REPUBLIQUE ALGERIENNE
DEMOCRATIQUE & POPULAIRE
MINISTERE DE L'ENSEIGNEMENT
SUPERIEUR & DE LA RECHERCHE
SCIENTIFIQUE

UNIVERSITE DJILLALI LIABES
FACULTE DES SCIENCES EXACTES
SIDI BEL-ABBES
BP 89. 22000 SBA ALGERIE
TEL/FAX 048-54-43-44

Méthodes Numériques Implémentées sous Fortran 90

12/02/2022

CHIKHI Malika

Avant Propos

Le présent document a pour objectif d'implémenter sous Fortran 90 quelques algorithmes des méthodes numériques proposées pour les étudiants de la 2eme et 3eme année de la filière physique.

Le manuscrit peut également être un outil important offrant aux étudiants l'opportunité de maitriser le langage Fortran 90 et de l'utiliser à la résolution des problèmes physiques rencontrés dans la recherche scientifique. La première partie est consacrée à un rappel sur le langage fortran 90. Les quatre autres parties sont dédiées à l'étude des méthodes à travers un bref rappel suivi des programmes fortran correspondants. Des travaux de base dont les références sont citées ci-dessous ont fait l'objet de source d'inspiration dans la réalisation de ce document.

Références

1. Hahn, Brian D *Fortran 90 for Scientists and Engineers* I. Title 005.13 ISBN 0-340-60034-9
2. Olivier Louisnard, Jean-Jacques Letourneau, Paul Gaborit *Initiation au Fortran* Ecole Des Mines D'Albi Carmaux
3. Guide 138 : *An introduction to programming in Fortran 90* University of Durham Information Technology Service 2011
4. M. Boumahrat et A.Gourdin *Méthodes Numériques Appliquées avec nombreux problèmes résolus en Fortran IV*. OPU Alger 1983
5. M. Lakrib *Cours d'Analyse Numérique.* OPU Alger 2008

Sommaire

Rappel sur le Langage Fortran 1
I. Introduction 1
1. Compilation 1
II. Généralités 1
1. Structure d'un programme Fortran 1
2. Types de données 2
2.1 Constantes 2
2.2 Variables 3
3. **_Déclaration des variables_** 3
4. Règles de typage implicite 4
4.1 Directive implicite 4
4.2 *Affectation d'une variable* 5
5. Les Instructions Entrées/Sorties Ecran clavier 5
5.1 Format de lecture 5
III. Les Structures conditionnelles 8
1. Condition sans négation 8
2. Condition avec négation 9
3. Suite de Conditions 10
IV. Les Boucles 12
1. Boucle avec compteur 12
2. Boucle conditionnelle 13
V. Fonctions et Subroutines 14
1. Les fonctions 14
3. Les Subroutines 17
VI. Les tableaux 19
1. Tableaux statiques 19
2. Tableaux dynamiques 20
3. fonctions intrinsèques manipulant les tableaux 20
Rappel sur l'interpolation polynomiale 23
I. Introduction 23
II. Méthode de Lagrange 24

III. Méthode de Newton 26

1. Différences divisées 26

2. Erreur d'interpolation 28

3. Différences finies 29

3.1 Différences finies progressive 29

Résolution d'un système d'équation linéaire 32

I. Méthodes directes 32

1. Méthode de Gauss 32

2. Méthode de Gauss Jordan 35

II. Méthodes itératives 36

1. Méthode de Jacobi 36

Intégration approchée 39

I. Méthode des trapèzes 39

II. Méthode de Simpson 40

Résolution d'équations non linéaires 45

I. Racines d'équations 45

II. Méthode de la dichotomie 45

III. Méthode de Newton-Raphson 48

1. Convergence de la méthode 49

2. Critère d'arrêt 49

Rappel sur le Langage Fortran

I. Introduction

Le Fortran est langage de programmation très populaire développé pour le calcul scientifique. Le nom Fortran est composé par la concaténation des initiales des termes FORmula et TRANslator. Fortran 90 est une version créée en 1990.

1. Compilation

La compilation est l'opération de traduire le langage écrit par l'utilisateur en langage machine La compilation d'un programme permet la génération d'un fichier exécutable à partir d'un ou de plusieurs programmes Fortran. Elle peut être effectuée sous les deux systèmes d'exploitation LINUX ou WINDOWS.

Un compilateur est un programme permettant de créer un fichier exécutable à partir d'un ou plusieurs fichiers Fortran. Dans ce travail nous utilisons le logiciel **Microsoft Developer Studio Fortran Powerstation 04** (compilateur intégré) permettant la compilation et l'exécution des programmes Fortran 90.

II. Généralités

Dans ce qui suit, nous utilisons la convention suivante: en **gras** les instructions Fortran.
Les couleurs utilisées dans les programmes sont telles qu'elles apparaissent dans les fichiers Fortan

1. Structure d'un programme Fortran

Un programme Fortran est une succession d'instructions. Il se compose obligatoirement d'un programme principal et peut comporter des sous programmes (voir paragraphe V).

Le programme principal commence par l'instruction **program** (optionnel) et se termine obligatoirement par **end**

Tout ce qui suit le symbole ! n' est pas compilé par la machine. On l'utilise généralement pour écrire des commentaires.

```
! Commentaire : Ce programme permet de calculer …..
program  Nom-du-programme
            [Déclarations des données]
            [Instructions]
end program Nom-du-programme
```

2. *Types de données*

2.1 Constantes

Toute constante doit avoir un *type* pour faciliter la manipulation et le stockage. Les types principaux pour les constantes sont :

- **Integer** : Contient un entier codé sur 4 octets (31 bits pour la valeur +1 bit pour le signe) les valeurs possible se trouvent dans l'intervalle [2^{31}, 2^{31} - 1]

- **Real** : représente un réel codé en virgule flottante sur 4 octets. Le codage se fait sous la forme suivante : $x = \pm 0.m\, 2^e$ où m est la mantisse codée sur 23 bits et e est l'exposant codé sur 8 bits. Les valeurs se trouvent dans l'intervalle [1.401×10^{-45}, 3.403×10^{38}]
- **Double precision :** Représente un réel codé en virgule flottante sur 8 octets, une mantisse codée sur 52 bits et un exposant codée sur 11 bits. Les valeurs (en valeur absolue) sont comprises entre [4.951×10^{-32} , 1.798×10^{308}]

- **Complex :** contient deux nombres réels (4 octets) stockés sous forme de paire et traités comme les parties réelle et imaginaire d'un nombre complexe
- **Logical :** Les constantes logiques ont une valeur de .true. ou fausse. . Ils occupent un espace de stockage de 4 octets.

- **Character :** représente une chaine de caractères texte/ASCII standard avec un octet par caractère et N octets au total, où N est un entier.

2.2 Variables

Une variable est une information donné par un nom associé à un espace mémoire. Au cours du déroulement d'un programme, les variables peuvent prendre plusieurs valeurs numériques logiques ou arithmétiques.

Similairement aux constantes les variables doivent avoir un type. Toute valeur constante peut être affectée à une variable Fortran, à condition que le type de la constante soit compatible avec le type de la variable.

Le nom d'une variable peut contenir des caractères alphabétiques et numériques et des traits de soulignement, mais doivent commencer par un caractère alphabétique et ne pas dépasser 31 caractères au total. Les majuscules et minuscules ne sont pas différentiées

3. *Déclaration des variables*

La déclaration d'une variable se fait entre **program** et la première instruction.

