Université Paul Sabatier **Année 2007-2008**

Licence Mathématiques et Applications

3e année : Mathématiques Pour l'Ingénierie

Programmation - TP de Fortran 90

---

1 Exercices de base

Exercice 1. Parmi les lignes suivantes, dites lesquelles respectent les conventions de format d'instructions Fortran 90. Dites lesquelles contiennent un commentaire. Dites lesquelles sont des lignes qui commentent une instruction, et lesquelles sont des lignes suites.

```
X=Y
A=B+C ! addition
MOT='chaine'
A=1.0 ; B=2.0
A=15. ! Initialise A ; B=22. ! et B
PHRASE_1='nous␣sommes␣aujourd'hui &
& 'le␣13␣novembre'
PHRASE_2='qui␣a␣gagne␣l␣election
presidentielle␣americaine␣?'
C(10)=4.5
```

Exercice 2. Classez les constante littérales suivantes selon les cinq types de variables de base vues en cours. Signalez les constantes incorrectement écrites.

```
−43          4.39              0.0001E+20
4 9          (1.E3,2)          '(4.3E9,6.2)'
E5           1.2_pr            z10
'mot'        1.9−4             'Laurel␣&␣Hardy'
(0.,1.)      'c'est faux'␣␣␣␣␣.true.
```

Exercice 3. Parmi les mots suivants, dites lesquels sont des noms valides en F90 :

```
nom     quotient     a18c3     %stop!%
no_go   nom32        123       %no−go%
brule   long_nom
```

Exercice 4. Le programme suivant calcule et affiche les racines d'une équation quadratique :

```
program racines_p2

  implicit none

  integer :: a, b, c, D
  real :: Partie_re, Partie_im

  print*, 'donner les valeurs entières de a,b,c'
  read*,a,b,c

  if (a/=0) then
     !--- discriminant
     d=b**2-4*a*c
     if (d==0) then     !-- une seule racine
        print*, 'racine=',-b/(2.*a)
     else if (d>0) then !-- racines réelles
        print*, 'racines=', (-b+sqrt(real(D)))/(2.*a), &
             &  '      et', (-b-sqrt(real(D)))/(2.*a)
     else                !-- racines complexes
        Partie_re=-b/(2.*a)
        Partie_im=sqrt(real(-d))/(2.*a)
        print*, 'racines=',Partie_re,'+i*',Partie_im, &
             &  '      et',Partie_re,'-i*',Partie_im
     end if
  else
     print*,'equation de degré<=1'
  end if

end program racines_p2
```

1. Comprendre les opérations effectuées par ce programme.
2. Avec Emacs ou un autre éditeur, taper ce programme dans un fichier nommé *racines_p2.f90*.
3. Compiler et lancer ce programme. Vérifier qu'il fonctionne bien avec les tests suivants :
   - $a = 1, b = -3, c = 2$
   - $a = 1, b = -2, c = 1$
   - $a = 1, b = 1, c = 1$
   - $a = 0, b = 2, c = 3$
4. Copier le fichier sous le nouveau nom *racines_p2_bis.f90*. Editer ce fichier et déclarer une nouvelle variable `s2a`. Dans le programme, donner à `s2a` la valeur $\frac{1}{2a}$, et remplacer partout l'expression `1./(2.*a)` par `s2a`. En quoi est-ce utile ?
5. Compiler le fichier pour produire l'executable *racines_p2_bis*, et tester ce nouveau programme.

Exercice 5. Écrire, compiler, et exécuter un programme qui affiche le message *Bonjour tout le monde* à l'écran.

Exercice 6. Lesquelles des déclarations suivantes sont incorrectes et pourquoi ? Ecrivez les dans le programme `declarations`, puis compilez ce programme pour voir les messages d'erreurs associés.

```
ReAl :: x
character :: nom
character(len=10) :: long
real :: var-1
integer :: 1a
boolean :: loij
double : x
integer :: pi_entier=22/7
real , parameter :: pi=22./7.
```

Exercice 7. Écrire un programme permettant de résoudre le système de 2 équations à 2 inconnues :

$$\begin{cases} u_1 x + v_1 y = w_1 \\ u_2 x + v_2 y = w_2 \end{cases}$$

On pourra imprimer les solutions sous la forme `x = ..., y = ...`.

Exercice 8.

