Initiation à Fortran

Mehmet Ersoy[1,2]

[1]SEATECH–Ecole d'ingénieurs de l'Université de Toulon,
[2]IMATH–Institut de Mathématiques de Toulon

Sommaire

Sommaire

Quelques règles de bases

- Programme général
  - la “recompilation” doit devenir exceptionnelle
  - toutes modifications doivent être facilitée

Quelques règles de bases

- Programme général
  - la “recompilation” doit devenir exceptionnelle
  - toutes modifications doivent être facilitée
- Programme structuré
  - structuré, commenté et aéré
  - utilisation de nom de variable significatif
  - rendre le programme modifiable “aisément” par d’autres

Quelques règles de bases

- Programme général
  - la “recompilation” doit devenir exceptionnelle
  - toutes modifications doivent être facilitée
- Programme structuré
  - structuré, commenté et aéré
  - utilisation de nom de variable significatif
  - rendre le programme modifiable “aisément” par d’autres
- Programme “défensif”
  - tests sur les données d’entrée
  - tests d’arrêts pour d’éventuelles erreurs avec affichage de message
  - tests si débogage (avec impressions multi-niveaux)

Quelques règles de bases

- Programme général
  - la “recompilation” doit devenir exceptionnelle
  - toutes modifications doivent être facilitée
- Programme structuré
  - structuré, commenté et aéré
  - utilisation de nom de variable significatif
  - rendre le programme modifiable “aisément” par d’autres
- Programme “défensif”
  - tests sur les données d’entrée
  - tests d’arrêts pour d’éventuelles erreurs avec affichage de message
  - tests si débogage (avec impressions multi-niveaux)
- Programme économique
  - le moins de temps de calculs possible
  - le moins d’espace disque possible

Quelques règles de bases

- Programme général
  - la “recompilation” doit devenir exceptionnelle
  - toutes modifications doivent être facilitée
- Programme structuré
  - structuré, commenté et aéré
  - utilisation de nom de variable significatif
  - rendre le programme modifiable “aisément” par d’autres
- Programme “défensif”
  - tests sur les données d’entrée
  - tests d’arrêts pour d’éventuelles erreurs avec affichage de message
  - tests si débogage (avec impressions multi-niveaux)
- Programme économique
  - le moins de temps de calculs possible
  - le moins d’espace disque possible
- Programme transportable
  - portable
  - documenté
  - post-traitement soigné

Principe du Fortran

FORTRAN = langage compilé

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)
  - est traduit par un programme annexe = compilateur (édition de lien)

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)
  - est traduit par un programme annexe = compilateur (édition de lien)
  - le compilateur génère un fichier autonome = exécutable (création d’un exécutable)

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)
  - est traduit par un programme annexe = compilateur (édition de lien)
  - le compilateur génère un fichier autonome = exécutable (création d’un exécutable)
- Avantage par rapport à un langage interprété (e.g. MATLAB)

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)
  - est traduit par un programme annexe = compilateur (édition de lien)
  - le compilateur génère un fichier autonome = exécutable (création d’un exécutable)
- Avantage par rapport à un langage interprété (e.g. MATLAB)
  - nulle besoin de programme annexe pour s’exécuter
  - rapidité d’exécution
  - sécurité du code exécutable

FORTRAN = langage compilé

i.e.

- Un programme “compilé” (édition)
  - est traduit par un programme annexe = compilateur (édition de lien)
  - le compilateur génère un fichier autonome = exécutable (création d’un exécutable)
- Avantage par rapport à un langage interprété (e.g. MATLAB)
  - nulle besoin de programme annexe pour s’exécuter
  - rapidité d’exécution
  - sécurité du code exécutable
  - TOUTEFOIS chaque modification du programme source requiert une recompilation

FORTRAN = (Mathematical) FORmula TRANslating

FORTRAN = (Mathematical) FORmula TRANslating

- Historique

1954	: début de projet (IBM)
1958	: FORTRAN II
1966	: FORTRAN 66
1977	: FORTRAN 77 $\star$
1991/92	: FORTRAN 90
...	: plus d'informations sur le net!!!