Exemple :

```
program  somme
integer : : i, j, a
real : : x, y, z
complex : : c1, c2
character :: c
logical :: b
a = i+ j     ! première instruction
        ⋮
end
```

4. *Règles de typage implicite*

Par défaut une variable est automatiquement de type :

- **integer** si son nom commence par i, j, k, l, m, n
- **real** si son nom commence par tout autre lettre (a , b, c…., h et o, p, q….., z)

4.1 Directive implicite

Les règles de typage implicite sont modifiées par la directive implicite selon la syntaxe suivante :

```
Implicit type ( lettre1-lettre2, lettre3 )
```

Toute variable dont le nom commence par une lettre comprise entre lettre1 et lettre2 ou par la lettre 3 est du *type* indiqué.

```
Implicit non
```

Aucune variable n'est déclarée par défaut. Toute variable doit être déclarée.

Exemple :

```
Implicit  integer (a-g, p –r, z)
Implicit real (h-o)
Implicit complex (s-w)
```

Tout nom de variable commençant par a, b, c, d, e, f, g, p, q, r, z est **integer**

Tout nom de variable commençant par h, i, j, k, l, m, n, o est **real**

Tout nom de variable commençant par s, t, w est **complex**

4.2 *Affectation d'une variable*

L'opération d'affectation consiste à attribuer une valeur numérique a une variable.
L'opérateur d'affectation est le signe (=) qui assigne la valeur de l'opérande droit à l'opérande gauche.

Exemple :

```
x =  235.96 ! affectation d'une constante réelle à la variable x
i = 14          ! affectation d'une constante entière à la variable i
y = x           ! affectation de la variable x  à la variable y
z=y+x           ! affectation du résultat de l'opération y+x à la variable z
```

5. *Les instructions Entrées/Sorties Ecran clavier*

Deux types d'instructions son distingués :

- Instruction de lecture : **read*, read** (*unité de lecture*, *paramètre format*)

- Instruction d'écriture ; **print*** et **write***, **write** (*unité de écriture*, *paramètre format*)

  Pour une lecture au clavier, *l'unité de lecture* est *
  Pour un affichage sur l'écran, *l'unité d'écriture* est *

5.1 Format de lecture

Le format d'écriture indique comment la variable va être lue en type et en nombre.

a) format libre

Le symbole * est réservé au format libre.

Syntaxe :

```
read*, x            ! lecture de la variable x a partir du clavier
```

```
read (*,*) x        ! lecture de la variable x à partir du clavier avec format
                    !  libre

read (5, *)         ! lecture de la variable x à partir d'un fichier en utilisant
                    ! un format libre
                    !5 représente l'unité de lecture
```

b) Ecriture et lecture formatées

Généralement *le paramètre de format* **fmt** est utilisé
Syntaxe :

```
read (*, fmt ='chaine') x

write (*, fmt =   'chaine') y
```

Où *chaine* indique le type de variable

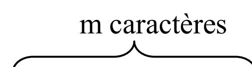
Type **integer** : **fmt**=' nIm' pour n entiers de m chiffres

Type **real** :

- **fmt**=' nFm.d' pour n réels sous forme décimale formés de m caractères (y compris le point décimal et le signe) et d chiffres après la virgule

- **fmt**=' nEm.d' pour n réels en virgule flottante de m caractères et d chiffres après la virgule (voir schéma ci-dessous)

m caractères



d caractères

$\pm 0.\sqcup\sqcup \ldots \sqcup E \pm \sqcup\sqcup$

Type **double precision :** =‘ nDm.d’ pour n réels double precisond de m caractères y

Type **character: fmt**=‘ nAm’ pour n chaines de m caractères

Type **boolean : fmt**=‘ nLm’ pour n *booleans* constitués de m caractères

Exemple

```
write(*,fmt = ' (2I4) ' ) i, j
read(*,fmt = ' ( f10.4) ' ) b
write(*,fmt = ' ( 3e15.3) ' ) x, y, z
read(*,fmt = ' ( 3d10.2) ' ) u,v,w
```

- I, E, F, D, L son appelés des descripteurs. Les descripteurs x est / sont utilisés respectivement pour laisser l’espace et passer à la ligne suivante.

- Les écritures fortran suivantes sont également possibles :

```
! elimination de fmt
write(*, ' (e10.2) ') b
! utilisation du format mixte
write(*,' (f10.5,3x,a8,4x,e10.2,3x,i6) ' ) a, char, b , c
! En définissant form
form=  '(i2)'
read(*, fmt=form) i
 ! En utilisant une étiquette
read (*,2) k
2 format (i5)
```

```
! utilisation du descripteur  slash (/)
write (*,4) p, q
4 format (i5/e15.6)
```

III. Les Structures conditionnelles

Elles sont utilisées lorsque l'exécution du programme dépend de la vérification d'une condition donnée

1. Condition sans négation

Syntaxe :

```
if (Expression-logique) then
        ⋮
[instructions]
        ⋮
end if
```

Exercice 1: Faire un programme qui permet l'affichage de x pour -1 <x<2

Programme 1.1 : Résolution de l'exercice 2

```
program Condition1
double precision :: x
write(*,*) 'Donner x'
read(*,*) x
if ( x > -1 .and. x < 2) then
write (*,*) x
endif
end program Condition1
```

2. Condition avec négation

L'utilisateur peut spécifier les instructions à faire si la négation de la condition logique est varie.

Syntaxe:

```
if (Expression-logique) then
[Instructions]
      ⋮
else
      ⋮
 [Instructions]
end if
```

Exercice 2: Faire un programme qui calcule la valeur absolue de x

Programme 1.2 : Résolution de l'exercice 2

```
program Condition2
double precision :: x,y
write(*,*) 'Donner x'
read(*,*) x
if ( x<0 ) then
y = - x
else
y = x
endif
write (*,*) y
end program Condition2
```

3. Suite de Conditions

Syntaxe:

```
if  (Expression-logique) then
        ⋮
[instructions]
        ⋮
else if (Expression-logique) then
        ⋮
[Instructions]
        ⋮
else if (Expression-logique) then
        ⋮
 [Instructions]
        ⋮
else
        ⋮
end if
```

Exercice 3 : Calcul de la valeur y pour une valeur de x lue au clavier :

Rappel sur le Langage Fortran

$$y = \begin{cases} cos(x\pi) & si -0.5 \leq x < 0 \\ 0 & si \ x = 0 \\ tang(x\pi) & si \ 0 < x \leq 1 \end{cases}$$

Programme 1.3 : Résolution de l'exercice 3

```
program Condition3
double precision :: x, y
parameter :: pi=3.14
write(*,*) 'Donner x'
read(*,*) x
if (x<0.and. x>=-0.5)  then
y=cos(x*pi)
write (*,*)y
else if (x==0) then
y=0
write(*,*) y
  else if (x<=1.and. x>0)  then
y = tan(x)
write(*,*) y
else
write(*,*) ' Hors domaine '
endif
end program  Condition3
```

Exercice 4 : (A la charge de l'étudiant)

Ecrire un programme qui permet de calculer la valeur de y pour $x \in [0,2]$.

$$y = \frac{e^{2\pi x}}{x-1}$$

Donner un message d'erreur si $x = 1$

IV. Les Boucles

Une boucle est utilisée pour exécuter une ou plusieurs instructions plusieurs fois d'affilées.