1. Écrire un programme qui lit une valeur entière n, puis calcule et écrit les n premiers termes de la suite $u_{n+1} = 2u_n + 3$, sachant que $u_0 = 1$.
2. Même question pour la suite de Fibonacci $u_{n+1} = u_n + u_{n-1}$, sachant que $u_0 = u_1 = 1$.

Exercice 9.

1. Écrire un programme pour lire le rayon d'un cercle au clavier et renvoyer l'aire et le volume du cercle et de la sphère correspondants (rappel : πr^2 et $4/3\pi r^3$).
2. Compiler, tester.
3. Rajouter une boucle infinie **do** dans laquelle, après l'affichage, on demande à l'utilisateur s'il désire continuer ou pas. Si la réponse est oui, on demande a nouveau un rayon, on recalcule l'aire et le volume, on affiche, et on repose la question. Si la réponse est non, le programme s'arrête.

Exercice 10. Écrire un programme qui lit une valeur réelle x, calcule et écrit la valeur $x/(1+x)$. Le cas $x = -1$ devra produire un message d'erreur et devra etre suivi d'une nouvelle opération de lecture de la valeur de x.

Exercice 11. Écrire un programme permettant d'effectuer le produit de 2 matrices A et B dont les dimensions sont stockées dans des constantes. Afficher à l'écran la matrice résultat C ligne par ligne.

Exercice 12. Écrire un programme qui :
- déclare un tableau de 100 entiers ;
- affecte aux éléments le valeurs $1, 2, 3, \ldots, 100$;
- lit deux valeurs entières comprises entre 1 et 100 ;
- inverse l'ordre des éléments du tableau qui sont compris dans l'intervalle donné.

Exercice 13. Écrire un programme permettant de trier une liste de nombres (stockée dans un vecteur) en ordre croissant. Utiliser l'algorithme de tri à bulle suivant :

1. comparer les 2 premiers éléments consécutifs et les intervertir s'ils ne s'ont pas dans l'ordre croissant ;
2. continuer ainsi de suite (les éléments 2 et 3, puis 3 et 4, etc.)
3. après avoir parcouru tout le vecteur, si au moins une interversion a été effectuée, recommencer à 1, sinon le tri est terminé.

Le vecteur sera donné en début de programme. Il sera affiché à l'écran avant le début de l'algorithme et à la fin.

Exercice 14. Ecrire un programme qui :

1. choisit un nombre au hasard entre 0 et 100 (utiliser la subroutine intrinsèque `random_number` qui renvoie un nombre n au hasard entre 0 et 1, et la fonction `nint` qui renvoie la partie entière d'un réel) ;
2. permet à l'utilisateur de jouer à retrouver ce nombre de la façon suivante : dans une boucle, tant que l'utilisateur n'a pas trouvé n, lui demander de taper un entier, puis lui répondre s'il est inférieur ou supérieur au nombre n.

2 Un peu plus loin ...

Exercice 15. Ecrivez des types dérivés qui permettent de représenter :
- l'enregistrement administratif d'une voiture (nom et prénom du propriétaire, numéro de plaque minéralogique, marque de la voiture) ;
- un cercle (coordonnées du centre et rayon) ;
- un livre (titre, auteur, nombre de pages).

Écrire le programme `types_deriv` qui utilise ces types dérivés, et écrit à l'écran un exemple de constante pour chacun des types dérivés ainsi définis.

Exercice 16.

1. Écrire un programme principal avec une subroutine interne qui prend en entrée deux réels a b et renvoie en sortie la somme $a + b$ dans la variable `c`. Compiler, tester.
2. Faire la même chose avec une fonction interne (arguments d'entrée a et b).

Exercice 17. Qu'est-ce qui ne va pas dans la procedure interne suivante ? Après l'avoir écrite dans un programme principal, compiler le programme pour voir les messages d'erreur correspondants.

```
subroutine erreur(a,b,c)
 implicit none
 real, intent(in) :: a
 real, intent(out) :: c
 a=2*c
end subroutine erreur
```

Exercice 18. Etant données les déclarations suivantes :

```
real, dimension(5,6) :: A,B
real, dimension(5) :: C
```

indiquer les instructions qui sont correctes parmi les suivantes :

```
A = B          ;   C = A(:,2) + B(5,1:5)
A = C + 1.0    ;   C = A(2,:) + B(:,5)
A(:,3) = C     ;   B(2:6,3) = C + B(1:5,3)
```

Exercice 19.

