Licence Mathématiques et Applications
3e année : Mathématiques Pour l'Ingénierie
Programmation - Correction des TP de Fortran 90

---

1 Exercices de base

Exercice 1.

Exercice 2.

Exercice 3.

Exercice 4.

Exercice 5.

```
program bonjour

  print*,'Bonjour tout le monde'

end program bonjour
```

Exercice 6.

Exercice 7.

```
program systeme

  implicit none

  !-------- variables
  real :: u1,u2,v1,v2,w1,w2
  real :: x,y,d
  !-------


  !--- lecture des coefficients
  print*,'donner les coefficients u1,u2,v1,v2,w1,w2'
  read*,u1,u2,v1,v2,w1,w2

  !--- calcul du determinant
  d=u1*v2-u2*v1

  !--- resolution du systeme
  if (d/=0.0) then

     x=-(v1*w2-v2*w1)/d
     y=(u1*w2-u2*w1)/d

     print*,'x=',x,'y=',y

  else

     print*,'determinant nul'

  end if

end program systeme
```

Exercice 8.

```
program fibonacci

  implicit none

  !-------- variables
  integer :: i,n
  integer :: u,v,w
  !-------


  !--- lecture des données
  print*,'donner les nombre n de termes voulus'
  read*,n

  !--- initialisation de la suite
  u=1
  v=1
  print*,u

  !--- calcul des termes de la suite
  do i=1,n-1
     w=v+u
     print*,w
     u=v
     v=w
  end do

end program fibonacci
```

Exercice 9.

```
program rayon

  implicit none

  !-------- variables
  real :: r,pi
  !-------

  !--- calcul de pi
  pi=acos(-1.0)

  !--- lecture des données
  print*,'donner le rayon'
  read*,r

  !--- affichage aire et volume
  print*,'aire du cercle =',pi*r**2
  print*,'volume de la sphere =',4./3.*pi*r**3

end program rayon
```

```
program rayon_bis

  implicit none

  !-------- variables
  character(len=3) :: reponse
  real :: r,pi
  !-------

  !--- calcul de pi
  pi=acos(-1.0)

  !--- boucle
  boucle:do

     !--- lecture des données
     print*,'donner le rayon'
     read*,r

     !--- affichage aire et volume
     print*,'aire du cercle =',pi*r**2
     print*,'volume de la sphere =',4./3.*pi*r**3

     !--- question
     print*,'voulez-vous continuer (oui/non) ?'
     read*,reponse
     if (reponse=='non') then
        exit boucle
     end if

  end do boucle

end program rayon_bis
```

Exercice 10.

```
program fraction

  implicit none

  !-------- variables
  real :: x
  !-------

  !--- boucle
  boucle:do

     !--- lecture de x
     print*,'donner x'
     read*,x

     !--- calcul, seulement si x different de -1
     if (x/=-1.0) then
        print*,'x/(1+x)=',x/(1.0+x)
        exit boucle
     end if

  end do boucle

end program fraction
```

Exercice 11.

```
program produit_mat

  !------------ but du programme ----------------------------!
  ! ce programme permet d'effectuer le produit de 2 matrices
  ! A et B dont les dimensions sont stockées dans des
  ! constantes. Affiche à l'écran la matrice résultat C ligne
  ! par ligne.
  !-------------------------------------------------------------!

  implicit none

  !-------- variables
  integer, parameter :: m=2, n=2, p=3
  integer :: i,j,k
  integer :: coeff
  integer, dimension(m,n) :: A
  integer, dimension(n,p) :: B
  integer, dimension(m,p) :: C
  !-------


  !--- lecture de A et B
  print*,'rentrer les éléments de A'
  do i=1,m
     do j=1,n
        print*,'A(',i,',',j,')='
        read*,A(i,j)
     end do
  end do

  print*,'rentrer les éléments de B'
  do i=1,n
     do j=1,p
        print*,'B(',i,',',j,')='
        read*,B(i,j)
     end do
  end do

  !--- calcul du produit
  !- boucle sur les elements de C
  do i=1,m
     do j=1,p
        !- calcul de C(i,j) = somme sur k des A(i,k)*B(k,j)
        coeff=0.0
        do k=1,n
           coeff=coeff+A(i,k)*B(k,j)
        end do
        C(i,j)=coeff
     end do
  end do

  !--- affichage de C
  do i=1,m
     print*,C(i,:)
  end do

end program produit_mat
```

