fcoppex | quanta 1

Séance 1: - commence par changer le mot de passe : PASSWD
- ordre :

```
cp  ~bardi/.cshrc  ~/
```
(cp : copie ; ~bardi/ : répertoire ; .cshrc : nom du fichier ; ~/ : dans notre répertoire principal)

- éditeur : nedit
- si on écrit : nedit & (le "et" commercial)
- nedit Fibo.f90 → édite un fichier particulier

(1) { f90 fibo.f90 → a.out
{ f90 fibo.f90 -o fibo.exe → fibo.exe

- autre manière (en 2 étapes)

(2) { f90 -c fibo.f90 → fibo.o
{ f90 fibo.o -o fibo.exe → fibo.exe

→ avantage : si on modifie que certains fichiers, on peut recompiler que les fichiers modifiés.

compiler une biblio : (p.54)

{ f90 -c biblio.f90
{ f90 -biblio.o prog1.f90 -o prog1.exe

ou bien :

{ f90 prog1.f90 biblio.f90 -o prog1.exe

NEDIT &
Utilisation de la disquette : mtools, mdir, ...
copies :
ordi → disq. mcopy prg.f90 a:
disq → ordi. mcopy a:prg.f90 *.*
(*.* : toutes fichiers)

éjecter : eject floppy0

dos2unix ; unix2dos

Séance 2: - remarque sur les tableaux. Tableaux de n dimension : c = a+b. phys 1712
- déclaration :
type, dimension( ) (max : 7 indices)

disq. → ordi :
mcopy a:prg.f90 prg.f90

- 1 dimension : 4:7 p.exemple
- si on ne donne que la fin, alors le début est automatiquement 1.
- exemples : (4,7) : 1,2,3,4 / 1,2,3,4,5,6,7
(4:7) : 4,5,6,7
(-1:2 ; 1:3) : -1,0,1,2 / 1,2,3

autre gestionnaire de fichiers :
programs — apps manager
— open windows
— a.w. file manager
⇒ file — check for floppy

dimension	rang	profil	taille
(4)	1	4	4
(5,3)	2	5,3	15
(10:20)	1	11	11
(-1:5, 0:2)	2	5,3	15

f90 fabo1.f90 -o fabo1.exe
-xlic_lib=sunperf

- opérations globales sur les tableaux : exemple : real, dimension(5) : a,b,c

c = a·b ⟺ do i = 1,5 ; c(i) = a(i)·b(i) ; end do (P.S. usuel)
c = a+b ⟺ do i = 1,5 ; c(i) = a(i)+b(i) ; end do

- exemple :

```
logical, dimension (5,5) :: compare
integer, dimension (5,5) :: V1, V2
compare = V1 < V2
```
← idée d'utiliser des ordinateurs parallèles

résultat : le tableau compare contient
$\begin{pmatrix} t & f & f & \dots \\ t & t & f & \\ f & t & \dots & \\ \vdots & & & \end{pmatrix}$ (i.e. compare chaque V1(i,j) et V2(i,j))

- grandes tailles des tableaux : p.34,35 : gestion dynamique : 1.10.7.

```
real, allocatable, dimension (:,:) :: x
                                  (laisse les valeurs libres)
read (*,n)        (connaît l'étendue du tableau)
allocate (x(n,n))   (crée la dimension du tableau)
deallocate (x)      (libère la mémoire)
```
etc, et on peut utiliser allocate n'importe quand
→ pas oublier de désallouer la mémoire.

Séance 3: - modules et subroutines. : opérations arithmétiques élémentaires
Y = (ax+b)·x+b : 4 op. élém.
Y = ax·x+bx+c : 5 op. élémentaires
} $Y = ax^2 + bx + c$

- unités de programmes : p.26 → 41.
- procédures intrinsèques : cos, sin, max, ... etc.
- procédures externes et internes :

program [nom]
use module → fait la connexion avec les modules qui font la base de donnée
le compilateur a besoin de savoir le plus vite possible quelles sont les données auxquelles il a accès
implicit none
-déclaration
interface → c.f. plus loin: liaison entre les paramètres arguments d'un sous-prg. et les valeurs de ces paramètres
end interface ← prg.
contain → procédures INTERNES : toutes les var. qu'elle utilise sont globales à l'unité de prg. ces procédures sont connues que à l'unité de prg. qui la contient.
end program

