Chapitre 1

Notes sur le langage Fortran 90

Contenu

1.1 Avant-propos 3
1.2 Le recyclage des vieux programmes Fortran 77 . . 4
1.3 Les déclarations de variables 5
1.3.1 Types 5
1.3.2 Les tableaux 6
1.4 Les instructions itératives 7
1.5 Les instructions conditionnelles : if, select, case . . 9
1.6 Opérateur de comparaison, opérateurs d'assignation 10
1.7 Les modules 10
1.8 Allocation dynamique de mémoire 11
1.8.1 Introduction 11
1.8.2 Exemple 12
1.9 Les entrées, les sorties, les formats 12
1.10 Les fonctions intrinsèques 13
1.11 Passage des variables, tableaux, des fonctions . . . 13
1.12 Conclusion et références 15

1.1 Avant-propos

Le langage Fortran 77 est un langage obsolète, même si une partie de la communauté scientifique a continué, par conservatisme excessif, à utiliser ce langage. Le Fortran a subi une évolution notable avec l'apparition du Fortran 90, qui intégrait les notions modernes de la programmation en intégrant par exemple l'allocation dynamique de mémoire, les notions de structures... que l'on connaissait déjà avec le langage C depuis longtemps. Par volonté de garder une compatibilité quasi-totale avec la version précédente, on peut continuer à garder les mauvaises habitudes du Fortran 77, mais le but de ces notes est d'encourager vivement à perdre celles-ci et pour les plus jeunes à ne jamais commencer à les utiliser.

Il existe aujourd'hui une bibliothèque non négligeable de programmes qui sont écrits en Fortran 90 (voire Fortran 95). L'intérèt de cette évolution pour ceux qui ont pris depuis longtemps l'habitude de programmer en Fortran est de retrouver une partie de leurs habitudes, tout en permettant une évolution "douce" par rapport à un changement complet de langage.

Il est évidemment nécessaire d'avoir un compilateur spécifique pour le Fortran 90. Longtemps absent du domaine public. Depuis 2005, le compilateur GNU gfortran permet de compiler des programmes en Fortran 90.

Il existe aussi un compilateur disponible et gratuit sur plateforme Linux qui est le compilateur Intel qui est un compilateur de Fortran 90 (et 95). Il s'appelle ifort.

1.2 Le recyclage des vieux programmes Fortran 77

Dans la mesure où l'écriture du programme ne contient pas des instructions qui ont été supprimés dans la version Fortran 90, on peut vérifier que des programmes simples de Fortran 77 sont compilés par un compilateur de Fortran 90.

Cette compatibilité ascentante des langages présente tout de meme un gros défaut, à savoir que le Fortran ne demande pas la déclaration des variables et qu'il ajouter l'instruction

Pour une meilleure efficacité, il est bien sur souhaitable de substituer la syntaxe d'un programme de Fortran 77 en un vrai programme de Fortran 90. Pour cela, il existe des outils comme f2f.pl (qui est un script écrit en Perl) qui permet de passer de manière très rapide d'un programme Fortran 77 en Fortran 90. Bien évidemment, cela reste moins efficace qu'une réécriture complète en Fortran 90, mais cela permet de gagner du temps dans la traduction, meme si les spécificités du Fortran 90 ne seront pas incluses. On peut par la suite progressivement introduire des spécificités du Fortran 90 pour améliorer le code.

A cause de la compatibilité, on peut toujours éviter de déclarer des variables, ce qui est fortement déconseillé. Pour obliger le compilateur à vérifier que les variables sont déclarées avec le bon type, il est nécessaire d'ajouter la directive **implicit none** dans chaque programme et module.

Les avantages de l'utilisation du Fortran 90 sont des fonctionnalités suivantes qui sont disponibles.