Exercice 12.

```
program inverse_tableau

  !------------ but du programme ----------------------------!
  ! ce programme déclare un tableau de 100 entiers ;
  ! affecte aux éléments le valeurs 1,2,3,...,100 ;
  ! lit deux valeurs entières comprises entre 1 et 100 ;
  ! inverse l'ordre des éléments du tableau qui sont compris dans
  !  l'intervalle donné.
  !-------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: i,m,n,milieu
  integer :: aux
  integer, dimension(100) :: tab
  !-------


  !--- remplissage de tab
  tab=(/(i,i=1,100)/)

  !--- lecture de m et n
  print*,'rentrer m et n'
  read*,m,n

  !--- calcul du numero du "milieu" de la section a inverser
  milieu=(n+m)/2    ! =(n+m)/2 si n+m pair, =(n+m-1)/2 sinon

  !--- inversion
  do i=m,milieu
     aux=tab(i)
     tab(i)=tab(n+m-i)
     tab(n+m-i)=aux
  end do

  !--- affichage
  print*,tab

end program inverse_tableau
```

Exercice 13.

```
program tri

  !------------ but du programme -----------------------------!
  ! Ecrire un programme permettant de trier une liste de nombres (stockée
  ! dans un vecteur) en ordre croissant. Utiliser l'algorithme de tri  à
  ! bulle suivant :
  !  - comparer les 2 premiers éléments consécutifs et les
  !  intervertir s'ils ne s'ont pas dans l'ordre croissant ;
  !  - continuer ainsi de suite (les éléments 2 et 3, puis 3 et 4,
  !    etc.)
  !  - 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.
  !-------------------------------------------------------------!

  implicit none

  !-------- variables
  integer, parameter :: n=100
  logical :: nointer
  integer :: i,aux
  integer, dimension(n) :: tab
  real, dimension(n) :: tabr
  !-------


  !--- remplissage de tab
  call random_number(tabr)  ! argument reel obligatoire
  tab=int(tabr*n)

  !--- affichage
  print*,tab

  !--- tri a bulle
  bulle:do
     nointer=.true.
     do i=1,n-1
        if (tab(i)>tab(i+1)) then ! interversion
           aux=tab(i+1)
           tab(i+1)=tab(i)
           tab(i)=aux
           nointer=.false. ! signifie qu'on a fait une interversion
        end if
     end do
     if (nointer) then
        exit bulle
     end if
  end do bulle

  !--- affichage
  print*,'------ tableau apres tri --------'
  print*,tab

end program tri
```

Exercice 14.

```
program nb_inconnu

  !------------ but du programme ----------------------------!
  ! ce programme choisit un nombre au hasard entre 0 et 100
  ! et permet à l'utilisateur de jouer à retrouver ce nombre
  !-------------------------------------------------------!

  implicit none

  integer :: n,nb
  real :: x

  !--- génération du nb aléatoire
  call random_seed
  call random_number(x)

  nb=nint(x*100)

  !--- recherche
  n=nb+1
  do while (n/=nb)

     print*,'donner un nombre n'
     read*,n

     if (n>nb) then
        print*,'trop grand'
     elseif (n<nb) then
        print*,'trop petit'
     end if

  end do
  print*,'vous avez trouve, bravo'

end program nb_inconnu
```