- procédures externes : - sous-programme
  - fonction
  - module
- construction procédures externes : function nom_de_la_fct / subroutine nom_de_subroutine } différence : dû le cas de la fct, le nom est une variable qui contient le résultat de la fonction
  ↖ le nom est simplement un identificateur ici
- Exemple : - on veut calculer un polynôme :

$$p = 3x^2 + 2x - 1$$

- avec une procédure de type fonction :

```
function p(x)
implicit none
real, intent(in)  :: p      ! pas nécessaire pour p car p est par définition intent(out)
real, intent(in)  :: x

p = 3.*x*x + 2.*x - 1
end function p
```

- dans le programme principal, on écrira :

```
z = p(a) + p(b)
```

équivalent

- exemple de la subroutine :

```
subroutine poly (p, x)
implicit none
real, intent(out) :: p
real, intent(in)  :: x
p = 3.*x*x + 2.*x - 1
end subroutine poly
```

- dans le programme principal, on aura :

```
call poly (pa, x)
call poly (pb, x)
z = pa + pb
```

- Remarque : module : base de données. Permet de rendre global un certain nb. d'opérations, p. exemple :
  - types (≠ important en fortran), variables ou constantes. Construction d'un module : identique à celle d'un prg.
  - le module ne fait que mettre ensemble des informations.

- Exemple : - module truc

```
complex , dimension (100,6)   :: gtx
real, dimension (100)         :: x
real, allocatable (:)         :: y
  :
```

biblio.f90

- utilisation :

```
use truc :  toutes les ressources accessibles
use truc, only : x, y
use truc, only : t => z, fct   ! utilise seulement z et fct, avec z qui va prendre le nom t.
```

- Remarque : - interface : définit une correspondance entre les arguments muets et réels. Utilisation :
  - procédures transmises en argument
  - exemple : mod_lib . dans une procédure subroutine : lorsque une procédure est un argument : subroutine ex_sub (z, f, u) → f est une fct. : le compilateur doit savoir que f n'est pas une variable mais une procédure → on doit dire explicitement que f est une fonction : dans l'interface on met la définition de la procédure : déclaration des arguments et des variables de f.
- Remarque : - on va mettre dans un même fichier tous les sous-programmes : une liste de modules d'un même fichier

- on fait un module : si on déclare une interface dans un module avec une fonction qui y réfère dans un autre module, on a pas besoin de faire un use module dans le premier module car la fonction fct n'est pas utilisée explicitement.

Séance 4 : utilisation de tableaux. Tableaux locaux d'étendue compatible.

Exemple :

```
subroutine échange (ta, tb)
    integer, dimension (:)                    :: ta, tb
    integer, dimension (size(ta,1))           :: temp
    temp = ta
    ta = tb
    tb = temp
end subroutine echange
```

utilise rarement : si on met rien = 1

p.34

à étendue que le tableau ta en argument

tableau sur lequel on veut faire la recherche

Remarque : - constructeur de tableau. p.14 Exemple : syntaxe : (/ liste_éléments /)

```
integer, parameter  :: lim = 9
integer             :: i
integer, dimension(5) :: t = (/2, 4, 3, 9, 11/)
:
t = (/(2*lim + i, i = 1, 3), 0, 0/)
```

remplit les 3 premières cases

p.14

- sections de tableaux : manipuler des parties de tableau : - l'indice, section, etc. (p. 15)

ex : section de tableau :

p.15

```
a(1, 1:n)      (portion de ligne)
a(1:m, j)      (portion de colonne)
a(i, :)        (ligne entière)
a(i, 1:n:3)    tous les 3e
a(i, 10:1:-1)  pas : à l'envers
```

de 1 à n

de 10 à 1

- expression tableaux et boucles (p.20)

```
do i=m,n
  tot = tot + a(i)
end do
```

⟺ `tot = sum(a(m:n))` (avec une instruction globale (procédure intrinsèque))

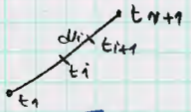
```
do i=1,n
  da(i) = a(i) - a(i-1)
end do
```

⟺ `da = a(1:n) - a(0:n-1)` (réparti autom. les élém. du tableau sur plusieurs processeurs : op. en //)

section de 1 à n ; 0 à (n-1)

relation de récurrence ⟺ boucle

Exemple : prg. longueur : N intervalles ; N+1 points



$\bar{t} = \{t_1, \dots, t_{N+1}\}$ pour 1 itération

$\bar{x} = \{x_1, \dots, x_{N+1}\} = \{f_x(t_1), \dots, f_x(t_{N+1})\} = \bar{f}_x(\bar{t})$