- L'écriture vectorielle des programmes (exemple x(1 :N)=y(1 :N)*COS(a(1 :N)))
- L'allocation dynamique de mémoire (ALLOCATE, DEALLOCATE)
- Les types dérivés et la surcharge d'opérateurs
- Le prototypage est possible.
- La notion de MODULE
- Les instructions conditionnelles étendues avec l'instruction SELECT CASE
- Le formatage libre de l'écriture du programme.

En ce concerne le formatage du code (libre ou fixe), il est possible de mélanger les deux formes à condition de remplacer pour les commentaires le caractère C par le point d'exclamation.

Le Fortran 95 est une version légèrement plus récente qui a introduit quelques

modifications relativement mineures et quelques extensions avec l'instruction FORALL et les procédures PURE et ELEMENTAL. Il existe a priori une version Fortran 2003, mais il n'existe pas encore de compilateur pour cette version de Fortran.

1.3 Les déclarations de variables

Le langage Fortran est un langage fortement typé, car il possède un grand nombre de type prédéfini. Pour la déclaration des variables, on peut utiliser soit la syntaxe du Fortran 77, mais on peut utiliser la syntaxe propre du Fortran 90

1.3.1 Types

Les prédéfinies

integer : :	entier de 4 octets
real : :	réel de 4 octets
character : :	caractère de 1 octet
logical : :	variable booléenne
complex : :	nombre complexe (2x4 octets)

L'initialisation des variables peut se faire soit par l'instruction **data** (façon un peu vieillotte), soit de façon directe dans la déclaration des variables soit directement

```
integer ::i=7,j=9
real :: x=5
```

Instruction kind

Si le besoin de choisir une précision particulière (souvent plus grande que le standard) est limité à quelques variables, le Fortran 90 permet de spécifier le type avec l'instruction kind.

```
integer(kind=1) ::i=7,j=9
real (kind=2):: x=5
```

Par défaut, pour les entiers kind=4, ce qui signfie que les entiers sont sur 4 octets. Si kind=8, les entiers sont sur 8 octest, ii kind=2, les entiers sur deux octets. et si kind=1, les entiers sont sur un octet et vont de -128 à 127.

Pour les réels, la double précision correspond à kind=8. La simple précision, le standard est sur 4 octets correspondant à kind=4

Le Fortran 90 offre la possibiblité de choisir la précision des types avec l'instruction selected_int_kind for les nombres entiers et selected_real_kind for les nombres réels.

En fournissant l'argument selected_int_kind(6), on oblige le compilateur à choisir un type d'entier (dépendant de la machine) qui pourra aller de -10^6 à 10^6.

En fournissant les arguments selected_reel_kind(9,99), on oblige le compilateur à choisir un type de réel (dépendant de la machine) qui pourra aller de -10^{99} à 10^{99} avec au moins une précision de 9 chiffres après la virgule.

En définissant une constante entière, on peut utiliser le nombre type ainsi défini dans le reste du programme.

```
integer, parameter ::dpi=selected_int_kind(2)
integer (kind=dpi) ::i=4_dpi,j=7_dpi
```

On note que les valeurs sont suffixés avec le type défini.

On voit bien qu'il suffit de modifier la définition du paramètre dpi pour changer dans la totalité du programme, la précision à laquelle les calculs sont effectués.

Structure

La notion de structure est définie en Fortran 90.

```
type num_point
   integer :: i
   real :: x,y
end type num_point
```

correspond à la définition d'une structure nouvelle constitué d'un entier et de deux réels. L'accès aux membres de la structure se fait à partir de l'opérateur %. La déclaration et l'initialisation de la structure peut se faire de la manière suivante

```
type (num_point) cc cc%i=6 cc%x=0.2 cc%y=2.7
```

1.3.2 Les tableaux

Déclarations

En Fortran 90, les tableaux commencent aussi par défaut avec un indice égal à 1. La déclaration d'un tableau de 100 éléments de réels à double précision est faite soit avec la syntaxe du Fortran 77 soit avec

```
real, dimension(100) :: a
```

Le premier élément commence toujours à l'indice 1.