FORTRAN = (Mathematical) FORmula TRANslating

- Historique
  - 1954 : début de projet (IBM)
  - 1958 : FORTRAN II
  - 1966 : FORTRAN 66
  - 1977 : FORTRAN 77 $\star$
  - 1991/92 : FORTRAN 90
  - ... : plus d'informations sur le net!!!
- Bibliographie (Fortran 77)
  - Metcalf, Michael. Effective FORTRAN 77. Oxford University Press, 1985.
  - Nyhoff, Larry and Sanford, Leestma. FORTRAN 77 and Numerical Methods for Engineers and Scientists. Prentice-Hall, Inc. New Jersey, 1995.
  - ...

FORTRAN = (Mathematical) FORmula TRANslating

- Historique
  - 1954 : début de projet (IBM)
  - 1958 : FORTRAN II
  - 1966 : FORTRAN 66
  - 1977 : FORTRAN 77 $\star$
  - 1991/92 : FORTRAN 90
  - ... : plus d'informations sur le net!!!
- Bibliographie (Fortran 77)
  - Metcalf, Michael. Effective FORTRAN 77. Oxford University Press, 1985.
  - Nyhoff, Larry and Sanford, Leestma. FORTRAN 77 and Numerical Methods for Engineers and Scientists. Prentice-Hall, Inc. New Jersey, 1995.
  - ...
- Documentations (tutoriels)
  - `http://www.stanford.edu/class/me200c/tutorial_77/` et `http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-5.htmlsh-5.5`
  - ...

Installation du compilateur

- Editeur de texte notepad++ sous windows
- sous windows (gfortran) via mingw ou cygwin (émulateurs unix)

Installation du compilateur

- Editeur de texte (notepad++ sous windows, gedit sous Linux

- sous windows (gfortran) via mingw ou cygwin (émulateurs unix)
- sous linux (gfortran) via l'installateur de paquet apt

Installation du compilateur

- Editeur de texte (notepad++ sous windows, gedit sous Linux, Fraise sous mac)
- sous windows (gfortran) via mingw ou cygwin (émulateurs unix)
- sous linux (gfortran) via l'installateur de paquet apt
- sous IOS via l'installateur de paquet fink ou suivre certains tuto sur le net

Sommaire

Sommaire

- Caractères
  - insensible à la casse (ToTo = TOTO)
  - blanc = aucun rôle
  - nom des variables = 6 caractères max

- Caractères
  - insensible à la casse (ToTo = TOTO)
  - blanc = aucun rôle
  - nom des variables = 6 caractères max
- Structure des lignes

colonne 1 à 5	=	champ étiquette
colonne 6 (muni de &)	=	césure de l'instruction (ligne N-1)
colonne de 7 à 72	=	champs instruction

- Caractères
  - insensible à la casse (ToTo = TOTO)
  - blanc = aucun rôle
  - nom des variables = 6 caractères max
- Structure des lignes

colonne 1 à 5	=	champ étiquette
colonne 6 (muni de &)	=	césure de l'instruction (ligne N-1)
colonne de 7 à 72	=	champs instruction

- Commentaire muni du caractère C en colonne 1

Un exemple : édition dans un fichier monprog.f

```
c234567
C-----------------------------------------
C CALCUL DU PRODUIT SCALAIRE (U,V) DANS R^N
C-----------------------------------------
       PROGRAM PRODSCAL
       IMPLICIT NONE
C declaration des variables
       INTEGER I,N
       DOUBLE PRECISION U(1000),V(1000)
      &  RES
C initialisation
       N = 5
       DO I=1,N
         U(I)=DBLE(I)
         V(I)=DBLE(I**2)
       ENDDO
C corps du programme
       RES = 0.0D0
       DO I=1,N
         RES=RES+U(I)*V(I)
       ENDDO
       WRITE(*,100) RES
100    FORMAT(1X,F5.3)
       END PROGRAM
```

. . . et compilation du fichier monprog.f

Dans un terminal unix, on compile
soit

```
gfortran -c monprog.f
gfortran -o prod monprog.f
```

et on exécute avec la commande

```
./prod
```

soit

```
gfortran monprog.f
```

et on exécute avec la commande

```
./a.out
```

soit

```
gfortran -o prod monprog.f && ./prod
```

Remarque: le nom du fichier, le nom du programme et le nom du code exécutable ne sont pas nécessairement identique.