$\bar{y} = \dots = \bar{f}_y(\bar{t})$

$d\bar{x} = \{dx_1, \dots, dx_N\} = \{x_2 - x_1 ; \dots, x_{i+1} - x_i\}$

$d\bar{y} = \dots$

$d\bar{s} = \{ds_1, \dots, ds_N\} = \sqrt{d\bar{x}^2 + d\bar{y}^2}$

$L = \sum_i ds_i = \text{sum}(d\bar{s})$

⟹

```
function longueur (fctx, fcty, ta, tb, itmax, epi)
  implicit none
  integer, intent(in)          :: itmax
  real, intent(in)             :: ta, tb, epi
  real                         :: longueur
  interface
    function fctx(x)
      real, dimension(:), intent(in) :: x
      real, dimension(size(x)),      :: fct x
    end function fctx
    :
  end interface
  :
  integer                      :: it, i, ni
  real                         :: longold, longnew, eps
  real, dimension(:), allocatable :: delx, dely, dels, t, xt, yt
```

```
    :
iteration et test sur itmax
    :
  it = 0
  longold = 0.0
  do
     it = it + 1
      :
     ni = 2 ** (it-1)
     allocate (delx(ni), dely(ni), dels(ni))
     allocate (t(ni+1), xt(ni+1), yt(ni+1))
     t = (/ (ta + (tb-ta)*(i-1)/ni, i=1, ni+1) /)
     xt = fctx(t)
     yt = fcty(t)
     delx = xt(2:ni+1) - xt(1:ni)
     dely = yt(2:ni+1) - yt(1:ni)
     dels = sqrt(delx*delx + dely*dely)
     longueur = sum(dels)
      :
     err = longnew - longold
      :
     longold = longnew
     deallocate (delx, dely, dels, t, xt, yt)

  end do
  longueur = longnew
end fonction longueur
```

Séance 5 : procédure Jimpron :

```
implicit none
integer, intend (in)    :: itmax
real , intend (in)      :: a, b, eps
real , intend (out)     :: somme

interface
   fonction fct (x)
   real , intend (in)     :: x
   real                   :: fct
   end fonction fct
end interface
   :
  var. locales
   :
real, dimension (:), allocatable :: x_4, f_4
   :
it = 1
ni = 1
h = b-a
sigma_1 = ...
sigma_2 = ...
sigma_4 = ...
sum_new = h * ...

do
   it = it+1
     ... test sur it ...
   sum_old = sum_new
   ni = 2*ni
   h = h/2
   allocate (x_4(ni), f_4(ni))
   x_4 = (/ .../)
   f_4 = (/

   sum_new = h*...
   deallocate (x_4, f_4)
   erreur = ...
   if (err <= eps) exit
end do
somme = sum_new
```

1. Fibonacci

```
program fibonacci

IMPLICIT NONE
integer             :: i, io, choix, nbmax
real                :: x_n, x_n1, x_n2, r_n
real                :: yn, cplus, cmoins, alpha
real                :: nbor1, nbor2

nbor1 = 1./2. *(1 - sqrt(5.))
nbor2 = 1./2. *(1 + sqrt(5.))
alpha = nbor1

! Calcul analytique des nombres de Fibonacci

do

  write (*, '(" On vérifie la convergence vers le nombre d''or. &
  &Cette fois on a alpha qui est θgal au nombre d''or.")', advance = 'yes')
  write (*, '(" Nombre d''itérations ? (entier)")')
  read (*,*, iostat = io) nbmax
  write (*,'("  (n)   ", "  (Calc.anal.nb fibo)", "  (xn)  ", "    (rn:nb.or)  ")')
  x_n2 = 1.0
  x_n1 = nbor1
  do i = 2, nbmax
    x_n = x_n1 + x_n2
    r_n = x_n/x_n1
    x_n2 = x_n1
    x_n1 = x_n
    cplus = (1./(2.*sqrt(5.)))*(sqrt(5.) + (2*alpha -1))
    cmoins = (1./(2.*sqrt(5.)))*(sqrt(5.) - (2*alpha -1))
    yn = cplus * nbor2**i + cmoins * nbor1**i
    write (*,'(I5, Es16.5, "   ", Es16.5, "  ", F12.8)') i, yn, x_n, r_n
  end do
  write(*,'(" Nouvel essai ? 0 = oui ")')
  read(*,*, iostat = io) choix
  if (io /= 0 .or. choix /= 0) exit
                ↳ si qqch pas correct.
end do

stop 'terminé'

end program fibonacci
```