2 Un peu plus loin ...

Exercice 15.

```
program types_deriv

  !------------ but du programme ----------------------------!
  ! 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).
  ! Ecrire le programme \fort{types_deriv} qui utilise ces types dérivés,
  ! et écrit à l'écran un exemple de constante pour chacun des types dérivés ainsi
  ! définis.
  !-------------------------------------------------------!

  implicit none

  !-------- variables : types derives
  type voiture
     character(len=30) :: nom, prenom, plaque, marque
  end type voiture
```

```
  type cercle
     real , dimension(2) :: centre
     real :: rayon
  end type cercle

  type livre
     character(len=200) :: titre
     character(len=30) :: auteur
     integer :: npages
  end type livre

  !-------- variables
  type(voiture), parameter :: v1=voiture('dupond','martin','123ABC31','renault')
  type(voiture) :: v2
  type(cercle), parameter :: c1=cercle((/0.0,1.0/),2.0)
  type(cercle) :: c2
  type(livre), parameter :: l1=livre('Diadorim','Joao Guimaraes Rosa',428)
  type(livre) :: l2


  !--- afichage
  print*,trim(v1%nom), ' ', trim(v1%prenom), ' ', trim(v1%plaque), ' ', trim(v1%marque)
  print*,c1
  print*,l1

  !--- remplissage
  print*,'informations sur votre voiture ...'
  print*,'nom du proprietaire : '; read*, v2%nom
  print*,'prenom du proprietaire : '; read*, v2%prenom
  print*,'numero de la plaque : '; read*, v2%plaque
  print*,'marque : '; read*, v2%marque

  print*,'coordonnees du centre du cercle (2 nombres)'
  read*,c2%centre
  print*,'rayon du cercle'
  read*,c2%rayon

  print*,'titre du livre' ; read*,l2%titre
  print*,'auteur' ; read*,l2%auteur
  print*,'nb de pages' ; read*,l2%npages


  !--- affichage
  print*,v2
  print*,c2
  print*,l2

end program types_deriv
```

Exercice 16.

```
program prog_somme

  !------------ but du programme ----------------------------!
  ! - Ecrire un programme principal avec une subroutine interne qui prend en
  !   entree deux réels a b et renvoie en sortie la somme a+b dans la
  !   variable \fort{c}. Compiler, tester.
  ! - Faire la même chose avec une fonction interne (arguments
  !   d'entrée a et b).
  !----------------------------------------------------------!
```

```
  implicit none

  !-------- variables
  real :: a,b,c
  !--------

  !--- donnees
  print*,'rentrer a,b'
  read*,a,b

  !--- appel subroutine
  call somme(a,b,c)

  !--- affichage
  print*,'a+b=',c

  !--- appel fonction et affichage
  print*,fsomme(a,b)

contains

  subroutine somme(x,y,z)

    implicit none

    !-------- variables in
    real, intent(in) :: x,y
    !--------

    !-------- variables out
    real, intent(out) :: z
    !--------

    !--- calcul
    z=x+y

  end subroutine somme

  function fsomme(x,y) result(z)

    implicit none

    !-------- variables in
    real, intent(in) :: x,y
    !--------

    !-------- variable resultat
    real :: z
    !--------

    !--- calcul
    z=x+y

  end function fsomme

end program prog_somme
```

Exercice 17.

Exercice 18.

Exercice 19.

```
program indices_matrice

  !------------ but du programme ----------------------------!
  ! - 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.
  ! - calculer la somme des carrés des éléménts de A, la moyenne
  ! des éléments, la norme 1 de A  (utiliser la fonction
  ! intrinsèque sum et les opérations usuelles étendues aux tableaux).
  !-------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: i,j
  integer, dimension(3,4) :: A
  !--------

  !--- donnees
  A=transpose(reshape((/0,-5,8,3,3,4,-1,2,1,5,6,-4/),(/4,3/)))

  !--- calcul et affichage
  do i=1,3
     print*,A(i,:)
  end do

  print*,'valeur element max de |A| : ',maxval(abs(A))
  print*,'indices element max de |A| : ',maxloc(abs(A))

  print*,'valeur element min de |A| : ',minval(abs(A))
  print*,'indices element min de |A| : ',minloc(abs(A))

  print*,'somme carres element de A',sum(A**2)
  print*,'moyenne elements',sum(A)/12
  print*,'norme 1 de A',maxval( (/ (sum(abs(A(:,j))), j=1,4) /) )

end program indices_matrice
```

Exercice 20.

```
program cpu

  !------------ but du programme ----------------------------!
  ! Dans un programme, initialiser deux tableaux a et b de taille
  ! 2000 par 2000. Calculer le produit ab avec une boucle externe sur
  ! les lignes, puis faire le même calcul en inversant l'ordre des
  ! boucles. A l'aide de la commande \fort{cputime}, calculer le temps
  ! calcul de chacune de ces boucles et expliquer le résultat.          !
  !-------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: i,j
  real, dimension(2000,2000) :: a,b,c
  real :: t1,t2
  !--------

  !--- donnees
  call random_number(a)
  call random_number(b)
```

```
  !--- calcul et affichage temps cpu, ordre i,j
  call cpu_time(t1)
  do i=1,2000
     do j=1,2000
        c(i,j)=a(i,j)*b(i,j)
     end do
  end do
  call cpu_time(t2)
  print*,t2-t1

  !--- calcul et affichage temps cpu, ordre j,i
  call cpu_time(t1)
  do j=1,2000
     do i=1,2000
        c(i,j)=a(i,j)*b(i,j)
     end do
  end do
  call cpu_time(t2)
  print*,t2-t1

  !--- calcul et affichage temps cpu, produit fonction intrinsèque
  call cpu_time(t1)
  c=a*b
  call cpu_time(t2)
  print*,t2-t1

end program cpu
```

