```
!Bibliothèque Fortran 90, Francois Coppex, DP, EPFL, 1998-1999.
!---------------------------------------------------------------------
!---------------------------------------------------------------------
!subroutine bissect(a,b,itmax,eps,racine,fct)
!subroutine newton(a,itmax,eps,racine,fct,fct_derivee)
!function longueur(fctx,fcty,ta,tb,itmax,eps)
!function minimum(a,b,itmax,eps,fct)
!subroutine simpson(a,b,itmax,eps,fct,somme)
!subroutine trois_points(a,b,itmax,eps,fct,somme)
!recursive subroutine simpson2(a,b,eps,fct,somme)
!recursive subroutine trois_points2(a,b,eps,fct,somme)
!subroutine simpson3(a,b,itmax,eps,fct,somme)
!subroutine rk4(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
!subroutine rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
!subroutine simptab(y,h,n,somme)


!Impression: taille 8 avec Word, texte.
!nom du module: c'est-la que on va mettre toutes les subroutines et fonctions
!d'ordre général

module biblio
     implicit none

!par défaut, tout est en PUBLIC, i.e. public :: bissect. Si on veut ne pas
!pouvoir accéder à des routines, on écrit: private :: bissect

  contains

! .
! .
! .
!end module biblio
```

```
!-----------------------------------------------------------------------
!Routine de m,thode de la bissection pour chercher les racines d'une ,qn
!Ce qu'il faut demander avant d'initialiser la routine:
!a,b: intervales o- on a localis, un z,ro
!itmax: le nombre maximum d'it,rations, apr$s itmax le prg stoppe
!eps: pr,cision relative entre l'it,ration n et n+1
!racine: c'est la racine de l',quation qui sera stoqu,e dans cette variable
!fct: c'est la fonction dont on cherche la racine: cette fonction doit ^tre
!uniquement fonction de la variable (x) et doit avoir la m^me d,claration
!d'inteface que ci-dessous
!limitations: pas capable de trouver les racines des ,quations du type x^2=0

subroutine bisect(a,b,itmax,eps,racine,fct)
real, intent(in) :: a,b,eps
integer, intent(in) :: itmax
real, intent(out) :: racine
real :: xn,xp,xm
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
end interface
integer :: nbit = 1

if (fct(a)*fct(b) > 0) then
  write (*,*) "bissection: condition aux extremit,s permettent pas &
  &d'appliquer la methode."
  stop 'erreur de bissection'
end if

if (fct(a) <= 0) then
  xn = a
  xp = b
  else
    xp = a
    xn = b
end if
xm = (a+b)/2.
write (*,*) "Iteration no          xm              g(xm)       Erreur relative"
write (*,*) "-----------------------------------------------------------------"
iteration: do
  write (*,'("     ",i4,"        ",f8.5,"       ",f8.5,"     ",f8.5)')&
  nbit,xm,fct(xm),Abs((xm-xp)/xm)
  if (fct(xm) <= 0) then
    xn = xm
  else
    xp = xm
  end if
  xm = (xn + xp) / 2.
  if (Abs((xm-xp)/xm) < eps) then
    racine = xm
    write (*,'("     ",i4,"        ",f8.5,"       ",f8.5,"     ",f8.5)')&
    nbit,xm,fct(xm),Abs((xm-xp)/xm)
    exit iteration
  end if
  if (nbit >= itmax) then
    write (*,*) "bissection: pas assez d'iterations pour obtenir epsilon desire"
    racine = xm
    exit iteration
  end if
  nbit = nbit + 1
end do iteration

write (*,*)
write (*,*) 'fin de bissection'
write (*,*)

return
end subroutine bisect
```

fonction à donner

variable d'entrée (non modifiée)

