Méthodes numériques et éléments de programmation

Guy Munhoven

Institut d'Astrophysique et de Géophysique (Bât. B5c)
Bureau 0/13
eMail: Guy.Munhoven@ulg.ac.be
Tél.: 04-3669771

22 septembre 2014

Méthodes numériques et éléments de programmation
Guy Munhoven
Eléments de base

Plan du cours 2014-2015

Cours théoriques

16-09-2014 *Méthodes numériques pour équations différentielles ordinaires: introduction*

22-09-2014 Méthodes de Runge-Kutta; Contrôle du pas; Equations raides

22-09-2014 Fortran 95 de base

- Eléments de base
- Structure d'un programme
- Boucles

4–5 cours Fortran 95: suite et compléments

Programme Fortran minimaliste : *Hello World*

Minimalistic Fortran Program : Hello World

Le programme *Hello World* en Fortran 90 :

Exemple

```
PROGRAM HelloWorld
WRITE(*,*) 'Hello World'
END PROGRAM HelloWorld
```

Méthodes numériques et éléments de programmation
Guy Munhoven
Eléments de base

Structure générale d'un programme Fortran

General Structure of a Fortran Program

```
PROGRAM [nom_du_programme]
instructions de spécification et de déclaration
instructions exécutables
[CONTAINS
procédures internes]
END [PROGRAM [nom_du_programme]]
```

Options indiquées entre [...], ces derniers étant à omettre

Instructions de spécification et de déclaration
Specifications and declarations

- Préciser des extensions à utiliser
  - ⇒ p. ex., `USE` *`module`*
- Adopter conventions particulières
  - ⇒ `IMPLICIT NONE` (pas de typage par défaut)
- Définir les *types* de données à traiter
  - ⇒ à valeurs entières, réelles, chaînes de caractères, ...
- Conférer des *attributs* (propriétés particulières) à certaines données
  - ⇒ `PARAMETER` (à valeur non-modifiable)
  - ⇒ `DIMENSION(...)` (tableaux à plusieurs dimensions)
- Définir interfaces avec d'autres parties du programme

Déclarations
Declarations

- Forme générique d'une déclaration

  type, *liste d'attributs* `::` *identificateur* = *valeur_initiale*
- `::` obligatoire si attributs ou valeurs initiales données

Exemples : déclarations de variables

```
INTEGER i, j, k
REAL :: x = 1.0, x2 = 1E-6
DOUBLE PRECISION :: y = -1.0d0, tol = 1D-12
LOGICAL :: l, option1 = .TRUE., option2 = .FALSE.
COMPLEX :: z
CHARACTER(LEN=80) :: nom_fichier, version_programme
```

Déclarations
Declarations

Exemples d'attributs : `PARAMETER`, `DIMENSION`, ...

Exemples : déclarations de variables avec attributs

```
DOUBLE PRECISION, PARAMETER :: pi = 3.14159265358979323846D+00
DOUBLE PRECISION, PARAMETER :: un_demi = 1D0/2D0
INTEGER, PARAMETER :: nj_semaine = 7
CHARACTER(LEN=*), PARAMETER :: version = 'Version 1.6 du 22SEP2014'
INTEGER, PARAMETER :: n_lons = 64, n_lats = 32, n_vert = 10
DOUBLE PRECISION, DIMENSION(12) :: temperature_moyenne_mensuelle
REAL, DIMENSION(n_lons, n_lats, n_vert) :: vitesse_vent, p_atmos
```

Jeu de caractères admis en Fortran 90
Character Set for Fortran 90 Writing Source Code

- Caractères utilisables pour les *instructions* et *identificateurs*
  - lettres A–Z, a–z, chiffres 0–9
  - caractères spéciaux: = + - * / ! ? : . ; , " ' (< >) % & $
  - le caractère espace (blanc)
  - le caractère _ (*underscore*)
- Autres caractères possibles pour contenu de chaînes de caractères
- Pas de distinction minuscules/majuscules, hormis pour le contenu de chaînes de caractères.