Exercice 21.

```
program prog_mult

  !------------ but du programme ----------------------------!
  ! - 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.
  ! - La subroutine interne matrix_mult_bis fait la même chose, sans que les
  ! les dimensions des matrices soient passees en arguments
  !-------------------------------------------------------!

  implicit none

  !-------- variables
  integer, parameter :: ma=2,na=3,mb=3,nb=2
  integer :: i
  integer, dimension(ma,na) :: A
  integer, dimension(mb,nb) :: B
  integer, dimension(ma,nb) :: C
  !--------

  !--- donnee
  A=reshape((/1,6,4,5,2,8/),(/2,3/))
  B=reshape((/6,4,2,8,4,9/),(/3,2/))


  !--- appel subroutine avec dimensions en arguments
  call matrix_mult(A,B,C,ma,na,mb,nb)

  !--- affichage
  print*,'calcul avec matrix_mult'
  do i=1,ma
     print*,C(i,:)
  end do

  !--- appel subroutine sans dimensions en arguments
  call matrix_mult_bis(A,B,C)

  !--- affichage
  print*,'calcul avec matrix_mult_bis'
  do i=1,ma
     print*,C(i,:)
  end do

contains

  subroutine matrix_mult(A,B,C,ma,na,mb,nb)

    implicit none

    !-------- variables in
    integer :: ma,na,mb,nb
    integer, dimension(ma,na), intent(in) :: A
    integer, dimension(mb,nb), intent(in) :: B
    !--------

    !-------- variables out
    integer, dimension(ma,nb), intent(out) :: C
    !--------
```

```
    !-------- variables locales
    integer :: i,j,k,coeff
    !--------


    !--- calcul du produit
    !- boucle sur les elements de C
    do i=1,ma
       do j=1,nb
          !- calcul de C(i,j) = somme sur k des A(i,k)*B(k,j)
          coeff=0.0
          do k=1,na
             coeff=coeff+A(i,k)*B(k,j)
          end do
          C(i,j)=coeff
       end do
    end do

  end subroutine matrix_mult

  subroutine matrix_mult_bis(A,B,C)

    implicit none

    !-------- variables in
    integer, dimension(:,:), intent(in) :: A
    integer, dimension(:,:), intent(in) :: B
    !--------

    !-------- variables out
    integer, dimension(:,:), intent(out) :: C
    !--------

    !-------- variables locales
    integer :: i,j,k,coeff
    !--------

    !--- calcul du produit
    !- boucle sur les elements de C
    do i=1,size(A,1)
       do j=1,size(B,2)
          !- calcul de C(i,j) = somme sur k des A(i,k)*B(k,j)
          coeff=0.0
          do k=1,size(A,2)
             coeff=coeff+A(i,k)*B(k,j)
          end do
          C(i,j)=coeff
       end do
    end do

  end subroutine matrix_mult_bis


end program prog_mult
```

Exercice 22.

```
program alloc_dyn

  !------------ but du programme ----------------------------!
  ! 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, ...
  !-------------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: m,n,i,j,k
  integer, dimension(:,:), allocatable :: A
  !--------


  !--- dimensions
  print*,'donner m et n'
  read*,m,n

  !--- allocation
  allocate(A(m,n))

  !--- construction de A : lignes paires
  do i=2,m,2
     A(i,:)=1
  end do

  !--- construction de A : lignes impaires
  k=1
  do i=1,m,2
     do j=1,n
        A(i,j)=k
        k=k+1
     end do
  end do

  !--- affichage
  print*,'A='
  do i=1,m
     print*,A(i,:)
  end do

end program alloc_dyn
```