1. Boucle avec compteur

Syntaxe :

```
do var= debut, fin, pas
          ⋮
  [Instructions]
          ⋮
end do
```

Var, debut, fin et pas sont des entiers

Exercice 5:

Faire un programme permettant de lire N et calculer $\boldsymbol{S} = \sum_{i=0}^{N} \boldsymbol{i}$

Programme 1.4 : Résolution de l'exercice 5

```
program Somme1
integer :: N, S
write (*,*) ' Donner N '
read (*,*) N
S=0
do  i =1, N
S = S + i
end do
write (*,*) S
end  program Somme1
```

2. *Boucle conditionnelle*

Syntaxe :

```
do while (condition logique)
         ⋮
         [instructions]
         ⋮
end do
```

Exercice 6: Faire un programme qui calcule : $S = \sum_{i=0} i$ avec S<100

Programme 1.5 : Résolution de l'exercice 6

```
program Somme2
S=0
do while ( S < 100)
write (*,*) S
S=S +i
end do
end program Somme2
```

Exercice 7: (A la charge de l'étudiant)

Ecrire un programme qui calcule et imprime la valeur approchée de Y en prenant 10 termes de la suite :

$Y=1-(1/ (x+1)) +(2/ (x^2+2)) -(3/ (x^3+3))+\ldots\ldots$

V. Fonctions et Subroutines

Le **Fortran 90** possède deux types de sous programmes :

1. Les fonctions

Plusieurs syntaxes sont proposées :

Syntaxe 1 :

```
type  function nom-de-la-fonction ( arg1, arg2,…,argn)
        ⋮
[Partie déclaration]
        ⋮
[Partie Exécution]
        ⋮
[Partie Sous programme]
        ⋮
end function nom-de-la-fonction
```

- **type** peut etre **integer, logical, real, double, precision, complex….**
- arg1, arg2,……, argn , sont les arguments de la fonction
- nom-de-la-fonction est le nom de la fonction

Syntaxe 2 :

```
 function nom-de-la-fonction ( arg1, arg2,…,argn)
type nom-de-la-fonction
        ⋮
[Instructions]
        ⋮
end function nom-de-la-fonction
```

Syntaxe 3 :

```
function nom-de-la-fonction ( arg1, arg2,…,argn)  result (R)
type :: R
        ⋮
[Instructions]
        ⋮
end function nom-de-la-fonction
```

Dans ce cas de la syntaxe 3 le résultat peut avoir un nom différent du nom de la fonction.

- L'appel de la fonction se fait directement par son nom (voir exemple 1 dans programme 1.6)

Exemple 1 : Appel de la fonction $f(x) = \sqrt{x}$

Programme 1.6 : Illustration de l'appel d'une fonction à travers l'exemple 1

``` program exemple real :: x,y write (*,*)' Donner x ' read (*,*)x y= f (x) write (*,*) 'La racine de x est :', y end program ```	``` real function f (x) real :: x f=sqrt (x) end function ```

Exercice 8 : En utilisant un sous programme fonction faire un programme qui permet de calculer la somme de deux réels.

Programme 1.7 : Résolution de l'exercice 8

```
program somme
real :: a, b
write (*,*) ' Donner a et b'
read (*,*) a,b
write (*,*) som (a,b)  ! affichage de la somme
end
```

```
real function  som (x,y)
real , intent(in) ::  x, y
som = x +y
end function som
```

Exercice 9 : (A la charge d l'étudiant)

Ecrire un programme principal avec une fonction qui prend en entrée deux coordonnés x et y coordonnes d'un point donnée et renvoie son module

Exercice 10 : (A la charge d l'étudiant)

En utilisant un sous programme fonction, écrire un programme qui calcule : $f(x) = \frac{1}{x^3 + 2} + 4$

3. *Les Subroutines*

Contrairement aux sous programmes fonctions, Le sous programme subroutine retourne plusieurs valeurs calculées.

Syntaxe :

```
subroutine nom-de-la-Subroutine ( arg1, arg2,..,argn)
[Déclaration]
 [Instructions]
end subroutine nom-de-la-Subroutine
```

- Dans la partie déclaration il est conseillé de préciser les attributs suivants :
  - **intent (in)** pour les arguments d'entrée
  - **intent (out)** pour les arguments de sortie
  - **intent (inout)** pour les arguments mixtes (voir exemple 2)

- La subroutine est appelée dans le programme principal en utilisant l'instruction :

```
call nom-de-la-Subroutine
```

Exemple 2: Permutation de deux réels x et y

Programme 1.8 : Illustration de l'utilisation de l'attribut **inout** à travers l'exemple 2

```
program Permut
write (*,*)' Donner x et y'
real : : x,y
read*, x,y
call Permutation (x,y)
write (*,*) 'x = ', x
write (*,*)  'y = ', y
end
Subroutine Permutation (a,b)
real , intent (inout) ::  a, b
real :: c
c = a
a = b        ! a prend la valeur b
b = c        ! b prend la valeur a
end  subroutine Permutation
```

Exercice 11 : En utilisant un sous programme subroutine faire un programme qui permet de calculer la somme et le produit de deux réels.

Programme 1. 9 : Résolution de l'exercice 11

```
program Operation
real : : a, b, c
write (*,*)' Donner a et b'
read (*,*) a, b
!    Appel de la Subroutine
call Op (a, b, c, d)
!  affichage de la somme
write (*,*) 'la somme est', c
! affichage du produit
write (*,*) 'le  produit est', d
end
subroutine Op (x, y, z, w)
real , intent (in) ::  x, y
real , intent (out) :: z, w
z=x+y
w=x*y
end subroutine Op
```

VI. Les tableaux

Deux types de tableaux sont distingués :

1. Tableaux statiques

Les dimensions des tableaux sont fixées

Syntaxe :

```
integer, dimension  (3, 4) : : A         ! 3 lignes et 4 colonnes
Real, dimension (1 :5, 3 :7) : : T     ! ligne de 1 à 5 et colonne de 3 à 7
```

2. Tableaux dynamiques

La taille du tableau est variable (tableaux allouables).

Syntaxe :

```
Real, dimension ( :,:), allocatable : : A
```

Pour manipuler les tableaux on peut leur réserver les tailles requises par :

Syntaxe

```
Allocate ( A(1 :8, 5 :9) )

Allocate ( T(1 : n, 1 : m) )   ! n et m sont deux entiers lus au clavier
```

3. fonctions intrinsèques manipulant les tableaux

Le Fortran 90 possède des fonctions intrinsèques. Quelque unes utilisées dans la manipulation des tableaux sont citées dan le tableau 1

Tableau 1 : Quelques fonctions manipulant les tableaux :

Fonction intrinsèque	Effet
Maxval (A)	Valeur maximale du tableau A
Minval (A)	Valeur minimale du tableau A
Maxloc (A)	Position du maximum dans le tableau
Minloc (A)	Position du maximum dans le tableau
Product (A)	Produit des éléments du tableau A
Sum (A)	Somme des éléments du tableau A
Dot_product (V1, V2)	Produit de deux vecteurs V1 et V2
Matmul (A1, A2)	Produit de deux matrices A1 et A2

Exercice 12

Ecrire un programme principal avec une subroutine qui prend en entrée deux matrices a(n,n) et b(n,n) et renvoie en sortie leur somme.

Programme 1.10: Résolution de l'exercice 12

