) \simeq f(t_1, y_1) \quad \text{(IV-7)}$$

Puis on définie une droite passant par le point (t_1, y_1) (voir la Figure IV.1) par :

$$d_1(t) = y_1 + f(t_1, y_1)(t - t_1) \quad \text{(IV-8)}$$

qui permettra d'estimer $y(t_2)$ par le même raisonnement précédent. Il vient que :

$$y(t_2) \simeq y_2 = d_1(t_2) = y_1 + hf(t_1, y_1) \quad \text{(IV-9)}$$

On constate que l'erreur commise à la première itération est réintroduite dans le calcul de la deuxième itération.
Si on applique la même procédure pour estimé respectivement les valeurs $y(t_3)$, $y(t_4)$, … . , $y(t_n)$ on obtient la formule itérative de la méthode d'Euler suivante :

$$y_{i+1} = y_i + hf(t_i, y_i) \quad \text{(IV-10)}$$

Avec : $t_0 = a$, $t_n = b$, $t_{i+1} = t_i + h$, $h = (b - a)/n$, $i = 0, 1, 2, \dots, n - 1$
Cette formule permet donc d'obtenir le point (t_{i+1}, y_{i+1}) à partir du point (t_i, y_i) en commençant par un point initiale donné (t_0, y_0).

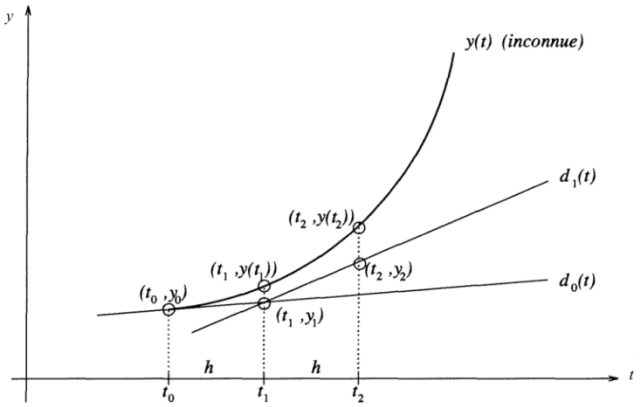


Figure IV.1 : Illustration graphique de la méthode approximative d'Euler

D'un autre coté, on remarque que les erreurs se propagent d'une itération à l'autre. Il en résulté de façon générale que l'erreur à chaque itération i est $|y(t_i) - y_i|$, qui augmente légèrement avec i.

IV. 3 Implémentation MATLAB de la méthode d'Euler

1) En appliquant l'algorithme de la méthode d'Euler, écrire un programme Matlab qui permet de calculer la solution approchée de l'équation :

$$y'(t) - \frac{3}{t}y(t) = t \quad , \quad y_0 = y(1) = 3 \quad , t \in [1\,,2] \qquad \text{(IV-11)}$$

par la méthode d'Euler pour n=20 sous-intervalles, et la solution exacte :

$$y(t) = -t^2 + 4t^3 \qquad \text{(IV-12)}$$

en même temps.

Note : Représenter les solutions approchées et exactes sur le même graphe afin de comparer.

2) Exécuter ce programme pour des nombres de sous-intervalles n=50 ,100 et 200. Conclure.

- Le programme MATLAB de la méthode d'Euler :

```
clc;clear all;close all;
a=1;
b=2;
n=20;
h=(b-a)./n;
t=a:h:b;
y=zeros(size(t));
y(1)=3;
f=@(t,y)t+(3./t)*y;
for i=1:n
t(i)=t(1)+(i-1)*h;
k1=h*f(t(i),y(i));
```

```
y(i+1)=y(i)+k1;
end
ys=-(t.^2)+4.*(t.^3);
plot(t,y,'*r',t,ys)
xlabel('t')
ylabel('y')
title('courbes des solutions approchée (pour n=20) et exacte.')
legend('approximée','exacte')
```

IV.4 Méthode de Runge-Kutta d'ordre 4

En reprenant le développement de Taylor de la fonction f, mais cette fois jusqu'à l'ordre 5, un raisonnement similaire à celui qui a mené aux méthodes de Runge-Kutta d'ordre 2 aboutit à un système de 8 équations non linéaires comprenant 10 inconnues. Le résultat final est la méthode de Runge-Kutta d'ordre 4 (RK4), qui représente un outil d'une grande utilité. Sa formule est donnée par :

$$y_{i+1} = y_i + \frac{1}{6} \times f(k_1 + 2k_2 + 2k_3 + k_4) \qquad \text{(IV-13)}$$

