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

29 septembre 2014

Méthodes numériques et éléments de programmation

Guy Munhoven

Plan du cours 2014-2015

Cours théoriques

16-09-2014 *Méthodes numériques pour E.D.O.: introduction*

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

22-09-2014 *Fortran 95: bases, boucles*

23-09-2014 *Fortran 95: branchements, sous-programmes*

23-09-2014 *Fortran 95: modules, opérations d'entrée-sortie*

29-09-2014 Fortran 95

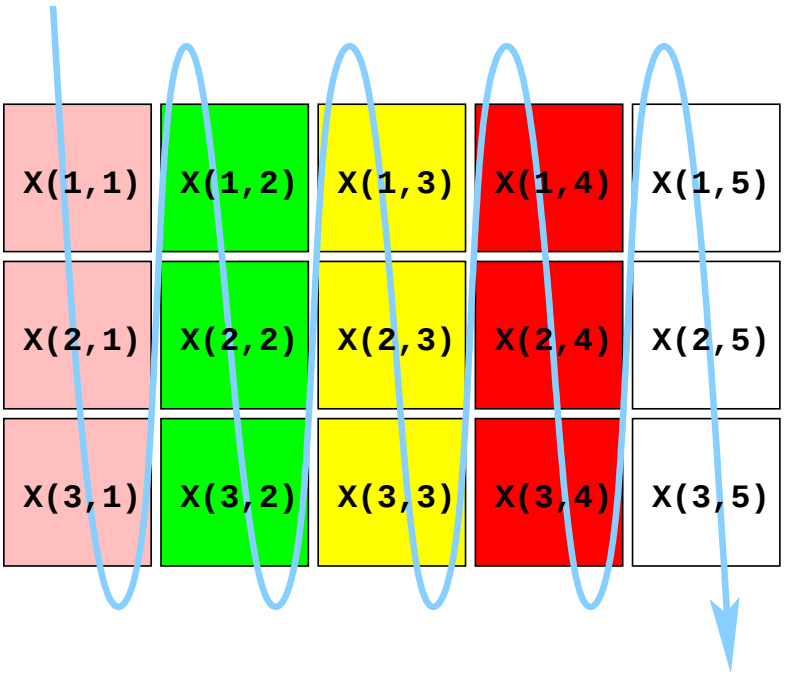
- Tableaux

07-10-2014 Fortran: librairies et collections de routines

Tableaux
Arrays

- nombre maximal de dimensions, appelé rang (*rank*): 7
  (attention: différent du rang en mathématiques)
- suite des dimensions d'un tableau définit sa forme (*shape*)
- tableaux possibles pour tous les types
- type `CHARACTER`: tous les éléments de même longueur
- pour chaque dimension, indices commencent à 1 par défaut
- stockage en mémoire: "par colonnes"
  (cf. matrices en mathématiques)
  - indice de la première dimension varie le plus vite
  - ensuite le deuxième ...
  - pour un tableau déclaré comme
    type`, DIMENSION(3,2) :: A`
    la suite de stockage est `A(1,1)`, `A(2,1)`, `A(3,1)`, `A(1,2)`, `A(2,2)`, `A(3,2)`

Tableaux : ordre de stockage
Arrays: storage order



..., DIMENSION(3,5) :: X

Tableaux : déclarations-type

Arrays: typical déclarations

Exemples de déclarations de tableaux

```
PROGRAM plenty_of_arrays
IMPLICIT NONE
INTEGER, DIMENSION(5,7) :: idx = -1 ! init. tous les éléments à -1
! équivalent à INTEGER, DIMENSION(1:5,1:7) :: idx
! rang de idx: 2
! forme de idx: (/ 5, 7 /)

DOUBLE PRECISION, DIMENSION(-1:1,3,0:7) :: x
! équivalent à DOUBLE PRECISION, DIMENSION(-1:1,1:3,0:7) :: x
! rang de x: 3
! forme de x: (/ 3, 3, 8 /)

CHARACTER(LEN=256), DIMENSION(2,9) :: txtlines
! tableau à 2x9 entrées, d'une longueur de 256 caractères chacune
! rang de txtlines: 2
! forme de txtlines: (/ 2, 9 /)
...
END PROGRAM plenty_of_arrays
```