Noter qu'en Fortran 90, les parenthèses servent toujours à la fois pour les tableaux et les arguments de fonctions, ce qui ne facilite pas la lecture d'un programme et nécessite d'autant plus d'être rigoureux dans l'écriture. Il est possible de faire démarrer un tableau avec un indice null ou négatif avec la déclaration suivante

```
real, dimension(-2:100):b
```

Le tableau contient alors 103 éléments et $b(-2)$ est le premier élément.

Les tableaux à plusieurs dimensions sont définis en Fortran 90 de la manière suivante

```
real, dimension(10,20):: a
```

Les tableaux sont aussi toujours stockés en mémoire en colonnes

L'initialisation des tableaux peut se faire aussi avec l'instruction data soit de la manière suivante

```
real, dimension(10,20)::a= (/ (0.1*i, i = 1, 200) /)
real, dimension(2,2)::b=(/3.0,0.7,-4.1,1.0/)
```

Notons que pour la matrice b l'affectation est éléments se fait selon l'ordre des colonnes.

Le Fortran 90 permet une initialisation conditionelle avec la séquence d'instruction where-elsewhere-endwhere en traitant l'ensemble des éléments d'un tableau Par exemple

```
integer, dimension(4,4) :: a
where (a<=0)
   a=0
elsewhere
   a=1/a
endwhere
```

L'utilisation des *common* est à proscrire, meme cette compatibilité avec le Fortran 77 permet théoriquement de garder cette ordre. La notion de module permet de remplacer tout common existant avec plus de sécurité et aussi plus de fonctionnalités.

L'utilisation des *parameter* permet de déclarer des variables qui ne changeront pas durant l'exécution du programme. La syntaxe du Fortran 90 permet en une seule ligne de code de déclarer et d'initialiser la variable.

```
integer, parameter ::dim=4
```

1.4 Les instructions itératives

Pour les boucles, la syntaxe est de la forme **do enddo** qui contient un indice de boucle (variable entière).

```
integer m,n,step
do i=m,n
...
enddo
```

où step correspond au pas d'incrémentation. Si n est strictement inférieur à m, aucune instruction de la boucle ne sera exécutée.

Il y a deux instructions **cycle** et **exit** qui permettent de sauter le reste des instructions de la boucle et de passer à l'itération suivante et de sortir de la boucle en allant à la première instruction placée juste après. Il faut faire attention que l'instruction exit ne correspond en rien à celle du langage C et C++ qui donne un moyen de sortir normalement ou anormalement du programme.

En Fortran l'instruction **exit** permet juste de continuer le programme au delà de la boucle.

Comme il arrive souvent d'avoir à faire des opérations répétitives, on dispose en programmation d'instructions qui vont entourer un bloc d'instructions pour les faire exécuter un grand nombre de fois. Si le nombre est connu à l'avance, on utilise l'instruction **do enddo** qui contient un indice de boucle (variable entière).

```
integer m,n,step
do i=m,n
...
enddo
```

où step correspond au pas d'incrémentation. Si n est strictement inférieur à m, aucune instruction de la boucle ne sera exécutée. Si le nombre d'itérations dépend d'une condition la syntaxe est

```
do while(condition)
...
enddo
```

Attention : Une boucle while n'est interrompue que si la condition devient fausse en un nombre d'itérations fini. Parmi les erreurs classiques de programmation, il arrive qu'un programme n'arrive pas à se finir car la condition reste indéfiniment vraie

Par exemple si l'on veut faire l'équivalent d'une boucle do-while du langage C ou C++, on doit écrire la condition suivante

```
do
 instructions
if(condition) exit
enddo
```

où la condition est le contraire de celle que l'on met dans la boucle do-while.

A noter que le fortran 90 permet d'éviter une partie d'écriture des instructions itératives avec l'emploi de la notation vectorielle ou de la boucle implicite.