Structure générale d'un code Fortran 77

```
c234567
C-----------------------------------------
C STRUCTURE D'UN CODE QUELCONQUE
C-----------------------------------------
       PROGRAM NOM_DU_PROGRAMME
C declaration des variables

C corps du programme =
C suite d'instructions (BOUCLE,TEST BOOLEEN)
C structurées (FONCTIONS OU SUBROUTINE)

       END PROGRAM NOM_DU_PROGRAMME
C ou encore END tout cout simplement
```

Sommaire

Variables & constantes

- Les variables, constantes et paramètres sont des noms permettent de manipuler des données en mémoire:

Entier	:	INTEGER
Réel	:	REAL, DOUBLE PRECISION
Complexe	:	COMPLEXE
Chaîne de caractères	:	CHARACTER, CHARACTER*20
Logique	:	LOGICAL

- Déclaration des variables: deux stratégies
  1. On "déclare tout"

```
IMPLICIT NONE
INTEGER I
DOUBLE PRECISION U(10),V(10)
Character fichier*20
```

  2. "Par défaut" (par défaut, toutes les variables dont le nom commence par I,J,K,L,M,N sont des variables entières et les autres des réels simple précision.)

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
Character fichier*20
```

Variables & constantes

- Les variables, constantes et paramètres sont des noms permettent de manipuler des données en mémoire:

Entier	:	INTEGER
Réel	:	REAL, DOUBLE PRECISION
Complexe	:	COMPLEXE
Chaîne de caractères	:	CHARACTER, CHARACTER*20
Logique	:	LOGICAL

- Déclaration des variables: deux stratégies
  1. On "déclare tout"

```
IMPLICIT NONE
INTEGER I
DOUBLE PRECISION U(10),V(10)
Character fichier*20
```

  2. "Par défaut" (par défaut, toutes les variables dont le nom commence par I,J,K,L,M,N sont des variables entières et les autres des réels simple précision.)

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
Character fichier*20
```

Tous les types précédents peuvent être structurés en tableaux en attribuant une dimension pour allouer l'espace en mémoire. Ainsi, dans l'exemple,

```
...
       DOUBLE PRECISION U(1000),V(1000)
      &  RES
...
       N = 5
       DO I=1,N
         U(I)=DBLE(I)
         V(I)=DBLE(I**2)
       ENDDO
...
       END PROGRAM
```

les vecteurs U et V sont de dimension 1000 (en mémoire) mais seulement les 5 premières composantes du tableaux sont utilisées.

les tableaux (suite)

- Déclaration

```
DOUBLE PRECISION U(1000)

DOUBLE PRECISION U
DIMENSION 1000

DOUBLE PRECISION U(0:1000)

PARAMETER (N=100)
PARAMETER (M=10)
DOUBLE PRECISION A,U
INTEGER TAB
DIMENSION A(N,M),I(M),U(N)
```

- Attention, le stockage en mémoire est par colonne descendante.

Si

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix}$$

alors en mémoire il est sous la forme $A = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{12} \\ a_{22} \\ a_{33} \end{pmatrix}$

Sommaire

Les opérateurs

- Opérateur unitaire: affectation (e.g. `n=10`)
- Opérateur arithmétiques:
  - exponentiation (e.g. `2.0D0**2.0D0`)
  - Multiplication `*`, Division `/`, Addition +, Soustraction −
- Opérateur relationnel:

  .LT. Lower Than
  .GT. Greater Than
  .LE. Lower or Equal
  .GE. Greater or Equal
  .EQ. Equal
  .NE. Non Equal
- Opérateur logique:

  .NOT. Négation logique
  .AND. Et logique
  .OR. ou logique

```
c234567
      DO ETQ I = K,L[,M]
         Instructions
ETQ   CONTINUE
```

ou

```
c234567
      DO  I = K,L[,M]
         Instructions
      CONTINUE
```

avec

I	:	compteur de boucle
K	:	première valeurs de I
L	:	dernière valeurs de I
M	:	pas
ETQ	:	étiquette pointant sur une instruction qui ferme la boucle.

```
c234567
       DOWHILE (TEST)
          Instructions
       ENDDO
```

qui signifie
tant que la condition TEST est vraie, on effectue les instructions
Instructions

les structures itératives et de contrôle : contrôle

```
c234567
      IF(test) Instruction

      IF(test) THEN
         suite d'instructions
      ENDIF

      IF(test) THEN
         suite d'instructions 1
      ELSE
         suite d'instructions 2
      ENDIF

      IF(test 1) THEN
         suite d'instructions 1
      ELSEIF(test 2) THEN
         suite d'instructions 2
      ELSE
         suite d'instructions 3
      ENDIF
```

Écrire des programmes pour

1. calculer $\Sigma_{i=0}^{1}00i^3$
2. calculer la valeur de $y = \begin{cases} 1 & \text{si} \quad x > 0 \\ -1 & \text{si} \quad x \leqslant 0 \end{cases}$
3. calculer la valeur de $y = \begin{cases} 0 & \text{si} \quad x < 0 \\ \sqrt{x} & \text{si} \quad x \in [0,1) \\ 1 & \text{si} \quad x > 1 \end{cases}$
4. calculer $\Sigma_{i=1,impaire}^{1}00i$
5. soient $N \in \mathbb{N}^+$, $a < b$, $(a,b) \in \mathbb{R}^2$ et $h = (b-a)/N$, créer le vecteur à incrément de pas constant h.
6. calcul approximative de $\sqrt{2}$ par l'algorithme $x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}$.
7. calcul approximative de la solution de l'équation $x - e^{-x}$ par la méthode de Newton.

Sommaire

entrées/sorties : interaction avec l'utilisateur

- lecture au clavier de la valeur d'une variable, quel que soit son type

```
READ*,variable1,variable2,\ldots,,variableN
```

- écriture à l'écran d'un message et/ou de la valeur d'une variable ou d'un tableau

```
DOUBLE PRECISION X,U(1000)
INTEGER N
CHARACTER PAYS*15

...

PRINT*, "hello"
WRITE(*,*) "hello"
C PRINT*, =  WRITE(*,*)
PRINT*, "la valeur de x est ",X
PRINT*, (U(I),I=1,N)


PRINT*, 'age?'
READ*, N
PRINT,* ' vous avez ' , N , ' ans '
PRINT*, 'pays?'
read * , PAYS
PRINT*, 'vous etes de', PAYS
```

Malheureusement, les messages d'erreurs ne sont pas toujours très clairs

```
Unclassifiable statement at ...
```

Cependant,

- le compilateur détecte certaines erreurs de syntaxe, conformité de types, de dimensions, . . .
- il affiche un message d'erreur indiquant le numéro de ligne concerné
- une erreur provoque une cascade d'erreur (e.g. une variable mal déclarée)
- la liste est donc bien souvent très longue: il faut commencer par la première

les mots clefs sont `READ(OU,COMMENT) QUOI` et `WRITE(OU,COMMENT) QUOI` pour lire/écrire les données `QUOI` dans le fichier étiqueté par `OU` (sauf si *) en utilisant le format d'étiquette `COMMENT`. Par exemple,

```
c234567
      X=2.0D0
      WRITE(*,*) 'valeur de X :',X

      WRITE(*,100) X
100   FORMAT('valeur de X :',e12.5)

      WRITE(*,'(f5.2)'),X
      WRITE(12,'(f5.2)'),x

      READ(*,*) a,b,c
      READ(*,*) (a(i),i=1,5)

      WRITE(12,10),A(2)
10    FORMAT('valeur de A(2) :',I2)
```

les mots clefs sont

```
OPEN(UNIT=unit, FILE=file, STATUS=status)
```

et

```
CLOSE(UNIT=unit, STATUS=status)
```

pour ouvrir/créer un fichier (qui portera l'identifiant entier `unit`) de nom `file`, de statut `status`

avec

`unit`	:	numéro d'unité logique
`file`	:	nom du fichier à ouvrir
`status`	:	'`OLD`','`NEW`' en mode `OPEN` ou '`KEEP`'(par défaut),'`DELETE`' en mode `CLOSE`