Exercice 23.

```
program alloc_dyn_formate

  !------------ but du programme ----------------------------!
  ! Comme alloc_dyn, avec des sorties formattees, le format etant
  ! fixe ou variable
  !-------------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: m,n,i,j,k
  character(len=30) :: cn,chaine
  integer, dimension(:,:), allocatable :: A
  !--------



  !------------ (1) FORMAT FIXE

  !--- allocation
  m=4; n=4; allocate(A(m,n))

  !--- construction de A : lignes paires
  do i=2,m,2
     A(i,:)=1
  end do

  !--- construction de A : lignes impaires
  k=1
  do i=1,m,2
     do j=1,n
        A(i,j)=k
        k=k+1
     end do
  end do

  !--- affichage
  print*,'A='
  do i=1,m
     write(*,fmt='(4I3)') A(i,:)
  end do

  !------------ (1) FORMAT VARIABLE

  !--- allocation
  deallocate(A)
  print*,'donner m et n'
  read*,m,n
  allocate(A(m,n))

  !--- construction de A : lignes paires
  do i=2,m,2
     A(i,:)=1
  end do

  !--- construction de A : lignes impaires
  k=1
  do i=1,m,2
     do j=1,n
        A(i,j)=k
```

```
        k=k+1
     end do
  end do

  !--- format d'affichage : n entiers de 3 chiffres
  write(cn,*) n   ! n est transforme en chaine cn
  chaine='('//trim(adjustl(cn))//'I3)'

  !--- affichage
  print*,'A='
  do i=1,m
     write(*,fmt=chaine) A(i,:)
  end do

end program alloc_dyn_formate
```

Exercice 24.

```
program prog_rptens

  !------------ but du programme ----------------------------!
  ! utilise 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.
  !------------------------------------------------------------!

  implicit none

  !-------- variables
  integer, parameter :: n=3
  integer :: i
  character(len=30) :: cn,chaine
  real, dimension(n) :: u,v
  real, dimension(n,n) :: w
  !--------


  !--- donnees
  u=(/1.,2.,3./) ; v=(/2.,1.,3./)

  !--- calcul : appel fonction
  w=rptens(u,v)

  !--- affichage
  write(cn,*) n
  chaine='('//trim(adjustl(cn))//'F10.5)'
  do i=1,n
     write(*,fmt=chaine) w(i,:)
  end do

contains

  function rptens(u,v) result(w)

    implicit none

    !-------- variables in
    real, dimension(:), intent(in) :: u,v
    !--------

    !-------- variable resultat
```

```
    real , dimension(size(u),size(v)) :: w
    !--------

    !-------- variables locales
    integer :: i,j
    !--------


    !--- calcul du produit tensoriel
    do i=1,size(u)
       do j=1,size(v)
          w(i,j)=u(i)*v(j)
       end do
    end do

  end function rptens

end program prog_rptens
```

Exercice 25.

Exercice 26.

```
program ligne_fichier

  !------------ but du programme ----------------------------!
  ! 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. Utilise le fichier source du programme
  ! pour le test
  !-------------------------------------------------------------!

  implicit none

  !-------- variables
  integer :: i,err
  character(len=30) :: fich,chaine,ncar
  character(len=250) :: ligne ! on suppose que ligne a au plus 250 caracteres
  !--------

  !--- nom fichier
  print*,'donner le nom du fichier'
  read*,fich

  !--- ouverture, ecriture ecran
  open(unit=10,file=fich)
  err=0   ! test de fin de fichier (code renvoye par read)
  i=1      ! compteur de lignes
  do while(err==0)

     !- la ligne est stockee dans 250 caracteres au plus
     read(10,fmt='(1a250)',iostat=err) ligne

     !- pour le format de sortie, on compte le nombre de caracteres
     !- (en enlevant les blancs de debut et de fin de ligne)
     !- et on transforme ce nombre en type character dans ncar
     write(ncar,*) len(trim(adjustl(ligne)))
     !- format : 1 entier de 3 chiffres, 1 caractere (pour l'espace),1 chaine de ncar caractere
     chaine='(1I3,1a,1a'//trim(adjustl(ncar))//')'

     write(*,fmt=chaine) i,' ',trim(adjustl(ligne))

     i=i+1

  end do

end program ligne_fichier
```

Exercice 27.