variable de sortie qui sera modifiée

pour donner une fonction en toute généralité qui se trouve à l'extérieur du fichier
Permet d'appliquer fct(x) comme une variable

```
!-----------------------------------------------------------------------
!Routine de m,thode de Newton pour chercher les racines d'une ,qn
!Ce qu'il faut demander avant d'initialiser la routine:
!a: le point d'o- on veut partir pour la recherche du z,ro
!itmax: le nombre maximum d'it,rations, apr%s itmax le prg stoppe
!eps: pr,cision relative entre l'it,ration n et n+1
!racine: c'est la racine de l',quation qui sera stoqu,e dans cette variable
!fct: c'est la fonction dont on cherche la racine: cette fonction doit ^tre
!uniquement fonction de la variable (x) et doit avoir la m^me d,claration
!d'inteface que ci-dessous
!fct_derivee: la d,riv,e de fct.-M^mes commentaires.
!limitations: - d,riv,e nulle
!             - fonctions … plusieurs points d'inflextion

subroutine newton(a,itmax,eps,racine,fct,fct_derivee)
real, intent(in) :: a,eps
integer, intent(in) :: itmax
real, intent(out) :: racine
real :: x_new,x_old
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
  function fct_derivee(x)
    real, intent(in) :: x
    real :: fct_derivee
  end function fct_derivee
end interface
integer :: nbit = 1

x_old = a
write (*,*) "Iteration no          xn                g(xn)         Erreur absolue"
write (*,*) "-------------------------------------------------------------------"
iteration: do
  x_new = x_old - fct(x_old)/fct_derivee(x_old)
  write (*,'("     ",i4,"          ",f8.5,"          ",f8.5,"        ",f8.5)')&
  nbit,x_new,fct(x_new),Abs(x_new-x_old)
  if (Abs(x_new-x_old) < eps) then
    racine = x_new
    exit iteration
  end if
  if (nbit >= itmax) then
    write (*,*) "Newton: pas assez d'iterations pour obtenir epsilon desire"
    racine = x_new
    exit iteration
  end if
  x_old = x_new
  nbit = nbit + 1
end do iteration

write (*,*)
write (*,*) 'fin de Newton'
write (*,*)

return
end subroutine newton
```

```
!-----------------------------------------------------------------------
!Fonction qui calcule la longueur d'un arc param,tr, par une ,quation
!param,trique G = (fctx(t);fcty(t))
!Ce qu'il faut demander avant d'initialiser la routine:
!ta,tb: intervale de d,finition de la variable t
!itmax: le nombre maximum d'it,rations, aprSs itmax le prg stoppe. Le nb
!d'intervales N est doubl, … chaque it,ration
!eps: pr,cision relative entre l'it,ration n et n+1
!fctx,fcty: fonctions d,finissant l'arc param,trique: ces fonctions doivent
!avoir le m^me interface que ci-dessous
!si on veu une fonction y = f(t) normale, alors la parametrisation pour
!fctx est fctx = t

function longueur(fctx,fcty,ta,tb,itmax,eps)
real :: longueur
real, intent(in) :: ta,tb,eps
integer, intent(in) :: itmax
interface
  function fctx(t)
    real, dimension(:), intent(in) :: t
    real, dimension(size(t)) :: fctx
  end function fctx
  function fcty(t)
    real, dimension(:), intent(in) :: t
    real, dimension(size(t)) :: fcty
  end function fcty
end interface
integer :: it,i,ni
real :: longold, longnew
real, dimension(:), allocatable :: delx,dely,dels,t,xt,yt

write (*,*) "Iteration      Points      Integrale        Erreur relative"
write (*,*) "-------------------------------------------------------------"
it = 0
longold = 0.0
iteration: 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)
  longnew = sum(dels)
  deallocate(delx,dely,dels,t,xt,yt)
  write (*,'(i3,"              ",i6,"      ",f10.5,"          ",f8.5)') it,ni,longnew,Abs((longold-
longnew)/longnew)
  if (Abs((longold-longnew)/longnew) < eps) then
    exit iteration
  end if
  if (it >= itmax) then
    write (*,*) "Longueur: pas assez d'iterations pour obtenir epsilon desire"
    exit iteration
  end if
  longold = longnew
end do iteration

longueur = longnew
write (*,*)
write (*,*) 'fin de longueur'
write (*,*)

return
end function longueur
```

variable d'entrée : vecteur de taille variable

fonction : elle doit être de la même taille que t, donc on met (size(t))

on peut faire plusieurs allocate en 1 ligne : la dimension est de ni : dels(ni)

équivalent à faire : do $i=1, ni+1$
$t(i) = ta + (i-1)\frac{tb-ta}{ni}$ (i.e. $t(i)=ta+i\Delta t$)
end do
syntaxe normale : $t = (/(\ldots, j=1, max)/)$