Exemples :

```
program simple
implicit none
real, dimension(5,5) :: a,b
a=0
b=a+1
write (*,*) a
write (*,*) b
end program simple
```

Un autre exemple

```
program simple
implicit none
real, dimension(5,5) :: a,b
integer ::i
do i=1,5
   a(i,:)=i
enddo
b=cos(a)
write (*,*) a(1:5,1:5)
write (*,*) b
end program simple
```

1.5 Les instructions conditionnelles : if, select, case

Pour les instructions conditionnelles simples, on a la structure usuelle if-then-else-endif.

```
if (condition)
then
 instructions1
else
 instructions2
endif
```

Cette instruction signifie que si la condition est vraie les instructions du groupe 1 sont exécutées, tandis que si la condition est fausse les instructions du groupe 2 sont exécutées. On a aussi la possibilité de faire des structures imbriquées de manière analogue à celle présentée dans le Fortran 77.

Le Fortran 90 a introduit la notion de select case qui permet de faire des branchements multiples.

```
select case  (valeur entière)
{
case (1)
 instructions1
case (2)
 instructions2
case (3,5)
 instructions3
case default
 instructionsN
}
```

La signification de ces instructions est la suivante : si la valeur entière est 1 le groupe d'instruction 1 est exécuté, si la valeur entière est 2 le groupe d'instruction 2 est exécuté, si la valeur entière est 3 ou 5, le groupe d'instruction 3 est exécuté, pour tout autre valeur entière, le groupe d'instruction N est exécuté.

1.6 Opérateur de comparaison, opérateurs d'assignation

Pour formuer une expression logique, il est souvent nécessaire d'avoir un opérateur de comparaison, le langage Fortran 77 fournit plusieurs opérateurs que nous rassemblons ci-dessous

$<$	inférieur(strictement) à
$>$	supérieur (strictement) à
$<=$	inférieur ou égal à
$>=$	supérieur ou égal à
$/=$	différent de
$==$	égal à

Il existe aussi des opérateurs de comparaison logiques

$.AND.$	et logique
$.OR.$	ou logique
$.XOR.$	ou exclusif

Dans l'exemple suivant la boucle do while est exécutée 6 fois

```
program toto
implicit none
integer  ::i=4,j=1
do while((i<10).and.(j<12))
   write (*,*) i,j
   i=i+1
   j=j+1
enddo
end program toto
```

1.7 Les modules

Les modules sont des objets qui regroupe des données et/ou des procédures et qui peuvent insérer dans des portions du programme principal ou à l'intérieur d'autres procédures ou fonctions : la structure de base est la suivante

L'exemple ci-dessous montre l'utilisation d'un module contenant que des functions à l'intérieur d'un programme principal. De plus la procédure echange ne peut être appelé qu'à partir de la procédure order définie dans le module. Ce concept permet d'éviter les interactions entre différentes parties non prévues.

```
module ModuleSimple
 implicit none
  private echange   ! echange n'est visible que dans le module
contains
  subroutine order( x, y )   ! Publique par défaut
    integer, intent( inout ) :: x, y
```

```
    if ( abs( x ) < abs( y ) )  call echange( x, y )
  end subroutine order


  subroutine echange( x, y )
    integer, intent( inout ) :: x, y
    integer tmp
    tmp = x; x = y; y = tmp ! Echange x et y.
  end subroutine echange
end module ModuleSimple

program Simple
  use modulesimple
  implicit none
  ! Declarer and initialiser x and y.
  integer :: x = 10, y = 20

  write (*,*) x, y
  call order( x, y )
  write(*,*) x, y
end program Simple
```

La directive private signifie que le sous-programme ne peut pas être appelé en dehors d'une fonction ou sous-programme appartenant au module Module-Simple.