2. Feigenbaum

```
!valable pour les exercices 6 à 9 et 11 à 12

program Feigenbaum6

IMPLICIT NONE
real :: a
integer, parameter :: nb = 3
real, allocatable, dimension(:,:) :: xn
integer :: i,nmax,io
integer, dimension(nb) :: per

write (*,'("Feigenbaum6: xn+1 = a*xn*(1 - xn)")')

boucle_principale: do

  write (*, '("Valeur du paramètre a (réel entre 0 et 4) ")', advance = 'no')
  read (*,*, iostat = io) a
  if (a < 0 .or. a > 4) exit

  write (*, '("Valeur de nmax ")', advance = 'no')
  read (*,*, iostat = io) nmax
  if (io /= 0) exit
    ocate (xn(nb,0:nmax))
  valeurs: do i = 1, nb
    write (*, '("Valeur de x0", i1," (réel entre 0 et 1) ")', advance = 'no') i
    read (*,*, iostat = io) xn(i,0)
    if (xn(i,0) < 0 .or. xn(i,0) > 1) exit
  end do valeurs

  write (*, '(" itération numéro     xn(1)     xn(2)     xn(3)     ")')

  calculs: do i = 1, nmax
    xn(:,i) = a*xn(:,i-1)*(1. - xn(:,i-1))
    write (*, '("    ",i5,"              ",f7.5,"     ",f7.5, "     ", f7.5)') &
    i, xn(1,i), xn(2,i), xn(3,i)
  end do calculs

  periode1: do i = 1,nmax
    if (xn(1,nmax) == xn(1,nmax-i)) then
      per(1) = i
      exit periode1
    end if
    if (i == nmax) then
      per(1) = 666
      exit periode1
    end if
  end do periode1

  periode2: do i = 1,nmax
    if (xn(2,nmax) == xn(2,nmax-i)) then
      per(2) = i
      exit periode2
    end if
    if (i == nmax) then
      per(2) = 666
      exit periode2
    end if
  end do periode2
```

allocate

les 3 lignes de la matrice automatiquement

comparaison : ==

```
  periode3: do i = 1,nmax
    if (xn(3,nmax) == xn(3,nmax-i)) then
      per(3) = i
      exit periode3
    end if
    if (i == nmax) then
      per(3) = 666
      exit periode3
    end if
  end do periode3

  write (*,*)
  write (*, '("Période1: ",i5,"     Période2: ",i5,"     Période3: ",i5)') &
  per(1), per(2), per(3)
  write (*,*)
  write (*, '("Si la période est 666, alors le comportement est chaotique, &
  &ou bien il n''y a pas encore convergence.")')
  write (*, '("Voulez-vous recommencer? 1 pour oui, autre chose sinon ")',&
  advance = 'no')
  read (*,*, iostat = io) i
  if (i /= 1) exit

end boucle_principale

deallocate (xn)

stop 'terminé'

end program Feigenbaum6
```

2. Feigenbaum 2. feig10. f90

```
!Attention: prg pour a = 3.61; x01 = 0.3999; x02 = 0.4; x04 = 0.4001

program Feigenbaum10

IMPLICIT NONE
real :: a
integer, parameter :: nb = 3
real, dimension (nb,0:500) :: xn
integer :: i, nmax, io, j
integer, dimension (nb) :: per
logical :: intervale

write (*,'(" FeigenBaum 6: xn+1 = a*xn*(1 - xn) ")')   ou bien :  write (*,*) "texte"

boucle_principale: do
  write (*,'("Valeur du paramètre a (réel, entre 0 et 4) ")', advance = 'no')
  read (*, *, iostat = io) a
  if (a < 0 .or. a > 4) exit

  valeurs: do i = 1, nb
    write (*,'("Valeur de x0", i1," (réel, entre 0 et 1) ")', advance = 'no') i
    read (*, *, iostat = io) xn (i,0)
    if (xn (i,0) < 0 .or. xn (i,0) > 1) exit
  end do valeurs

  write (*,'("Valeur de nmax (entier) ")', advance = 'no')
  read (*, *, iostat = io) nmax
  if (io /= 0) exit
  write (*, '(" itération numéro     xn(1)      xn(2)       xn(3)    ")')

  calculs: do i = 1, nmax
    xn (1,i) = a * xn (1,i-1) * ( 1 - xn (1,i-1))
    xn (2,i) = a * xn (2,i-1) * ( 1 - xn (2,i-1))
    xn (3,i) = a * xn (3,i-1) * ( 1 - xn (3,i-1))
    write (*, '("  ", i3,"                ", f7.5,"     ", f7.5, "     ", f7.5)')&
    i, xn (1,i), xn (2,i), xn (3,i)
  end do calculs

  intervale = .true.
  Verif_intervale: do i = 1, nb
    do j = 1, (nmax / 2)
      if ( (xn (i,2*j + 1) < 0.78247) .or. (xn (i,2*j + 1) > 0.9025) ) then
        intervale = .false.
        exit Verif_intervale
      end if
      if ( (xn (i,2*j) < 0.31765) .or. (xn (i,2*j) > 0.61445) )  then
        intervale = .false.
        exit Verif_intervale
      end if
    end do
  end do Verif_intervale

  i = 1
  periode1: do
    if (xn (1,nmax) == xn (1,nmax - i)) then
      per (1) = i
      exit periode1
    end if
    i = i + 1
    if (i == nmax) then
```

