TE 8

Tableaux en fortran et C

Les exercices du TE sont à traiter **au choix en Fortran ou en C**. Les tableaux étant traités de façon très différente en Fortran et en C, certaines questions seront néanmoins spécifiques à un langage.
Le TE s'organise comme suit :
– 2 exercices consacrés au calcul vectoriel (tableaux de dimension 1 en C, rang 1 en Fortran) ;
– 2 exercices soulignant les spécificités du C et du Fortran pour le traitement des tableaux ;
– 2 exercices consacrés au calcul matriciel (tableaux de tableaux en C, tabelaux de rang 2 en Fortran).

En C, on pourra utiliser selon les exercices (voir annexe page 5) :
- soit des tableaux dynamiques alloués sur le tas, explicitement en 1D et grâce à la bibliothèque `mnitab` en 1D ou 2D ;
- soit des tableaux automatiques du C99 (par exemple pour l'exercice 5).

En fortran, on s'appliquera à proposer une solution avec et sans les fonctions intrinsèques spécifiques aux tableaux.

Exercice 1 : Norme d'un vecteur A

On souhaite calculer la norme d'un vecteur à N composantes qui sont des nombres réels. On cherche donc à écrire un programme qui :
– déclare un tableau de flottants de dimension 1 (rang 1 en fortran) représentant un vecteur ;
– puis lit (de préférence par redirection depuis un fichier) ou calcule les composantes du vecteur ;
– affiche ces composantes ;
– calcule la norme du vecteur ;
– affiche le résultat.
On commencera par un tableau de taille fixée avant de passer à un tableau alloué dynamiquement.

1.1 Programme simple où N est fixé à la compilation 5 min.

a) Écrire un programme `norme_vect1.f90` ou `norme_vect1.c` qui exécute les tâches listées ci-dessus, avec la taille N du tableau fixée à la compilation :
- **En C**, on utilise par exemple la directive `#define N 5` destinée au préprocesseur.
- **En Fortran**, l'attribut `PARAMETER` pour la taille est suffisant.

b) Tester le programme sur les cas suivants :
- vecteur à 4 composantes : 1. ; 2. ; 3. ; 4. ;
- vecteur à 5 composantes : 1. ; 2. ; 3. ; 4. ; 5. ;

c) En **Fortran**, écrire une variante `norme_vect1b.f90` utilisant la fonction intrinsèque `dot_product`.

1.2 Calcul de la norme dans une fonction 10 min.

On se propose de déplacer le calcul (et uniquement le calcul [1]) de la norme du vecteur à l'intérieur d'une fonction `norme`. On souhaite de plus que cette fonction soit utilisable pour un vecteur de taille quelconque n (y compris dans la partie 1.3 quit suit, où le tableau verra sa taille fixée à l'exécution).

a) Quel(s) argument(s) doit-on fournir à cette fonction ? Préciser les différences entre langage C et Fortran. Quel est le type du résultat ?

1. L'affectation des composantes et l'affichage doivent demeurer dans le programme principal.

b) Modifier le programme précédent pour obtenir `norme_vect2` qui utilise la fonction `norme`.
- **En Fortran,** placer la fonction `norme` dans un module à compiler avant le programme principal.
- **Dans chaque langage,** indiquer dans la fonction `norme` qu'elle n'est pas autorisée à modifier les composantes du vecteur passé en argument.

1.3 Allocation dynamique du vecteur à `n` composantes **20 min.**

À partir de `norme_vect2`, écrire un programme `norme_vect3` qui, lit au clavier la taille `n` du vecteur, alloue dynamiquement le tableau qui le représente et affecte à ses composantes les valeurs des `n` premiers entiers. Le programme appelle ensuite la fonction `norme`, affiche le module du vecteur, et enfin libère la mémoire allouée. Tester quelques cas dont `n`=4, 5 et 24. On rappelle que $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$.

Exercice 2 : Moments d'une variable aléatoire **A**

a) Effectuer dans le programme principal le tirage de `n` nombres réels uniformément répartis dans l'intervalle [0,1[. Ils seront stockés dans un tableau de taille `n` fournie par l'utilisateur à l'exécution. Les tirages se feront par un appel à une procédure intrinsèque de tirage aléatoire dont la nature et la syntaxe dépendent du langage choisi :
- **En Fortran,** *un seul* appel au sous-programme intrinsèque `random_number` avec un argument de type tableau de réels. Il fournit des réels pseudo-aléatoires distribués uniformément dans l'intervalle [0,1[.
- **En C,** une suite d'appels à la fonction `rand` qui rend un **entier** distribué uniformément dans l'intervalle [0, `RAND_MAX`], que l'on ramenèra dans l'intervalle [0,1[.