Le deuxième exemple contient des données

1.8 Allocation dynamique de mémoire

1.8.1 Introduction

L'allocation dynamique de mémoire permet de créer en cours d'éxécution du programme des tableaux dont la taille n'est pas fixée a priori. La séquence des instructions à effectuer est la suivante

- Déclarer le type de tableau qui devra être créé dynamiquement ainsi que sa nature (une dimension, deux dimensions,...)
  ```
  real, allocatable :: A( :, : )
  ```
  Ici, A sera un tableau à deux dimensions
- Faire l'allocation proprement dite
  ```
  allocate( A( 7, 8 ) )
  ```
  Un tableau de 7 lignes et 8 colonnes.
- A la fin de l'utilisation du tableau A, il faut libérer la mémoire
  ```
  deallocate (A)
  ```
  cela est réalisé par l'instruction deallocate. Les compilateurs ne signalent pas toujours l'absence de l'instructin desallocate, ce qui peut conduire à un problème de fuite de mémoire (c'est-à-dire que le programme en fin d'éxécution ne libère toute la mémoire utilisée ; ce travail est alors

réservé au système d'exploitation qui généralement le fait correctement, mais si cela n'est aussi fait par lui, l'ordinateur finit alors par disposer qu'une partie de plus en plus réduite de sa mémoire et un redémarrage est nécessaire.

1.8.2 Exemple

```
program AllocatableMatrices
      implicit none
   real, allocatable :: A( :, : )
   integer n
   write(*,*), 'Donner un entier: '
   read (*,*) n

   ! Allouer dynamiquement une matrice  n x n.
   allocate( A( n, n ) )
   call random_number( A )   ! Remplir la matrice  A avec des nombres
                             ! aléatoires entre [0,1) .

   write(*,*), 'A = '
   write(*,*)  A
   deallocate (A)
end program AllocatableMatrices
```

Signalons l'une des vertus du Fortran 90, qui permet très simplement de pouvoir écrire un tableau complet avec l'instruction write sans faire une boucle sur les indices.

1.9 Les entrées, les sorties, les formats

Pour lire des données au clavier, on peut utiliser le format par défaut suivant

```
read (*,*) variable
```

Pour écrire à l'écran, on utilise

```
write(*,*) variable
```

Il exite aussi l'instruction **print** dans le style obsolète !

On peut aussi écrire en alternant les chaines de caractères et variables

```
write (*,*) ' var1= 'variable1,'var2 =',variable2
```

Si l'on souhaire écrire avec un format bien spécifique on peut écrire l'instruction suivante pour deux variables une entière et une réelle.

```
integer i
real a
 write(*,'(I5 E15.5)')i,a
```

On ne peut que recommander l'abandon de l'instruction format pour l'utilisation du Fortran 90, puisque nous avons donné les ces règles pour le Fortran 77 alors qu'il s'agissait d'extension.

1.10 Les fonctions intrinsèques

Le langage Fortran possède un nombre significatif de fonctions dans la bibliothèque standard. Il en existe de deux types principalement : les fonctions qui permettent de convertir les données d'un type en autre type (par exemple entier en flottant) et les fonctions mathématiques, à la fois les fonctions transcendantes usuelles et une partie des fonctions spéciales rencontrées assez fréquemment.

On retrouve les fonctions intrinsèques du Fortran 77 et nous ne les rappelons pas ici. Il existe des fonctions intrinsèques supplémentaires en Fortran 90, pour la manipulation de matrices

matmul(A,B) où A est une matrice $n \times m$ et B une matrice $m \times k$

pour la manipulation de vecteurs

dot_product(A,B)	produit scalaire avec A et B des vecteurs de même taille
sum(A)	calcule la somme des éléments d'un vecteur
maxval(A)	plus grand élément d'un vecteur
minval(A)	plus petit élément d'un vecteur
product(A)	produit des éléments d'un vecteur
sum(A)	somme des élément d'un vecteur

1.11 Passage des variables, tableaux, des fonctions

Le passage des variables du Fortran 90 peut être sécurisé. Les variables transmises peuvent avoir un attribut qui spécifie si la variable peut être modifiée ou non dans la fonction appelée.