$xn(:,i) = a \cdot xn(:,i-1) \cdot (1 - xn(:,i-1))$

```
      per (1) = 666
      exit periode1
    end if
  end do periode1

  i = 1
  periode2: do
    if (xn (2,nmax) == xn (2,nmax - i)) then
      per (2) = i
      exit periode2
    end if
    i = i + 1
    if (i == nmax) then
      per (2) = 666
      exit periode2
    end if
  end do periode2

  i = 1
  periode3: do
    if (xn (3,nmax) == xn (3,nmax - i)) then
      per (3) = i
      exit periode3
    end if
    i = i + 1
    if (i == nmax) then
      per (3) = 666
      exit periode3
    end if
  end do periode3

  write (*,*)
  write (*, '("Période1: ",i3,"    Période2: ",i3,"     Période3: ",i3)') &
  per(1), per(2), per(3)
  write (*,*)
  write (*, '("Si la période est 666, alors le comportement est chaotique, &
  &ou bien il n''y a pas encore convergence.")')
  write (*, '("Est-ce que les termes des suites appartiennent alternativement à &
  &l''intervale [0.317657437..., 0.614456961] et α l''intervale [0.782471..., 0.9025] ?")')
  if (intervale) then
  write (*, '("   OUI   ")')
  else
  write (*, '("   NON   ")')
  end if
  write (*, '(" Voulez-vous recommencer? 1 pour oui, autre chose sinon ")',&
  advance = 'no')
  read (*, *, iostat = io) i
  if (i /= 1) exit

end do boucle_principale
stop 'terminé'
end program Feigenbaum10
```

3flott1.f90

```
module fonctions
  IMPLICIT NONE
  contains

    function fct1(x)
      real, intent(in) :: x
      real :: fct1
      fct1 = x**2 - 2.
      return
    end function fct1

end module fonctions

!------------------------------------------------------------------------
!------------------------------------------------------------------------

program flotteur1
  use biblio
  use fonctions
  implicit none

  real :: racine,a,b,eps
  integer :: itmax,io

  write (*,"(/////////////////////////)")
  write (*,*) "Experimentation numerique: flotteur spherique"
  write (*,*)
  write (*, '("Extremites de l''intervalle [a,b] ? ")', advance = 'no')
  read (*,*, iostat = io) a,b
  if (io /= 0) stop 'Erreur sur l''intervalle'
  write (*, '("Nombre maximum de bissections (entier) ? ")', advance = 'no')
  read (*,*, iostat = io) itmax
  if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
  write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
  read (*,*, iostat = io) eps
  if (io /= 0) stop 'Erreur sur epsilon'
  call bissect(a,b,itmax,eps,racine,fct1)

  write (*,'("Racine de l''equation : ",f8.5)') racine
  write (*,*) "Appuyer sur ENTER pour terminer le programme"
  read (*,*)
  stop 'termine'
end program flotteur1
```

définit fct1(x) qui est la fonction qui pourra être utilisée avec bissect. Il faut que fct1 et celle de bissect aient la même interface

utilise la bibliothèque générale

efface l'écran (fait / fois en return)