1. Soit la matrice suivante : $A = \begin{pmatrix} 0 & -5 & 8 & 3 \\ 3 & 4 & -1 & 2 \\ 1 & 5 & 6 & -4 \end{pmatrix}$. Dans un programme, définir cette matrice, et déterminer les valeurs et les indices du plus grand et du plus petit élément en valeur absolue de A. Utiliser pour cela les fonctions intrinsèques `maxloc`,`minloc`,`maxval`,`minval`,`abs`.
2. calculer la somme des carrés des éléments de A, la moyenne des éléments, la norme 1 de A ($\max_j \sum_i |a_{i,j}|$) (utiliser la fonction intrinsèque `sum` et les opérations usuelles étendues aux tableaux).

Exercice 20. Dans un programme, initialiser deux tableaux a et b de taille 1000 par 1000. Calculer le produit terme à terme de a par b avec une boucle externe sur les lignes, puis faire le même calcul en inversant l'ordre des boucles. A l'aide de la commande `cputime`, calculer le temps calcul de chacune de ces boucles et expliquer le résultat.

Exercice 21. La subroutine interne `matrix_mult` effectue le produit de deux matrices A et B dont les dimensions sont respectivement ma, na et mb, nb. Le résultat est retrourné dans une matrice C de dimension ma, nb. Les arguments d'entree de `matrix_mult` sont A, B et leurs dimensions. l'argument de sortie est C.

- écrire cette subroutine
- écrire et tester le programme `prog_mult` qui effectue une multiplication avec cette subroutine.
- modifier la subroutine pour éviter de passer les dimensions des matrices en arguments. Modifiez le programme principal en conséquence et retestez le.

Exercice 22. Écrire un programme permettant de remplir une matrice de n lignes et m colonnes (n et m n'étant connus qu'au moment de l'exécution) de la façon suivante :

- les lignes de rang pair seront constituées de l'entier 1 ;
- les lignes de rang impair seront constituées des entiers successifs 1, 2, 3, ...

exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

Le programme affichera à l'écran la matrice obtenue ligne par ligne.

Exercice 23. Reprendre le programme précédent pour faire des sorties formattées :

- dans un premier temps, fixer n et m à 4, puis afficher correctement la matrice avec un format adapté ;
- ensuite, tenir compte du fait que m et n ne sont en fait pas fixés à la compilation, et écrire donc un format "variable" qui dépende de m et n. Utiliser pour cela **write**(`va`,$*$) `a` pour stocker l'entier a sous forme de chaîne dans `va` ainsi que les fonctions **trim**, **adjustl** et l'opérateur // pour manipuler les chaînes.

Exercice 24. Créer un programme principal contenant la fonction `rptens` qui renvoie le produit tensoriel de deux vecteurs réels (de même taille) donnés en entrée. La taille des vecteurs n'est pas un argument de la fonction. Tester cette fonction dans le programme principal.

Exercice 25. Etant donné la déclaration de tableau :

```
real , dimension(50,20) :: A
```

écrire les sections de tableau qui représentent :

- la première colonne de A ;
- la dernière colonne de A ;
- l'ensemble des éléments de A situés dans une colonne et ligne paires ;
- l'ensemble des éléments de A situés dans une ligne et colonne paires mais en sens inverse dans les deux dimensions ;

Exercice 26. Écrire un programme qui demande à l'utilisateur le nom d'un fichier, puis ouvre ce fichier, et l'écrit à l'écran, ligne par ligne, en faisant précéder chaque ligne par son numéro. Utiliser le fichier source du programme pour le test.

Exercice 27. Modifier le programme de l'exercice **??** pour qu'il demande à l'utilisateur une chaîne de caractères `chaine`, et écrive dans le fichier de nom `fich` ce qui était écrit à l'écran dans la version initiale du programme. Le nom fich devra être formé ainsi `chaine_`'a'`_va_`'b'`_vb_`'c'`_vc`, où `va`,`vb`,`vc` sont des variables chaînes contenant la valeur de a, b, c. Exemple, si $a = 1, b = 2, c = 4$ et `chaine`='toto', alors le nom du fichier est *toto_a1_b2_c4*.

Utiliser pour cela les fonctions **trim**, **adjustl**, // pour manipuler les chaînes, et **write**(`va`,$*$) `a` pour stocker l'entier a sous forme de chaîne dans `va`.