manipulation de tableaux :
vecteur de 2 à (ni+1) - vecteur de 1 à ni

somme de tous les éléments d'un tableau

on peut faire plusieurs deallocate en 1 ligne.

```
!-----------------------------------------------------------------------
!fonction qui calcule le minimum d'une fonction.
!La fonction donne la valeur x du minimum de f(x)
!A donner:
!l'intervale [a,b] dans lequel on a localis, un minimum
!itmax: le nb maximum d'it,rations
!eps: la pr,cision relative … l'it,ration n et n+1
!fct: la fonction de memé interface que ci-dessous
!remarque: attention a l'intervale de definition si a = 0 !!!
!remarque: pour calculer le maximum, on r,soud pour -f(x)

function minimum(a,b,itmax,eps,fct)
real :: minimum
real, intent(in) :: a,b,eps
integer, intent(in) :: itmax
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
end interface
real :: g,d,x,x2,an,bn
integer :: i

write (*,*) "   Iteration        Minimum             Erreur relative"
write (*,*) "-------------------------------------------------------"
i = 0
an = a
bn = b
iteration: do
  i = i+1
  x = (bn+an)/2.
  g = an + (bn-an)/3.
  d = bn - (bn-an)/3.
  if (fct(g) >= fct(an)) then
    bn = g
  else
    if (fct(d) >= fct(g)) then
      bn = d
    else
      if (fct(bn) >= fct(d)) then
        an = g
      else
        an = d
      end if
    end if
  end if
  x2 = (bn+an)/2.
  write (*,'("  ",i4,"             ",f10.5,"           ",f10.5,"      ")') &
  i,x2,abs((x-x2)/x2)
  if (abs((x-x2)/x2) < eps) then
    exit iteration
  end if
  if (i >= itmax) then
    write (*,*) "Minimum: pas assez d'iterations pour obtenir epsilon desire"
    exit iteration
  end if
end do iteration

minimum = x2
write (*,*)
write (*,*) 'fin de minimum'
write (*,*)

return
end function minimum
```

```
!-----------------------------------------------------------------------
!subroutine qui calcule une int,grale selon la formule de Simpson d'ordre 4
!r,utilise les valeurs pr,c,dentes lors des it;rations
!A donner:
!a,b : intervales d'int,gration
!itmax: nb maximum d'it,rations
!eps: erreur relative exig,e sur la convergence
!fct: fonction: !!!! DOIT ETRE DU MEME INTERFACE QUE CI-DESSOUS !!!!!
!somme: valeur de sortie de l'int;grale

subroutine simpson(a,b,itmax,eps,fct,somme)
real, intent(out) :: somme
real, intent(in) :: a,b,eps
integer, intent(in) :: itmax
interface
  function fct(x)
    real, dimension(:), intent(in) :: x
    real, dimension(size(x)) :: fct
  end function fct
end interface
real :: sold,snew,s1,s2,s4,hn
integer :: it,ni,i
real, dimension(1:2) :: bords
real, dimension(:), allocatable :: t4

write (*,*) "Iteration   Points     Integrale   Erreur Richardson    Erreur relative"
write (*,*) "-------------------------------------------------------------------------"
it = 0
sold = 0.0
hn = b-a
bords = (/a,b/)
s1 = sum(fct(bords))
iteration: do
  it = it+1
  ni = 2**(it-1)
  hn = hn/2.
  allocate(t4(ni))
  s2 = s2 + s4
  t4 = (/(a + (i-1/2.)*hn,i=1,ni)/)
  s4 = sum(fct(t4))
  deallocate(t4)
  if (it == 1) then
    hn = b-a
    s2 = 0
    bords = (/(a+b)/2.,(a+b)/2./)
    bords = fct(bords)
    s4 = bords(1)
  end if
  snew = (hn/6.)*(s1+2*s2+4*s4)
  write (*,'(i3,"          ",i6,"   ",f12.5,"    ",f10.5,"          ",f10.5)') &
  it,ni,snew,Abs((sold-snew)/15.),Abs((sold-snew)/snew)
  if (Abs((sold-snew)/snew) < eps) then
    exit iteration
  end if
  if (it >= itmax) then
    write (*,*) "Simpson: pas assez d'iterations pour obtenir epsilon desire"
    exit iteration
  end if
  sold = snew
end do iteration
somme = snew
write (*,*)
write (*,*) 'fin de simpson'
write (*,*)

return
end subroutine simpson
```

```
!-----------------------------------------------------------------------
!subroutine qui calcule une int,grale sans le calcul de f(x) aux intervales
!d'int,gration, ce qui permet p.ex. le calcul dans le cas de singularit,s
!par exemple dans l'intervale [0,1] avec la fonction 1/sqrt(x) on ne peut pas
!calculer cette int,grale avec Simpson mais on le peut avec trois_points
!MAIS: convergence sensiblement plus lente
!ne recalcule pas les points d,j… calcul,s
!A donner:
!a,b : intervales d'int,gration
!itmax: nb maximum d'it,rations
!eps: erreur relative exig,e sur la convergence
!fct: fonction: !!!! DOIT ETRE DU MEME INTERFACE QUE CI-DESSOUS !!!!!
!somme: valeur de sortie de l'int,grale

subroutine trois_points(a,b,itmax,eps,fct,somme)
real, intent(out) :: somme
real, intent(in) :: a,b,eps
integer, intent(in) :: itmax
interface