quitte le programme

appelle la routine bissect. de biblio

3flotf7.f90

```
module fonctions
IMPLICIT NONE
real :: rho, rho2
contains

function fct1(x)
real, intent(in) :: x
real :: fct1
fct1 = x**3 - 3*x**2 + 4*rho
return
end function fct1

function fct2(x)
real, intent(in) :: x
real :: fct2
fct2 = x**3 - 3*x**2 + 4*rho -rho2*(1+x)*(2-x)**2
return
end function fct2

function test(x)
real, intent(in) :: x
real :: test
te = x**2 - 2
return
end function test

end module fonctions

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

program flotteur7
use biblio
use fonctions
IMPLICIT NONE

real :: racine,a,b,eps,r
integer :: itmax,io,i
real, dimension(2,10) :: valeurs

write (*,"(/////////////////////////)")
wr (*,*) "Experimentation numerique: flotteur spherique"
write (*,*)
write (*,*) "Calcul de la profondeur d'immersion h d'une sphère de masse volumique &
& relative rho = rhosphère/rhoH20, et de rayon r (avec poussee d'air)."
write (*,*)

write (*, '("Extremites de l''intervalle [a,b] ? ")', advance = 'no')
read (*,*, iostat = io) a,b
if (io /= 0) stop 'Erreur sur l''intervalle'
write (*, '("Nombre maximum de bissections (entier) ? ")', advance = 'no')
read (*,*, iostat = io) itmax
if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*, iostat = io) eps
if (io /= 0) stop 'Erreur sur epsilon'

!write (*,'("Masse volumique relative du solide rho ?")', advance = 'no')
!read (*,*, iostat = io) rho
!if (io /= 0) stop 'Erreur sur la donne de la masse volumique relatice'
```

real :: rho, rho2 ces variables déclarées dans tout module peuvent être réutilisées sans déclaration dans le programme qui les appelle. Cela permet de garder les mêmes déclarations d'interface que dans bisect.

} différentes fonctions. Chacune peut être utilisée avec bisect.

```
write (*,'("Masse volumique relative de l''air rho2 ?")', advance = 'no')
read (*,*, iostat = io) rho2
if (io /= 0) stop 'Erreur sur la donne de la masse volumique relatice'
write (*,'("Rayon de la sphère r[m] ?")', advance = 'no')
read (*,*, iostat = io) r
if (io /= 0) stop 'Erreur sur la donne du rayon'

rho = 0.01
boucle: do i = 1, 10
  rho = 0.01*i
  call bissect(a,b,itmax,eps,racine,fct2)
  valeurs(1,i) = racine * r
  call bissect(a,b,itmax,eps,racine,fct1)
  valeurs(2,i) = racine * r
end do boucle

write (*,*) "rho shpere       h1[m] sans pous. air    h1[m] avec pous. air  &
& h1-h2 [m]    "

affiche: do i=1,10
  write (*,'("  ",f9.6,"      ",f9.6,"              ",f9.6,"             ",f9.6)') &
  .01,valeurs(2,i),valeurs(1,i),Abs(valeurs(1,i)-valeurs(2,i))
end do affiche

write (*,*)
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)
stop 'termine'
end program flotteur1
```

3flott99.f90

```
module fonctions
IMPLICIT NONE
real :: rho
contains

function fct1(x)
real, intent(in) :: x
real :: fct1
fct1 = x*x*(x-3.) + 4.*rho
return
end function fct1

end module fonctions

!------------------------------------------------------------------------
!------------------------------------------------------------------------

program flotteur99
use biblio
use fonctions
IMPLICIT NONE
real :: racine,a,b,eps,r
integer :: itmax,io,i,nb,choix
character (len=15) :: fichier   va être le nom du fichier à créer

write (*,"(/////////////////////////)")
write (*,*) "Experimentation numerique: flotteur spherique"
write (*,*)
write (*,*) "Enregistre dans un fichier la profondeur d'immersion en fonction de rho&
& la masse volumique relative de la sphere"
write (*,*)

write (*,*) "Valeurs standart, donner 0, sinon autre chose ? "
read (*,*) choix
if (choix == 0) then
    r = .5
    a = 0
    b = 1
    itmax = 50
    eps = 1e-5
    nb = 10
    e
    write (*, '("Extremites de l''intervalle [a,b] ? ")', advance = 'no')
    read (*,*, iostat = io) a,b
    if (io /= 0) stop 'Erreur sur l''intervalle'
    write (*, '("Nombre maximum de bissections (entier) ? ")', advance = 'no')
    read (*,*, iostat = io) itmax
    if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
    write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
    read (*,*, iostat = io) eps
    if (io /= 0) stop 'Erreur sur epsilon'

    write (*,'("Nombre de points (entier > 0) ?")', advance = 'no')
    read (*,*, iostat = io) nb
    if (io /= 0 .or. nb <= 0) stop 'Erreur sur la donne de nbmax'
    write (*,'("Rayon de la sphere r [m] ?")', advance = 'no')
    read (*,*, iostat = io) r
    if (io /= 0) stop 'Erreur sur la donne du rayon'
end if
```

```
fichier = '3flott99.dat'
open (1, file = fichier)

boucle: do i = 1, nb
  rho = i/nb
  call bissect(a,b,itmax,eps,racine,fct1)
  write (1,*) rho, r*racine
end do boucle

close (1)

write (*,*)
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)
stop 'termine'
end program flotteur99
```