3 Programmation modulaire

Exercice 28.

```
program prog_rptens_module

  !------------ but du programme ---------------------------!
  ! utilise la fonction rptens du module mod_ptens qui renvoie
  ! le produit tensoriel de deux vecteurs réels
  !-----------------------------------------------------------!

  use mod_ptens

  implicit none

  !-------- variables
  integer, parameter :: n=3
  integer :: i
  character(len=30) :: cn,chaine
  real, dimension(n) :: u,v
  real, dimension(n,n) :: w
  !--------


  !--- donnees
  u=(/1.,2.,3./) ; v=(/2.,1.,3./)

  !--- calcul : appel fonction
  w=rptens(u,v)

  !--- affichage
  write(cn,*) n
  chaine='('//trim(adjustl(cn))//'F10.5)'
  do i=1,n
     write(*,fmt=chaine) w(i,:)
  end do

end program prog_rptens_module
```

```
module mod_ptens

  implicit none

contains

  function rptens(u,v) result(w)

  !------------ fonctionnement -----------------------------!
  ! 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.
  !-----------------------------------------------------------!

    implicit none

    !-------- variables in
    real, dimension(:), intent(in) :: u,v
    !--------

    !-------- variable resultat
    real, dimension(size(u),size(v)) :: w
    !--------
```

```
    !-------- variables locales
    integer :: i,j
    !--------


    !--- calcul du produit tensoriel
    do i=1,size(u)
       do j=1,size(v)
          w(i,j)=u(i)*v(j)
       end do
    end do

  end function rptens

end module mod_ptens
```

Exercice 29.

```
program prog_matlab

  !------------ but du programme ----------------------------!
  ! utilise les fonctions diag, triu, tri et tridiag
  !------------------------------------------------------------!

  use mod_diag
  use mod_tri
  use mod_tridiag

  implicit none

  !-------- variables
  integer, parameter :: n=6
  integer :: i
  integer, dimension(n-1) :: u,w
  integer, dimension(n) :: v,d
  integer, dimension(n,n) :: A,Up,Lw,T
  real, dimension(n,n) :: Ar
  !--------


  !--- donnees
  u=(/1,2,3,4,5/) ; w=(/8,6,4,2,0/) ; v=(/1,1,1,1,1,1/)
  call random_number(Ar)
  A=int(Ar*50)

  print*,'u='
  write(*,fmt='(5I3)') u
  print*,'v='
  write(*,fmt='(6I3)') v
  print*,'w='
  write(*,fmt='(5I3)') w

  print*,'A='
  do i=1,n
     write(*,fmt='(6I3)') A(i,:)
  end do

  !--- appels fonctions
  d=diag(A)
  Up=triu(A)
  Lw=tril(A)
```

```
  T=tridiag(u,v,w)


  !--- affichage
  print*,'d='
  write(*,fmt='(6I3)') d

  print*,'U='
  do i=1,n
     write(*,fmt='(6I3)') Up(i,:)
  end do

  print*,'L='
  do i=1,n
     write(*,fmt='(6I3)') Lw(i,:)
  end do

  print*,'T='
  do i=1,n
     write(*,fmt='(6I3)') T(i,:)
  end do

end program prog_matlab
```

```
module mod_diag

  implicit none

contains

  function diag(A) result(d)

  !------------ fonctionnement ------------------------------!
  ! pour extraire la diagonale de A (matrice carrée)
  ! renvoyee dans le vecteur d
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    integer, dimension(:,:), intent(in) :: A
    !--------

    !-------- variable resultat
    integer, dimension(size(A,1)) :: d
    !--------

    !-------- variables locales
    integer :: i
    !--------


    !--- creation de la diagonale d
    do i=1,size(A,1)
       d(i)=A(i,i)
    end do

  end function diag

end module mod_diag
```

```
module mod_tri

  implicit none

contains

  function triu(A) result(U)

  !------------ fonctionnement ------------------------------!
  ! renvoie la partie triangulaire superieure de A dans U
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    integer, dimension(:,:), intent(in) :: A
    !--------

    !-------- variable resultat
    integer, dimension(size(A),size(A,2)) :: U
    !--------

    !-------- variables locales
    integer :: i,j
    !--------


    !--- creation de U
    U=0.0
    do i=1,size(A,1)-1
       do j=i+1,size(A,2)
          U(i,j)=A(i,j)
       end do
    end do

  end function triu


  function tril(A) result(L)

  !------------ fonctionnement ------------------------------!
  ! renvoie la partie triangulaire inferieure de A dans L
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    integer, dimension(:,:), intent(in) :: A
    !--------

    !-------- variable resultat
    integer, dimension(size(A),size(A,2)) :: L
    !--------

    !-------- variables locales
    integer :: i,j
    !--------


    !--- creation de L
    L=0.0
    do i=2,size(A,1)
```

```
        do j=1,i-1
           L(i,j)=A(i,j)
        end do
     end do

  end function tril

end module mod_tri
```

```
module mod_tridiag

  implicit none

contains

  function tridiag(u,v,w) result(T)

  !------------ fonctionnement ------------------------------!
  ! crée la matrice tridiagonale T ayant v sur la diagonale et
  ! u et w sur les diagonales sup et inf (u et w doivent être
  ! de dimension inférieure de 1 à celle de v).
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    integer, dimension(:), intent(in) :: u,v,w
    !--------

    !-------- variable resultat
    integer, dimension(size(v),size(v)) :: T
    !--------

    !-------- variables locales
    integer :: i,n
    !--------


    !--- creation de T
    n=size(v)
    T(1,1)=v(1) ; T(1,2)=u(1)
    do i=2,n-1
       T(i,i-1)=w(i-1)
       T(i,i)=v(i)
       T(i,i+1)=u(i)
    end do
    T(n,n-1)=w(n-1) ; T(n,n)=v(n)

  end function tridiag

end module mod_tridiag
```