Afficher les nombres tirés pour des échantillons de petite taille (5, 20, ...) afin de contrôler le bon fonctionnement du programme.

b) Écrire une fonction `moyenne` qui calcule et renvoie la moyenne empirique $\frac{1}{n}\sum_{i=1}^{n} x_i$ des éléments d'un tableau de réels de taille fournie à l'exécution. Comparer la moyenne d'un échantillon avec la moyenne théorique. Que constate-t-on en augmentant la taille de l'échantillon ?

c) Le moment centré d'ordre p d'une variable aléatoire X de moyenne $\langle X \rangle$ est $\mu_p(X) = \langle (X - \langle X \rangle)^p \rangle$. Écrire une procédure `moments` qui estime les moments centrés d'ordre 2 (variance), 3 et 4 d'une variable aléatoire à partir des tirages stockés dans un tableau `x` de taille `n` fournie à l'exécution. Cette procédure utilisera la fonction `moyenne` et stockera les moments dans un tableau `mc` de taille fixe (de 3 réels) déclaré dans le programme principal. Calculer les moments centrés théoriques et comparer selon les valeurs de `n`.

Exercice 3 : Tableaux, pointeurs et fonctions en C **AB**

On se propose d'examiner les adresses des tableaux dans le programme suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define M 4
4  #define ULI (unsigned long int)
5  void affiche(double *t, int n);
6  void affiche(double *t, int n){
7         int i;
8         printf("adresse de t \t= %lu\n", ULI &t);
9         printf("valeur de t \t= %lu\n", ULI t);
10        printf("adr. de t[0] \t= %lu\n", ULI &t[0]);
11        printf("Nombre d'octets occupes par un double = %d\n",
12                (int) sizeof(double));
```

```
13        for (i = 0 ; i < n ; i++) {
14              printf("\tt[%d] = %f\n",i, *(t+i));
15              printf("\tadresse de t[%d] = %lu\n",i, ULI (t+i));
16        }
17  }
18  int main(void) {
19     double tab[M] ;
20     int i;
21     printf("valeur de tab \t= %lu\n", ULI tab);
22     printf("adr. de tab[0] \t= %lu\n", ULI &tab[0]);
23     printf("entrer les %d valeurs du tableau \n", M);
24     /* Boucle de lecture */
25     for (i=0; i< M ; i++) {
26         scanf("%lf", &tab[i]);
27     }
28     affiche(tab, M);
29     exit (EXIT_SUCCESS);
30  }
```

a) Comparer et expliquer les affichages obtenus dans la fonction `main` et la fonction `affiche`.

b) Compléter le programme avec une fonction `oppose` qui change le signe des valeurs du tableaux. Afficher le tableau avant et après l'appel à `oppose`. Expliquer le mécanisme de passage du tableau.

c) Remplacer le tableau de `double` par un tableau de `float` en modifiant types et formats partout où c'est nécessaire. Commenter.

Exercice 4 : Fonctions intrinsèques pour tableaux en Fortran B

a) On considère le vecteur de nombres réels `w = [-6.5, 7.4, 8.1, -3.2, 6.7]`. Écrire un programme Fortran qui fait usage de fonctions intrinsèques pour afficher les informations relatives à `w` : taille (nombre total d'éléments), profil (vecteur des étendues), indice minimal, indice maximal, élément maximal, élément minimal, indice de l'élément maximal, indice de l'élément minimal, somme des composantes, produit des composantes, et produit des composantes positives.

b) Compléter le programme de la question précédente pour déterminer la position relative de l'élément maximal par rapport à l'élément minimal. Commenter les erreurs de compilation éventuelles : le résultat des fonctions intrinsèques est-il scalaire ou vectoriel ? Corriger si nécessaire.

c) On désire vérifier que l'indice maximal de `w` moins l'indice minimal de `w` est égal à l'étendue du vecteur `w` moins 1. Compléter le programme pour afficher ces deux valeurs.

d) Introduire le vecteur `u` indexé entre `-2` et `+2` (inclus) tel que `u=w`. Afficher les informations relatives à `u` : indice minimal et maximal, indices des éléments minimal et maximal. Commenter.

Exercice 5 : Construction de matrices particulières A

5.1 Allocation 2D

Écrire un programme qui :
- demande la saisie de deux entiers `lignes` et `colonnes` ;
- vérifie qu'ils sont strictement positifs ;
- alloue un tableau 2D représentant une matrice d'entiers à `lignes` lignes et `colonnes` colonnes ;
- remplit ce tableau de 1.

5.2 Affichage d'un tableau 2D

Écrire une procédure `print_mat` qui affiche un tableau 2D quelconque d'entiers sous la forme de la matrice associée. Vérifier le contenu du tableau généré par le programme développé à la question précédente.
- **En Fortran :** penser à tirer profit du fait que fortran 90 sait afficher les tableaux... en ligne ! Pour pouvoir effectuer une allocation dans une procédure et retourner dans l'appelant sans désallouer, il est nécessaire d'utiliser le standard 2003 (via les alias `g2003-mni` ou `gfortran2003-mni`)
- **En C :** l'usage de deux boucles imbriquées est indispensable.

5.3 Matrice identité

Écrire une procédure `diag` qui remplit le tableau 2D `mat_id` de taille `n` par `n` de 1 sur la diagonale et de zéros ailleurs, de façon à représenter la matrice identité de dimension `n` par `n`. Vérifier à l'aide de la procédure `print_mat` que le programme produit le résultat attendu.

5.4 Matrice tridiagonale

Écrire une procédure `tridiag` qui remplit le tableau 2D `mat_tri` de taille `n` par `n` avec la matrice tridiagonale ci-contre. Vérifier à l'aide de `print_mat` que le programme produit le résultat attendu.

$$A = \begin{pmatrix} -2 & 3 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 3 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 3 & & \vdots \\ \vdots & & & & \ddots & \\ 0 & \cdots & 0 & 1 & -2 & 3 \\ 0 & \cdots & 0 & 0 & 1 & -2 \end{pmatrix}$$

Exercice 6 : Produit de matrices AB

Le but de cet exercice est de programmer la lecture de deux matrices (à éléménts réels) à partir de fichiers, leur multiplication matricielle, et l'affichage de la matrice résultat. Les dimensions des matrices n'étant connues qu'à la lecture des fichiers, les tableaux de dimension 2 utilisés pour les stocker devront donc être alloués dynamiquement [2].

Les fichiers contenant les matrices comportent une première ligne indiquant le nombre de lignes et le nombre de colonnes de la matrice, puis les lignes des coefficients. Par exemple, la matrice identité de taille 3 sera codée dans un fichier de la façon suivante :