donne un nom à fichier

ouvrir de 1 : le disque dur, le nom dans lequel a enregistre est ce qui est contenu de fichier

enregistre sur le disque (rho, r*racine) par ligne dans fichier

ferme le fichier

4cable1.f90

```
module fonctions
IMPLICIT NONE
contains

function fctx(t)
  real, dimension(:), intent(in) :: t
  real, dimension(size(t)):: fctx
  fctx = cos(t)
  return
end function fctx

function fcty(t)
  real, dimension(:), intent(in) :: t
  real, dimension(size(t)):: fcty
  fcty = sin(t)
  return
end function fcty

end module fonctions

!---------------------------------------------------------------------
!---------------------------------------------------------------------

program cable1
use biblio
use fonctions
implicit none

real :: ta,tb,eps,long,pi
integer :: itmax,io

write (*,"(/////////////////////////)")
write (*,*) "Experimentation numerique: TP.4: cable souple homogene"
write (*,*)

pi = 4*Atan(1.)
ta = 0
tb = 2*pi

write (*, '("Nombre maximum d''iterations (entier) ? ")', advance = 'no')
read (*,*, iostat = io) itmax
if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*, iostat = io) eps
if (io /= 0) stop 'Erreur sur epsilon'

long = longueur(fctx,fcty,ta,tb,itmax,eps)

write (*,*)
write (*,'("Longueur de la courbe parametree : ",f8.5)') long
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)
stop 'termine'

end program cable1
```

size(t,1) = size(t) par défaut, i.e. le rang est 1

manière de définir π

utilisation de la fonction longueur

4 cable 5.f90

```
module fonctions
IMPLICIT NONE
contains

function fct(x)
  real, intent(in) :: x
  real :: fct
  fct = 4*x**3 -9*x**2 -30*x + 5
  return
end function fct

function tst(x)
  real, intent(in) :: x
  real :: tst
  tst = x**2 - 1
  return
end function tst

end module fonctions

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

program cable5
use biblio
use fonctions
implicit none

real :: a,b,eps,min
integer :: itmax,io

write (*,"(/////////////////////////)")
write (*,*) "Experimentation numerique: TP.4: cable souple homogene"
write (*,*)

!write (*, '("Intervale de localisation [a,b] ? ")', advance = 'no')
!read (*,*, iostat = io) a,b
!if (io /= 0) stop 'Erreur sur la donne de l''intervalle'

a = 0
b = 4

write (*, '("Nombre maximum d''iterations (entier) ? ")', advance = 'no')
read (*,*, iostat = io) itmax
if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*, iostat = io) eps
if (io /= 0) stop 'Erreur sur epsilon'

min = minimum(a,b,itmax,eps,fct)  utilisation de la fonction minimum

write (*,'("Minimum de l''equation : ",f8.5)') min
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)
stop 'termine'

end program cable5
```