4 Utilisation avancée

Exercice 30.

```
program prog_resistance

  use mod_resistance

  implicit none

  print*,'montage parallele : ',10.0.parallele.5.0
  print*,'montage serie : ',10.0.serie.5.0

end program prog_resistance
```

```
module mod_resistance

  implicit none

  interface operator(.parallele.)
     module procedure par
  end interface

  interface operator(.serie.)
     module procedure ser
  end interface

contains

  function par(R1,R2) result(R)

    implicit none

    real, intent(in) :: R1,R2
    real :: R

    R=1./(1./R1+1./R2)

  end function par

  function ser(R1,R2) result(R)

    implicit none

    real, intent(in) :: R1,R2
    real :: R

    R=R1+R2

  end function ser

end module mod_resistance
```

Exercice 31.

```
program prog_ptens_generique

  !------------ but du programme ----------------------------!
  ! utilise la fonction generique ptens et l'operateur .ptens.
  ! du module mod_ptens_bis qui renvoie le produit tensoriel
  ! de deux vecteurs réels ou entiers
  !------------------------------------------------------------!

  use mod_ptens_bis

  implicit none

  !-------- variables
  integer, parameter :: n=3
  integer :: i
  character(len=30) :: cn,format_r,format_i
  integer, dimension(n) :: a,b
  integer, dimension(n,n) :: c
  real, dimension(n) :: u,v
  real, dimension(n,n) :: w
  !--------


  !--- donnees
  a=(/1,2,3/) ; b=(/2,1,3/)
  u=(/1.,2.,3./) ; v=(/2.,1.,3./)

  !--- calcul : appel fonction generique
  c=ptens(a,b)
  w=ptens(u,v)

  !--- affichage
  write(cn,*) n
  format_r='('//trim(adjustl(cn))//'F10.5)'
  format_i='('//trim(adjustl(cn))//'I3)'
  do i=1,n
     write(*,fmt=format_r) w(i,:)
  end do
  print*,''
  do i=1,n
     write(*,fmt=format_i) c(i,:)
  end do
  print*,''

  !--- calcul : appel operateur
  c=a.ptens.b
  w=u.ptens.v

  !--- affichage
  do i=1,n
     write(*,fmt=format_r) w(i,:)
  end do
  print*,''
  do i=1,n
     write(*,fmt=format_i) c(i,:)
  end do



end program prog_ptens_generique
```

```
module mod_ptens_bis

  implicit none

  !----- declaration fonction generique ptens
  interface ptens
     module procedure rptens,iptens
  end interface

  !----- declaration operateur .ptens.
  interface operator(.ptens.)
     module procedure rptens,iptens
  end interface

contains

  function rptens(u,v) result(w)

  !------------ fonctionnement ------------------------------!
  ! 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.
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    real, dimension(:), intent(in) :: u,v
    !--------

    !-------- variable resultat
    real, dimension(size(u),size(v)) :: w
    !--------

    !-------- variables locales
    integer :: i,j
    !--------


    !--- calcul du produit tensoriel
    do i=1,size(u)
       do j=1,size(v)
          w(i,j)=u(i)*v(j)
       end do
    end do

  end function rptens

  function iptens(u,v) result(w)

  !------------ fonctionnement ------------------------------!
  ! renvoie le produit tensoriel de deux vecteurs entiers (de
  ! même taille) donnés en entrée. La taille des vecteurs
  ! n'est pas un argument de la fonction.
  !----------------------------------------------------------!

    implicit none

    !-------- variables in
    integer, dimension(:), intent(in) :: u,v
    !--------
```

```
    !-------- variable resultat
    integer, dimension(size(u),size(v)) :: w
    !--------

    !-------- variables locales
    integer :: i,j
    !--------


    !--- calcul du produit tensoriel
    do i=1,size(u)
       do j=1,size(v)
          w(i,j)=u(i)*v(j)
       end do
    end do

  end function iptens


end module mod_ptens_bis
```