```
3 3
1. 0. 0.
0. 1. 0.
0. 0. 1.
```

En pratique, il faudra donc lire les deux premiers entiers, puis allouer la mémoire, puis lire les lignes de la matrice. Les deux matrices de nombres réels A de dimension (n, p) et B de dimension (p, m) sont fournies dans deux fichiers texte séparés [3] `matA.dat` et `matB.dat`. On souhaite calculer la matrice $C = A.B$ de dimension (n, m).
- **En C**, on utilisera la bibliothèque `mnitab`, décrite dans l'annexe page 5.

a) Écrire un programme `produit_mat1.f90` ou `.c`, permettant de lire les fichiers, de stocker A dans un tableau `a` et B dans un tableau `b`, et d'afficher A et B.

b) Copier le fichier contenant ce programme dans un fichier `produit_mat2`. Modifier le programme pour effectuer la lecture et l'allocation dynamique dans une procédure `lecture_matrice` et l'affichage dans une autre procédure `print_mat`.

c) Écrire une procédure `mul_mat` de multiplication des deux matrices. Calculer le produit $C = A.B$ et afficher le résultat.
- **En Fortran :** comparer avec les résultats obtenus via la fonction intrinsèque `matmul`.

2. On pourra cependant commencer la résolution par une version avec des tableaux de dimensions fixes.
3. Voir `~lefrere/M1/2012-2013/etu/mni/f90+c/te/te8/`

Annexe : tableaux de taille variable en C

Tableaux 2D automatiques de taille variable en C99

La norme C99 a introduit les tableaux de taille variable alloués sur la pile dans les fonctions ou dans les blocs. Grâce à la déclaration tardive, il est possible de créer en C99 des tableaux de taille ajustable à l'exécution y compris dans le programme principal. S'ils sont déclarés dans une fonction, leur portée est limitée à cette fonction. Ils se manipulent comme les tableaux statiques. On pourra appliquer cette méthode à l'exercice 5, mais elle n'est pas adaptée à l'exercice 6.

Rappel : allocation dynamique de tableaux 2D sur le tas en C

Les tableaux 2D dynamiques ne sont pas définis par le langage C89 lui-même. La représentation des tableaux 2D utilisée dans la bibliothèque `mnitab` consiste à introduire un tableau de pointeurs des débuts de ligne de la matrice (cf cours). Le calcul des adresses est fait une fois pour toutes et la double indexation `[i][j]` permet ensuite l'accès aux éléments. Une matrice de flottants sera donc représentée par un pointeur de pointeur sur des flottants. L'espace mémoire correspondant sera alloué en appelant une fonction `float2d` fournie du type : `float ** float2d(int nlignes, int ncolonnes)` qui :
- alloue l'espace mémoire pour les coefficients, au moyen d'une instruction du type : `coeffs = calloc( nlignes * ncolonnes , sizeof(float) );` ,
- alloue un tableau 1D (de type `float **`) de `plignes` de pointeurs qui pointeront sur les débuts de ligne ;
- affecte ces pointeurs aux adresses des débuts de ligne ;
- rend le pointeur des pointeurs de début de lignes.

En fin de calcul, l'espace mémoire des coefficients, puis celui du tableau de pointeurs seront libérés par une fonction `float2d_libere`, elle aussi fournie. Il sera prudent de mettre ensuite à `NULL` le pointeur de pointeur qui a désigné la matrice ainsi libérée.

La bibliothèque `mnitab`

La bibliothèque `mnitab` est installée via le compte de l'UE ; elle comporte :
- Le fichier d'entête `mnitab.h` destiné au préprocesseur pour la phase de compilation. Il est installé dans le répertoire `~lefrere/include/`
- Le fichier binaire d'archive de la bibliothèque elle-même `libmnitab.a` pour la phase d'édition de liens avec `ld`. Il est installé dans le répertoire `~lefrere/lib/`

Si les chemins d'accès ont été fournis par les options `-I~/lefrere/include` et `-L~lefrere/lib` du compilateur, l'utilisateur doit donc seulement :
- inclure le fichier `mnitab.h` destiné au préprocesseur pour la phase de compilation : `#include "mnitab.h"`
- demander d'attacher la bibliothèque par l'option `-lmnitab` lors de la production de l'exécutable par l'éditeur de liens.

En pratique, on utilisera les alias `gcc+mni-c99` ou `gcc+mni-c89` qui définissent les chemins d'accès au fichier d'entête de la bibliothèque `mintab.h` et au fichier `libmnitab.a`.

Prototypes de la bibliothèque `mnitab`

Les prototypes de la bibliothèque `mnitab` fournis par `~lefrere/include/mnitab.h` sont aussi décrits dans le fichier `~lefrere/M1/Doc/f90+c/libmnitab/mnitab.liste_utf8`. Voici par exemple les déclarations des deux fonctions d'allocation `float1d` et `float2d` et leurs compagnons de libération :