Quelques exercices simples

1. Reprendre les programmes des exercices précédents et proposer un programme qui permet d'interagir avec l'utilisateur (exemple pour prodscal, demander à l'utilisateur la valeur de N).
2. Soit $A \in M_N(\mathbb{R})$. Proposer une méthode pour afficher la matrice à l'écran.

Sommaire

Sommaire

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)

  - Compilation : `gfortran -o code prog.f && ./code`

  - Compilation :
    `gfortran -o code prog.f fcts.f subrout.f && ./code`
    (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
  - Compilation :
    `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ..`
    (si les fonctions et subroutines sont toutes dans des fichiers séparés)

.

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)
  - les fonctions
  - les subroutines (procédures)

    - Compilation : `gfortran -o code prog.f && ./code`

    - Compilation :
      `gfortran -o code prog.f fcts.f subrout.f && ./code`
      (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
    - Compilation :
      `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ...`
      (si les fonctions et subroutines sont toutes dans des fichiers séparés)

.

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)
  - les fonctions
  - les subroutines (procédures)
- Un code fortran peut être édité
  - soit sous la forme "programme" suivi de(s) fonction(s) et/ou subroutine(s) dites internes (dans un seul et unique fichier prog.f)
    - Compilation : `gfortran -o code prog.f && ./code`
    - Compilation :
      `gfortran -o code prog.f fcts.f subrout.f && ./code`
      (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
    - Compilation :
      `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ...`
      (si les fonctions et subroutines sont toutes dans des fichiers séparés)

.

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)
  - les fonctions
  - les subroutines (procédures)
- Un code fortran peut être édité
  - soit sous la forme "programme" suivi de(s) fonction(s) et/ou subroutine(s) dites internes (dans un seul et unique fichier prog.f)
    - Compilation : `gfortran -o code prog.f && ./code`
  - soit dans plusieurs fichiers séparés, on parle alors de sous-programmes externes.
    - Compilation :
      `gfortran -o code prog.f fcts.f subrout.f && ./code`
      (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
    - Compilation :
      `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ...`
      (si les fonctions et subroutines sont toutes dans des fichiers séparés)

.

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)
  - les fonctions
  - les subroutines (procédures)
- Un code fortran peut être édité
  - soit sous la forme "programme" suivi de(s) fonction(s) et/ou subroutine(s) dites internes (dans un seul et unique fichier prog.f)
    - Compilation : `gfortran -o code prog.f && ./code`
  - soit dans plusieurs fichiers séparés, on parle alors de sous-programmes externes.
    - Compilation :
      `gfortran -o code prog.f fcts.f subrout.f && ./code`
      (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
    - Compilation :
      `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ...`
      (si les fonctions et subroutines sont toutes dans des fichiers séparés)

.

- Dès lors qu'un bloc d'instructions est utilisé plusieurs fois $\longrightarrow$ on "factorise" (appel à de sous programmes)
  - les fonctions
  - les subroutines (procédures)
- Un code fortran peut être édité
  - soit sous la forme "programme" suivi de(s) fonction(s) et/ou subroutine(s) dites internes (dans un seul et unique fichier prog.f)
    - Compilation : `gfortran -o code prog.f && ./code`
  - soit dans plusieurs fichiers séparés, on parle alors de sous-programmes externes.
    - Compilation :
      `gfortran -o code prog.f fcts.f subrout.f && ./code`
      (si les fonctions sont toutes contenues dans un fichier fcts.f et idem pour les subroutines)
    - Compilation :
      `gfortran -o code prog.f fct1.f ... fctN.f subrout1.f ..`
      (si les fonctions et subroutines sont toutes dans des fichiers séparés)

Remarques: il est préférable de distinguer les sous-programmes internes des sous-programmes externes. Plus il y a des fichiers à compiler et plus c'est compliqué $\longrightarrow$ "Makefile" script redoutable.