Formats de code
Code formats

- Codes: fichiers texte
- Deux formats sont possibles
  - format fixe (héritage FORTRAN 77 et prédécesseurs)
  - format libre (recommandé, innovation Fortran 90)
- Distinction via extension des noms de fichier
  - `*.for`, `*.f`, `*.F`: format fixe
  - `*.f90`, `*.F90`: format libre

Format libre
Free Format

- Jusqu'à 132 caractères par ligne
- Instructions trop longues peuvent être encodées sur plusieurs lignes
  - chaque ligne incomplète est alors terminée par un caractère de continuation `&`, qui peut être répété en début de la ligne suivante (ligne de continuation)
  - maximum 39 lignes de continuation
  - pour couper une chaîne de caractères : faire précéder premier caractère significatif sur la ligne de continuation d'un `&` aussi
- Possibilité de mettre plusieurs commandes sur une ligne, séparées par des ;

Format libre
Free Format

Exemple

```
PROGRAM pi_calc_0
! Calcul de pi, version 0

DOUBLE PRECISION :: pi

! pi = 4 * arctan(1), par serie de Taylor
pi = 4D0 * (1 - 1D0/3D0 + 1D0/5D0 - 1D0/7D0 + 1D0/9 &
             - 1D0/11D0 + 1D0/13D0 - 1D0/15D0 + 1D0/17D0 &
             - 1D0/19D0 )
WRITE(*,*) 'Pi approché ', pi
WRITE(*,*) 'Cette manière de calculer pi est à déconseiller, &
           &car il faut inclure plus de 500 termes pour avoir&
           & une précision de ... trois déci&
           &males.'           ! couper ici ^^ n'est pas malin,
                              ! mais c'est parfois nécessaire
                              ! et c'est permis.
END PROGRAM pi_calc_0
```

Format fixe
Fixed Format

- Héritage du temps des cartes perforées
- Chaque ligne subdivisée en 3 zones différentes
  1. colonnes 1–5 : réservées aux étiquettes (*label*), suite de 1 à 5 chiffres non tous nuls, ⇒ repères du programme
  2. colonne 6 : caractère non-blanc en colonne 6
     ⇒ continuation de la ligne précédente ;
     dans ce cas, les colonnes 1 à 5 doivent rester vides
  3. colonnes 7–72 : instructions et mots-clé
- FORTRAN 77 standard : maximum 9 lignes de continuation consécutives
- Fortran 90 standard, format fixe : maximum 19 lignes de continuation consécutives
- Fortran 90 standard, format fixe : possibilité de mettre plusieurs commandes sur une ligne, séparées par des ; (pas possible en FORTRAN 77)

Commentaires dans le code
Source Code Comments

- Format libre
  - caractère `!` indique que la suite de cette ligne est un commentaire et ne doit pas être traitée par le compilateur
  - exception : caractère `!` dans une chaîne de caractères
  - `!` interdit après un signe de continuation (`&`) qui coupe une chaîne de caractères (permis derrière les autres `&`)
- Format fixe FORTRAN 77
  - lignes de commentaires caractérisées par `c`, `C`, ou `*` en première colonne
- Format fixe Fortran 90
  - avec `c`, `C`, ou `*` en première colonne (id. FORTRAN 77)
  - avec `!` comme en format libre
  - exception : `!` en colonne 6 caractérise toujours une ligne de continuation et n'initie pas un commentaire
- En toute généralité : une ligne de commentaire ne peut être continuée que par une nouvelle ligne de commentaire

Méthodes numériques et éléments de programmation

Guy Munhoven

Eléments de base

Identificateurs: noms permis
Identifiers: Possible Names

Un identificateur permet de donner un nom à

- une variable,
- une constante,
- un sous-programme
  (sous-routine ou fonction $\rightarrow$ plus tard).

Spécifications :

- défini par une suite de caractères alphanumériques (lettres non accentuées, chiffres, *underscore*),
- son premier caractère doit être une lettre,
- longueur limitée à 31 caractères,
- pas de distinction majuscules/minuscules.

