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

UNIVERSITE MUSTAPHA STAMBOULI DE MASCARA

FACULTÉ DES SCIENCES ET TECHNOLOGIE

Polycopié de Cours
PROGRAMMATION SCIENTIFIQUE

Présenté par **:**

ZENGAH Sahnoun

Ce cours est destiné aux étudiants de Parcours Sciences et Techniques / spécialité 2ème Master Mécanique et science pour l'ingénieur

Algérie
2015

Avant Propos

Ce cours s’adresse aux étudiants de fin de cycle (licence et master) des sciences techniques des disciplines génie mécanique et physique énergétique,

La matière Programmation scientifique a pour but de vous donner les connaissances de base nécessaires à la compréhension des algorithmes les plus couramment utilisés dans de modélisation en mécanique. La référence est le Fortran qui reste (malheureusement, diraient certains) un langage très utilises dans le domaine du calcul numérique par l'industrie et le monde de la recherche. La raison principale est les milliers de bibliothèques contenant des millions de lignes de code Fortran qui ont été développées, testées et validées depuis plus de 50 ans. C'est un investissement lourd qu'il faut continuer d'exploiter

Table des matières

Résumé

L'ordinateur permet de calculer et d'ordonner. Pour effectuer cette tâche principale l'ordinateur est composé d'un microprocesseur pour calculer et de la mémoire pour ranger les données et les résultats. Le microprocesseur n'effectue que des opérations simples sur des nombres codés en binaire (1 et 0) : additionner, soustraire, multiplier, diviser, lire dans la mémoire, écrire dans la mémoire, ……etc. .

D'autre part ces calculs même si simple ne peuvent être effectué sans l'intervention de l'utilisateur:

Pour calculer les racines d'une équation 2ème degré, le microprocesseur ne sait pas de lui-même manipuler les données, analyser tous les cas possibles et trouver les résultats correspondant à chaque cas.

Pour calculer la constante des gaz parfaits à partir des conditions standard, le microprocesseur doit recevoir toutes les données et la formule permettant d'avoir ce résultat.

La communication entre l'homme et le microprocesseur qui utilise le langage machine s'effectuée à travers des langages commun (langages informatiques).

Un langage informatique est constitué par :

- un ensemble d'instructions ou mots-clés
- un ensemble d'objets manipulables
- des règles de syntaxe
- de structures logiques

Le programme élaboré doit répondre à toutes les exigences et les règles du langage pour qu'il puisse être exécuté et fournir des résultats corrects. Parmi les langages informatiques existants on site : Fortran, C, Visual Basic, C++, Delphi, Matlab …, Le Fortran a été adopté comme langage de programmation car il s'avère très puissant dans les calculs scientifiques et ainsi pouvant répondre a nos besoins dans la résolution numérique des modèles mathématiques régissant les phénomènes physiques étudiés dans les modules d'électronique, optique, mécanique,….

Mots-clés : fortran, programmation, numérique, calcule, methode.

Introduction

Le FORTRAN est un langage structuré, de syntaxe simple et facile à apprendre, qui a su évoluer, a été adopté comme langage de programmation car il s'avère très puissant dans les calculs scientifiques et ainsi pouvant répondre a nos besoins dans la résolution numérique des modèles mathématiques régissant les phénomènes physiques étudiés dans les modules d'électronique, optique, mécanique,….

Le FORTRAN C'est un langage universel: la plupart des scientifiques mathématiciens, physiciens, chimistes, mécanique, biologistes, ingénieurs, etc.

Le FORTRAN C'est un langage utilisé depuis 60 ans : les programmes et bibliothèques de routines disponibles se chiffrent en millions de lignes. On trouve toujours le bout de code que l'on cherche dans une bibliothèque existante.

Le premier compilateur FORTRAN fut conçu et implémenté par John Backus (IBM Lab, San José California) en 1954. Il s'agissait du FORTRAN I. L'objectif de Backus était de concevoir un langage de programmation qui permettrait de réaliser des programmes en utilisant une syntaxe la plus proche possible de la syntaxe mathématicienne. Le terme FORTRAN traduit cette ambition, puisqu'il signifie FORmula TRANslation.

FORTRAN I était un peu primaire, mais quand même très supérieur aux assembleurs et langages machines utilisés au début des années 50. Il évolua rapidement en FORTRAN II et FORTRAN III.

FORTRAN prit son essor dans les années 60, surtout à partir de 1966, date à laquelle l'ANSI (organisme de standardisation américain) publia la norme ANSI66 qui normalisa le FORTRAN IV. A partir de cette date, tous les compilateurs utilisaient la même syntaxe. Un chercheur pouvait donc utiliser un code source écrit en FORTRAN IV sur n'importe quelle machine

Le FORTRAN est donc un langage qui a un demi-siècle d'existence, mais qui reste un des plus utilisés en calcul scientifique.

Le langage fortran évolue régulièrement, mais la véritable évolution a été faite en passant du fortran 77 au fortran 90. Les nouvelles normes fortran 95 (puis fortran 2003, fortran 2008) n'apportent que des fonctionnalités nouvelles.

Chapitre 1 **Les Bases du Fortran**

1.1 Organisation d'un programme FORTRAN

Comme tous les programmes, un programme FORTRAN est un ensemble organisé d'instructions données par le programmeur à la machine. Nous parlerons ici essentiellement de code source (ou programme source, on dit aussi "source").

Un programme FORTRAN, du moins son code source, le plus simple est organisé comme suit:

1.1.1 La particularité d'un code source FORTRAN

Les contenu des lignes en Fortran doit respecter les quelques règles suivantes:

- les instructions doivent commencer en colonne 7 et leur longueur ne doit pas dépasser la colonne 72
- Une ligne de commentaire doit commencer par un C ou une * en première colonne. Tout ce qui suit le point d'exclamation ! dans une ligne est considéré comme un commentaire.
- Les étiquettes (Labels) doivent être écrites dans les colonnes 2 à 5
- pour couper une instruction en plusieurs lignes, on positionne un caractère quelconque (je mets un!) dans la colonne 6 de chacune des lignes suivant la première ligne.

FORTRAN n'était pas sensible à la casse minuscule ou majuscule. Mais vous avez sans doute remarqué aussi que j'écrivais tous les mots clé en majuscule. Ce n'est pas obligatoire, mais c'est une coutume en FORTRAN (elle provient des premiers standards de FORTRAN).

1.1.2 Structure générale d'un programme :

Dans le cas général un programme fortran se présente sous la structure suivante :

PROGRAM < nom du programme>

Déclaration des données et des résultats

Ouverture des fichiers de données et de résultats

Lecture des données

Calcul et instructions de commande

Ecriture des résultats

END

Les mots PROGRAM et END sont les mots clé FORTRAN qui délimitent le programme pricipal.

Le programme peut inclure des sous programmes appelés par le programme principal.

- PROGRAM

C'est une instruction facultative qui définit le nom du programme, sa syntaxe est :

PROGRAM < Nom du programme>

- Déclaration

C'est dans cette partie qu'on définit les objets qui seront manipuler par le programme, la syntaxe de déclaration est :

TYPE < Liste des objets>

- Ouverture des fichiers

L'ouverture des fichiers est nécessaire lorsque le programme est alimenté par des données inscrites dans des fichiers externes, et dans le cas ou les résultats doivent être stocker dans des fichiers externes, dans la syntaxe on indique au minimum le numéro du fichier et son chemin :

OPEN (<n°>, FILE = ' <Chemin et nom du fichier>')

- Lecture des données

Cette partie permet d'alimenter le programme par des données introduites soit directement sur clavier ou à partir d'un fichier déjà ouvert par l'instruction OPEN. On utilise l'instruction READ dont la syntaxe précise le numéro d'unité de lecture des données et le format de lecture.

- Instructions et calcul

Les données sont manipulées par les opérations arithmétiques et logiques, et suivant l'algorithme du problème étudié (conditionnel, répétitif, …), on utilise des instructions Fortran avec une syntaxe spécifique : IF, DO, WHILE, GOTO, CONTINUE, …..etc. Cette partie peut contenir des appels aux sous programme.

- Ecriture des résultats

Les résultats d'exécution des instructions du programme restent dans la mémoire de l'ordinateur, pour les afficher sur écran ou sur un fichier déjà ouvert par l'instruction OPEN, on emploi l'instruction WRITE avec une syntaxe qui définit l'unité de stockage des résultats et le format d'affichage.

- END

Instruction obligatoire a la fin d'un programme ou d'un sous programme.