3 Programmation modulaire

Exercice 28. Reprendre la fonction `rptens` de l'exercice **??** et l'inclure dans un module appelé `mod_ptens`. Utiliser cette fonction dans un programme principal et la tester. Le module et le programme principal devront être placés dans des fichiers séparés. Utiliser alors les commandes de compilation séparée vues en cours.

Exercice 29. On veut créer un ensemble de fonctions "à la matlab" pour manipuler des matrices réelles :

- fonction `diag(A)` pour extraire la diagonale de `A` (matrice carrée) ;
- fonctions `triu(A)` et `tril(A)` pour extraire les parties triangulaires supérieures et inférieures de `A` ;
- fonction `tridiag(u,v,w)` pour créer la matrice tridiagonale ayant `v` sur la diagonale, et `u` et `w` sur les diagonales sup et inf (qui devraient être de dimension inférieure de 1 à celle de `v`).

Créer trois modules correspondants, les utiliser dans un programme principal. Compiler le tout de façon séparée (on pourra utiliser un makefile).

4 Utilisation avancée

Exercice 30. Créer le module `mod_resistance` qui contient deux opérateurs `.parallele.` et `.serie.` tels que pour deux réels R_1 et R_2 représentant les résistances élèctriques d'appareils, ces opérateurs renvoient la résistance R du montage en parallèle ou en série. On rappelle que dans le cas du montage en parallèle, on a $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$, et dans le cas de la série $R = R_1 + R_2$. Tester ce module dans un programme principal avec les valeurs $R_1 = 10$ et $R_2 = 5$.

Exercice 31.
- dans le module `mod_ptens` de l'exercice **??**, rajouter la fonction `iptens` qui effectue la même opération que `rptens` pour des vecteurs entiers ;
- créer ensuite la fonction générique `ptens` associée à ces deux fonctions ;
- enfin, créer l'opérateur `.ptens.` associé à cette fonction générique.

Chaque étape devra être testée dans le programme principal.

Exercice 32. Soit A une matrice quelconque. On décide de stocker uniquement les éléments non nuls de A, pour cela on considère la structure suivante :

```
type element
   real    :: coef
   integer :: indl, indc
end type element

type(element), dimension(:), allocatable :: a
```

Dans la suite de l'exercice, toutes les matrices utilisées sont stockées sous cette forme.

1. Donnez la nature de ces objets :

```
A,  A(i),  A(i)%coef,  A(i)%indl, A(i)%indc
```

   On désigne par `A(i)%coef` le coefficient non nul de `A` et par `A(i)%indl` (respectivement `A(i)%indc`) l'indice de ligne (resp. l'indice de colonne) de ce coefficient. A titre d'exemple, la matrice suivante

$$\begin{pmatrix} 10 & 0 & 0 & -1 \\ 4 & 9 & 3 & 0 \\ 1 & 0 & 5 & 0 \\ 0 & -3 & 0 & 8 \end{pmatrix}$$

   est stockée dans le fichier *a.mat* comme suit

```
9              * nombre de coefficients non nuls de la matrice
-1.   1   4    * coef  indL  indC
 4.   2   1
 9.   2   2
10.   1   1
 1.   3   1
-3.   4   2
 3.   2   3
 8.   4   4
 5.   3   3
```

2. Écrire un procédure permettant de lire une matrice stockée sous cette forme.
3. Écrire une fonction nommée `trace(b)` permettant de calculer la trace d'une matrice b.
4. Écrire une fonction `norme(b)` permettant de calculer la norme infinie d'une matrice b.

5. Écrire une fonction : `matvect(A,x)` permettant de faire le produit d'une matrice A par un vecteur x.

   On pourra écrire dans un module une surdéfinition de l'opérateur $*$, afin que l'opération `c=A*x` soit valide.

6. Soit la suite des vecteurs définie par :

$$
\begin{aligned}
& q_0 \text{ donne tel que} \|q_0\| = 1 \\
& x_k = A q_{k-1}, \quad k \geq 1 \\
& q_k = \frac{x_k}{\|x_k\|} \\
& \lambda_k = q_k^T A q_k
\end{aligned}
$$

   où $\|\|$ désigne la norme euclidienne. La suite λ_k converge sous certaines conditions vers la plus grande valeur propre en module de la matrice (algorithme de la puissance itérée).

   Écrire un programme permettant de calculer une approximation de cette valeur propre pour la matrice A. On considère comme test d'arrêt pour cette méthode itérative $\|x_k - x_{k-1}\| \leq \varepsilon$.