```
program SomMat
integer, parameter ::  max1=3,max2=3
 real : : a(max1,max2), b(max1,max2), c(max1,max2)
write (*,*)'Donner les composantes de matrice a'
read (*,*) ( ( a(i,j), i=1, max1), j = 1, max2)
write (*,*)'Donner les composantes de  la matrice b'
read (*,*) ((b(i,j),i=1,max1),j=1,max2)
 call somme (a, b, max1, max2, c)
 do i=1,max1
write (*,*) (c(i,j),j=1,max2)
 enddo
end  program
```

```
subroutine somme ( a, b, m, n, c)
  real, dimension (m, n):: a, b, c
   integer, intent(in) :: m, n
   intent(in) : : a, b
   intent(out) : : c
   do i=1,m
   do j=1,n
   c ( i, j) = a( i, j ) + b  (i, j)
    enddo
    enddo
    end  subroutine somme
```

Rappel sur l'interpolation polynomiale

I. Introduction

Soit f une fonction inconnue explicitement, définie sur $[a, b]$ de $\mathbb{R}$ $(a < b)$, connue pour (n+1) valeurs distinctes de x_i $(i = 0,1 \ldots . n)$.

Soit,

$$f(x_i) = y_i \text{ pour } i = 0,1, \ldots . n \qquad (2.1)$$

la fonction est inconnue pour une valeur intermédiaire entre x_i et x_{i+1}.

L'interpolation polynomiale d'une fonction consiste à approcher la fonction f à un polynôme de degré inférieur ou égal à n qui passe par les n+1 points x_i tel que :

$$P_n(x_i) = f(x_i) = y_i \text{ pour } i = 0,1, \ldots . n \qquad (2.2)$$

P_n et f coincident alors en n+1 points x_0, x_1,……, x_n de $[a, b]$.

Méthode de résolution naïve :

Un polynôme de degré m s'écrit passant par les n+1 point x_i s'écrit :

$$P_n(x_i) = \sum_{j=0}^{m} a_j\, x_i^j \text{ , } a_i \in \mathbb{R} \text{ et } i = 0,1, \ldots . n \qquad (2.3)$$

En utilisant (2.2), il vient :

$$P_n(x_i) = f(x_i) = \sum_{j=0}^{m} a_j\, x_i^j \qquad (2.4)$$

qui s'écrit :

$$\begin{cases} a_0 + a_1 x_0^1 + a_2 x_0^2 \ldots a_m x_0^m = f(x_0) \\ a_0 + a_1 x_1^1 + a_2 x_1^2 \ldots a_m x_1^m = f(x_1) \\ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\ a_0 + a_1 x_n^1 + a_2 x_n^2 \ldots a_m x_n^m = f(x_n) \end{cases} \qquad (2.5)$$

Rappel sur l'interpolation polynomiale

Soit :

$$\begin{bmatrix} 1 & \cdots & x_0^m \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{bmatrix} \quad (2.6)$$

Les coefficients a_i sont déterminés en faisant la résolution du système (2.5).

Trois cas sont possibles :

a) $m > n$: aucune solution
b) $m = n$: une solution unique
c) $m < n$: aucune solution

La résolution du système nécessite un très grand nombre d'opération d'où la nécessité de chercher des méthodes plus simples.

II. Méthode de Lagrange

Considérons une fonction connue en (n +1) valeurs distinctes de $x_i (i = 0,1 \ldots . n)$, et considérons les polynômes de Lagrange suivants :

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad \text{avec, } 0 \leq i, j \leq n, \ et \ \ j \neq i$$

$$= \frac{(x-x_0)\,(x-x_1)\,(x-x_2)\ldots(x-x_{i-1})\,(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)\,(x_i-x_1)\,(x_i-x_2)\ldots(x_i-x_{i-1})\,(x_i-x_{i+1})\ldots(x_i-x_n)} \quad (2.7)$$

Nous avons : $\begin{cases} L_i(x_j) = 0 & i \neq j \\ L_i(x_i) = 1 \end{cases}$ (2.8)

Ce qui peut s'écrire,

$L_i(x_j) = \delta_{ij}$, (δ_{ij} étant le symbole de Kronecker) (2.9)

Ces polynômes sont de degré n et sont linéairement indépendants

En effet, pour tout $k = 0,1, \ldots . n$

$$\sum_{i=0}^{n} \lambda_i L_i(x_k) = 0 \quad (2.10)$$

Rappel sur l'interpolation polynomiale

Ce qui donne $\sum_{i=0}^{n} \lambda_i \delta_{ik} = 0$

Ou encore $\lambda_k = 0$ pour $k = 0,1, \dots . n$

Les $n + 1$ polynômes de Lagrange sont linéairement indépendants, ils forment par conséquent une base dans l'espace vectoriel des polynômes.

Sur cette base le polynôme d'interpolation $P_n(x)$ s'écrit :

$$P_n(x) = \sum_{i=0}^{n} P_n(x_i) L_i(x) \qquad (2.11)$$

Ou encore,

$$P_n(x) = \sum_{i=0}^{n} f(x_i) L_i(x) \qquad (2.12)$$

Exrcice 1 : Le programme permet de calculer le polynôme de Lagrange d'ordre inferieur ou égal à n qui interpole les points X(i), Y(i) (i=0, N)) pour une abscisse données lue au clavier.

Programme 2.1: Programme permettant la résolution de l'exercice 1