Tableaux : opérations

Arrays: operations

- en FORTRAN 77: codage manuel des opérations entre éléménts, élément par élément
- en Fortran 90: opérations possibles sur des tableaux entiers
  - `A = B + 4` : addition de 4 à chaque élément de `B`
  - `A = B + C` : addition élément par élément de `B` et `C`
  - `A = B * C` : multiplication élément par élément
  - condition: `A`, `B` (et `C`) de même forme
- opérations matricielles
  - addition: `A = B + C`
  - multiplication: `A = MATMUL(B,C)`
  - condition: dimensions compatibles

Tableaux : sections

Arrays: sections

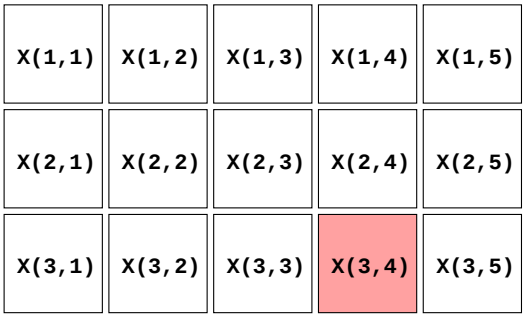
Exemples d'extraction de sections de tableaux

```
DOUBLE PRECISION, DIMENSION(-1:1,3,0:7) :: x
```

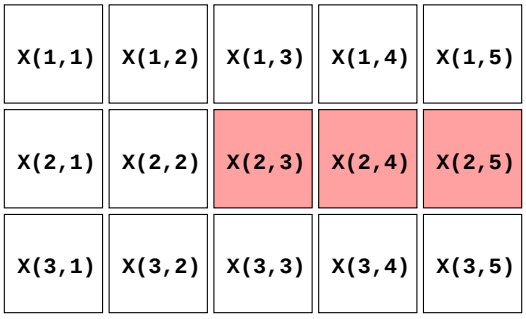
- élément individuel: par ses indices (ex.: `x(0,1,6)`)
- sections complexes par indices et/ou triplets d'indices de la forme [*indice_inf*] : [*indice_sup*] [: *pas*], où, par défaut, *indice_inf* et *indice_sup* sont égaux aux bornes inférieure et supérieure déclarées et *pas* (*stride*) égal à 1
  - `x(:,2,:1) = (/ x(-1,2,0), x(0,2,0), x(1,2,0), &`
    `x(-1,2,1), x(0,2,1), x(1,2,1) /)`
  - `x(1,2,1:7:3) = (/ x(1,2,1),x(1,2,4),x(1,2,7) /)`
  - `x(-1,2:3,::3) = (/ x(-1,2,0), x(-1,3,0), &`
    `x(-1,2,3), x(-1,3,3), &`
    `x(-1,2,6), x(-1,3,6) /)`

Tableaux : sections

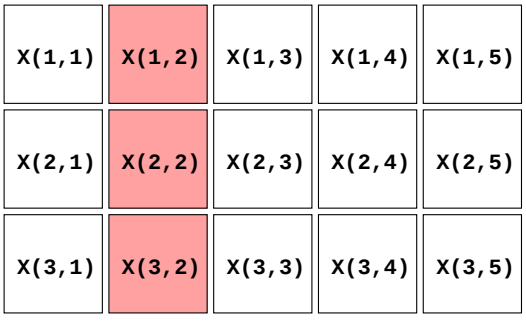
Arrays: sections



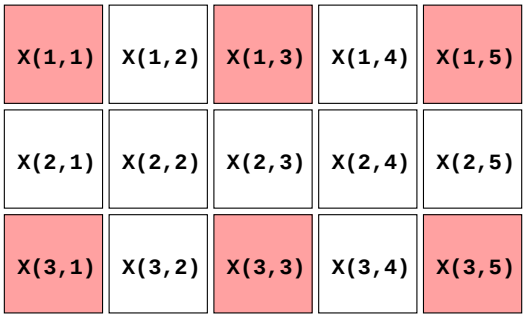
`X(3,4)`



`X(2,3:5)` or `X(2,3:)`



`X(:,2)` or `X(1:3,2)`



`X(1:3:2,1:5:2)` or `X(::2,::2)`

Tableaux : sections

Arrays: sections

Exemples d'extraction de sections de tableaux

```
DOUBLE PRECISION, DIMENSION(-1:1,3,0:7) :: x
```