Sommaire

la règle:

- arguments entrée obligatoire
- renvoie un seul résultat qui porte le nom de la fonction

```
c234567
      FUNCTION nom_fonction(...)
      TYPE nom_fonction
      ....
      nom_fonction = ...
      END FUNCTION nom_fonction
```

les fonctions : un exemple

- Calcul de la partie positive d'un réel:

```
c234567
      FUNCTION POS(X)
      IMPLICIT NONE
      DOUBLE PRECISION POS,X
      POS = MAX(0.0D0,X)
      END FUNCTION nom_fonction
```

- Appel de cette fonction dans un programme

```
c234567
      PROGRAM BIDON
      IMPLICIT NONE
C-------------------------------
      DOUBLE PRECISION POS
      EXTERNAL POS
      DOUBLE PRECISION X,Y
C-------------------------------
      WRITE(*,*) 'ENTREZ X'
      READ(*,*) X
      Y = POS(X)
      WRITE(*,*) 'LA PARTIE POSITIVE DE X EST ',X
      END PROGRAM BIDON
```

Remarques:

- les variables X et Y du programme `BIDON` sont des variables locales.
- la variable X de la fonction `POS` est une variable locale.

Sommaire

la règle:

- arguments entrée et/ou entrée-sortie et/ou sortie
- renvoie un ou plusieurs résultats

```
SUBROUTINE nom_subroutine(...)
....
 = ...
END SUBROUTINE nom_subroutine
```

les subroutines : un exemple

- Calcul une approximation de la racine caré

```
      SUBROUTINE RACINE(U,N)
      IMPLICIT NONE
      DOUBLE PRECISION U
      INTEGER I,N
      DO I=1,N
        X =  X/2+1/X
      ENDDO
      END SUBROUTINE RACINE
```

- Appel de cette subroutine dans un programme

```
c234567
      PROGRAM BIDON
      IMPLICIT NONE
C-------------------------------
      DOUBLE PRECISION X
      INTEGER N
C-------------------------------
      WRITE(*,*) 'ENTREZ LE NOMBRE D''ITERATIONS SOUHAITE'
      READ(*,*) N
      WRITE(*,*) 'INITIALISATION DE L'ALGORITHME'
      READ(*,*) X
      CALL RACINE(X,N)
      WRITE(*,*) 'LA VALEUR APPROCHEE DE SQRT(2) EST ',X
      END PROGRAM BIDON
```

Remarques:

- les variables X et N du programme `BIDON` sont des variables locales.
- les variables N (entrée) et U(entrée-sortie) de `RACINE` des variables locales.

Sommaire

Remarques

- Pour passer un vecteur dans un sous-programme, on n'a pas besoin de déclarer la taille, il suffit d'écrire dans la déclaration *. par exemple : `DOUBLE PRECISION V(*))`
- Pour une matrice, on peut soit
  - passer en argument *A* et la déclarer

```
PROGRAM BIDON
PARAMETER (N=10)
DOUBLE PRECISION A(N,N),Y(N)
A(i,j)=
Y=FCT(A,N...)
END PROGRAM BIDON

FUNCTION FCT(A,N,...)
DOUBLE PRECISION A(N,N),Y(N)
...
Y=FCT1(A,N...)
END FUNCTION BIDON
```

Remarques

- sinon il faut la premère taille de déclaration de la matrice

```
PROGRAM BIDON
PARAMETER (NMAX=1000)
DOUBLE PRECISION A(NMAX*NMAX),Y(NMAX)
N = ...
A(I+(J-1)*N) = ...
Y=FCT(A,N...)
END PROGRAM BIDON

FUNCTION FCT(A,N,...)
DOUBLE PRECISION A(N,*),Y(*)
A(I,J)
Y=FCT1(A,N...)
END FUNCTION BIDON
```