```
Program LagrangeInterpolation
parameter (N=10)
real:: x(N), y(N) ,k, p, s
print*, 'Donner k' !pour  une abscisse donné
                        donnée
read (*,*)k
 print*," Donner les points d'interpolation"
do i=1, N+1
read (*, *) x(i), y(i)
enddo
 lp=1
do i=1,n
p=1.0
do j=1,n
if (i .ne. j) then
p=p*((k-x(j))/(x(i)-x(j)))
end if
end do
 s=s+( p*y(i) )
end do
print *,"Le polynôme calculé pour ", k,  "est : ", s
stop
end
```

Remarque : Pour implémenter l'équation (2.12) en langage fortran la somme sera de 1 à n+1 au lieu de 0 à n et ce pour éviter des erreurs à la compilation

III. Méthode de Newton

1. Différences divisées

Soit f une fonction dont on connait les valeurs $f(x_0)$, $f(x_1)$,..$f(x_n)$ aux abscisses x_0, x_1,…x_n. On définit les différences divisées par la relation de récurrence :

Rappel sur l'interpolation polynomiale

$$\begin{cases} \delta(x_i) = f(x_i) \\ \delta(x_i, x_{i+1}) = \dfrac{f(x_i)-f(x_{i+1})}{x_i-x_{i+1}} = \dfrac{\delta(x_i)-\delta(x_{i+1})}{x_i-x_{i+1}} \\ \delta(x_i, x_{i+1}, x_{i+2}) = \dfrac{\delta(x_i,x_{i+1})-\delta(x_{i+1},x_{i+2})}{x_i-x_{i+2}} \\ \vdots \\ \delta(x_i, x_{i+1}, \dots, x_{i+p}) = \dfrac{\delta(x_i,x_{i+1},\dots x_{i+p-1})-\delta(x_{i+1},x_{i+2},\dots x_{i+p})}{x_i-x_{i+p}} \end{cases} \qquad (2.13)$$

La dernière relation du système (2.13) est appelée différence divisée d'ordres p de la fonction f aux points x_i, $x_{i+1}, \dots x_{i+p}$.

Pour calculer des différences divisées, on forme le tableau suivant :

x_i	$f(x_i)$	δ_1	δ_2	-----	δ_{n-1}	δ_n
x_0	$f(x_0)$					
		$\delta(x_0, x_1)$				
x_1	$f(x_1)$		$\delta(x_0, x_1, x_2)$	-----		
		$\delta(x_1, x_2)$				
x_2	$f(x_2)$		$\delta(x_1, x_2, x_3)$			
		$\delta(x_2, x_3)$				
x_3	$f(x_3)$				$\delta(x_0, x_1, \dots, x_{n-1})$	
$\vdots$	$\vdots$	$\vdots$		-----		$\delta(x_0, x_1, \dots x_n)$
x_{n-2}	$f(x_{n-2})$				$\delta(x_1, x_2, \dots, x_n)$	
		$\delta(x_{n-2}, x_{n-1})$				
x_{n-1}	$f(x_{n-1})$		$\delta(x_{n-2}, x_{n-1}, x_n)$			
		$\delta(x_{n-1}, x_n)$				
x_n	$f(x_n)$					

Le polynôme $P_n(x)$ qui prends les valeurs $f(x_i)$ aux abscisses x_0, x_1,…x_n, peut s'écrire :

$$P_n(x) = \delta(x_0) + \delta(x_0, x_1)(x - x_0) + \delta(x_0, x_1, x_2)(x - x_0)(x - x_1) + \cdots .. \delta(x_0, x_1, \dots, x_n)(x - x_0)(x - x_1)..(x - x_{n-1}) \quad (2.14)$$

2. *Erreur d'interpolation*

Soit $[a, b]$ un intervalle contenant x_0, x_1,…x_n. On suppose que f est $(n + 1)$ fois continument dérivable sur $[a, b]$.

Alors pour tout x ∈ $[a, b]$, ∃ un $\xi \in [a, b]$ te que :

$$f(x) - P_n(x) = \frac{f^{n+1}}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \quad (2.15)$$

Exercice 2 : Ecrire un programme permettant de calculer les différences divisées pour les points de coordonnées x(i) et y(i), i= 1, n+1.

Programme 2.2 : Résolution de l'exercice 2

```
program NewtonDivisees

 parameter ( max = 4 )

 real :: x( max), y( max, max)

integer :: p

write (*,* ) ' donner x et y '

do i= 1, max

read (*,'(f10.6)')  x( i ) , y (i, 1)
```

```
enddo
k = max
p = 1
do  j = 1,max
    do  i = 1, k-1
        y(i,j+1)=(y(i+1,j)-y(i,j))/(x(i+p)-x(i)) ! difference divisée
       write (*,' (a50, i2, a6, f10.6)' )  ' Les differences divisées a l ordre ', j, 'sont', y(i, j+1)
    end do
    k = k - 1
    p = p + 1
end do
end
```

3. *Différences finies*

Dans le cas des points équidistants l'algorithme de l'interpolation se simplifie énormément. En effet, les abscisses x_i s'écrivent en fonction de la distance h, (h un réel non nul)

$$x_1 = x_0 + h,$$

$$x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots . . x_n = x_0 + nh$$

3.1 Différences finies progressive

Les différences finies progressives sont définies par :

$\Delta^0 y_i = y_i \quad i = 0,1, \ldots . , n \quad ,$ à l'ordre 0

$\Delta y_i = y_{i+1} - y_i \quad i = 0,1, \ldots . , n-1 \; ,$ au premier ordre

$\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i \; , \; i = 0,1, \ldots . , n-2 \; ,$ au second ordre

Rappel sur l'interpolation polynomiale

$$\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i \ , \ i = 0,1, \ldots, n-k \quad \text{à l'ordre k}$$

Le polynôme d'interpolation de f aux points x_i avec $x_i = x_0 + ih$ et $i = 0,1, \ldots, n$ s'écrit :

$$P_n(x) = f(x_0) + \frac{\Delta f(x_0)}{1!\,h}(x - x_0) + \frac{\Delta^2 f(x_0)}{2!\,h^2}(x - x_0)(x - x_1) + \cdots$$

$$\ldots + \frac{\Delta^n f(x_0)}{n! h^n}(x - x_0)(x - x_1)..(x - x_{n-1}) \qquad (2.16)$$

Exercice 3 : Ecrire un programme permettant de calculer les différences finies progressives pour les points de coordonnées x(i) et y(i), i= 1, n+1.

Programme 2.3 : Résolution de l'exercice 3

```
program NewtonFinies
 parameter ( max = 4 )
 real :: x( max), y( max, max)
integer :: p
write (*,* ) ' donner x et y '
do i= 1, max
read (*,'(f10.6)') x( i ) , y (i, 1)
enddo
k = max
do  j = 1,max
    do  i = 1, k-1
         y(i,j+1)=(y(i+1,j)-y(i,j))               ! difference finies
      write (*,' (a50, i2, a6, f10.6)' )  ' Les differences finies a l ordre ', j, 'sont', y(i, j+1)
    end do
    k = k - 1
end do
end
```

Résolution d'un système d'équation linéaire

I. Méthodes directes

Une méthode numérique est directe si elle aboutit à la solution exacte du problème.

1. Méthode de Gauss

Soit le système d'équations linéaire suivant :

$$Ax = b \tag{3.1}$$

A est une matrice (n, n) régulière x et b sont deux vecteurs à n composantes.

Le but de la méthode est de transformer la matrice A en une matrice triangulaire supérieure. C'est-à-dire faire des opérations sur A et b de telle sorte que A soit transformée en A' et b en b'.

Ou encore la matrice augmentée $[A, b]$ soit transformée en $[A', b']$.

$$[A, b] = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{22} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{1n} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix} \tag{3.2}$$

Transformation de Gauss

$$[A', b'] = \begin{pmatrix} a'_{11} & 'a_{12} & \dots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{nn} & b'_n \end{pmatrix} \tag{3.3}$$

A l'ordre k, dans le cas où $a_{kk}^{(k)} \neq 0$, la transformation de Gaus s'écrit:

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} & i = \overline{1, k} \;\; j = \overline{1, n} \\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} . a_{ij}^{(k)} & i = \overline{k+1, n} \;\; j = \overline{1, n} \end{cases} \tag{3.4}$$

et,

$$\begin{cases} b_i^{(k+1)} = b_i^{(k)} \quad i = \overline{1,k} \\ b_i^{(k+1)} = b_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} . b_k^{(k)} \quad i = \overline{k+1,n} \end{cases} \tag{3.5}$$

Si à un ordre donné $a_{kk}^{(k)} = 0$ on effectue une permutation des lignes $L_k^{(k)}$ et $L_p^{(k)}$ (où, $k + 1 \leq p \leq n$,

$L_k^{(k)}$ et $L_p^{(k)}$ sont des lignes d'indice k et p respectivement à l'ordre k).

Programme 3. 1 : Résolution d'un système d'équation linéaire donné par l'équation (3.1) par la méthode de Gauss.