- par vecteur d'indices :
  ```
  idx = (/ 1, 4, 5, 6 /)
  subx = x(0, 2, idx)
  ```
- possibilité de répéter des indices dans `idx`
- `idx` peut aussi apparaître à la gauche d'une attribution — ne peut pas contenir d'indices répétés dans ce cas

Tableaux : fonctions enquêtrices

Arrays: inquiry functions

- `SHAPE(`*`source`* `[,` *`dim`*`])` ⇒ forme de la *`source`* si elle est un tableau (résultat: tableau `INTEGER` de rang 1), 0 si la *`source`* est un scalaire;
- `SIZE(`*`tableau`* `[,` *`dim`*`])` ⇒ nombre d'éléments du *`tableau`* entier ou le long de la dimension *`dim`* si spécifiée (résultat: scalaire `INTEGER`);
- `LBOUND(`*`tableau`* `[,` *`dim`*`])` ⇒ bornes inférieures des indices du *`tableau`* (résultat: tableau `INTEGER` de rang 1), ou la borne inférieure de la dimension *`dim`* (résultat: scalaire `INTEGER`);
- `UBOUND(`*`tableau`* `[,` *`dim`*`])` ⇒ analogue à `LBOUND`, pour les bornes supérieures.

Tableaux : opérations

Arrays: operations

- `SUM(`*`tableau`* `[,` *`dim`*`])` ⇒ somme des éléments du *`tableau`* entier (résultat scalaire), ou le long de la dimension *`dim`* si spécifiée (résultat: tableau dont le rang est le rang du *`tableau`* diminué de 1, et la forme celle du *`tableau`*, sans la dimension *`dim`*);
- `PRODUCT(`*`tableau`* `[,` *`dim`*`])` ⇒ analogue à `SUM`, pour le produit des éléments;
- `MAXVAL(`*`tableau`* `[,` *`dim`*`])` ⇒ valeur maximale du *`tableau`* (résultat scalaire) ou le long de la dimension *`dim`* si spécifiée (résultat: tableau dont le rang est le rang du *`tableau`* diminué de 1, et la forme celle du *`tableau`*, sans la dimension *`dim`*);
- `MINVAL(`*`tableau`* `[,` *`dim`*`])` ⇒ analogue à `MAXVAL`, pour la valeur minimale.

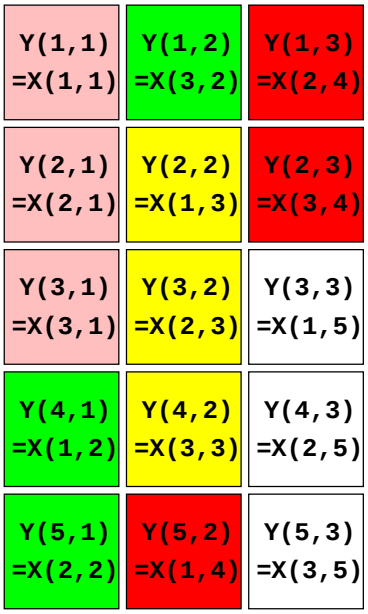
Tableaux : reformes

Arrays: reshaping

Y(1,1) =X(1,1)	Y(4,1) =X(1,2)	Y(2,2) =X(1,3)	Y(5,2) =X(1,4)	Y(3,3) =X(1,5)
Y(2,1) =X(2,1)	Y(5,1) =X(2,2)	Y(3,2) =X(2,3)	Y(1,3) =X(2,4)	Y(4,3) =X(2,5)
Y(3,1) =X(3,1)	Y(1,2) =X(3,2)	Y(4,2) =X(3,3)	Y(2,3) =X(3,4)	Y(5,3) =X(3,5)

```
Y = RESHAPE(X, (/ 5, 3 /) )
```

Tableaux : reformes
Arrays: reshaping




```
Y = RESHAPE(X, (/ 5, 3 /) )
```

Constructeurs de tableau
Array constructors

Forme générique d'un constructeur de tableau

`(/` *liste_de_valeurs* `/)`

où *liste_de_valeurs* est donnée par

- une succession de valeurs, séparées par des virgules

  `1.0, 2.4, 5.6, −1.5`

- une succession de sections d'autres tableaux déjà initialisés auparavant, séparées par des virgules
- une liste à boucle implicite

  `(`*expression*`(i) , i = -5, 6, 2)`

- une liste à boucle implicite de listes à boucles implicites . . .