Remarque importante

En raison du stockage d'une matrice sous forme uni-dimensionnelle par colonne descendante les deux extraits de code suivants ne sont pas équivalents:

```
DO I=1,N
  DO J=1,M
    A(I,J)=I*J
  ENDDO
ENDDO

DO J=1,M
  DO I=1,N
    A(I,J)=I*J
  ENDDO
ENDDO
```

Remarque importante

En raison du stockage d'une matrice sous forme uni-dimensionnelle par colonne descendante les deux extraits de code suivants ne sont pas équivalents:

```
DO I=1,N
  DO J=1,M
    A(I,J)=I*J
  ENDDO
ENDDO

DO J=1,M
  DO I=1,N
    A(I,J)=I*J
  ENDDO
ENDDO
```

le premier bloc est beaucoup plus coûteux en temps!!!

En vous basant sur la notion de sous-programme, écrire les programmes permettant de

1. tracer une fonction $x \in [a, b] \to f(x) \in \mathbb{R}$ (à l'aide de gnuplot ou matlab, ...)
2. résoudre l'équation $f(x) = 0$ avec l'agorithme de Newton et Quasi-Newton.
3. d'approcher l'intégrale $I(f) = \int_0^1 f(x)\,dx$ par une formule de quadrature $I_n(f)$ (rectangle, point milieu et trapèze).
4. résoudre une équation différentielle ordinaire $u'(t) = F(t, u(t))$.
5. résoudre un système linéaire $Ax = b$ où A est une matrice tridiagonale de taille n. On suppose que A admet une décomposition LU.
6. Résoudre l'équation différentielle $-\frac{d^2}{dx^2}u(x) = f(x),\ x \in (0, 1)$ ordinaire munit des conditions aux limites de Dirichlet homogène.

Sommaire

On a vu que les variables utilisées jusqu'à présent étaient locales. Pour rendre, une variable (ou paramètre) locale en gloable il suffit d'utiliser un fichier externe qu'on appelle common. L'exemple suivant permet de montrer comment un tel fichier permet de réaliser ceci.

Sommaire

Sommaire

Code fortran

Un exemple de code FORTRAN 77 avec MAKEFILE ET COMMON est disponible à l'adresse `http://ersoy.univ-tln.fr/documents/teaching/CodeTP1.zip`

Sommaire

Que doit-on répondre à

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Que doit-on répondre à

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Alors . . .

- la plupart des codes industriels sont développés en Fortran (son nom est quand même révélateur de son utilisation, non?)

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Alors . . .

- la plupart des codes industriels sont développés en Fortran (son nom est quand même révélateur de son utilisation, non?)
- Fortran est beaucoup plus simple d'utilisation physiques et conceptuels que le C++.

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Alors . . .

- la plupart des codes industriels sont développés en Fortran (son nom est quand même révélateur de son utilisation, non?)
- Fortran est beaucoup plus simple d'utilisation physiques et conceptuels que le C++.
- la programmation orientée objet du C++ sont maintenant utilisables en fortran (depuis fortran 90).

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Alors . . .

- la plupart des codes industriels sont développés en Fortran (son nom est quand même révélateur de son utilisation, non?)
- Fortran est beaucoup plus simple d'utilisation physiques et conceptuels que le C++.
- la programmation orientée objet du C++ sont maintenant utilisables en fortran (depuis fortran 90).
- en développement constant . . .

Vous programmez encore avec Fortran, . . . ce langage est dépassé et je te conseille de programmer en C++

Alors . . .

- la plupart des codes industriels sont développés en Fortran (son nom est quand même révélateur de son utilisation, non?)
- Fortran est beaucoup plus simple d'utilisation physiques et conceptuels que le C++.
- la programmation orientée objet du C++ sont maintenant utilisables en fortran (depuis fortran 90).
- en développement constant . . .

Pour terminer,

le seul moyen d'apprendre à programmer c'est de programmer tout(e) seul(e) (c'est un peu comme apprendre à faire du vélo?)! Donc, usez et abusez de votre temps en salle machine, c'est la clef de la succès!