  function fct(x)
    real, dimension(:), intent(in) :: x
    real, dimension(size(x)) :: fct
  end function fct
end interface
real :: sold,snew,s1,s2,hn
integer :: it,ni,i
real, dimension(2) :: temp
real, dimension(:), allocatable :: t2

write (*,*) "Iteration   Points     Integrale   Erreur Richardson    Erreur relative"
write (*,*) "-----------------------------------------------------------------------"
it = 0
sold = 0.0
hn = b-a
iteration: do
  it = it + 1
  ni = 2**(it-1)
  hn = hn/2.
  s1 = s2
  allocate(t2(2*ni))
  t2 = (/(a+(2*i-1)*hn/4.,i=1,2*ni)/)
  s2 = sum(fct(t2))
  deallocate(t2)
  if (it == 1) then
    hn = b-a
    temp = (/(a+b)/2.,(a+b)/2./)
    temp = fct(temp)
    s1 = temp(1)
    temp = (/(3*a+b)/4.,(a+3*b)/4./)
    s2 = sum(fct(temp))
  end if
  snew = (hn/3.)*(2*s2-s1)
  write (*,'(i3,"          ",i6,"   ",f12.5,"    ",f10.5,"          ",f10.5)') &
  it,ni,snew,Abs((sold-snew)/15.),Abs((sold-snew)/snew)
  if (Abs((sold-snew)/snew) < eps) then
    exit iteration
  end if
  if (it >= itmax) then
    write (*,*) "Trois_points: pas assez d'iterations pour obtenir epsilon desire"
    exit iteration
  end if
  sold = snew
end do iteration
somme = snew
write (*,*)
write (*,*) 'fin de trois_points'
write (*,*)

return
end subroutine trois_points
```

```
!-----------------------------------------------------------------------
!recursive subroutine qui calcule une int,grale selon la m,thode de
!simpson mais en faisant une nouvelle subdivision par deux d'un intervale
!seulement si dans ces r,gions la condition sur l'erreur calcul,e n'est
!pas satisfaite. Converge trSs rapidement, PEU DE TEMPS DE CALCUL.
!Remarque: ici on arrête en fonction de l'erreur de richardson, qui diminue
!plus rapidement que l'erreur relative pour de pr,cisions exigeantes.
!A donner: idem simpson, mais:
!1-Pas de itmax
!2-ATTENTION: l'interface sur la fonction n'est pas la mme que dans simpson
(méthode adaptive)
recursive subroutine simpson2(a,b,eps,fct,somme)   recursive subroutine : la subroutine peut être appelée par elle-même.
real, intent(in) :: a,b,eps
real, intent(out) :: somme
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
end interface
real :: h,mid,fourth_1,fourth_3
real :: one_simpson, two_simpson, somme_g, somme_d

if (a >= b) then
  stop 'Simpson2: Trop grande pr,cision exig,e: a >= b'
end if
h=b-a
mid=(a+b)/2.
fourth_1=(a+mid)/2.
fourth_3=(mid+b)/2.
one_simpson=h*(fct(a) + 4.*fct(mid) + fct(b))/6.
two_simpson=h/2.*(fct(a) + 4.*fct(fourth_1) + fct(mid))/6. + &
            h/2.*(fct(mid) + 4.*fct(fourth_3) + fct(b))/6.
if (abs(one_simpson - two_simpson) < 15.*eps) then
  somme = (16.*two_simpson - one_simpson)/15.
else
  call simpson2(a,mid,eps,fct,somme_g)   la subroutine s'appelle elle-même
  call simpson2(mid,b,eps,fct,somme_d)
  somme = somme_g + somme_d
end if
return
end subroutine simpson2
```

```
!-----------------------------------------------------------------------
!recursive subroutine qui calcule une int,grale selon la m,thode de
!3points mais en faisant une nouvelle subdivision par deux d'un intervale
!seulement si dans ces r,gions la condition sur l'erreur calcul,e n'est
!pas satisfaite.
!Remarque: ici on arr^te en fonction de l'erreur de richardson, qui diminue
!plus rapidement que l'erreur relative pour de pr,cisions exigeantes.
!A donner: idem simpson, mais:
!1-Pas de itmax
!2-ATTENTION: l'interface sur la fonction n'est pas la m^me que dans 3points
(méthode adaptive)
recursive subroutine trois_points2(a,b,eps,fct,somme)
real, intent(in) :: a,b,eps
real, intent(out) :: somme
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
end interface
real :: h,mid,fourth_g,fourth_d,eigth_gg,eigth_gd,eigth_dg,eigth_dd
real :: one_points,two_points,somme_g,somme_d

if (a >= b) then
  stop 'trois_points2: mauvais intervale d''int,gration: a >= b'
end if
h = b-a
mid = (a+b)/2.
fourth_g = (a+mid)/2.
fourth_d = (mid+b)/2.
eigth_gg = (a+fourth_g)/2.
eigth_gd = (fourth_g+mid)/2.
eigth_dg = (mid+fourth_d)/2.
eigth_dd = (fourth_d+b)/2.
one_points = h*(2*fct(fourth_g) - fct(mid) + 2*fct(fourth_d))/3.
two_points = (h/2.)*(2*fct(eigth_gg) - fct(fourth_g) + 2*fct(eigth_gd))/3.+&
             (h/2.)*(2*fct(eigth_dg) - fct(fourth_d) + 2*fct(eigth_dd))/3.
if (abs(one_points - two_points) < 15.*eps) then
  somme = (16.*two_points - one_points)/15.
else
  call trois_points2(a,mid,eps,fct,somme_g)
  call trois_points2(mid,b,eps,fct,somme_d)
  somme = somme_g + somme_d
end if