1.2 Declarations des objets

La mémoire de l'ordinateur est partagée en un certain nombre de cellules. Une fois le programme affecte un nom a une cellule, cette dernière se prête à recevoir une donné. La déclaration d'une donnée consiste à lui réserver une place dans la mémoire par son nom, la taille réservée dépend du type de la donnée (Entier, réel, caractère, ……..).

La syntaxe d'une déclaration en fortran est :

TYPE < Liste des objets>

Les objets (données, constantes ou variables) sont séparés par des virgules.

1.2.1 Les données :

Les données sont soit de type numérique (Entier, réel, double précision et complexe), soit de type caractère ou de type logique :

1.2.2 Données de type entier :

Un entier est représenté base binaire sur 4 octets (31 bits pour la valeur plus un bit pour le signe). Les valeurs représentées par le type entier sont situées sur l'intervalle $[-2^{31},2^{31}-1]$.

La syntaxe de déclaration est :

INTEGER <liste des données entières>

1.2.3 Données de type réel:

Un réel est représenté en base 2 en virgule flottante sur 4 octets. Les valeurs représentées par le type réel simple précision sont situées dans l'intervalle $[1.401x10^{-45},3.403x10^{38}]$.

La syntaxe de déclaration est :

REAL < liste des objets>

1.2.4 Données de type double précision :

La double précision est un réel plus précis, codé en virgule flottante sur 8. Les valeurs (en valeur absolue) sont comprises entre $[4.941x10^{-324},1.798x10^{308}]$ avec 15 chiffres significatifs.

La syntaxe de déclaration est :

DOUBLE PRECISION < Liste des données >

1.2.4.1 Données de type complexe :

Une donnée de type complexe occupe deux octets ; un octet pour la partie réelle et un octet pour la partie imaginaire.

La syntaxe de déclaration est :

COMPLEX <liste des données complexes>

1.2.5 Données de type caractère :

Le Fortran a été développé principalement pour manipuler des données de type numérique a travers des formules mathématique (Formula Translator). Cependant Fortran peut manipuler même des chaînes de caractères par des opérations appropriées.

La syntaxe de déclaration d'une chaîne de caractère consiste à spécifier le nombre maximal de caractères de la chaîne :

CHARACTER*n < Liste des données caractères>

n c'est la longueur maximale des données caractères.

1.2.5.1 Données de type Logique :

Une donnée de type logique occupe 4 octets de mémoire, sa valeur est .True. ou .False.
La syntaxe de déclaration est :

LOGICAL <Liste des données>

1.2.6 Les constantes numériques:

1.2.7 Constantes intègre : constantes entières

Exemples :

2002 300000 -1422

1.2.7.1 Constantes real:

Une constante de type réel comporte soit un point décimal, soit le caractère e en notation virgule flottante

Exemples :

1. 1.0 3.1415 31415e-4 1.6e-19 1e12 .001

1.2.7.2 Constantes double précision :

Dans ce type la lettre e est remplacée par d

Exemples :

0.d0 1.d0 1d0 3.1415d0 31415d-4 1.6d-19 1d12 -36.d0

1.2.7.3 Constantes complexe :

Dans la constante complexe on présente la partie réelle et imaginaire, qui sont de type réel entre parenthèses

Exemples :

(-2.73, 3.1415) (2.2e4, 3.14e-2)

1.2.8 Les variables:

Une variable est un emplacement en mémoire référencé par un nom, dans lequel on peut lire et écrire des valeurs au cours du programme. D'autre part les variables permettent de manipuler des symboles et de programmer des formules.

1.2.8.1 Noms des variables:

Le nom d'une variable peut contenir des lettres alphabétiques, des chiffres et quelques caractères.

1.2.8.2 Déclaration de variables:

Une variable est déclarée par son non et son type qui peut être integer, real, double precision, complex, logical ou character. Si une variable n'est pas déclarer au début du programme elle sera considérée par défaut comme entière si elle commence par les six lettres i, j, k, l, m, n, une variable commençant par les autres lettres (de a à h et de o à z) est considérée comme réelle.

1.2.8.3 Affectation d'une variable:

Lors de la déclaration d'une variable, un espace mémoire lui y réserver, au cours du programme cette espace est rempli tout en affectant des valeurs a ces variables.

1.2.8.4 Syntaxe:

var = constante	Ex : charge=1.6E-19
var = autre variable	Ex : charge1=charge2
var = opération	Ex : E=m*C**2

Chapitre 2 **Operateurs et Fonctions Mathematiques**

2.1 Opérateurs arithmétiques

Les opérateurs arithmétiques sont les opérateurs utilisés dans les calculs courants. Mais l'évaluation d'une expression mathématique qui contient plusieurs symboles, exige un certain ordre de priorité de calcul.

FORTRAN supporte les opérateurs arithmétiques suivants:

- Addition +
- Soustraction -
- Multiplication *
- Division /
- Puissance **

Les sous expressions regroupées entre parenthèses sont évalués en premier.

Lorsque les opérateurs sont de même priorité, les calculs sont effectués de la droite vers la gauche.

Exemple1:

$x = a + \frac{b}{c} - d$ En langage fortran x=a+b/c-d

$y = \frac{a+b}{c-d}$ En langage fortran y=(a+b)/(c-d)

$f = a^b + \frac{c}{d}$ En langage fortran f=a**b+c/d

Exemple2 **:**

a = 2,
b = 4
c = a + b / 2
donne c = 4
(la division est prioritaire sur l'addition)
c = (a + b)/2
donne c = 3

2.1.1 Conversion de type :

Comme les opérateurs arithmétiques, les types d'objets suivent un ordre de priorité, ce qui affecte profondément le résultat des calculs.

Les différents types par ordre de priorité croissante sont:

Logical, integer, real, double precision, complex

Exemples:

- i, j deux entiers tel que i = 1, j = 2,

  le résultat de i / j sera du type entier donc 0

- a, b deux réels tel que ; a = 1, b = 2,

  le résultat de a / b sera du type réel donc 0.5

- On considère a et c de type Real et b de type Double precision,

  on calcul delta = b ** 2 - 4 * a* c

  b ** 2 sera du type Double precision

  4 * a* c sera du type le plus fort de a et de c, donc Real.

  Delta sera du type le plus fort, donc Double precision, mais le produit 4*a*c sera effectué en simple précision, d'où une perte de précision possible.

- 1 / 2 donne un résultat entier qui est 0
- 1./ 2 1/2. ou 1. / 2. donne un résultat réel qui est 0.5
- k = 3.1415, si k n'est pas déclarer comme réel, il sera considéré par défaut comme entier, ce qui affectera seulement la valeur 3 à k

2.2 Fonctions mathématiques

Certaines fonctions mathématiques sont prédéfinies dans le langage. On n'utilisera pas la même fonction selon le type de l'argument. Par exemple la fonction sinus sera SIN pour un argument réel, DSIN pour un argument double précision, CSIN pour un argument complexe.

Le tableau suivant donne les fonctions de la bibliothèque FORTRAN avec leurs types (fonctions arithmétiques) :

real	double precision	complex	Fonction
SIN	DSIN	CSIN	sin(x)
COS	DCOS	CCOS	cos(x)
TAN	DTAN		tg(x)
ASIN	DASIN		arcsin(x)
ACOS	DACOS		arccos(x)
ATAN	DATAN		arctg(x)
SINH	DSINH		sh(x)
LOG10	DLOG10		log10(x)
LOG	DLOG	CLOG	ln(x)
EXP	DEXP	CEXP	exp(x)
SQRT	DSQRT		

2.3 Fonctions de conversion

Ces fonctions permettent de convertir explicitement des données d'un type en un autre type. Le tableau suivant donne la liste des fonctions de conversion

Fonction	Entière	real	Double précision	Complexe
Max(x,y)	MAX0(x,y)	AMAX1(x,y)	DAMX1(x,y)	
Min(x,y)	MIN0(x,y)	AMIN1(x,y)	DMIN1(x,y)	
$\|x\|$ ou $\|z\|$	IABS(x)	ABS(x)	DABS(x)	CABS(x)
Partie entière	INT	AINT	DINT	
Reste dans la division entière	MOD	AMOD	DMOD	

2.4 Entrees / Sorties

Voyons les instructions qui nous permettent d'afficher un résultat sur l'écran ou de saisir un paramètre depuis notre clavier. A ce stade nous en resterons à ce genre d'entrée/sortie. Nous parlerons des fichiers plus loin. on se concentre sur les sorties ou l'écriture des résultats d'un programme car les données sont généralement introduites sous format libre sauf dans des cas particuliers. Le format d'écriture des résultats est applicable a la lecture des données.