Avec :

$$k_1 = h \times f(t_i, y_i)$$
$$k_2 = h \times f\,(t_i + h/2, y_i + k_1/2)$$
$$k_3 = h \times f\,(t_i + h/2, y_i + k_2/2)$$
$$k_4 = h \times f(t_i + h, y_i + k_3)$$
$$t_{i+1} = t_i + h$$

IV.5 Implémentation MATLAB de la méthode de Range-Kutta d'ordre 4

1) En appliquant l'algorithme de la méthode de Range-Kutta d'ordre 4 (RK4), écrire un programme Matlab qui permet de calculer la solution approchée de l'équation (IV-11) par la méthode de Range-Kutta pour $n = 20$ sous-intervalles, et la solution exacte (IV-12) en même temps.

Note : Représenter les solutions approchées et exactes sur le même graphe afin de comparer.

2) Exécuter ce programme pour des nombres de sous-intervalles $n = 50$, 100 et 200. Conclure.

3) Matlab dispose des commandes prédéfinies permutant de calculer la solution approchée des équations différentielles ordinaires. Les commandes ode23 et ode45 sont très utiles, leurs syntaxes sont :

$$[t\,, y] = \text{ode23}(f, [a\,, b], y0) \qquad \text{et} \qquad [t\,, y] = \text{ode45}(f, [a\,, b], y0)$$

En utilisant ces commandes calculer des solutions approchées (représenter les résultats graphiquement avec la solution exacte afin de comparer).

1) Le programme Matlab de la méthode RK4 :

```
clc;clear all;close all;
a=1;
b=2;
n=20
h=(b-a)./n;
t=a:h:b;
y=zeros(size(t));
y(1)=3;
f=@(t,y)t+(3./t)*y;
for i=1:n
t(i)=t(1)+(i-1)*h;
k1=h*f(t(i),y(i));
k2=h*f(t(i)+(h./2),y(i)+(k1./2));
k3=h*f(t(i)+(h./2),y(i)+(k2./2));
k4=h*f(t(i)+h,y(i)+k3);
y(i+1)=y(i)+(1./6)*(k1+2.*k2+2.*k3+k4);
end
ys=-(t.^2)+4.*(t.^3);
plot(t,y,'*r',t,ys)
xlabel('t')
ylabel('y')
title('courbes des solutions approchée (pour n=20) et exacte.')
legend('approximée','exacte')
```

3) Le calcul par des commandes prédéfinies :

```
a=1;
b=2;
y0=3;
f=@(t,y)t+(3./t)*y;
[t,y]=ode23(f,[a ,b],y0)
ys=-(t.^2)+4.*(t.^3);
plot(t,y,'*r',t,ys)
xlabel('t')
ylabel('y')
title('courbes des solutions approchée (commande ode23) et exacte.')
legend('approximée par commande ode23','exacte')
```

IV. 6 Enoncé du TP:

1) Quand on fait la programmation des méthodes numérique d'Euler et de RK4 ?

2) Ecrire un algorithme de calcul pour la méthode d'Euler.

3) Ecrire un programme Matlab qui permet de calculer la solution approchée de l'équation suivant :

$$y'(t) - \frac{3}{t}y(t) = t \quad , \quad y_0 = y(1) = 3 \quad , t \in [1\,,2]$$

Par la méthode d'Euler pour $n = 20$ sous-intervalles, et la solution exacte de cet équation : $y(t) = -t^2 + 4t^3$ en même temps. Note : Représenter les solutions approchées et exactes sur le même graphe afin de comparer.

4) Exécuter ce programme pour des nombres de sous-intervalles $n = 50\,,100\,,200$. Conclure.

5) Répondre aux questions précédentes en utilisant la méthode RK4 au lieu de la méthode d'Euler.

6) Calculer la solution approchée en utilisant des commandes Matlab prédéfinies corresponds à différentes méthodes numériques.