```
program Gauss
 parameter (max=3)
 double precision ::a(max,max), ab(max,max+1), b(ma
write(*,*)'Donner les composantes  a(i,j) '
do i=1,max
   do j=1,max
      read(*,*)a(i,j)
    enddo
enddo
write(*,*)'Donner les composantes  b(i)'
```

```
! Triangularisation
 k=1
 do while (k<=max-1)
       i=k+1
       do while (i<=max)
           j=k
           do while (j<max+1)
 ab(i,j)=ab(i,j)-(ab(i,k)/ab(k,k))*ab(k,j)
```

```
do i=1,max
read(*,*)b(i)
enddo
! Matrice augmenté [ab]
do i=1,max
  do j=1,max
      ab(i,j)=a(i,j)
   enddo
enddo


do i=1,max
ab(i,max+1)=b(i)
enddo
! Affichage de la matrice augmenté ab
do i=1,max
write(*,'(4F10.5)')(ab(i,j), j=1,max+1)
 enddo
```

```
   j=j+1
        enddo
       write(*,'(4F10.5)')(ab(i,j), j=1,max+1)
        i=i+1
   enddo
      k=k+1
 enddo
!Resolution
x(max)=b(max)/a(max,max)
do i = max-1,1,-1
x(i) = b(i)
  do j = i+1,max
      x(i) = x(i)-a(i,j)*x(j)
   enddo
x(i) = x(i) / a(i,i)
enddo
! Affichage de la solution
do i=1,max
write(*,*)'x(',i,')',x(i)
enddo
end
```

2. *Méthode de Gauss Jordan*

$$\begin{cases} a_{kj}^{(k+1)} = \dfrac{a_{kj}^{(k)}}{a_{kk}^{(k)}} & j = \overline{k,n} \\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} . a_{kj}^{(k+1)} & i = \overline{1,n}\,,\ i \neq k,\ j = \overline{k+1,n} \end{cases} \qquad (3.6)$$

$$\begin{cases} b_{k}^{(k+1)} = \dfrac{b_{i}^{(k)}}{a_{kk}^{(k)}} & i = \overline{1,k} \\ b_{i}^{(k+1)} = b_{i}^{(k)} - a_{ik}^{(k)} . b_{k}^{(k+1)} & i = \overline{k+1,n} \end{cases} \qquad (3.7)$$

Programme 3.2 : Résolution par la méthode de Gauss-Jordan

```
program GaussJordan
 !Matrice augmentée
real, allocatable :: a (:,:)
 real :: fact, solution
 write (*,*) ' donner le nombre de variable n '
  read (*,*) n
  allocate (a ( n, n+1) )
 write (*,*)'Donner la matrice augmentée a:'
!lecture de la matrice
do i=1,n
read (*,*) (a(i,j),j=1, n+1)
end do
! Diagonalisation
 do k = 1,n
```

```
 ! Affichage de la matrice
 write (*,'(/)')
 write (*,*)'La matrice diagonale est:'
do i = 1,n
write (*, '(4F10.6)') (a(i,j) /a(i,i),j =1,n+1)
enddo
! Affichage des résultats
write (*,'(/)')
write (*,*)'la solution est :'
 do i = 1,n
   solution = a (i, n+1) / a(i,i)
write(*,'(a5,i2,a5,F10.6)')'x(',i,') =',  solution
 end do
```

```
    do i=1,n
      if (i. ne. k) then
        fact = a(i,k) / a(k,k)
        do j=1,n+1
          a(i,j) = a(i,j) - a(k,j)*fact
        end do
      end if
    end do
  end do
end program GaussJordan
```

II. Méthodes itératives

Une méthode numérique est dite itérative si elle utilise commençant par une solution initiale choisie et aboutissant à une solution approchée après un certain nombre d'itérations successives.

1. Méthode de Jacobi

Soient deux vecteurs $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ et $b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

L'équation $Ax = b$ peut s'écrire ;

$$\begin{cases} x_1 = (b_1 - a_{12}\, x_2 - a_{13}x_3 - \cdots - a_{1n}\, x_n)\dfrac{1}{a_{11}} \\ x_2 = (b_2 - a_{22}\, x_1 - a_{13}x_3 - \cdots - a_{2n}\, x_n)\dfrac{1}{a_{22}} \\ \vdots \\ x_n = (b_n - a_{n1}\, x_1 - a_{n2}x_2 - \cdots - a_{n-1n}\, x_{n-1})\dfrac{1}{a_{nn}} \end{cases} \tag{3.8}$$

Ou encore,

$$x_i = \left(b_i - \sum_{\substack{j \\ i \neq j}}^{n} a_{ij} x_j\right) . \frac{1}{a_{ij}} \tag{3.9}$$

Algorithme :

$$x_i^{k+1} = \left(b_i - \sum_{\substack{j \\ i \neq j}}^{n} a_{ij} x_i^k\right) . \frac{1}{a_{ij}} \tag{3.10}$$

On commence par un choix de x^0 (ordre k=0)

On obtient les ordres supérieurs à partir de l'équation de récurrence (3.10).

A chaque ordre k et à chaque valeur de i on fait le teste de convergence $\frac{\left|x_i^{k+1} - x_i^k\right|}{x_i^k} \leq \varepsilon$

Programme 3.3 : Résolution par la méthode de Jacobi