return
end subroutine trois_points2
```

```
!-----------------------------------------------------------------------
!subroutine qui calcule une int,grale selon la formule de Simpson d'ordre 4
!Diff,rences avec SIMPSON:
!       - ne r,utilise pas les valeurs pr,c,dentes lors des it,rations
!       - d,claration de la fonction: pas de dimension (:)
!-> au minimum 2 fois plus lent que simpson
!A donner:
!a,b : intervales d'int,gration
!itmax: nb maximum d'it,rations
!eps: erreur relative exig,e sur la convergence
!fct: fonction: !!!! DOIT ETRE DU MEME INTERFACE QUE CI-DESSOUS !!!!!
!somme: valeur de sortie de l'int,grale

subroutine simpson3(a,b,itmax,eps,fct,somme)
real, intent(out) :: somme
real, intent(in) :: a,b,eps
integer, intent(in) :: itmax
interface
  function fct(x)
    real, intent(in) :: x
    real :: fct
  end function fct
end interface
real :: sold,snew,s1,s2,s4,hn
integer :: it,ni,i
real, dimension(1:2) :: bords
real, dimension(:), allocatable :: t4

write (*,*) "Iteration   Points     Integrale   Erreur Richardson    Erreur relative"
write (*,*) "-------------------------------------------------------------------------"
it = 0
sold = 0.0
hn = b-a
bords = (/a,b/)
s1 =fct(bords(1)) + fct(bords(2))
iteration: do
  it = it+1
  ni = 2**(it-1)
  hn = hn/2.
  allocate(t4(ni))
  s2 = s2 + s4
  t4 = (/(a + (i-1/2.)*hn,i=1,ni)/)
  s4 = 0
  do i=1,ni
    s4 = s4 + fct(t4(i))
  end do
  deallocate(t4)
  if (it == 1) then
    hn = b-a
    s2 = 0
    s4 = fct((a+b)/2)
  end if
  snew = (hn/6.)*(s1+2*s2+4*s4)
  write (*,'(i3,"          ",i6,"   ",f12.5,"    ",f10.5,"          ",f10.5)') &
  it,ni,snew,Abs((sold-snew)/15.),Abs((sold-snew)/snew)
  if (Abs((sold-snew)/snew) < eps) then
    exit iteration
  end if
  if (it >= itmax) then
    write (*,*) "Simpson3: pas assez d'iterations pour obtenir epsilon desire"
    exit iteration
  end if
  sold = snew
end do iteration
somme = snew
write (*,*)
write (*,*) 'fin de simpson3'
write (*,*)

return
end subroutine simpson3
```

```
!-----------------------------------------------------------------------
!Int,gration de l',quation diff,rentielle ordinaire:
! Dy/Dx = f(x,y) ,y(xi) = yi
!On doit donner:
!xi,yi : conditions initiales xi et yi=y(xi)
!xf    : valeur xf de x o- on veut calculer la fonction y
!itmax : nombre maximum d'it,rations permis
!eps   : erreur relative
!
!On fait appel …:
!function fct(x,y)
!
!Le programme sort les valeurs:
!yf    : valeur cherch,e de y(xf)
!n     : nombre d'intervales qui a permis d'atteindre la pr,cision d,sir,e
!        (la dimension des tableaux x et y est donc n+1)
!x,y   : tableaux de dimension (n+1) contenant les valeurs interm,diaires
!        de la variable ind,pendante x et celles correspondant … y(x)
!        on a un tableau de dimension (1:n+1) avec les valeurs 0,1,...,n
!
!ATTENTION: x,y sont … d,clarer comme suit:
!             real, pointer, dimension(:) :: x,y
!           Le allocate est fait ici, mais il ne faut pas oublier de faire
!           deallocate(x,y) dans le programme principal.

subroutine rk4(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
integer, intent(out) :: n
integer, intent(in) :: itmax
real, intent(in) :: xi,yi,xf,eps
real, intent(out) :: yf
real, pointer, dimension(:) :: x,y
interface
  function fct(x,y)
    real, intent(in) :: x,y
    real :: fct
  end function fct
end interface
integer :: i,it,npas
real :: h,xc,yc,t1,t2,t3,t4,err,yc_old
real, dimension(:), allocatable :: xt,yt

write (*,*) "Iteration       Points           Valeur y(xf)        Erreur   "
write (*,*) "-------------------------------------------------------------"
npas = 1
it = -1
boucle: do
  it = it + 1
  if (it > itmax) then
    write (*,*) "rk4: pas de convergence"
    stop
  end if
  npas = 2**it
  h = (xf-xi)/npas
  xc = xi
  yc = yi
  allocate (xt(npas),yt(npas))
  do i=1,npas
    t1=h*fct(xc,yc)
    t2=h*fct(xc+h/2.,yc+t1/2.)
    t3=h*fct(xc+h/2.,yc+t2/2.)
    t4=h*fct(xc+h,yc+t3)
    yc = yc + (t1 + 2*t2 + 2*t3 + t4)/6.
    xc = xi + i*h
    xt(i) = xc
    yt(i) = yc
  end do
  if (it >= 1) then
    err = abs((yc-yc_old)/(15.*yc))          !richardson
!   err = abs(yc-yc_old/15.)                 !absolue
    write (*,'("    ",i3,"       ",i10,"            ",f10.5,"        ",f10.7)') &
    &it,npas,yc,err
    if (err <= eps) then
      yf = (16.*yc - yc_old)/15.
      n = npas
      allocate(x(n+1),y(n+1))
      x(1:n+1) = (/xi,xt(1:n)/)
      y(1:n+1) = (/yi,yt(1:n)/)
```