Chapitre V

Résolution Numérique Des Systèmes D'équations Linéaires

Chapitre 5 : Résolution Numérique Des Systèmes D'équations Linéaires

V.1 Introduction

Les systèmes d'équations algébriques jouent un rôle très important en ingénierie. On peut classer ces systèmes en deux grandes familles : les systèmes *linéaires* et les systèmes *non linéaires.* Ici encore, les progrès de l'informatique et de l'analyse numérique permettent d'aborder des problèmes de taille prodigieuse. On résout couramment aujourd'hui des systèmes de plusieurs centaines de milliers d'inconnues. On rencontre ces applications en mécanique des fluides et dans l'analyse de structures complexes. On peut par exemple calculer l'écoulement de l'air autour d'un avion ou l'écoulement de l'eau dans une turbine hydraulique complète. On peut également analyser la résistance de la carlingue d'un avion à différentes contraintes extérieures et en vérifier numériquement la solidité.

Dans ce chapitre nous nous sommes intéressées aux quelques méthodes de résolution des systèmes d'équation linéaires.

V.2 Définitions

Systèmes d'équations linéaires

Un système carré d'équations linéaires à coefficients réels s'écrit sous forme des équations :

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ &a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ &a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \\ &\vdots \\ &\vdots \\ &\vdots \\ &a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{aligned} \qquad \text{(V-1)}$$

Ou sous forme matricielle :

$$AX = \mathrm{B} \qquad \text{(V-2)}$$

Avec :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \ddots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 & x_2 & x_3 & \vdots & x_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 & b_2 & b_3 & \vdots & b_n \end{bmatrix}$$

Ce système est donc constitué de n équations et n inconnus. Les éléments (a_{ij}) de la matrice A et ceux (b_i) du vecteur B sont des réels donnés, alors que ceux (xi) du vecteur X sont des réels inconnus. La résolution du système d'équations linéaire se ramène donc à la détermination du vecteur X.

Dans la plupart des cas, nous traitons des *matrices non singulières* ou *inversibles*, c'est-à-dire dont la matrice inverse A^{-1} existe. Nous ne faisons pas non plus de révision systématique de l'algèbre linéaire élémentaire sur les matrices que nous supposons connue.

V.3 Méthodes de résolution des systèmes d'équations linéaires

Le système (V-2) ne possède d'une seule solution que si la matrice A est inversible, c'est-à-dire que la matrice inverse A^{-1} existe. Dans le cas où l'un des coefficients diagonaux aii est nul, la matrice A n'est pas inversible.

La solution du système (V-2) peut donc s'écrire :

$$X = A^{-1}\,\mathrm{B} \qquad \text{(V-3)}$$

Cependant le calcul de la matrice inverse A^{-1} est plus difficile et plus long que la résolution du système linéaire de départ. Pour cela il existe différentes méthodes de résolutions qui sont en générale classées en deux catégories :

V.3.1 Méthodes directes

Ce sont les méthodes où la solution du système peut être obtenue en effectuant un nombre fini et prédéterminé d'opérations. Dans ce chapitre nous allons étudier une méthode directe dite *méthode d'élimination de Gauss*.

V.3.2 Méthodes itératives

Ce sont les méthodes qui peuvent converger en quelques itérations. La convergence des méthodes itératives n'est réalisée que dans certaines conditions que nous préciserons. De plus, les méthodes itératives, lorsqu'elles convergent, ne deviennent vraiment avantageuses que pour les systèmes linéaires de très grande taille. Dans ce chapitre nous allons étudier une méthode itérative dite *méthode de Gauss Seidel*.

V.3.3 Opérations élémentaires sur les lignes

Le système (V-2) peut se transformer à un autre système sans modifier sa solution, en effectuant des opérations sur ses lignes. Ceci est possible puisque on peut toujours multiplier les deux membres du système (V-2) par une matrice W inversible :

$$WAX = WB \qquad \text{(V-4)}$$

La solution n'est pas modifiée puisque l'on peut multiplier par W^{-1} pour revenir au système de départ. Particulièrement, pour transformer un système quelconque en un système triangulaire, il suffit d'utiliser trois opérations élémentaires sur les lignes de ce système. Ces trois opérations élémentaires correspondent à trois types de matrices W différentes.

On note Li la ligne i de la matrice A, les trois opérations élémentaires dont on a besoin sont les suivantes :

1. Opération ($Li \leftarrow \lambda Li$) : remplacer la ligne i par un multiple d'elle-même.

2. Opération ($Li \leftrightarrow Lj$) : intervertir la ligne i et la ligne j.

3. Opération ($Li \leftarrow Li + \lambda Lj$) : remplacer la ligne i par la ligne i plus un multiple de la ligne j.

Exemple 1

Soit le système :

$$\begin{bmatrix} 3 & 1 & 2 \\ 6 & 4 & 1 \\ 5 & 4 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \\ 10 \end{bmatrix} \quad \text{(V-5)}$$

Dont la solution est $X = [1\ 1\ 1]^T$. Si on souhaite multiplier la ligne 2 par un facteur 3, on obtient :

$$\begin{bmatrix} 3 & 1 & 2 \\ 18 & 12 & 3 \\ 5 & 4 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 6 \\ 33 \\ 10 \end{bmatrix} \quad \text{(V-6)}$$

La solution de ce nouveau système reste la même que celle du système de départ.