Lire: Pour lire les informations saisies depuis un clavier, il existe deux méthodes:

1ère méthode :

Format spécifié directement dans l'instruction Write. La donnée saisie prend le type de la variable dans laquelle on la stocke. Cela s'écrit:

Write (n, ' format') liste à afficher

2ème méthode :

La lecture formatée, On indique au programme un certaine nombre d'informations en particulier le nombre de chiffres avant et après la virgule et bien d'autres choses.

Format spécifier à l'extérieur de Write ; cela s'écrit,

```
      Write (n, m) r1, r2,…….
      ……
m   Format ( ……..)
```

Dans la ligne d'étiquette m on spécifie les différents champs qui correspondent aux types des résultats à afficher.

Dans la syntaxe d'écriture **WRITE (n,m)** :

n pose la question 'ou ?', ou seront affichés les résultats du programme.

Si **n** est remplacé par le caractère '*', les résultats seront affichés directement sur Ecran.

Si **n** est un nombre entier, Les résultats seront stockés sur le fichier dont le numéro n, déjà ouvert par l'instruction OPEN.

m pose la question 'comment ?', c'est à dire spécifier le format d'écriture des résultats.

Si **m** est remplacé par le caractère '*', le format d'écriture est libre et les résultats seront transférés sur l'unité d'écriture **n** octet par octet.

Si **m** est un nombre entier les résultats seront affichés avec le format spécifier à la ligne d'étiquette **m**.

Chapitre 3 **Les Instructions de Controle**

Ce sont des instructions qui permettent de faire des itérations ou des tests sur des variables.

- les conditions
- les boucles
- les tableaux
- les fonctions et subroutines
- les fichiers

3.1 INSTRUCTIONS CONDITIONNELLES IF

L'intérêt d'un programme provient essentiellement de la possibilité d'effectuer; des choix (comportement différent suivant les circonstances) et des boucles (répétition d'un ensemble donne d'instructions).

En Fortran, la condition est imposée par l'instruction IF :

Pour aborder les conditions, il faut d'abord parler d'expression logique.

3.1.1 L'expression Logique:

Il existe en FORTRAN un type de variable LOGICAL, Une variable de type LOGICAL peut prend deux valeurs: .TRUE. ou .FALSE.

FORTRAN définit les expressions logiques suivantes, dans lesquelles x et y sont deux variables :

- x .EQ. y signifie x = y
- x .NE. y signifie x ≠ y
- x .GT. y signifie x > y
- x .LT. y signifie x < y
- x .GE. y signifie x >= y
- x .LE. y signifie x <= y

Il définit aussi les opérateurs logiques unaires:

- .OR. OU
- .AND. ET
- .XOR. OU exclusif
- .NOT. NON

3.1.2 Les différents types de la condition IF :

- IF Logique :

Teste la valeur d'une expression logique, si elle est vraie l'instruction qui suit sera exécutée.

```
IF (Expression logique) instruction
……
```

Exemple :

Calcul de la racine d'un nombre réel Q

```
Program Racine
read(*,*)Q
IF(Q.LT.0)Q = - Q
Write(*,*) 'la racine de Q=', Sqrt(Q)
End
```

- Si la valeur de Q est négative, le programme change son signe.
- On peut aussi résoudre ce problème en utilisant l'instruction GOTO :

```
    Program Racine
20  read(*,*)Q
    IF(Q.LT.0) Goto 20
    Write(*,*)'la racine de Q=', Sqrt(Q)
    End
```

Dans ce cas la racine n'est calculer que si Q est positif

- Le Bloc IF :

Syntaxes :

1) Pour exécuter une série d'instructions uniquement si une condition logique est vraie:

```
IF (condition logique) THEN
.....
ENDIF
```

2) Pour spécifier une autre série d'instructions à exécuter si la condition logique est fausse :

```
IF (condition logique) THEN
…….....
```

ELSE

……....

ENDIF

3) Pour enchaîner plusieurs conditions logiques :

IF (condition logique 1) THEN

…….....

ELSE IF (condition logique 2) THEN

……

ELSE IF (condition logique 3) THEN

…….....

ELSE

……

ENDIF

Si au lieu des ELSE IF on utilise ELSE suivit par le bloc IF, on sera obligé de fermer chaque bloc par ENDIF

- IF Arithmétique :

Teste le signe d'une expression arithmétique.

Syntaxe :

IF(Expression arithmétique) l , m , n

……..…

l ensemble d'instructions et calculs à exécuter lorsque la valeur de l'expression arithmétique est négative

m ensemble d'instructions et calculs à exécuter lorsque la valeur de l'expression arithmétique est nulle

n ensemble d'instructions et calculs à exécuter lorsque la valeur de l'expression arithmétique est positive

Exemple :

Résolution d'une équation 2ème degré $ax^2 + bx + c = 0$

Les constants caractères :

PROGRAM EQUATION2

IMPLICIT NONE

REAL a,b,c,delta,x1,x2,x

```
COMPLEX z1,z2
READ(*,*)a,b,c
IF(a.NE.0)THEN
        delta=b**2 - 4*a*c
        IF(delta)10,20,30
30      WRITE(2,*) ‘ Deux racines réelles ‘
        x1=- b - SQRT(delta)/2*a
        x2=- b + SQRT(delta)/2*a
        WRITE(2,150)x1,x2
        GOTO 40
20      WRITE(2,*) ‘ Racine double ‘
        x=-b/2*a
        WRITE(2,200)x
        GOTO 40
10      WRITE(2,*) ‘ Deux racines complexes ‘
        z1=- b - CSQRT(CMPLX(delta))/2*a
        z2= CONJG(z1)
        WRITE(2,300)z1,z2
        GOTO 40
        ELSE
                IF(b.ne.0)THEN
                x=-c/b
                WRITE(2,400)x
                ELSE
WRITE(2,*)'pas de solution'
                END IF
        END IF
40      CONTINUE
        150 FORMAT('x1=',1x,f7.3,/,'x2=',1x,f7.3)
```

```
200 FORMAT('x1=x2=',1x,f7.3)

300 FORMAT('z1=(',f7.3,'+',f7.3,'i)',/,'z2=(',f7.3,'+',f7.3,'i)')

400 FORMAT('solution equation 1er degrée ....x=',1x,f7.3)

END
```

3.2 LA BOUCLE DO

Lorsqu'un ensemble d'instructions et de calculs se répètent plusieurs fois, on utilise la boucle DO.

3.2.1 Syntaxe de la boucle DO :

- Première Syntaxe :

```
DO e i = Vint , Vfin , ps
      ……….
e  CONTINUE
```

Les instructions et les calculs situés entre l'instruction DO et l'instruction CONTINUE d'étiquette e, se répètent i fois (indice de la boucle) à partir de Vint (valeur initiale de i), jusqu'à Vfin (valeur finale de i). Les calculs s'effectuent avec un Pas 'Ps '.

- Deuxième syntaxe :

```
DO i = Vint , Vfin , Ps
……….
END DO
```

Exemple

```
PROGRAM  Somme
somme = 0

DO i = 1, 100

somme = somme + i

ENDDO
```

Cette boucle calcule la somme de 100 premiers entiers.

3.2.2 Syntaxe de la boucle DO... WHILE :

```
DO WHILE (condition)

……..
<liste d'instructions>
```

```
      ENDDO
```

où condition est une expression logique.

Exemple

```
C     déclaration des variables

      INTEGER n

      DOUBLE PRECISION somme, epsilon

        LOGICAL fin

C     Initialisation des variables

      epsilon = 1.d-10
      somme = 0d0
      fin = .FALSE.

C     Boucle de calcul. La condition est .NOT. fin de telle sorte que la condition soit vraie à l'entrée

      DO WHILE (.NOT. fin)

      somme = somme + 1d0/n
      n = n +1

C     On sort de la boucle si fin devient TRUE

      fin = (1d0/n .LT. epsilon*somme)

      ENDDO
```

Cette boucle calcule la suite des inverses jusqu'à ce que le terme 1/n soit inférieur à epsilon fois la somme partielle.

3.3 Instructions Goto Et Continue

Les instructions de branchement conduisent à des programmes illisibles et difficiles à corriger. Certaines instructions du FORTRAN utilisent des branchements de manière implicite pour gérer des erreurs. Dans ce cas, et seulement dans celui-ci, le branchement est obligatoire.