on n'a pas besoin de mettre intent(out)

xt : tableau temporaire ⟷ x
yt : tabl. " ⟷ y

⚠ peut poser parfois des problèmes

pour un pointeur

--> nouvel exemple du constructeur de tableau

```
      deallocate(xt,yt)
      exit boucle
    end if
  end if
  yc_old = yc
  deallocate(xt,yt)
end do boucle

write (*,*)
write (*,*) "fin de rk4"
write (*,*)

return
end subroutine rk4
```

```
!-----------------------------------------------------------------------
!Int,gration de l',quation diff,rentielle ordinaire:
! Dyk/Dx = f(x,y1,...,yN) ,yk(xi) = yik   ,k=1,N (C.I.) i=i=cte
!On doit donner:
!xi,xf : extr,mit,s de l'intervale d'int,gration [xi,xf]
!yi : conditions initiales t.q. yik=y(xi). Tableau de dimension N
!itmax : nombre maximum d'it,rations permis (a la fin de chaque it,ration,
!        le nombre d'intervales est doubl,
!eps    : erreur relative: tableau de dimension N avec eps(i) l'erreur
!        relative sur la i-Šme ,quation diff,rentielle
!
!On fait appel …:
!function fct(x,y)      : fonction qui pour chaque ensemble des valeurs des
!                       variables x et y = (y1,...,yN) fournit le tableau
!                         fct contenant les valeurs des fonctions
!                       f1(x,y),...,fN(x,y).
!
!Le programme sort les valeurs:
!yf     : valeur cherch,e de y(xf): tableau contenant les valeurs des N
!        variables d,pendantes calcul,es pour x=xf
!n      : nombre d'intervales qui a permis d'atteindre la pr,cision d,sir,e
!x      : tableau de dimension (n+1) contenant les (n+1) valeurs interm,diaires
!         de la variable x
!y      : tableau de dimension (N,n+1) contenant les valeurs des N ,quations
!         diff,rentielles aux n+1 points d',valuation.
!         on a un tableau de dimension (1:n+1) avec les valeurs 0,1,...,n
!
!ATTENTION: x,y sont … d,clarer comme suit:
!             real, pointer, dimension(:) :: x
!             real, pointer, dimension(:,:) :: y
!           Le allocate est fait ici, mais il ne faut pas oublier de faire
!           deallocate(x,y) dans le programme principal.
!           Il faut toujours se rappeler que la d,claration d'interface est
!           en bijection avec la d,claration du programme principal

subroutine rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
integer, intent(out) :: n
integer, intent(in) :: itmax
real, intent(in) :: xi,xf
real, intent(in), dimension(:) :: yi,eps
real, intent(out), dimension(:) :: yf
real, pointer, dimension(:) :: x
real, pointer, dimension(:,:) :: y
interface
  function fct(x,y)
    real, intent(in) :: x
    real, intent(in), dimension(:) :: y
    real, dimension(size(y)) :: fct
  end function fct
end interface
integer, parameter :: ninit = 8
integer :: npas, it, i
logical, dimension(size(yi)) :: masque
real, dimension(:), allocatable :: xt
real, dimension(:,:), allocatable :: yt
real :: h,xc
real, dimension(size(yi)) :: err,yc,yc_old,t1,t2,t3,t4

write (*,*) "Iteration         Points "
write (*,*) "-------------------------"
npas = ninit
it = 0
boucle: do
  it = it + 1
  if (it > itmax) then
    write (*,*) "rk4sys: pas de convergence"
    stop
  end if
  npas = 2*npas
  h = (xf-xi)/npas
  xc = xi
  yc = yi
  allocate (xt(npas),yt(size(yi),npas))
  do i=1,npas
    t1=h*fct(xc,yc)
    t2=h*fct(xc+h/2.,yc+t1/2.)
    t3=h*fct(xc+h/2.,yc+t2/2.)
```

} tableaux de travail temporaires

dimension du nb. d'équations du système

```
    t4=h*fct(xc+h,yc+t3)
    yc = yc + (t1 + 2*t2 + 2*t3 + t4)/6.
    xc = xi + i*h
    xt(i) = xc
    yt(:,i) = yc
  end do
  if (it >= 2) then
    masque = .false.
!    err = abs((yc-yc_old)/(15.*yc))          !richardson
    err = abs((yc-yc_old)/15.)               !absolue
!-----------------------------------
!    where (err <= eps) masque = .true.
!-----------------------------------
!la fonction where n'etant pas accept,e par le fortran PC, on remplace
!cette instruction par les 4 lignes suivantes
!-------------------------------
    do i=1,size(masque)
      if (err(i) <= eps(i)) then
        masque(i) = .true.
      end if
    end do
!-------------------------------
    write (*,'("   ",i3,"       ",i10)') it,npas
    if (all(masque)) then
      yf = (16*yc-yc_old)/15.
      n = npas
      allocate(x(n+1),y(size(yi),n+1))
      x(1:n+1) = (/xi,xt(1:n)/)
      y(1:size(yi),1) = yi                   !conditions initiales
      y(1:size(yi),2:n+1) = yt               !solutions cherch,es
      deallocate(xt,yt)
      exit
    end if
  end if
  yc_old = yc
  deallocate(xt,yt)
end do boucle

write (*,*)
write (*,*) "fin de rk4sys"
write (*,*)

return
end subroutine rk4sys
```