```
program JacobiMethod
real, allocatable :: a(:,:), b(:), xnouv(:), xancien(:)
write (*,*) '  donner n'
read (*,*) n
allocate ( a(n,n) )
allocate (b(n))
allocate (xnouv(n))
allocate (xancien(n))
eps=0.1
do i=1,n
  do j=1,n
      read (*,*) a(i,j)
  enddo
 enddo
do i=1,n
   read (*,*) b(i)
enddo
do  i=1,n
    read (*,*) xancien(i)
enddo
 k=1  ! à l'ordre 1
1   do i=1,n
        do j=1,n
          if (i.ne.j) then
          s=s+a(i,j)*xancien(j)
          endif
        enddo
     xnouv(i)=(b(i)-s)/a(i,i)
   enddo
do i=1,n
test=abs(xnouv(i)-xancien(i))/ abs(xancien(i))
write(*,*) 'test=', test
  if ( test > eps) then
    write (*,*) 'Ordre k','test plus grand que eps'
    k=k+1 ! ordre suivant
      do j=1,n
          xancien(j)= xnouv(j)
      enddo
    goto 1
    else
        write (*,*)' Ordre k', k, xnouv
    endif
  enddo
end
```

Intégration approchée

I. Méthode des trapèzes

L'arc AB est remplacé par le segment AB.

Dans ce cas l'intégrale $I = \int_a^b f(x)\,dx$ est donnée par

l'aire du trapèze aABb,

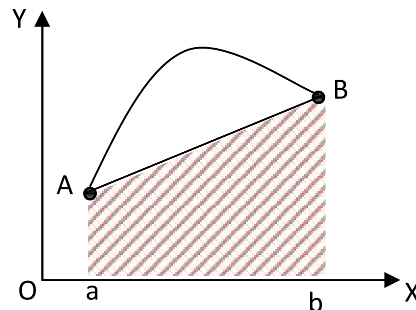


Figure 4.1 : l'intégrale $I = \int_a^b f(x)\,dx$ approché à l'aire du trapèze aABb

$$\int_a^b f(x)\,dx \cong \frac{b-a}{2}[f(a) + f(b)] \qquad (4.1)$$

Pour beaucoup plus de précision, on divise le segment $[a, b]$ en n segments égaux et on applique à chacun la formule des trapèzes.

Soient, $x_0 = a$, $x_1 = x_0 + h$, ……$x_n = x_{n-1} + h = b$, $h = \frac{b-a}{n}$

On trouve :

$$\int_{x_0}^{x_n} f(x)\,dx \cong \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)\,dx + \cdots \ldots \int_{x_{n-1}}^{x_n} f(x)\,dx \qquad (4.2)$$

$$\cong h\left[\frac{f(x_0)+f(x_n)}{2} + f(x_0) + f(x_1) + f(x_2) \ldots\ldots f(x_{n-1})\right] \qquad (4.3)$$

Ou encore :

$$\int_a^b f(x)\,dx \cong h = \frac{b-a}{n}\left[\frac{f(a)+f(b)}{2} + f(x_0) + f(x_1) + f(x_2) \ldots\ldots f(x_{n-1})\right] \quad (4.4)$$

II. Méthode de Simpson

Posons : $x_0 = a$, $x_1 = \frac{a+b}{2}$, $x_2 = b$

Calculons l'intégrale :

$$I = \int_{x_0}^{x_2} f(x)\,dx \quad (4.5)$$

Pour ce faire on fait l'approximation de f par le polynome de Lagrange (pour n=2) (voir $2^{ème}$ partie :TP 2)

$$P_2(x) = \sum_{i=0}^{2} f(x_i)\,L_i(x) \quad (4.6)$$

Donc,

$$I = \int_{x_0}^{x_2} \sum_{i=0}^{2} f(x_i)\,L_i(x)\,dx = \sum_{i=0}^{2} f(x_i) \int_{x_0}^{x_2} L_i(x)\,dx = \sum_{i=0}^{2} f(x_i)\,A_i \quad (4.7)$$

Avec $A_i = \int_{x_0}^{x_2} L_i(x)\,dx$. L_i étant le polynôme de Lagrange au point x_i

$$L_i(x) = \frac{\prod_{j=0, j\neq i}^{n}(x-x_j)}{\prod_{j=0, j\neq i}^{n}(x_i-x_j)} \quad (4.8)$$

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-x_1)(x-x_2)}{2h^2} \quad (4.9)$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-x_0)(x-x_2)}{2h^2} \quad (4.10)$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x-x_0)(x-x_1)}{2h^2} \quad (4.11)$$

$$A_0 = \int_{x_0}^{x_2} L_0(x)\, dx = \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{2h^2}\, dx = \int_{x_0}^{x_2} \frac{(x-x_0-h)(x-x_0-2h)}{2h^2}\, dx \quad (4.12)$$

$$= \frac{1}{2h^2} \int_{-h}^{h} u(u-h)du \quad (4.13)$$

Avec $u = x - x_0 - h$ (4.14)

$$A_0 = \frac{1}{2h^2}\left[\frac{u^3}{3} - \frac{u^2}{2}\right]_{-h}^{h} = \frac{h}{3} \quad (4.15)$$

$$A_1 = \int_{x_0}^{x_2} L_1(x)\, dx = \frac{4h}{3} \quad (4.16)$$

$$A_2 = \int_{x_0}^{x_2} L_2(x)\, dx = \frac{h}{3} \quad (4.17)$$

L'approximation d'ordre 2 de $\int_{x_0}^{x_2} f(x)\, dx$ est donné donc par :

$$\int_a^b f(x)\, dx \cong \frac{b-a}{6}[f(a) + 4f(c) + f(b)] \quad (4.18)$$

Avec,

$$c = \frac{a+b}{2} \quad (4.19)$$

Dans le cas général, affin de trouver des résultats plus précis, on décompose l'intervalle $[a, b]$ en n intervalles égaux aux quels on applique séparément la méthode de Simpson.

Soient $x_0 = a$, $x_1 = x_0 + h$, ……$x_n = x_{n-1} + h = b$, $h = \frac{b-a}{n}$

On trouve :

$$\int_{x_0}^{x_n} f(x)\, dx \cong \int_{x_0}^{x_1} f(x)\, dx + \int_{x_1}^{x_2} f(x)\, dx + \cdots \ldots \int_{x_{n-1}}^{x_n} f(x)\, dx \quad (4.20)$$

$$\cong \frac{h}{3}\left[f(x_0) + f(x_n) + 4\big(f(x_1) + f(x_3) + \cdots . f(x_{n-1})\big) + 2\big(f(x_2) + f(x_4) + \cdots . f(x_{n-2})\big)\right]$$

(4.21)

Ou encore :

$$\int_a^b f(x)\,dx \cong$$

$$\frac{b-a}{3n}\left[f(a) + f(b) + 4\big(f(x_1) + f(x_3) + \cdots . f(x_{n-1})\big) + 2\big(f(x_2) + f(x_4) + \cdots . f(x_{n-2})\big)\right]$$

(4.22)

Exercice 1 : calcul de l’intégrale entre deux bornes a et b donnés d’une fonction définie par :

$$f(x) = x^2$$

En utilisant la méthode des trapèzes. N étant le nombre se subdivision de l’intervalle [a, b]

Programme 4.1 : Résolution de l'exercice 1

```
program Trapeze
implicit none
double precision:: a, b, x ,y, h, S, f
integer:: N, i
write (*,*)'Donner N, a et b'
read(*,*) N, a, b
h=(b-a)/N
S=h*(f(a)+f(b))/2
do i=1,N-1
x=a+ i*h
y =f(x)
S= S+ y*h
enddo

! Affichage du résultat
write (*,*) ' l intégrale de f est   ', S
end

function f(x)
double precision:: x, f
f=x**2
end function
```

Exercice 2: En utilisant la méthode de Simpson, écrire un programme qui permet de calculer l'intégrale entre deux bornes a et b donnés de la fonction définie par :

$$f(x) = x^2$$

Programme 4.2 : Résolution de l'exercice 2