Schoir3.f90

```
module fonctions
IMPLICIT NONE
contains

function fexacte(x)
  real, dimension(:), intent(in) :: x
  real, dimension(size(x)) :: fexacte
  fexacte = 1./(x**5*(exp(1./x)-1.))
  return
end function fexacte

function fgrand(x)
  real, dimension(:), intent(in) :: x
  real, dimension(size(x)) :: fgrand
  fgrand = x**3/(exp(x)-1.)
  return
end function fgrand

function fdl(x)
  real, dimension(:), intent(in) :: x
  real, dimension(size(x)) :: fdl
  fdl = x**2*(1.-x/2.)
  return
end function fdl

end module fonctions

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

program cnoir3
use biblio
use fonctions
implicit none

real :: p1,p2,eps,I2cte
integer :: itmax,io,i
real, dimension(4) :: a,I1,I2

write (*,"(/////////////////////////)")
write (*,*) "Experimentation numerique: TP.5: Rayonnement du corps noir"
write (*,*)

write (*, '("Nombre maximum d''iterations (entier) ? ")', advance = 'no')
read (*,*, iostat = io) itmax
if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*, iostat = io) eps
if (io /= 0) stop 'Erreur sur epsilon'

a = (/(0.25*2**i,i=0,3)/)
p1 = 0.025
p2 = 0.005
!car on a vu a l'exercice precedent que la fct est nulle pour des valeurs plus petites que 0.025
do i=1,4
  call simpson(p1,a(i),itmax,eps,fexacte,I1(i))
end do
do i=1,4
  call simpson(p2,1/a(i),itmax,eps,fgrand,I2(i))
```

```
end do
p1=0
call simpson(p1,p2,itmax,eps,fdl,I2cte)
I2 = I2 + I2cte

write (*,*) "  a         I1(a)       I2(a)        Intégrale"
do i=1,4
  write (*,'(" ",f5.3,"    ",f8.5,"      ",f8.5,"     ",f8.5)') a(i),I1(i),I2(i),I1(i)+I2(i)
end do
write (*,*)
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)

stop 'termine'
end program cnoir3
```

5cnoir9.f90

```
module fonctions
IMPLICIT NONE
contains

function f1(x)
  real, dimension(:), intent(in) :: x
  real, dimension(size(x)) :: f1
  f1 = log(x)**3/(x-1)
  return
end function f1

end module fonctions

!-----------------------------------------------------------------------
!-----------------------------------------------------------------------

program cnoir9
use biblio
use fonctions
implicit none

real :: a,b,eps,Int
integer :: itmax,io

write (*,"(/////////////////////////)")
write (*,*) "Experimentation numerique: TP.5: Rayonnement du corps noir"
write (*,*)

write (*, '("Nombre maximum d''iterations (entier) ? ")', advance = 'no')
read (*,*, iostat = io) itmax
if (io /= 0) stop 'Erreur sur le nombre maximum d''iterations'
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*, iostat = io) eps
if (io /= 0) stop 'Erreur sur epsilon'

a=0
b=1
call trois_points(a,b,itmax,eps,f1,Int)

write (*,*)
write (*,'("Integrale 1 = ",f8.5)') int
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)

stop 'termine'
end program cnoir9
```

5cnoir 90.f90

Excellent exemple qui montre l'utilisation des 5 routines d'intégration.

```
!Pour tester les méthodes d'intégration

module fonctions
IMPLICIT NONE
contains

function f1(x)
  real, dimension(:), intent(in) :: x
  real, dimension(size(x)) :: f1
  f1 = (log(x)**3)/(x-1)
  return
end function f1

function f2(x)
  real, intent(in) :: x
  real :: f2
  f2 = (log(x)**3)/(x-1)
  return
end function f2

end module fonctions

!--   ----------------------------------------------------------------
!----------------------------------------------------------------------

program cnoir90
use biblio
use fonctions
implicit none

real :: a,b,eps
real :: Int1,Int2,Int3,Int4,Int5
integer :: itmax

write (*,"(/////////////////////////)")
write (*, '("Nombre maximum d''iterations (entier) ? ")', advance = 'no')
read (*,*) itmax
write (*, '("Erreur relative epsilon (epsilon = (xn - nx-1)/xn ) ? ")', advance = 'no')
read (*,*) eps

a=0.00001
b=6.   999
call simpson(a,b,itmax,eps,f1,Int1)
call simpson2(a,b,eps,f2,Int2)
call simpson3(a,b,itmax,eps,f2,Int5)
a=0
b=1
call trois_points(a,b,itmax,eps,f1,Int3)
call trois_points2(a,b,eps,f2,Int4)

write (*,*)
write (*,'("Integrale Simpson1 = ",f12.8)') int1
write (*,'("Integrale Simpson2 = ",f12.8)') int2
write (*,'("Integrale trois_points1 = ",f12.8)') int3
write (*,'("Integrale trois_points2 = ",f12.8)') int4
write (*,'("Integrale Simpson3 = ",f12.8)') int5

write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)
```

pour :
- simpson
- trois_points

pour :
- simpson2
- trois_points 2
- simpson 3

} méthodes adaptives

stop 'termine'
end program cnoir90