Syntaxe

Goto N° de label

En arrivant sur une telle ligne, le programme est branché directement sur la ligne comportant le label mentionné. En général, pour faire beau (il faut le dire vite...), cette ligne contient seulement l'instruction qui ne fait rien continue.

Exemple

Pour vous donner des mauvaises idées :

```
      character rep

10    continue
       write(*,*) 'Repondez oui ou non'
        read(*,*) rep
       if (rep.ne.'o'.and.rep.ne.'n') then
      goto 10


      endif
      END
```

Chapitre 4 **Fonctions et Subroutines**

4.1 LES SUBROUTINES

Une subroutine est une séquence d'instructions appelable d'un point quelconque du programme. Elle peut être appelée depuis le programme principal ou depuis une autre subroutine ou fonction. Une subroutine est dénie par un nom

4.1.1 Ecriture d'une subroutine

```
 subroutine nomsub(pf1, pf2, pf3, ...)
type pf1
type pf2
type pf2
  ...
Déclaration des variables locales
   ...
 Instructions exécutables
 ...
return
end
```

4.1.2 Appel d'une subroutine

L'appel d'une subroutine se fait depuis un bloc fonctionnel quelconque (programme principal, subroutine, fonction) avec l'instruction call.

Exemple

Ecrire une subroutine qui calcule les coordonnées polaires associées à des coordonnées cartésiennes (x; y).

$r = \sqrt{x^2 + y^2}$

$\theta = arctan(|y/x|$si x>0

$\theta = -arctan(|y/x|$si x<0

$\theta = \frac{\pi}{2}$si x=0 et y>0

$\theta = -\frac{\pi}{2}$si x=0 et y<0

Repense

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C          Programme coordonnées polaires associées
c     à des coordonnées cartésiennes (x; y).                          CCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      double precision x, y, r, theta
      double precision pi
      double precision temp

      call polar(x, y, r, theta)



      End
        subroutine polar(x, y, r, theta)
        double precision x, y, r, theta
       double precision pi
      double precision temp
      Write(*,*) ' donne x='
      read(*,*) x
      Write(*,*) ' donne y='
      read(*,*) y
       pi=4*datan(1d0)
        r=dsqrt(x**2+y**2)
       temp=datan(y/x)
      if (x.gt.0d0) then
       theta=temp
      write(*,*) theta
       else if (x.lt.0d0) then
          theta=-temp
      write(*,*) theta
        else if (x.eq.0d0) then
        if (y.ge.0d0) then
         theta=pi/2
      write(*,*) theta
            else
         theta=-pi/2
      write(*,*)  theta
        endif
        endif
        return
        end
```

4.2 Les Fonctions

Une fonction est en tout point identique à une subroutine, mais son nom contient en plus une valeur. C'est-à-dire qu'au lieu de l'appeler par call, on l'utilise à droite du signe = dans une instruction d'adéquation.

4.2.1 Déclaration EXTERNAL

Les paramètres formels a, b et valint seront double précision, mais de quel
type est f ?
C'est le nom d'une fonction FORTRAN, et pour déclarer un paramètre formel
aussi bizarre, on écrira :
EXTERNAL F
Et aussi :
Real F

4.2.2 Ecriture d'une fonction

Programme

EXTERNAL nomfonc

End

typefunction nomfonc (pf1, pf2, ...)

type pdf1

type pdf2

déclarations des variables locale

instructionsexécutables

! devrait contenir quelque chose comme : nomfonc = ...

...

return

end

4.2.2.1 Exemple

On considère un débit turbulent (Re>3500) dans une conduite et l'on veut calculer le coefficient λ de perte de charge linéaire en utilisant la relation de Colebrook :

$$\lambda^{-1/2} = -2\log\left(\frac{\varepsilon}{3.71 * D} + \frac{2.51}{(R_e * \lambda^{1/2})}\right)$$

Avec :

λ est coefficient de perte de charge linéaire de Moody

ε est la rugosité relative du tuyau

D est le diamètre de la conduite

Re est le nombre de Reynolds

Ecrire un programme en fortran évaluant λ à partir de l'équation de Colebrook. Pour obtenir une estimation initial, on pourra se servir des formules empirique de :

Blasius : $\lambda = 0{,}316R_e^{-0{,}25}$ valide pour 3500<R_e et $\varepsilon = 0$

Leeds : $\lambda = 0{,}00714 + 0{,}6104R_e^{-0{,}35}$ pour R_e<4,7.10^5

Herman : $\lambda = 0{,}0054 + 0{,}395R_e^{-0{,}3}$ pour 1^{05}<Re<2*10^6

Nikuradse: $\lambda = 0{,}0032 + 0{,}221R_e^{-0{,}237}$ pour 10^6<Re<10^8

4.2.2.2 Application numérique

Re	10^4	10^5	10^6	10^7	10^6
ε/D	0,05	0,003	0,0003	0,05	0,00001

N.B : Les valeurs ainsi calculées seront compères avec celles du diagramme de MoodyAlgorithme de Wegstein

1- $x^{(0)}, \varepsilon, nmax$

2- $x^{(n)} = F(x^{(n-1)})$

3-$\Delta = \frac{F(x^{(n)}) - F(x^{(n-1)})}{x^{(n)} - x^{(n-1)}}$

4-$\alpha = \frac{1}{1-\Delta}$

5-$x^{(n+1)} = x^{(n)} + \alpha(F(x^{(n)} - x^{(n)})$

6-Arrêter si $\left|\frac{x^{(n)} - x^{(n-1)}}{x^{(n)}}\right| < \varepsilon$ ou $\left|x^{(n)} - x^{(n-1)}\right| < \varepsilon$

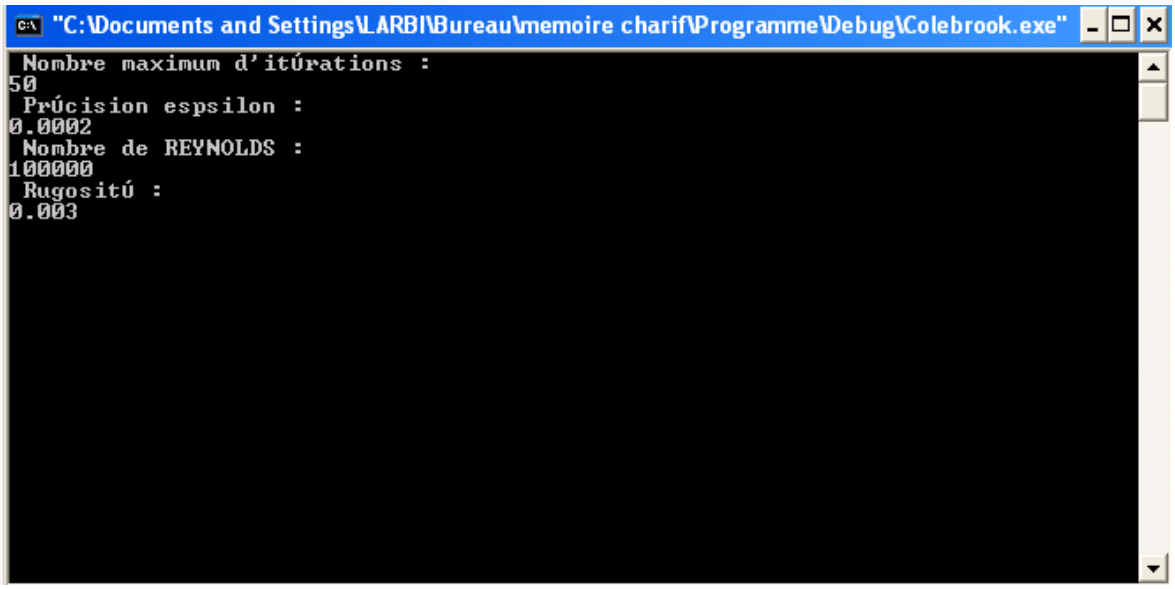
Avec n=1, 2, 3,…, nmax