Exemple 2

On vaut intervenir la ligne 2 et la ligne 3 du système de l'exemple précédent, on obtient :

$$\begin{bmatrix} 3 & 1 & 2 \\ 5 & 4 & 1 \\ 6 & 4 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \\ 10 \end{bmatrix} \quad \text{(V-7)}$$

Exemple 3

Dans le système (V-7), on souhaite remplacer la deuxième ligne par la deuxième ligne ($i = 2$) moins deux fois ($\lambda = -2$) la première ligne ($j = 1$), ce qui donne :

$$\begin{bmatrix} 3 & 1 & 2 \\ 0 & 2 & -3 \\ 5 & 4 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 6 \\ -1 \\ 10 \end{bmatrix} \quad \text{(V-8)}$$

V .4 Méthode d'élimination de Gauss

La méthode d'élimination de Gauss permet de trouver la solution du système (V-2) en le transformant en un système triangulaire supérieur qui possède la même solution. Cette méthode consiste donc à éliminer tous les termes sous la diagonale de la matrice A.

V.4.1 Notion de la matrice augmentée

La matrice augmentée du système linéaire (V-2) est la matrice de dimension n sur $n+1$ que l'on obtient en ajoutant le membre de droite B à la matrice A, c'est-à-dire :

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} & b_n \end{array}\right] \qquad \text{(V-9)}$$

Puisque les opérations élémentaires doivent être effectuées à la fois sur les lignes de la matrice A et sur celle du vecteur B, cette notation est très utile.

La méthode d'élimination de Gauss est appliquée seulement dans le cas où aucune permutation de ligne n'est effectuée. Pratiquement, la méthode d'élimination de Gauss est composée de deux parties :

1. Transformation du système un autre système triangulaire possédant la même solution, en appliquant, sur les lignes, les opérations suivantes :

$$Li \leftarrow Li - (aik/akk)Lk\,, \qquad k = 1{:}\, n-1\,,\, i = k+1{:}\,\mathrm{n} \qquad \text{(V-10)}$$

L'indice k est le numéro de l'étape, k varie de 1 à $(n-1)$. L'indice i est le numéro de la ligne Li . Pour chaque valeur de k, i varie de $k+1$ à n.

On obtient alors un système triangulaire supérieur de la forme :

$$\left[\begin{array}{ccccc} a'_{11} & a'_{12} & a'_{13} & \dots & a'_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a'_{33} & \dots & a'_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & a'_{nn} \end{array}\right] \left\| \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{array} \right\| = \left[\begin{array}{c} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{array}\right] \qquad \text{(V-11)}$$

Où $a'ij = 0$ pour tout $i > j$

2. Calculer la solution (xi) du système triangulaire (V-11)qui est :

$$x_i = \begin{cases} \dfrac{b'_n}{a'_{nn}} & si \quad i = n \\ \dfrac{\left(b'_i - \sum_{j=i+1}^{n} a'_{ij}\, x_j\right)}{a'_{ii}} & si \quad i \neq n \end{cases} \quad , \quad (i = n\,, \dots, 2\,, 1) \qquad \text{(V-12)}$$

V.4.2 Implémentation MATLAB de la méthode d'élimination de Gauss

1) En utilisant l'algorithme de la méthode d'élimination de Gauss, écrire un programme MATLAB qui permet de résoudre le système d'équation suivant :

$$\begin{bmatrix} 2 & 1 & 2 \\ 6 & 4 & 0 \\ 8 & 5 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 10 \\ 26 \\ 35 \end{bmatrix} \qquad \text{(V-13)}$$