→ car avec Yc on fait des opérations globales pour toutes les éqn. du syst

Véqn valeur au pt. Xc

utilisation équivalente à, marche uniq. avec UNIX.

si masque(i) == .true. ∀i

```
!-------------------------------------------------------------------------
!M,thode de simpson pour calculer une int,grale connaissant uniquement
!quelques points discrets. Explication des variables:
!  y       : tableau de dimension (n+1) contenant les valeurs y(xi)
!  h       : largeur du pas d'int,gration
!  n       : nombre de pas d'int,gration
!  somme   : valeur de l'int,grale sur la base des points fournis

subroutine simptab(y,h,n,somme)
implicit none
real, intent(in), dimension(:) :: y
real, intent(in) :: h
integer, intent(in) :: n
real, intent(out) :: somme
integer :: i
real :: wgt

somme = y(1)
wgt = 2.0
do i=1,n
  if (wgt == 2.0) then
    wgt = 4.0
  else
    wgt = 2.0
  end if
  somme = somme + wgt*y(i)
end do
somme = somme + y(n+1)
somme = h*somme/3.0

write (*,*)
write (*,*) "fin de simptab"
write (*,*)

return
end subroutine simptab
```

- remarque: ne jamais commencer un calcul avec une tolérance trop exigeante, commencer p. ex. avec 10^{-2}, etc.

- procédure adaptative d'intégration.
  → fonction récursive: la procédure se rappelle elle-même.

Remarque: - programs tests:
- bissection (racine d'une équation): 3flott1.f90
- Longueur (longueur d'une courbe paramétrée ou d'une fonction): 4cable1.f90
- minimum (minimum d'une fct, ou maximum de -fct): 4cable5.f90

Remarque: - 2 méthodes pour le calcul du maximum d'une fonction:
1. minimum ($-f(x)$)
2. bissection ($\frac{df}{dx}=0$)

Remarque: - semestre 5

Remarque: - exercice 1a)

(1)
```
subroutine rk4 ( fct, xi, yi, xf, n, yf, x,y)

implicit none
integer, intent (in)      :: n
real, intent (in)         :: xi, yi, xf
real, intent (out)        :: yf
real, dimension ( : )     :: x,y
```

→ version améliorée: (p.24)

(2)
```
subroutine rk4 ( fct, xi, yi, xf, xf,   itmax, eps, yf, n, x, y)
reste: idem
integer, intent (out)     :: n
real, pointer, dimension ( : )   :: x,y
```
↳ puis: allocate x(n)

Pourquoi on ne doit pas mettre intent (out) ?

(avec tolérance ε : #n inconnu)

Remarque: - sur les pointeurs: p. 21, 22: de plus:

real :: p ⟺ allocation en mémoire
real, pointer :: p ⟺ aucune allocation de mémoire

- allocation de mémoire:

```
real, pointer    :: p
allocate (p)          ← allocation de la mémoire ; si pas allocate, alors p reste inutilisable.
p = 15.               ← peut être fait maintenant
:
deallocate (p)        ← après utilisation
```

- autre moyen d'utilisation du pointeur: allocation:

```
real, pointer    :: p      (pas obligatoire)
real, target     :: t = 17.
p => t                ← p va contenir le contenu de t : on fait pointer p sur l'adresse-mémoire de t.
```

- états du pointeur:
  - indéfini: avant allocate
  - nul: pas utilisé ⇒ ou allocate (nullify)
  - associé: relation d'affectation

→ savoir si l'état est associé: (true ou false)

```
if ( associated (p) )   nullify (p)
if ( associated (p) )   deallocate (p)
```

⚠ Ne pas oublier de désallouer de la mémoire
⚠ Ne pas faire nullify avant deallocate (car annule la correspondance entre le pointeur et les variables réelles)

- utilisation des pointeurs:

```
        Yc_old = Yc
        deallocate (xt, Yt)
    end do
end subrouhine rk4
```

Remarque – complément sur Simpson

– complément sur GNUPlot

– appel : GNUPLOT

– sortie : exit ou quit

– aide : help → help { <topic> }

nom du fichier qui contient les valeurs { x ; y }

```
plot [0:1] √(1-x²) , 3.x.√(1-x²) , 3.(1-x²) , " p22 "
```

→ format du fichier " p22 " : p. ex. : c.f. fauvcule

– créer un fichier tabulé

3flott99.f90

```
open (unit=1 , file = 'p22')
do i=1,n
    x = ...
    Y = 3.0*(1.0 - x.x)
    write (1,*) x,Y
end do
close(1)
```

– sauvetage du fichier Postscript : (avant de sortir de GNUPLOT)

```
set term postscript
set output "file.ps"      ← nom du fichier de l'image
replot
```