```
ccccc ____________________________________________________________ *****
******    Calcul du coefficent de MOODY de perte de              *****
*******   Charge linèaire dans un conduite par                   *****
******* Résolution de l'équation de COLEBROOK                     *****
****  ____________________________________________________________ *****
      |
      |  EXTERNAL COLEBROOK
      |  DOUBLE PRECISION COLEBROOK
      |  DOUBLE PRECISION  reynlods, rugosite, espsilon , lambda,x
      |  COMMON   rugosite, reynlods
      |
*     |  Saisie des parmètres
      |  Write (*,*) 'Nombre maximum d''itérations :'
      |  Read (*,*)  NbMaxIterations
      |  Write (*,*) 'Précision espsilon :'
      |  Read (*,*)  espsilon
      |  Write (*,*) 'Nombre de REYNOLDS :'
      |  Read (*,*)  reynlods
      |  Write (*,*) 'Rugosité : '
      |  Read (*,*)   rugosite
      |
*     |   Estimé de d'HERMAN
      |  Lambda= 54.0E-4+0.395*reynlods**(-0.3)
*     |  Résolution de l'équation de COLEBROOK
      |   x=1.0/SQRT(Lambda)
      | Call WEGSTEIN(COLEBROOK, NbMaxIterations, espsilon, NbIterations)
***   |
****  |  Affichage de résultat
      |  write (1,500) Lambda
      |  write (1,600) NbIterations
500   |FORMAT ('Coefficient de perte de charge=',G12.5)
600   |FORMAT ('Nombre d''itérations=',G12.5)
      |
      |     END
      |
```

4.2.2.3 soubroutine

```
      SUBROUTINE WEGSTEIN(F,NbMaxIterations, espsilon, NbIterations)

       DOUBLE PRECISION espsilon,alpha, delta
       DOUBLE PRECISION x,x1,F
       INTEGER NbMaxIterations, NbIteration
       DO 10 NbIterations=1,NbMaxIterations
       x1=F(x)
       delta=(F(x1)-F(x))/(x1-x)
       alpha=1./(1.-delta)
       x=x1+alpha*(F(x1)-x1)
      IF(ABS((x1)-x).LE.espsilon) GO TO 30
10      CONTINUE
30      CONTINUE

       RETURN
      END

**      La fonction de COLEBROOK
      DOUBLE PRECISION FUNCTION COLEBROOK (x)

      DOUBLE PRECISION x, rugosite, reynlods
      COMMON rugosite, reynlods

      COLEBROOK=-2.*LOG10(rugosite/3.71+2.51*x/reynlods)
          RETURN
      END
```

4.2.2.4 Résultat




```
Coefficient de perte de charge= 0.17891E-01
Nombre d'itérations=           2
```

Chapitre 5 **Les Tableaux Et Les Fichiers**

5.1 Les Tableaux

5.1.1 Déclaration :

Pour déclarer un tableau en utilisant soit

- Dimension variable (chiffre)
  Ou
- Double precision variable (chiffre)

Pour créer des tableaux, il est conseille de déclarer les tailles dans des constantes symboliques

Exemple

parameter (max=chiffer)

double precision variable (max,max)

ou

parameter (max=chiffer)

Dimension variable (max,max)

Avant de déclarer un tableau, pensez à la taille mémoire qu'il va occuper. Le tableau a ci-dessus occupe par exemple 100*100* 8 = 80000 octets.

5.1.1.1 Exemple

La suite de Fibonacci est générée par l'équation récurrente linéaire d'ordre 2 :

U(i)=U(i-1)+U(i-2)

Aunant pour les conditions initiales :

U(0)=1, U(1)=1

Ecrire un programme imprimant les valeurs de i et U(i) pour i variant de 1 à 22

Repense

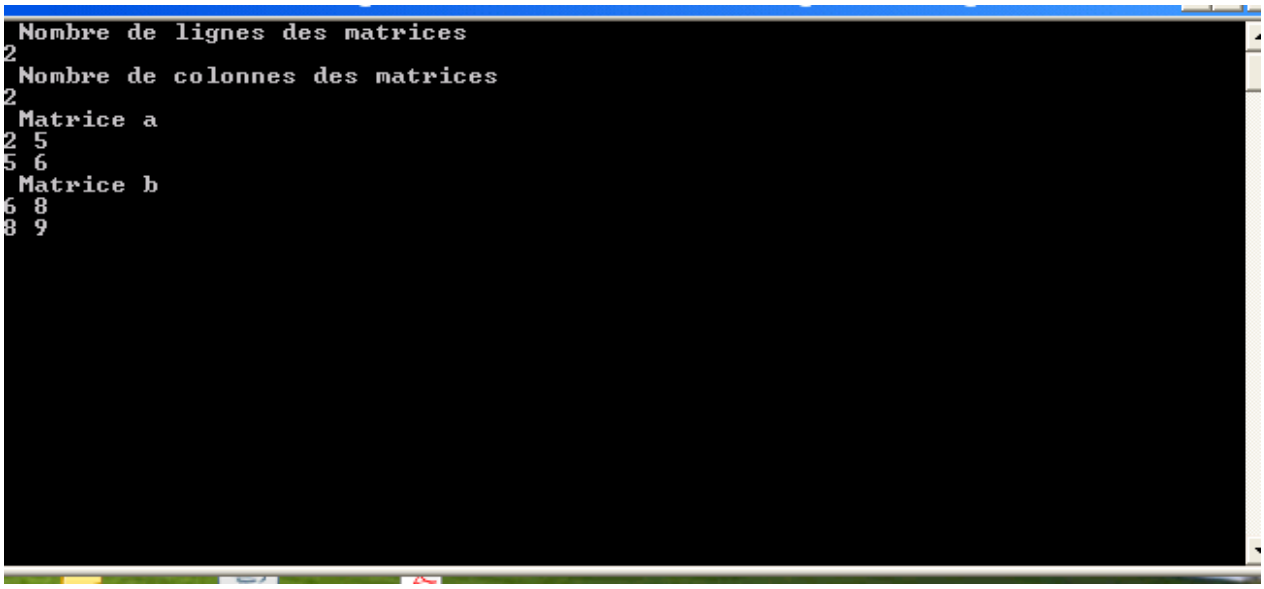
```
      Dimension u(100)

      Write(*,*) 'suit de Fiboonnci'
      U(1)=1
      u(2)=1
      do 10 i=3,22
      U(i)=u(i-1)+u(i-2)
      write (2,*) i-1,U(i-1)
10    Continue

      stop
      End
```

```
 2    1.000000
 3    2.000000
 4    3.000000
 5    5.000000
 6    8.000000
 7    13.00000
 8    21.00000
 9    34.00000
10    55.00000
11    89.00000
12    144.0000
13    233.0000
14    377.0000
15    610.0000
16    987.0000
17    1597.000
18    2584.000
19    4181.000
20    6765.000
21    10946.00
```

5.1.1.2 Exemple

Ecrire un programme complet qui lit au clavier deux matrices de même taille **m×n** et effectue leur somme. On part du principe que ces matrices ont au plus 50 lignes et 80 colonnes.

```
      parameter (mligne=50, mcolon=80)
     double precision a(mligne, mcolon)
     double precision b(mligne, mcolon)
     double precision c(mligne, mcolon)
      integer nligne, ncolon
      write(*,*)
&'Nombre de lignes des matrices'
     read(*,*) nligne
         write(*,*)
&'Nombre de colonnes des matrices'
 read(*,*) ncolon
write(*,*) 'Matrice a '
 do i=1,nligne
 read(*,*) (a(i,j), j=1,ncolon)
 enddo
write(*,*) 'Matrice b '
do i=1,nligne
 read(*,*) (b(i,j), j=1,ncolon)
 enddo
 do i=1,nligne
do j=1,ncolon
c(i,j)=a(i,j)+b(i,j)
enddo
enddo
do i=1,nligne
write (1,*) (c(i,j),j=1,ncolon)
 Enddo
  end
```



5.1.1.3 Résultats