Exercice 32. Attention : la seule véritable difficulté de cet exercice est de concevoir des algorithmes adaptés à la structure de stockage de la matrice (en particulier le produit de A par un vecteur). Dans le programme ci-dessous, je n'utilise jamais la matrice pleine associée, mais uniquement sa taille.

```
program prog_matrice_creuse

  use mod_matrice_creuse
  use mod_puiss

  implicit none

  !--- variables
  integer :: d,i,n,taille
  character(len=10) :: c
  real :: l
  real, dimension(:), allocatable :: x
  type(element), dimension(:), allocatable :: a

  !--- lecture
  d=11
  open(unit=d, file='a.mat')
  read(d,*) n
  allocate(a(1:n))
  call lecture(a,d)

  !--- calcul taille de la matrice pleine associee
  taille=0
  do i=1,n
     taille=max(taille,a(i)%indl,a(i)%indc)
  end do

  !--- trace
  print*,'trace de a = ',trace(a)

  !--- norme infinie
  print*,'norme infinie de a = ',norme_inf(a)

  !--- multiplication matrice-vecteur
  allocate(x(1:taille))
  call random_number(x)
```

```
  x=int(10.*x)

  print*,'Ax=',matvect(A,x)
  print*,'Ax=',A*x

  !--- calcul val.propre
  call puiss(A,l)
  print*,'lambda=',l

end program prog_matrice_creuse
```

```
module mod_matrice_creuse

  implicit none

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

  interface operator(*)
     module procedure matvect
  end interface

contains

  subroutine lecture(a,d)

    implicit none

    !--- arguments
    integer, intent(in) :: d
    type(element), dimension(:), intent(out) :: a

    !--- variables locales
    integer :: i

    !--- lecture dans le fichier connecte a d
    do i=1,size(a)
       read(d,*) a(i)%coef, a(i)%indl, a(i)%indc
    end do

  end subroutine lecture

  function trace(a) result(t)

    implicit none

    !--- arguments
    type(element), dimension(:), intent(in) :: a
    real :: t

    !--- variables locales
    integer :: i

    !--- calcul de la trace
    t=0.
    do i=1,size(a)
       if (a(i)%indl==a(i)%indc) then
          t=t+a(i)%coef
       end if
    end do
```

```
  end function trace

  function norme_inf(a) result(norme)

    implicit none

    !--- arguments
    type(element), dimension(:), intent(in) :: a
    real :: norme

    !--- variables locales
    integer :: i,taille
    real, dimension(:), allocatable :: y

    !--- calcul taille de la matrice pleine associee
    taille=0
    do i=1,size(a)
       taille=max(taille,a(i)%indl,a(i)%indc)
    end do

    !--- calcul de la norme
    norme=0.
    allocate(y(1:taille))
    y=0.
    do i=1,size(a)
       y(a(i)%indl)=y(a(i)%indl)+abs(a(i)%coef)
    end do
    norme=maxval(y)

    deallocate(y)

  end function norme_inf

  function matvect(a,x) result(y)

    implicit none

    !--- arguments
    real, dimension(:), intent(in) :: x
    type(element), dimension(:), intent(in) :: a
    real, dimension(1:size(x)) :: y

    !--- variables locales
    integer :: i

    !--- produit
    y=0.
    do i=1,size(a)
       y(a(i)%indl)=y(a(i)%indl)+a(i)%coef*x(a(i)%indc)
    end do

  end function matvect


end module mod_matrice_creuse
```