```
float * float1d(int n);
void float1d_libere(float *vect);
float ** float2d(int nlignes, int mcolonnes);
void float2d_libere(float ** mat);
```

Générateurs pseudo-aléatoires

Appel du générateur

Les générateurs pseudo-aléatoires fournissent des tirages successifs statistiquement indépendants pris dans une suite *finie* de très longue période.

En C

Le générateur `rand()` rend un entier de type `int` compris entre `0` et `RAND_MAX` défini dans `stdlib.h` et de même valeur que `INT_MAX`, le plus grand entier signé représenté dans le type `int` (en général 2147483647). Pour obtenir des valeurs réelles entre 0 et 1, il faudra normaliser ce tirage par division flottante par `RAND_MAX`. Si on souhaite remplir un tableau de réels, il faut faire autant d'appels à `rand` qu'il y a d'éléments dans le tableau.

En fortran

Le générateur `RANDOM_NUMBER` est un sous-programme élémentaire (que l'on peut donc appeler avec des arguments tableau) qui calcule des valeurs pseudo-aléatoires comprises entre 0 et 1. Si l'argument est scalaire, une seule valeur est calculée alors que s'il s'agit d'un tableau, autant de valeurs pseudo-aléatoires sont calculées qu'il y a d'éléments dans le tableau.

Initialisation du générateur

Le générateur possède une mémoire, ou germe, qui « progresse » naturellement à chaque invocation, et les tirages successifs apparaissent indépendants lors d'une seule exécution du code. Mais, sans intervention sur ce germe, chaque exécution d'un programme appelant le générateur reproduira exactement la même séquence de tirages.
Cette mémoire peut aussi être manipulée de façon à choisir le point de départ de la séquence, en s'appuyant sur des éléments qui varient d'une exécution à l'autre (date, numéro du processus lancé, ...). Il est enfin possible de stocker le germe pour réinitialiser le générateur en cours d'exécution par exemple à chaque début d'une boucle pour reproduire plusieurs fois une même séquence. Bien noter que ces manipulations du germe produisent des tirages qui ne sont plus indépendants. Réinitialiser le germe avec les mêmes valeurs à chaque appel constituerait une maladresse grave.

En C

La fonction `srand` permet d'initialiser le générateur `rand` : son argument est de type `unsigned int`.

En fortran

Le sous-programme `RANDOM_SEED` admet des arguments optionnels `GET` pour récupérer la valeur courante du germe et `SET` pour la réinitialiser. Comme le germe est un tableau d'entiers, il faut au préalable utiliser l'argument optionnel `SIZE` pour connaître sa taille et allouer le tableau.