```
 8.00000000000000        13.0000000000000
 13.0000000000000        15.0000000000000
```

5.2 Les Fichiers

Les instructions nécessaires à créer, lire et écrire les fichiers qui conviennent pour stocker nos résultats de calcul ou nos données d'entrée.

On ne considérera donc que les seuls fichiers séquentiels formattés (autrement appelés fichier texte). Ces fichiers ont la propriété intéressante de pouvoir être lus et modifiés avec n'importe quel éditeur de texte.

5.2.1 Créer et ouvrir un fichier séquentiel formaté

La syntaxe de l'instruction pour ouvrir un fichier est:

OPEN (unité,

FILE= chaine,

FORM = chaine,

STATUS = chaine

ERR = numlabel)

où

- unité : numéro d'identification de l'unité du fichier. C'est un nombre entier compris entre 10 et 99, choisi arbitrairement par le programmeur. Chaque fois qu'une instruction voudra adresser le fichier ouvert, elle le désignera par ce numéro d'unité.
- FILE : chaîne de caractères contenant le nom et le chemin du fichier, par exemple 'd:\NumLab\Labmecanique\data.txt'
- FORM: chaîne de caractères: 'formatted' pour les fichiers formatés (valeur par défaut) ou 'unformatted' pour les fichiers binaires.
- STATUS : chaîne de caractères :
    - 'new': le fichier n'existe pas, il est créé.S'il existe déjà le programme retourne une erreur
    - 'old'le fichier existe déjà. S'il n'existe pas, le programme retourne une erreur
    - 'unknow': ouvre le fichier, s'il n'existe pas il le créé, s'il existe il écrit à la fin de celui-ci. C'est le mode par défaut
- ERR : numéro de label vers lequel le programme continuera son exécution s'il rencontre une erreur

5.2.1.1 Exemple:

OPEN(10, FILE = 'd:\NumLab\Labmecanique\data.txt') ouvre le fichier data.txt, qu'il existe ou non (par défaut). En cas d'erreur, le programme se plante...

5.2.2 Lire un fichier séquentiel formaté

La syntaxe de l'instruction de lecture est:
READ(unité,
format,
ERR = numlabel,
END = numlabel) liste de données

où

- unité : numéro d'identification de l'unité du fichier alloué à son ouverture
- format : dans notre cas (fichier formaté), on indiquera *
- ERR : numéro de label vers lequel le programme continuera son exécution s'il rencontre une erreur
- END : numéro de label vers lequel le programme continuera son exécution lorsqu'il n'aura plus rien à lire dans le fichier (optionnel)

Exemple :

READ (10,*,ERR=90, END =100) toto lit une donnée dans le fichier data.txt et la stocke dans la variable toto.

5.2.3 Ecrire une fichier séquentiel formaté

La syntaxe de l'instruction de lecture est:

WRITE(unité,
format,
END = numlabel) liste de données

Exemple :

WRITE (10,*) zz écrit la variable zz dans le fichier data.txt.

5.2.4 Fermer un fichier

On utilise l'instruction CLOSE(unité).

En principe, tous les fichiers sont fermés automatiquement à la fin d'un programme. Mais c'est une bonne habitude à prendre que de les fermer par un CLOSE

5.2.5 Exemple d'utilisation

Supposons que nous ayons stocké les résultats d'un calcul dans deux vecteurs x et y de dimension n. Nous voulons sauvegarder ces valeurs dans un fichier pour les plotter avec un logiciel comme Matlab, Scilab ou GnuPlot.

Voilà le bout de code correspondant.

On supposera que le fichier data.txt n'existe pas.

```
INTEGER n, i

PARAMETER (n = 100)

DOUBLE PRECISION x(n), y(n)

CHARACTER*80 nomfic

nomfic = 'data.txt'

OPEN(10, FILE=nomfic)

DO i=1,n

WRITE(10,*) REAL(x(i)), REAL(y(i))

ENDDO

CLOSE(10)

END
```

Chapitre 6 **Programme de Calcul**

6.1 La Methode D'euler

6.1.1 Résolution d'une équation différentielle par la méthode d'Euler

Nous voilà maintenant en possession de tous les éléments pour réaliser un programme de calcul en FORTRAN.

Soit à résoudre une EDO équation différentielle ordinaire avec une condition initiale de type:
y' = f(x, y)
y(x0) = y0

La méthode d'Euler consiste à estimer l'accroissement de y entre les abscisses x_i et x_{i+1} par un développement de Taylor d'ordre 1 (on dit que c'est une méthode à un pas):

$y_{i+1} = y_i + h*f(x_i, y_i)$ où $h = x_{i+1} - x_i$

Cette méthode n'est pas très précise. De plus, elle génère et amplifie les erreurs. Mais dans les cas simples, si f n'est pas trop variable et h très petit, elle sera suffisante en première approximation.

Pour traiter des EDO d'ordre supérieur, on procéde à un changement de variables qui aboutit à un système d'équations différentielles. Par exemple, si l'on considère l'EDO:

y'' = f(x,y,y')

avec y(x0) = y0 et y'(x0) = y'0

On peut la décomposer en un système de 2 équations du premier ordre en posant y' = u(x), d'où:

y' = u(x)

u'(x) = f(x,y,u)

et résoudre ce système par la méthode d'Euler.

6.1.1.1 Exemple

Nous allons illustrer l'utilisation de la méthode d'Euler par l'étude de la dynamique du pendule simple. Comme vous le savez sans doute, l'équation différentielle de son mouvement est :

$d^2(theta)/dt^2 = -(g/l)sin(theta)$

où l = longueur du fil de suspension.

Pour appliquer la méthode d'Euler, je vais décomposer cette EDO en un système à 2 équations en posant:

d(theta)/dt = omega la vitesse angulaire

a(theta) = d(omega)/dt l'accelération angulaire, qui vaut -(g/l)*sin(theta)

L'application du schéma d'Euler me donne les équations:

$theta_{i+1} = theta_i + h*omega_i$ (1)

$omega_{i+1} = omega_i + h*a_{i=} omega_i + h*(-(g/l)sin(theta_i))$ (2)

Nous retrouverons tout ce petit monde dans le programme ci-dessus:

- le tableau X1 contient les valeurs d'angle calculées
- le tableau X2 contient les valeurs de vitesse angulaire calculées
- le tableau t contient les pas temporels.
- les conditions initiales sont : angle initial 60° (attention à l'exprimer en rd) et vitesse initiale nulle.

6.1.1.2 Le programme

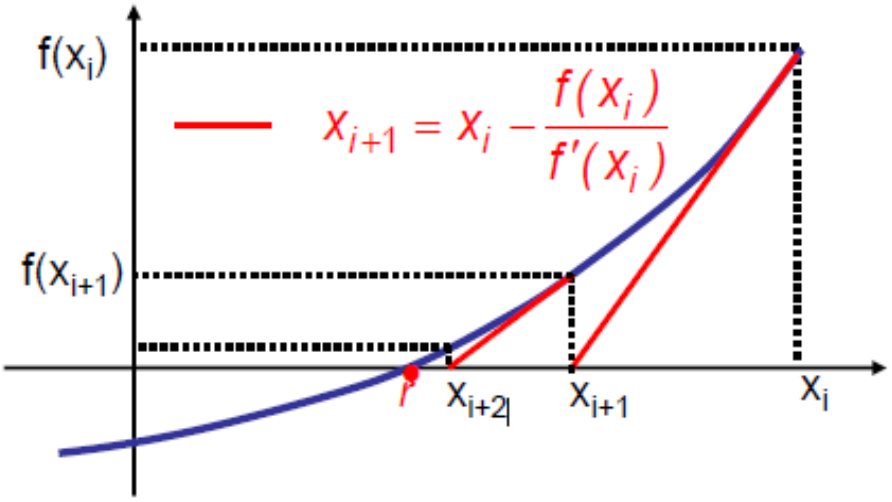
```
C Programme de demonstration de l'usage de la methode d'EULER
PROGRAM PenduleEuler
IMPLICIT NONE
C Declaration des constantes
REAL g, PI, n, P
PARAMETER (g = 9.81, PI = 3.141592654)
PARAMETER (n = 2) ! Nb de cycles de calcul
PARAMETER (P = 1000) ! Nb de pas de calcul par cycle
C Declaration des variables
INTEGER i
REAL l, Theta, h
REAL X1(n*P), X2(n*P), t(n*P) ! resp. l'angle, la vitesse angulaire, le temps
C declaration des zone commune
COMMON /Systeme/l
C Initialisation des variables
l = g/(4*PI*PI) ! je choisis une longueur arbitraire qui m'arrange
Theta = 2*PI*SQRT(l/g) ! la periode vaut 1 pour l'approximation des petits angles
h = Theta/P ! ! pas temporel pour le calcul
```

```
C Declaration des conditions initiales d'integration
X1(1) = 60*PI/180 ! angle initial en radian
X2(1) = 0 ! vitesse angulaire initiale
t(1) = 0.0 ! Origine des temps
C Calcul du mouvement selon l'algorithme d'Euler
DO i=1, n*P-1
t(i+1) = t(i)+h ! incrementation du temps
C Le systeme differentiel descriptif du pendule
CALL Derivee(X1(i), X2(i), f1, f2)
C L'algorithme d'EULER
CALL Euler(X1,X2,f1,f2,i,h)
ENDDO
C Sauvegarde des donnees dans le fichier Euler.dat
C On sauvegarde les données pour tracer la courbe theta = f(t) mais on pourrait
C tracer d'autre courbes, en particulier omega = f(theta). Il suffit de bien
C choisir les données à stocker
OPEN(10, FILE='Euler.dat')
DO i=1, n*P
WRITE(10,*) REAL(t(i)), REAL(X1(i)*180./PI)
ENDDO
CLOSE(10)
END
C Subroutine de calcul du systeme differentiel
SUBROUTINE Derivee(x,y,fx,fy)
REAL x,y,fx,fy
COMMON /Systeme/l
REAL g, l
PARAMETER (g = 9.81)
fx = y
```