2) Comparer avec la solution donnée par les commandes suivantes :

$$X = \text{inv}(A) * B \text{ et } X = A \backslash B$$

- Le programme MATLAB de la méthode d'élimination de Gauss :

```
clear all;close all;clc;
%Programme de la méthode d'élimination de Gauss
A=[2 1 2;6 4 0;8 5 1];B=[10;26;35];n=length(B);
%Calcul du système triangulaire supérieur
for k=1:n-1
if A(k,k)==0
disp('Attention: pivot nul')
break
end
for i=k+1:n
C=A(i,k)/A(k,k);
B(i)=B(i)-B(k)*C;
for j=k:n
A(i,j)=A(i,j)-A(k,j)*C;
end
end
end
disp('La matrice triangulaire supérieur obtenue est :')
disp(A)
disp('Le vecteur B devient :')
disp(B)
%Calcul de la solution du système
for i=n:-1:1
if i==n
x(i)=B(i)/A(i,i);
else
    s1=0;
for j=i+1:n
s1=s1+A(i,j)*x(j);
end
x(i)=(B(i)-s1)/A(i,i);
end
end
disp('solution est :')
disp(x)
```

Après exécution on obtient

La matrice triangulaire supérieure obtenue est :
2 1 2
0 1 -6
0 0 -1
Le vecteur B devient :
10
-4
-1

La solution est :
3 2 1

2) On fait ce calcul directement sur la zone de commandes :

```
>> A = [2 1 2;6 4 0;8 5 1];B=[10;26;35];
>> X=A\B
X =
3
2
1
>> X=inv(A)*B
X =
3
2
1
```

Ces commandes prédéfinies équivalentes donnent le même résultat que notre programme réalisé.

V.5. La méthode de Gauss-Seidel

On utilise encore chacune des équations pour recalculer une nouvelle estimation de la solution. La différence avec la méthode de *Jacobi* réside dans le fait que l'on utilise les dernières valeurs obtenues (de l'itération en cours) et non celles de l'itération précédente.

Pour cette méthode on a :

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} * x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} * x_i^k) \qquad \text{(V-14)}$$

Avec i=1 :n

Il s'écrit aussi :

$$x^{k+1} = (D-E)^{-1}(b + Fx^k) \qquad \text{(V-15)}$$

Dans ce cas, le splitting de A est :

$$P = D - E; N = F \qquad \text{(V-16)}$$

Et la matrice d'itération associée est :

$$B_{GS} = (D-E)^{-1}U \qquad \text{(V-17)}$$

L'algorithme de Gauss-Seidel ne nécessite qu'un vecteur de stockage, étant remplacé par au cours de l'itération. Il est en général plus rapide que l'algorithme de Gauss, donc il est préférable en termes de rapidité.

V.6 : Implémentation MATLAB de la méthode de Gauss-Seidel

1) Calculer la solution exacte du système (V-18) en utilisant la commande :

$X = \text{inv}(A) * B$ ou $X = A \setminus B$

$$\begin{bmatrix} 3 & 1 & -1 \\ 1 & 5 & 2 \\ 2 & -1 & -6 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 2 \\ 17 \\ -18 \end{bmatrix} \qquad \text{(V-18)}$$

2) En utilisant l'algorithme de la méthode de Gauss-Seidel, écrire un programme MATLAB qui permet de résoudre le système d'équation (V-18), à une précision $\varepsilon = 10^{-2}$, en utilisant le vecteur de départ :

$X_0 = [0 \ 0 \ 0]^T$.

3) Exécuter ce programme pour les précisions 10^{-3} , 10^{-4} et 10^{-6} . Conclure.

1) On fait ce calcul directement sur la zone de commandes :

```
>> A=[3  1 -1;1  5  2;2  -1  -6];B=[2; 17; -18];
>> X=inv(A)*B
   X =
   1
   2
   3
>> X=A\B
X =
1.00000000000000
2.00000000000000
3.00000000000000
```

2) Le programme MATLAB de la méthode de Gauss-Seidel :

```
clear all;close all;clc;
format long
%Programme de la méthode de Gauss-Seidel
A=[3 1 -1;1 5 2;2 -1 -6];B=[2;17;-18];n=length(B);
x0=[0;0;0];
x1=x0;
eps=10^-2
e=1+eps;
k=0;
while e>eps
for i=1:n
s1=A(i,1:i-1)*x1(1:i-1);
s2=A(i,i+1:n)*x0(i+1:n);
x1(i)=(B(i)-s1-s2)/A(i,i);
```

```
end
e=max(abs(x1-x0));
x0=x1;
k=k+1;
end
disp('le vecteur solution est')
disp(x1)
disp('le nombre d''itérations est')
disp(k)
```