Types de données (variables et constantes)
Data Types (Variables and Constants)

Type	Octets /unité	Etendue	Précision
`LOGICAL`	1–4	`.TRUE.`, `.FALSE.`	—
`INTEGER`	2	$-32768 < i < 32767$	exacte
	4	$-2.147.483.648 < i < 2.147.483.647$	exacte
`REAL`	4	$1.2 * 10^{-38} < \|r\| < 3.4 * 10^{+38}$	6–7
`DOUBLE PRECISION`	8	$2.2 * 10^{-308} < \|r\| < 1.8 * 10^{+308}$	15–16
`COMPLEX`	2*4	voir `REAL`	
`CHARACTER`	1/caract.	—	—

Identificateurs et typage par défaut
Identifiers and Default Typing

- Par défaut, les identificateurs typés (p.ex., variables) mais non déclarés sont de type
  - `INTEGER`, si leurs noms commencent par `I`, `J`, `K`, `L`, `M`, ou `N`
  - `REAL` sinon
- Typage par défaut offre la commodité de ne pas devoir déclarer toutes les variables
- Origine de beaucoup d'erreurs de programmation
  $\Rightarrow$ erreurs de frappe peuvent engendrer des variables "fantômes" ou confondues
- Recommandation: désactiver le typage par défaut en faisant précéder la section des déclarations de l'instruction `IMPLICIT NONE`

Types de données : gamme minimale
Data Types : Minimal Set

- Norme Fortran 90 : minimum deux types de `REAL`
  - `REAL` (simple précision)
  - `DOUBLE PRECISION` (précision étendue)
  - un type de `INTEGER`
  - pas de spécifications précises quant à la précision et l'étendue de la gamme de valeurs

  $\Rightarrow$ implémentations différentes selon les compilateurs
- Spécifications précises à l'aide des fonctions intrinsèques
  - `SELECTED_INT_KIND(r)` pour les entiers
  - `SELECTED_REAL_KIND(p,r)` pour les réels
  - `p` et `r` de type `INTEGER`, fixant resp. la précision décimale (*precision*) et l'étendue (*range*) de la gamme de valeurs

Constantes littérales
Literal Constantes

Représentations de 10 comme constante

`10`	`INTEGER`, espèce par défaut
`10.`	`REAL`, espèce par défaut
`1E1`	`REAL`, espèce par défaut
`10.E0`	`REAL`, espèce par défaut
`10.D0`	`REAL`, espèce `DOUBLE PRECISION`
`'10'`	`CHARACTER`
`"10"`	`CHARACTER`

Opérations courantes
Common Operations

Variables numériques	
Opération	Opérateur
Exponentiation	**
Multiplication	*
Division	/
Addition	+
Soustraction	-
Identité	+
Opposé	-

Variables logiques	
Opération	Opérateur
négation	`.NOT.`
et logique	`.AND.`
ou logique	`.OR.`
équivalence logique	`.EQV.`
non équivalence logique	`.NEQV.`

Chaînes de caractères	
Opération	Opérateur
Concaténation	//

Opérateurs relationnels
Relational Operators

Opération	opérateurs
strictement plus petit	`.LT.` ou <
inférieur ou égal	`.LE.` ou <=
strictement supérieur	`.GT.` ou >
supérieur ou égal	`.GE.` ou >=
égal	`.EQ.` ou ==
différent	`.NE.` ou /=

Particularités pour les relations entre chaînes de caractères

- Résultats pour `.LT.`, `.LE.`, `.GT.` et `.GE.` dépendent de l'ordre des caractères adopté par le processeur (la machine)
- Pour des résultats basés sur l'ordre ASCII, utiliser les fonctions logiques intrinsèques `LLT(chaine1, chaine2)`, `LLE(...)`, `LGT(...)` et `LGE(...)`

Opérations : ordre de priorité
Operations: precedence

1. `(...)`
2. `**`, de droite à gauche si successives

$$\begin{aligned} 2**3**2 \Leftrightarrow 2**(3**2) &= 2^{(3^2)} = 2^9 = 512 \\ (2**3)**2 &= 8^2 = 64 \end{aligned}$$

3. `/`, de gauche à droite si successives

$$\begin{aligned} 4/2/2 \Leftrightarrow (4/2)/2 = 2/2 = 1 \\ 4/(2/2) = 4/1 = 4 \end{aligned}$$

4. `*`
5. `+`, `-`
6. opérations relationnelles
7. opérations logiques

$\Rightarrow$ en cas de doute : utiliser des parenthèses `(...)`

Opérations entre types différents : conversions
Operations Between Different Types: Conversions

Conversion de type vers le type le plus fort

- `INTEGER` $\prec$ `REAL` $\prec$ `DOUBLE PRECISION`
- `REAL` $\prec$ `COMPLEX` de même espèce
- *Exception*: si `**` est suivie d'un `INTEGER`, ce dernier ne sera pas converti, quelque soit le type de la base de puissance
  - `(-1.)**2` sera évalué correctement à `(-1.)*(-1.) = 1.`
  - `(-1.)**2.` donne lieu à une exception arithmétique, car calculée comme `EXP(2.*LOG(-1.))`