```
fy = -(g/l)*SIN(x)

END

C Subroutine Algorithme d'Euler

SUBROUTINE Euler(x,y,f1,f2,i,dt)

REAL x(*), y(*), f1, f2, dt

INTEGER i

x(i+1) = x(i)+f1*dt ! on retrouve ici les formules du schéma d'Euler

y(i+1) = y(i)+f2*dt

END
```

6.2 Methode De Newton-Raphson

Cette méthode suppose que la fonction f est dérivable dans l'entourage de la fonction et que la dérivée, peu importe où dans l'entourage, n'est pas zéro.
En supposant que x0 est un point suffisamment près de la racine de la fonction, le graphe de la fonction y = f (x) est approximé par la tangente de la courbe au point.



6.2.1 Exemple

1)-f(x) = exp(x)+cos(x)

3)- f(x)=x^3-3*x^2

2)- f(x)=cos(x)-x^3

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C          Programme de la Méthode de Newton Raphson        CCCCCCC
c                                                           CCCCCC
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      EXTERNAL F,F1

      CALL NewtonRaphson(i,x,F,F1)
      End
*-------------------------------------------------------
      function F(x)
       F=x**3-3*x**2+2*x-1
       return
       end
*-------------------------------------------------------
      function F1(x)
       F1=3*x**2-6*x+2
       return
       end
*-------------------------------------------------------
      SUBROUTINE NewtonRaphson(i,x,F,F1)


      write(*,*)'enter initial guess'
          read(*,*) x0
         write(*,*)'enter tolerrence'
        read(*,*) eps
        i=1
 20      x1=x0-f(x0)/f1(x0)
        if(abs((x1-x0)/x0).lt.eps) then
         write(*,*) x1,i

        STOP

       else

       x0=x1
      i=i+1
        goto 20
        endif
           end
*-------------------------------------------------------
```

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
C
C          Programme de la Méthode de Newton Raphson        CCCCCCC
c                                                           CCCCCC
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      EXTERNAL F,F1, F2,F3, F4,F5
Cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      Write(*,10)                                                      !ccc
10     FORMAT (/////,10x,'Donner votre chiox,'//                       !ccc
     $10x,'1-Solution de la fonction f(x) = exp(x)+cos(x) Appuyer sur 1 !ccc
     $',//,10x,'2-Solution de la fonction f(x)=x^3-3*x^2  Appuyer sur 2
     $',//,10x,'2-Solution de la fonction f(x)=cos(x)-x^3 Appuyer sur 3 !cc
     $ ',//////)                                                       !ccc
                                                                       !ccc
Cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
          READ (*,*) I1
      IF (I1.EQ.1) GOTO 4
      IF (I1.EQ.2) GOTO 3
      IF (I1.EQ.3) GOTO 5
3     CALL NevtonRaphson(i,x,F,F1)
4     CALL NevtonRaphson1(i,x,F2,F3)
5     Call NevtonRaphson2(i,x,F4,F5)

      End
*-------------------------------------------------------
      function F(x)
       F=x**3-3*x**2+2*x-1
       return
       end

*-------------------------------------------------------
      function F1(x)
       F1=3*x**2-6*x+2
       return
       end
*-------------------------------------------------------
      function F2(x)
       F2=exp(x)-2*cos(x)
       return
       end

*-------------------------------------------------------
```

```
*-------------------------------------------------------
      function F3(x)
       F3=exp(x)+2*sin(x)
       return
       end
*-------------------------------------------------------
*-------------------------------------------------------
      function F4(x)
       F4=cos(x)-x**3
       return
       end

*-------------------------------------------------------
      function F5(x)
       F5=-sin(x)-3*x**2
       return
       end
*-------------------------------------------------------
```

```
*-------------------------------------------------------------
      SUBROUTINE NewtonRaphson(i,x,F,F1)


      write(*,*)'enter initial guess'
          read(*,*) x0
         write(*,*)'enter tolerrence'
        read(*,*) eps
        i=1
 20      x1=x0-f(x0)/f1(x0)
        if(abs((x1-x0)/x0).lt.eps) then
         write(*,*) x1,i

        STOP

       else

       x0=x1
      i=i+1
        goto 20
        endif
           end
*-------------------------------------------------------------
```

```
*-------------------------------------------------------------
      SUBROUTINE NewtonRaphson1(i,x,F2,F3)


      write(*,*)'enter initial guess'
          read(*,*) x0
         write(*,*)'enter tolerrence'
        read(*,*) eps
          i=1
 20      x1=x0-f2(x0)/f3(x0)
        if(abs((x1-x0)/x0).lt.eps) then
         write(*,*) x1, i

        STOP

       else
       x0=x1
      i=i+1
        goto 20
        endif
           end
*-------------------------------------------------------------
```

```
*-------------------------------------------------------------
          SUBROUTINE NewtonRaphson2(i,x,F4,F5)


      write(*,*)'enter initial guess'
          read(*,*) x0
         write(*,*)'enter tolerrence'
        read(*,*) eps
          i=1
 20      x1=x0-f4(x0)/f5(x0)
        if(abs((x1-x0)/x0).lt.eps) then
         write(*,*) x1, i



        STOP

       else
       x0=x1
      i=i+1
        goto 20
        endif
           end
*-------------------------------------------------------------
```

6.3 *Methode Des Moindres Carres*

6.3.1 *Exemple*

```
      PROGRAM LINE_1
      Dimension X(100), Y(100)
         Write (*,*) 'doone N='
          read(*,*) N
      OPEN( 10, FILE = 'pts.dat')
       do i=1,N
           READ(10,*) x(i), y(i)
        End do
*****************************************************************************

              SUMX = 0.0
              SUMY = 0.0
               SUMXY = 0.0
             SUMXX = 0.0
               SUMYY=0.0
*******************************************************************************


      DO I = 1, N

             SUMX = SUMX + X(I)
             SUMY = SUMY + Y(I)
             SUMXY = SUMXY + X(I) * Y(I)
             SUMXX = SUMXX + X(I) **2
             SUMYY=SUMYY+Y(I)**2

      END DO
*************************************************************************************************
          XBAR = SUMX/N
          YBAR = SUMY/N
          sigmaxy=(SUMXY/N-XBAR*YBAR)
          sigmax=sqrt(SUMXX/N-XBAR**2)
          sigmay=sqrt(SUMYY/N-YBAR**2)
          r=sigmaxy/(sigmax*sigmay)
*************************************************************************************************
      a = ( SUMXY/N - XBAR* YBAR )/( SUMXX / N - XBAR ** 2 )
          b = YBAR - a*XBAR
*************************************************************************************************

          PRINT *, 'viscosité plastique  = ', a
          PRINT *, 'seuil d''écoulement  = ', b

          PRINT *, 'Coefficient de corrélation= ', r

          End
```

6.4 La Methode De Runge-Kutta D'ordre 4

Les méthodes de Runge-Kutta (ou RK), l'ordre 2 ou 4, sont très couramment utilisées pour la résolution d'équations différentielles ordinaires (EDO). Ce sont des méthodes à pas unique, directement dérivées de la méthode d'Euler. Elles ont l'avantage d'être simples à programmer et d'être assez stables pour les fonctions courantes de la physique.

6.4.1 description de la méthode RK4.

Donc, l'algorithme RK4:

On part de la formule d'Euler, qui donne :

$y_{n+1} = y_n + h*f(x_n, y_n)$, et $x_{n+1} = x_n + h$

La méthode RK du deuxième ordre produit deux coefficients k_1 et k_2, qui permettent d'écrire:

$k_1 = h*f(x_n, y_n)$

$k_2 = h*f(x_n + h/2, y_n + k_1/2)$

$y_{n+1} = y_n + k_2 + O(h^3)$

Cette méthode exige donc deux évaluations de f. L'erreur de consistance est en $O(h^3)$ et l'erreur globale de convergence est d'ordre $O(h^2)$

Pour obtenir plus de précision, mais en doublant le temps de calcul puisqu'on procéde à 4 évaluations de f, voici la méthode RK4:

$k_1 = h*f(x_n, y_n)$

$k_2 = h*f(x_n+ h/2, y_n + k_1/2)$

$k_3 = h*f(x_n+ h/2, y_n + k_2/2)$

$k_4 = h*f(x_n+ h, y_n + k_3)$

$y_{n+1} = y_n + k_1/6 + k_2/3 + k_3/3 + k_4/6 + O(h^5)$

La méthode RK4 exige donc 4 évaluations de f, ce qui peut être gênant si f est compliquée. L'erreur de consistance est en $O(h^5)$ et l'erreur globale de convergence est d'ordre $O(h^4)$.

6.4.2 Implémentation de l'algorithme RK4