On obtient après exécution le résultat suivant :

Le vecteur solution est :

0.99682770502025

2.00214981016155

2.99858426664649

Le nombre d'itérations est : 7

3) Les résultats obtenus pour différentes précisions sont donnés dans le tableau suivant :

ε	Nombre d'itérations	x_1	x_2	x_3
10^{-2}	7	0.99682770502025	2.00214981016155	2.99858426664649
10^{-3}	9	0.99955527762739	2.00030101282468	2.99980159040502
10^{-4}	12	0.99997667778165	2.00001578354740	2.99998959533598
10^{-6}	14	0.99999673260932	2.00000221124840	2.99999854232837

On conclut que lorsque la précision augmente, le nombre d'itérations effectué augmente plus en plus, et la solution converge vers la solution exacte.

V-7 Enonce du TP

1) Méthode d'élimination de Gauss

a. Ecrire un algorithme de calcul pour la méthode d'élimination de Gauss.

b. Ecrire un programme Matlab qui permet de calculer la solution du système d'équation (V-19) par la méthode d'élimination de Gauss.

$$\begin{bmatrix} 2 & 1 & 2 \\ 6 & 4 & 0 \\ 8 & 5 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 10 \\ 26 \\ 35 \end{bmatrix} \qquad \text{(V-19)}$$

c. Comparer avec la solution donnée par les commandes suivantes : $X=\text{inv}(A)*B$ et $X=A\backslash B$

2) Méthode de Gauss-Seidel

a. Calculer la solution exacte du système d'équation (V-20) en utilisant la commande : $X=\text{inv}(A)*B$ ou $X=A\backslash B$

$$\begin{bmatrix} 3 & 1 & -1 \\ 1 & 5 & 2 \\ 2 & -1 & -6 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 2 \\ 17 \\ -18 \end{bmatrix} \quad \text{(V-20)}$$

b. Ecrire un algorithme de calcul pour la méthode de Gauss-Seidel.

c. Ecrire un programme Matlab qui permet de calculer la solution du système (V-20) par la méthode de Gauss-Seidel, à une précision $\varepsilon=10^{-2}$, en utilisant le vecteur de départ : $X_0=[0 \quad 0 \quad 0]^T$.

d. Exécuter ce programme pour les précisions 10^{-3}, 10^{-4} et 10^{-6}. Conclure.

Références bibliographiques

[1] C. T. Kelley, "Iterative Methods for Linear and Nonlinear Equations", SIAM, Philadelphia, 1999.

[2] Manfred GILLI, Méthodes numériques, Université de Genève, 2006.

[3] S. benkouda, Méthodes Numériques (cours et TD), université frères Mentouri.

[4] C. Brezinski, Introduction à la pratique du calcul numérique, Dunod, Paris, 1988.

[5] G. Allaire et S.M. Kaber, Algèbre linéaire numérique, Ellipses, 2002.

[6] G. Allaire et S.M. Kaber, Introduction à Scilab. Exercices pratiques corriges d'algèbre linéaire, Ellipses, 2002.

[7] G. Christol, A. Cot et C.M. Marle, Calcul différentiel, Ellipses, 1996.

[8] M. Crouzeix et A.L. Mignot, Analyse numérique des équations différentielles, Masson, 1983.

[9] S. Delabriere et M. Postel, Méthodes d'approximation. Equations différentielles Applications Scilab, Ellipses, 2004.

[10] J.P. Demailly, Analyse numérique et équations différentielles. Presses Universitaires de Grenoble, 1996.

[11] E. Hairer, S. P. Norsett et G. Wanner, Solving Ordinary Differential Equations, Springer, 1993.

[12] D. PENNEQUIN, Méthodes Numériques, Université Paris 1, 2015-2016

[13] P. G. Ciarlet, Introduction à l'analyse numérique matricielle et à l'optimisation, Masson, Paris, 1982.

[14] S. KENOUCHE, Méthodes Numériques via MATLAB, université de Biskra.

[15] S. Karoui, Polycopié de TP : Méthodes Numériques, USTO, 2016-2017.

[16] A. Quarteroni,R. Sacco, F. Saleri, Méthodes Numériques Algorithmes, analyse et applications, Springer, 2007.

[17] Paola GOATIN, Analyse Numérique, Université du Sud Toulon-Var.

[18] Stéphane Canu et Gilles Gasso, « Méthodes itératives pour la solution d'un système linéaire », Novembre 19, 2016.