Opérations entre types différents : conversions

Operations Between Different Types: Conversions

Conversions effectuées au fur et à mesure de l'évaluation d'expressions complexes : attention aux pièges et surprises

`x = 2/4/2.0`	`x = 2/4.0/2`
$\Leftrightarrow \quad x = (2/4)/2.0$	$\Leftrightarrow \quad x = (2/4.0)/2$
$\rightsquigarrow \quad x = 0/2.0$	$\rightarrow \quad x = (2./4.0)/2$
$\rightarrow \quad x = 0./2.0$	$\rightsquigarrow \quad x = 0.5/2$
$\rightsquigarrow \quad x = 0.$	$\rightarrow \quad x = 0.5/2.$
	$\rightsquigarrow \quad x = 0.25$

$\Rightarrow$ Inclure les constantes littérales avec leur type approprié

Fonctions mathématiques courantes

Common Mathematical Functions

Fonction mathématique	Fonction Fortran	Remarque
$\sin(x)$	`SIN(X)`	x en radians
$\cos(x)$	`COS(X)`	x en radians
$\tan(x)$	`TAN(X)`	x en radians
$\text{asin}(x)$	`ASIN(X)`	résultat en radians
$\text{acos}(x)$	`ACOS(X)`	résultat en radians
$\text{atan}(x)$	`ATAN(X)`	résultat en radians
$\exp(x)$	`EXP(X)`	
$\ln(x)$	`LOG(X)`	
$\log_{10}(x)$	`LOG10(X)`	
$\sqrt{x}$	`SQRT(X)`	

Toutes ces fonctions fournissent un résultat du même type que l'argument (x); si x est un tableau, le résultat sera un tableau de même dimension que x.

Boucles: actions répétitives
Loops: Repetitive Action

- Forme générique

```
[nom_de_boucle:] DO [contrôle_de_boucle]
    Instructions de la boucle
END DO [nom_de_boucle]
```

- Le *nom_de_boucle* optionnel fournit une référence pour la boucle; doit être unique dans l'unité de programme
- Trois grands types, suivant type de *contrôle_de_boucle*
  - à nombre indéterminé d'itérations
  - de type `DO WHILE`
  - à nombre fixe d'itérations

Boucle à nombre indéterminé d'itérations
Unlimited Loop

contrôle_de_boucle vide

Exemple de boucle à nombre indéterminé d'itérations

```
PROGRAM pi_calc_01a
IMPLICIT NONE
INTEGER          :: i = 1
DOUBLE PRECISION :: pi = 1D0, delta, pm_un = -1D0

helice: DO
  i = i + 1
  delta = 1D0/(2D0*DBLE(i)-1D0)
  pi = pi + pm_un*delta
  pm_un = -pm_un
 ! Boucler indéfiniment
END DO helice
! On ne va jamais arriver ici

END PROGRAM pi_calc_01a
```

Boucle à nombre indéterminé d'itérations
Unlimited Loop

contrôle_de_boucle vide

Exemple de boucle à nombre indéterminé d'itérations

```
PROGRAM pi_calc_01a
IMPLICIT NONE
INTEGER          :: i = 1
DOUBLE PRECISION :: pi = 1D0, delta, pm_un = -1D0

helice: DO
  i = i + 1
  delta = 1D0/(2D0*DBLE(i)-1D0)
  pi = pi + pm_un*delta
  pm_un = -pm_un
  IF(delta<1D-6) EXIT helice
END DO helice
! Continuation ici après abandon de la boucle par EXIT
WRITE(*,*) i, delta, pi*4.D0
END PROGRAM pi_calc_01a
```


Boucle à nombre déterminé d'itérations
Limited Loop

contrôle_de_boucle de la forme

DO compte = *compte_ini*, *compte_fin* [, *pas_compte*]

Exemple de boucle à nombre déterminé d'itérations

```
PROGRAM pi_calc_01b
IMPLICIT NONE
INTEGER          :: i, n_termes = 100000
DOUBLE PRECISION :: pi = 1D0, delta, pm_un = -1D0

helice: DO i = 2, n_termes
  delta = 1D0/(2D0*DBLE(i)-1D0)
  pi = pi + pm_un*delta
  pm_un = -pm_un

END DO helice
! Continuation ici lorsque i>n_termes
pi = 4.D0*pi
WRITE(*,*) i, delta, pi
END PROGRAM pi_calc_01b
```