```
SUBROUTINE ResolutionRK4(x,y,dx,NbEquation,Derivee)
C x : abscisse
C y : l'appel ce tableau contient les valeurs y initiales. Au retour, il contient les
C valeurs calculees
C dx : pas de calcul
C Nb Equation : nombre d'equations dans le système dif.
C Derivee : nom de la fonction decrivant le systemedif.
IMPLICIT NONE
```

```
EXTERNAL Derivee ! fonction de calcul de la derivee
C Typage de l'interface de ResolutionRK4
INTEGER NbEquation
REAL x, y(NbEquation), dx
C Declaration des variables locales
REAL pred1(NbEquation), pred2(NbEquation), pred3(NbEquation),1 pred4(NbEquation), ytemp (NbEq-
uation), halfdx
INTEGER i
halfdx = dx/2
C Premiereprediction
CALL Derivee(x,y,pred1,NbEquation)
DO i=1, NbEquation
ytemp(i) = y(i) + pred1(i)*halfdx
ENDDO
C Seconde prediction
CALL Derivee (x+halfdx,ytemp,pred2,NbEquation)
DO i=1, NbEquation
ytemp(i) = y(i) + pred2(i)*halfdx
ENDDO
C Troisieme prediction
CALL Derivee(x+halfdx,ytemp,pred3,NbEquation)
DO i=1, NbEquation
ytemp(i) = y(i) + pred3(i)*dx
ENDDO
C  Quatrieme prediction
CALL Derivee(x+dx,ytemp,pred4,NbEquation)
C Estimation de y pour le pas suivant
DO i=1, NbEquation
y(i)=y(i)+(dx/6.0)*(pred1(i)+2.0*pred2(i)+2.0*pred3(i) + pred4(i))
ENDDO
END
```

6.4.3 Exemple d'application du RK4

Le programme PenduleRK4 maintenant:

```
PROGRAM PenduleRK4
IMPLICIT None
C Declaration des routines externes
C ResolutionRK4 : routine d'implementation du schema RK4
C Derivee : description du systemedifferentiel de l'oscillateur
EXTERNAL ResolutionRK4
EXTERNAL Derivee
C Declaration des parametres RK4
C Le systemedifferentiel comporte deux equations
INTEGER NbEquation
PARAMETER (NbEquation = 2)
C Declaration des constantes
REAL g, PI
DATA g/9.81/, PI/3.141592654/
C Declaration des variables
INTEGER i, NbPas
REAL y(NbEquation), t, l, PasTemps, a0
C Declaration des variables communes
COMMON /Pendule/l
C Saisie des parametres du mouvement
WRITE(*,'(a,$)') 'Longueur du pendule (en m) : '
READ(*,*) l
WRITE(*,'(a,$)') 'angle initial du pendule (en degres) : '
READ(*,*) a0
WRITE(*,'(a,$)') 'Pas temporel de calcul (en s) : '
READ(*,*) PasTemps
WRITE(*,'(a,$)') 'Nombre de pas : '
READ(*,*) NbPas
C Declaration des conditions initiales pour l'integration
y(1) = a0*PI/180.
! angle initial du pendule
y(2) = 0.0
```

```
! vitesse angulaire initiale
C Ouverture du fichier de stockage des resultats de calcul
C Ce fichier sera trace par GnuPlot
OPEN(10, FILE='PenduleRK4.dat')
C Ecriture des conditions initiales
t = 0.0
WRITE(10,* ) t, y(1)
C Boucle de calcul - le coeur du programme
DO i=1, NbPas
t = i*PasTemps
CALL ResolutionRK4(t, y, PasTemps, NbEquation, Derivee)
WRITE(10,*) t, y(1)*180./PI
! notez la conversion en degrés
ENDDO
C Fermeture du fichier de donnees
CLOSE(10)
STOP
END
```

6.4.4 Subroutine de resolution d'une equation differentielle par la methode de Rung-Kutta d'ordre 4

```
C               Subroutine de resolution d'une equation differentielle
C               par la methode de Rung-Kutta d'ordre 4
        SUBROUTINE ResolutionRK4(x,y,dx,NbEquation,Derivee)
C               x           : abscisse
C               y()         : valeurs retournees par le systeme dif.
C               dx          : pas de calcul
C               NbEquation : nombre d'equations dans le systeme dif.
C               Derivee     : nom de la fonction decrivant le systeme dif.

    IMPLICIT NONE
        EXTERNAL Derivee            ! fonction de calcul de la derivee
C       Typage de l'interface de ResolutionRK4
        INTEGER NbEquation
        REAL x, y(NbEquation), dx
```

```
C       Declaration des variables locales
        REAL pred1(NbEquation),pred2(NbEquation),pred3(NbEquation),
       1                    pred4(NbEquation), ytemp(NbEquation), halfdx
        INTEGER i
        halfdx = dx/2
C               Premiere prediction
        CALL Derivee(x,y,pred1,NbEquation)
        DO i=1, NbEquation
                ytemp(i) = y(i) + pred1(i)*halfdx
        ENDDO
C               Seconde prediction
        CALL Derivee(x+halfdx,ytemp,pred2,NbEquation)
        DO i=1, NbEquation
                ytemp(i) = y(i) + pred2(i)*halfdx
        ENDDO
C               Troisieme prediction
        CALL Derivee(x+halfdx,ytemp,pred3,NbEquation)
        DO i=1, NbEquation
                ytemp(i) = y(i) + pred3(i)*dx
        ENDDO
C               Quatrieme prediction
    CALL Derivee(x+dx,ytemp,pred4,NbEquation)
C               Estimation de y pour le pas suivant
        DO i=1, NbEquation
                y(i)=y(i)+(dx/6.0)*(pred1(i)+2.0*pred2(i)+2.0*pred3(i) + pred4(i))
        ENDDO
        END
```

Références

[1]. IDRIS. Cours IDRIS Fortran, 2004. http ://www.idris.fr/data/cours/lang/f90/F90 cours.html.

[2]. Luc Mieussens. Cours unix-shell-makefile. Institut de Math´ematiques – Universite de Bordeaux, 2004.

[3]. Numerical Recipes in Fortran: The Art of Scientific Computing de William H. Press, et al

[4]. Programmer en Fortran 90. Guide complet de Claude Delannoy, 1996.

[5]. Luc Mieussens. Petit guide pour Emacs. Institut de Mathematiques - Universite de Bordeaux, 2004.

[6]. Adams, Brainerd, Hendrickson, Maine, Martin, Smith, The Fortran 2003 Handbook, Springer, 2009.

[7]. Adams, Brainerd, Martin, Smith et Wagener, Fortran 95 Handbook, MIT Press,1997.

[8]. Brainerd, Goldberg, Adams, Programmer's guide to Fortran 90, 3e édit. Unicomp,1996,

[9]. Delannoy Claude, Programmer en Fortran 90 – Guide complet, Eyrolles, 1997

[10]. Dubesset M., Vignes J., Les spécificités du Fortran 90, Éditions Technip, 1993

[11]. Ellis, Phillips, Lahey, Fortran 90 Programming, Addisson-Wesley, 1994.

[12]. Hahn B.D., Fortran 90 for the Scientist & Engineers, Edward Arnold, London,1994.

[13]. Kerrigan James F., Migrating to Fortran 90, O'Reilly & Associates Inc.,1994.

[14]. Lignelet P., Fortran 90 : approche par la pratique, Éditions Studio Image (sérieinformatique), 1993.

[15]. Lignelet P., Manuel complet du langage Fortran 90 et Fortran 95, calcul intensifet génie logiciel, Col. Mesures physiques,Masson, 1996.

[16]. Lignelet P., Structures de données et leurs algorithmes avec Fortran 90 et Fortran95, Masson, 1996.

[17]. Morgan and Schoenfelder, Programming in Fortran 90, Alfred Waller Ltd., 1993.

[18]. Metcalf M., Reid J.,_ Fortran 95/2003 explained, Oxford University Press, 2004.

[19]. Olagnon Michel, Traitement de données numériques avec Fortran 90,Masson, 1996.