  `( (`*expression*`(i, j, ...), j = ...), i=...)`

- des combinaisons des précédentes

Tableaux : utilisation de RESHAPE à l'initialisation

Arrays: use of RESHAPE at initialisation

Initialisation de tableaux à la déclaration

```
PROGRAM more_arrays
IMPLICIT NONE
INTEGER, DIMENSION(5,7) :: iarr1 = -1 ! init. tous les éléments à -1
! équivalent à INTEGER, DIMENSION(1:5,1:7) :: iarr1
! ou encore INTEGER :: iarr1(5,7)
! rang de iarr1: 2
! forme de iarr1: (/ 5, 7 /)

INTEGER, DIMENSION(5) :: iarr2 = (/ 2, 4, 8, 16, 32 /)
! équivalent à
! INTEGER :: i
! INTEGER, DIMENSION(5) :: iarr2 = (/ (2**i, i=1,5) /)

! iarr3 tableau de forme (/ 3, 5 /), chaque élément
! initialisé à la valeur de sa position en mémoire
INTEGER, DIMENSION(3,5) :: iarr3  &
        = RESHAPE( (/ (i, i=1,3*5) /), (/ 3, 5 /))
...
END PROGRAM more_arrays
```

Tableaux à dimensionnement différé

Allocatable arrays

Déclaration

type, ALLOCATABLE, DIMENSION(:[,:...]) :: *nom_tabl*

- Tableau nommé, dont uniquement le type et le rang sont déclarés
- Rang déclaré à l'aide du nombre approprié de : dans la liste d'indices de l'attribut DIMENSION
- Attribut ALLOCATABLE
- Après déclaration: statut *non-alloué*
- Statut d'allocation peut être obtenu à l'aide la fonction logique ALLOCATED(*nom_tabl*)

Tableaux à dimensionnement différé
Allocatable arrays

Allocation

`ALLOCATE(`*`nom_tabl`*`(`*`dim_1`*`, `*`dim_2`*`, ...) &`
`[, STAT=`*`stat_var`*`])`

- Forme requise pour *`nom_tabl`* est `(/`*`dim_1`*`, `*`dim_2`*`, .../)`
- Argument optionnel `STAT`, avec la variable *`stat_var`* de type `INTEGER` permet un contrôle d'erreur
- Au retour de la commande, *`stat_var`* sera
  - définie à 0 si l'allocation s'est produite sans erreur
  - définie à une valeur positive, dépendant du compilateur, sinon
- Statut de la variable *`nom_tabl`* passe à *alloué* (`ALLOCATED(`*`nom_tabl`*`)` retournera `.TRUE.`)
- Programme s'arrête en cas d'erreur si l'argument optionnel `STAT` n'est pas spécifié

Tableaux à dimensionnement différé
Allocatable arrays

Méthodes numériques et éléments de programmation

Guy Munhoven

Désallocation/Libération

`DEALLOCATE(`*`nom_tabl`*` [, STAT=`*`stat_var`*`])`

- L'espace mémoire réservé auparavant par `ALLOCATE` est libéré, le statut de la variable *`nom_tabl`* passe à *non allouée* (`ALLOCATED(`*`nom_tabl`*`)` retournera `.FALSE.`)
- Si *`stat_var`* est indiquée, elle sera, au retour de la commande,
  - définie à 0 si l'allocation s'est produite sans erreur
  - définie à une valeur positive, dépendant du compilateur, sinon.
- Programme s'arrête en cas d'erreur si l'argument optionnel `STAT` n'est pas spécifié

Ce qui n'a pas été couvert
Subjects not covered

- Pointeurs
- Structures et types dérivés
- `INTERFACE`s de sous-routines et fonctions
  - déclaration des caractéristiques des sous-routines au sein du programme appelant
  - obligatoires pour fonctions à résultat sous forme de tableau
  - obligatoires pour sous-programmes avec argument optionnels
  - générées automatiquement pour des sous-programmes dans des modules
  - permettent surcharge (extension, redéfinition) d'opérateurs
- Attributs `PUBLIC` et `PRIVATE` de variables et sous-programmes d'un `MODULE`