L'instruction intent avec un argument in ,out ou inout signifie que la variable est une variable d'entrée (non modifiable), une variable de sortie (donc récupérable) ou une variable qui joue les deux roles en meme temps respectivement. Par exemple

```
subroutine addition(a,b,x)
 real, intent(in)::a,b
 real, intent (out)::x
  x=a+b
 return
endsubroutine addition
```

Pour le passage des tableaux, il suffit de transmettre le nom du tableau pour avoir accès de ce tableau dans le sous-programme.

Pour les fonctions, il est nécessaire de déclarer la fonction appelée avec l'ordre external et de préciser son type. Le nombre de variables de la fonction appelée n'a pas besoin d'être spécifié. Cela peut être bien entendu un dangereux, puisque aucune vérification n'est faite à la compilation. Le programme suivant calcule deux integrales par la méthode des rectangles ; la même version en C a été donnée au premier chapitre, la seconde en Fortran 77 au chapitre 2, voici la version pour le Fortran 90

```
program integrale
 implicit none
```

```
    integer ::  n_point
    real ::  a_inf,b_sup
    real ::  rectangle,trapeze
    real ::  func_1,func_2
    real ::  res,res_ex
    integer :: nf
    external trapeze
    external func_1,func_2
! variables locales
    integer :: k
!*****************************************************************
! premiere fonction
!*****************************************************************
    a_inf=0.0
    b_sup=0.5
    n_point=10
    res_ex=1.0d0/3.0d0+log(3.0d0)/4.0d0
    write(*,*)' Iex=',res_ex
!*****************************************************************
    res=trapeze(a_inf,b_sup,n_point,func_1)
!*****************************************************************
! deuxieme fonction
!*****************************************************************
    b_sup=1.0
    n_point=10
    res_ex=acos(-1.0d0)/4.0d0
    write(*,*)' Iex=',res_ex

    res=trapeze(a_inf,b_sup,n_point,func_2)
!*****************************************************************
    end program
! fin du programme principal
!*****************************************************************

!*****************************************************************
    real function func_1(x)
    implicit none
    real, intent(in) ::  x
    func_1=1.0/((1.0-x*x)*(1.0-x*x))
    return
    end function func_1
!*****************************************************************
    real function func_2(x)
    implicit none
    real, intent (in) ::  x
    func_2=1.0/(1.0+x*x)
    return
```

```
    end function func_2
!*************************************************************
    real function trapeze(a,b,n,f)
    implicit none
    real, intent(in) ::  a,b
    integer, intent(in) :: n
    real, external ::f
! varianbles locales
    real ::  sum,delta
    integer :: i
! calcul par les trapezes
    sum=0.0
    delta=(b-a)/real(n)
    do i=1,n-1
        sum=sum+f(a+delta*i)
    enddo
    trapeze=delta*(sum+0.5*(f(a)+f(b)))
    return
    end function trapeze
!*************************************************************
```

1.12 Conclusion et références

Le Fortran est un langage aux possibilités nettement plus étoffées que celui de la génération précédente. Ces notes ont pour vocation de pouvoir commencer à écrire des programmes, mais pour pouvoir utiliser pleinement toutes les possibilités de ce langage les ouvrages sont un complément rapidement indispensable. A défaut, il existe de nombreux sites qui permettent d'aller plus loin. Voici une liste non exhaustive de ceux-ci
http ://cch.loria.fr/documentation/documents/F95/F95.html
http ://www.cs.mtu.edu/ shene/COURSES/cs201/NOTES/Fortran.html
http ://wwwasdoc.web.cern.ch/wwwasdoc/f90.html
http ://www.nsc.liu.se/ boein/f77to90/f77to90.html
http ://www.scd.ucar.edu/tcg/consweb/Fortran90/F90Tutorial/tutorial.html
http ://www.nas.nasa.gov/Groups/SciCon/Tutorials/FORTRAN90/
http ://www.liv.ac.uk/HPC/F90page.html
http ://www.idris.fr/