```
program Simpson
implicit none
double precision:: a, b, x ,y, h, S, f,S0,S1,S2
integer:: N, i
write (*,*) 'Donner N,a et b'
read (*,*) N,a,b
h = (b-a)/N
s0=f(a)+f(b)
! la somme des termes impairs
do i=1, N-1,2
x= a +i *h
y =f(x)
S1=S1+y
enddo
! la somme des termes pairs
do i=2, N-2,2
x = a+ i*h
y = f(x)
S2 = S2+ y
enddo
S= (h/3.)*(S0+4*S1+2*S2)
! Affichage du résultat
write (*,*) S
end
function f(x)
double precision:: x, f
f=x**2
 end
```

Résolution d'équations non linéaires

I. Racines d'équations

Soit f une fonction définie de $\mathcal{D}_f$ dans $\mathbb{R}$ ($\mathcal{D}_f \subset \mathbb{R}$).

On dit que α est racine de l'équation :

$$f(x) = 0 \qquad (5.1)$$

si $f(\alpha) = 0$.

On dit aussi que α est un zéro de f si f s'écrit sous forme polynomiale

Théorème des valeurs intermédiaires : Si f est une fonction continue sur un intervalle $[a, b]$ alors elle prend toute les valeurs comprises entre $f(a)$ et $f(b)$ au moins une fois.

II. Méthode de la dichotomie

La méthode de dichotomie ou méthode de bissection est un algorithme mathématique utilisé pour la recherche d'un zéro de la fonction f continue dans un intervalle donné. La méthode consiste à diviser l'intervalle en deux parties puis sélectionner le sous intervalle qui contient la racine de la fonction

Etant donnée une fonction continue sur un intervalle $[a, b]$ telle que $f(a)$ et $f(b)$ soient de signes opposés.

D'après le théorème des valeurs intermédiaires, il existe au moins un zéro de f dans $[a, b]$.

Soit c un point tel que,

$$c = \frac{a+b}{2} \qquad (5.2)$$

Deux cas sont possibles :

- $f(c).f(a) < 0$ la racine existe alors entre a et c. L'algorithme de dichotomie est appliqué à l'intervalle $[a, c]$

- $f(c).f(b) < 0$ la racine existe entre c et b. L'algorithme de dichotomie est appliqué à l'intervalle $[c, b]$

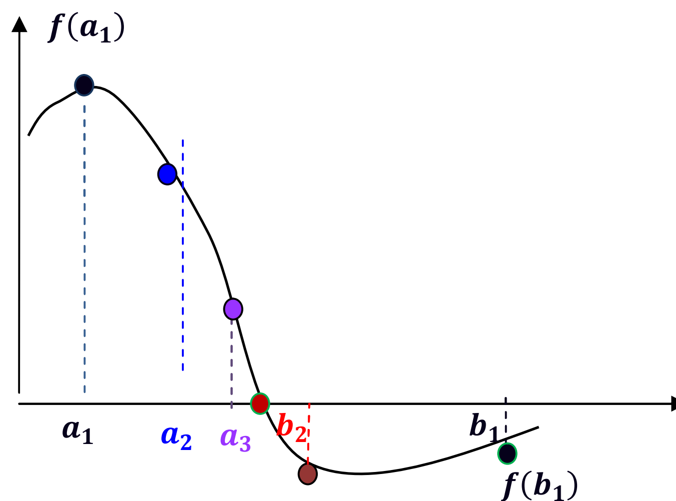


Figure 5.1 illustration des étapes successives de la méthode de dichotomie dans l'intervalle $[a_1, b_1]$. La racine de la fonction est en rouge.

Exercice 1 : En utilisant la méthode de dichotomie, écrire un programme qui fait la résolution de l'équation $f(x) = 3x - 1$

Programme 5.1 : Résolution de l'exercice 1

```
program dichotomie
double precision :: a,b,c
print*, 'Donner a et b tel que f(a).f(b)<0'
read (*,*) a,b

if (f(a)*f(b)>0) then     ! test  f(a).f(b)
   print*,' la solution est hors intervalle [a b] '
elseif ( f(a)*f(b) = = 0) then
   print*,'la solution peut etre', a, 'ou', b
else
   20 c= (a + b) / 2.
                 if ( f(c) = = 0 ) then  ! test f(c)
                    print*, "la racine de l'équation est  ", c
                  else if ( f(a)*f(c) > 0 ) then
                           a=c
                            goto 20
                   else
                             b=c
                             goto  20
                   end if  !  fin test f(c)
 end if ! fin test
stop
end

function f(x)
double precision :: x
f=3*x-1
end
```

III. Méthode de Newton-Raphson

C'est une méthode numérique itérative permettant de résoudre l'équation 4.1. La fonction f est supposée continue et dérivable (de classe C^2) sur l'intervalle $[a, b]$ entourant la racine α

On note x^*une racine exacte et x_0 une valeur approchée de x^*.

Le développement de Taylor de f à l'ordre deux s'écrit,

$$f(x^*) = f(x_0) + f'(x_0)(x^* - x_0) + \frac{f''(\xi)}{2}(x^* - x_0)^2 \qquad (5.\ 3)$$

où $\xi \in (x^*, x_0)$.

Puisque x^* est une racine exacte, $f(x^*) = 0$. En supposant $f'(x_0) \neq 0$, on obtient,

$$x^* = x_0 - \frac{f(x_0)}{f'(x_0)} - \frac{f''(\xi)}{2f'(x_0)}(x^* - x_0)^2 \qquad (5.4)$$

et en négligeant le reste $O_2 = \frac{f''(\xi)}{2f'(x_0)}(x^* - x_0)^2$ on obtient :

à l'ordre 1 : $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ (5.5)

x_1 constitue la valeur approchée améliorée de x^*. Cette valeur peut être

Encore améliorée en utilisant la formule de récurrence à l'ordre k,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 0,1,2, \dots \qquad (5.6$$

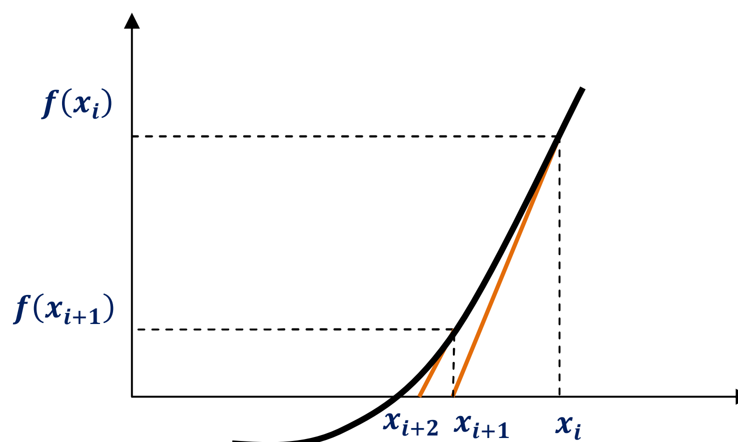


Figure 5.2 illustration géométrique de la méthode de Newton-Raphson

1. Convergence de la méthode

Dans l'équation 4.6, si en un point x_k la tangente $f'(x_k)$ s'annule dans $[x^*, x_k]$ alors x_{k+1} peut être plus éloigné de x^* que x_k

Les conditions suffisantes de convergence sont [4] :

Si $f'(x_k).f''(x_k) > 0$ et si $f'(x_k)$ et $f''(x_k)$ ne changent pas de signes dans $[x^0, x^*]$ alors la méthode converge vers x^*. Sinon la méthode peut diverger (converger ou non)

2. Critère d'arrêt

Dans le cas ou la méthode est convergente les solutions approchés x_k s'approchent de x^* à chaque itération. Il faut donc se donner un critère pour arrêter les calculs une fois que x_k soit suffisamment proche de x^*. En pratique, on utilise l'un des trois critères pour arrêter les calculs:

- Limiter le nombre maximal d'itération $n > n_{max}$
- $|x_{k+1} - x_k| < \varepsilon_1$ (ε_1 très petit)
- $|f(x_k)| < \varepsilon_2$. (ε_2 très petit) Lorsque x et très proche de la valeur exacte x^* , $f(x)$ est très proche de 0

Exercice 2 : En utilisant la méthode de Newton-Raphson, écrire un programme qui permet la résolution de l'équation :

$$f(x) = x^2 - 4$$

Programme 5.2 : Résolution de l'exercice 2

```
program NewtonRaphson
double precision :: x0,x1,err
print*, 'donner x0'
read (*,*) x0
err=0.0001
k=1
20 x1=x0 -f(x0)/fp(x0)
if (abs((x1-x0)/x0)<=err)       then
print*,' ordre ', k, x1
else
  x0=x1
  k=k+1
  goto 20
  endif
   stop
  end
```

```
! La function f
function f(x)
  double precision :: x
  f=x**2-4
  end function f
! La dérivée de la fonction
  function fp (x)
  double precision :: x
  fp=2*x
    end function fp
```