Boucle à nombre déterminé d'itérations
Limited Loop

contrôle_de_boucle de la forme

DO compte = *compte_ini*, *compte_fin* [, *pas_compte*]

Exemple de boucle à nombre déterminé d'itérations

```
PROGRAM pi_calc_01b
IMPLICIT NONE
INTEGER          :: i, n_termes = 100000
DOUBLE PRECISION :: pi = 1D0, delta, pm_un = -1D0

helice: DO i = 2, n_termes
  delta = 1D0/(2D0*DBLE(i)-1D0)
  pi = pi + pm_un*delta
  pm_un = -pm_un
  IF(delta<1D-4) EXIT helice ! Abandon prématuré permis
END DO helice
! Continuation ici après abandon de la boucle par EXIT
pi = 4.D0*pi
WRITE(*,*) i, delta, pi
END PROGRAM pi_calc_01b
```


Boucle DO WHILE
DO WHILE Loop

contrôle_de_boucle de la forme

DO WHILE(*condition_logique*)

Exemple de boucle DO WHILE

```
PROGRAM pi_calc_01c
IMPLICIT NONE
INTEGER          :: i = 2, n_termes = 100000
DOUBLE PRECISION :: pi = 1D0, delta, pm_un = -1D0

helice: DO WHILE (i<=n_termes)
  delta = 1D0/(2D0*DBLE(i)-1D0)
  pi = pi + pm_un*delta
  pm_un = -pm_un ; i = i+1
  IF(delta<1D-4) EXIT helice ! Abandon prématuré permis
END DO helice
... ! Continuation ici après abandon de la boucle par EXIT
pi = 4.D0*pi
WRITE(*,*) i, delta, pi
END PROGRAM pi_calc_01c
```

Boucles: condition d'exécution des boucles

Loops: Conditions of Execution

- DO indéterminé : exécution inconditionnelle
- DO i = *i_ini*, *i_fin* [, *i_inc*]
  1. Par défaut, *i_inc* = 1
  2. A l'arrivée à la ligne DO : i initialisé à *i_ini*
  3. Au départ de la ligne DO,
     - l'exécution continue avec les instructions de la boucle
       si i $\leq$ *i_fin* pour *i_inc* $>$ 0 et
       si i $\geq$ *i_fin* pour *i_inc* $<$ 0 ;
     - l'exécution continue à la première instruction suivant la ligne END DO de la boucle sinon
  4. A la ligne END DO ou lorsqu'une instruction CYCLE est rencontrée :
     - i = i + *i_inc*
     - puis retour à la ligne DO pour l'étape 3
  5. Lorsqu'une instruction EXIT est rencontrée:
     - l'exécution continue à la première instruction suivant la ligne END DO de la boucle auquel EXIT se rapporte.

Boucles : condition d'exécution des boucles

Loops: Conditions of Execution

- DO WHILE(*condition_logique*)
  1. Si la *condition_logique* s'évalue à .TRUE. à la ligne DO, les instructions de la boucle sont exécutées, sinon, l'exécution continue après la ligne END DO
  2. A la ligne END DO ou lors d'une instruction CYCLE : retour à la ligne DO pour l'étape 1
  3. Une modification de la *condition_logique* au cours de l'exécution de la boucle n'a pas d'influence — ce n'est que l'évaluation à la ligne DO qui est déterminante pour l'exécution ou non des instructions de la boucle

Boucles : à faire et à ne pas faire

Loops: Dos and Don'ts

- Si un *nom_de_boucle* précède `DO`, il doit obligatoirement suivre le `END DO` correspondant.
- Aussi bien l'indice `i` qui contrôle une boucle, que les *paramètres de boucle* *i_ini*, *i_fin* et *i_inc* doivent être de type `INTEGER`.
- Il n'est pas permis de modifier directement la valeur de l'indice `i` à l'intérieur de la boucle.
- La *condition_logique* d'une boucle `DO WHILE` ne peut pas se réduire à une simple variable logique: en modifiant la valeur de cette variable à l'intérieur de la boucle on modifierait le *critère* de la boucle, ce qui n'est pas permis.