– impression sur imprimante :

```
lp -d printname -o-h file.ps
```

Remarque : – cas d'un système : attribut dimension aux var. scalaires

```
integer, intent(in)  :: itmax
integer, intent(out) :: n
real, intent(in)     :: xi, xf
real, intent(in), dimension (:)  :: Yi , eps
real, intent (out), dimension (:) :: Yf
real, pointer, dimension (:)     :: x
real, pointer, dimension (:,:)   :: Y

interface
   function fct(x,n)
      real, intent (in)                    :: x
      real, intent (in) , dimension (:)    :: Y
      real, dimension (size(Y))            :: fct
   end function fct
end interface

integer                          ::  ...
integer, parameter               ::  ninit = 8
real                             ::  ...
real, dimension (size(Yi))       ::  ...
logical, dimension (size(Yi))    ::  marque
real, dimension(:), allocatable  ::  Xt
real, dimension (:,:), allocatable :: Yt


it = 0
npas = ninit
do
    it = it + 1
    if (it > itmax) then
        print *, ' pas de converg.'
        stop
    end if
    npas = 2* npas
    h = ...
    allocate ( xt(npas), Yt (size(Yi), npas))
```

Indications : - prog. principal

```
module procedures
  implicit none
  real :: z
  real, dimension (0:3) :: be
  ...
  contains
  ...
end module procedure
```

```
program atome
  use procedures
  implicit none
  ...
  character (len=10)        :: formule
  ...
  write (*,'("donner symbole atome : ")', advance='no')
  read *, formule
  select case (formule)
     case ('H')
        be = ( /0., 0., 0., 0./)
        z = 1
     case ('Li2+')
        be = ( /0., 0., 0., 0./)
        z = 3
     case ('Na')
        be = ( /0.5147, 0.18288, -0.02450, 0/)
        z = 1
        :
     case default   ← qqch. de non connu
        print *, 'cas non prevu'
  end select
  ... définir combien de fct. de bases on veut : n_alf = ...  ⇒  n = 2*n_alf + 1
  call basis (...)
  ...
  call overlap → calcul de la base, αi, la matrice de recouvrement
  call hamilt → éléments de matrice du hamiltonien
  call ssygv (...)
  ...
end program atome
```

boucle sur e

ITYPE = 1 ; UPLO = L
JOBZ = N ; N = 2*n_alf + 1

```
module procedures
  implicit none
  integer                  :: nbase
  real, allocatable, dimension(:)   :: alf
                           (:,:)    :: alf_s, alf_p
                           (:,:)    :: s, k
  logical, allocatable, dimension(:,:) :: mask

contains
  subroutine basis(q, alf_mid, n_alf)        alf = alpha = α
    implicit none
    integer, intent(in)    :: n_alf
    real, intent(in)       :: q, alf_mid
    :

    alf = ( / (alf_mid/q**i , i=n_alf,1,-1), alf_mid, &
              (alf_mid*q**i , i=1, n_alf)/)        αi = α0·q^i , i = 0, ±1, ..., ±n
    alf_s = 0.0
    alf_p = 0.0
    allocate (mask(nbase,nbase))
    mask = .false.
    do i=1, nbase
       alf_s(i,:) = alf(i) + alf
       alf_p(i,:) = alf(i) * alf
       mask(i,1:i) = .true.
    end do
  end subroutine basis
```

← on veut une matrice symétrique pour notre procédure Lapack ⇒ triangle inférieur

on a les matrices $\alpha_\nu + \alpha_\mu$ et $\alpha_\nu \cdot \alpha_\mu$

6eqdif2.f90 : exemple d'utilisation de rk4 (runge-kutta ordre 4)

```
module fonctions
IMPLICIT NONE
contains

function fct(x,y)
  real, intent(in) :: x,y
  real :: fct
  fct = x**2 + y
  return
end function fct

end module fonctions

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

program eqdif2
use biblio, only : rk4
use fonctions
implicit none

real :: xi,yi,xf,yf,eps
int  r :: itmax,n
real, pointer, dimension(:) :: x,y

write (*,"(//////////////////////)")
write (*,*) "eps = "
read (*,*) eps
write (*,*) "itmax = "
read (*,*) itmax

xi = 0.
yi = -1.
xf = 3.

call rk4(fct,xi,yi,xf,itmax,eps,yf,n,x,y)

write (*,*) "Intervales n                y(3,n)    "
write (*,*) "-------------------------------------"
write (*,'(I8,"                ",F12.8)') n,yf
!write (*,*) x,y
xi = sum(x) + sum(y)
deallocate(x,y)

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

stop 'termine'
end program eqdif2

!le nombre d'itérations est égal à ln(n)/ln(2)
```

⇔ $Y'(X) = fct(Y(X), x) = x^2 + Y(X)$

→ par nécessaire : si on veut utiliser que rk4 de biblio.f90

} est obligé de faire cette déclaration pour l'utilisation de rk4

x(1, n+1) ; Y(1, n+1)
↓ valeur des pts d'évaluation
↓ fct à ces divers points

} pour utiliser x et y : sinon : erreur. On peut aussi faire write (*,*), x,y

→ libère la mémoire du pointeur ; le allocate est déjà dans rk4

6eqdif3.f90 : exemple d'accès au fichier

```
module fonctions
IMPLICIT NONE
contains

function fct(x,y)
  real, intent(in) :: x,y
  real :: fct
  fct = x**2 + y
  return
end function fct

end module fonctions

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

program eqdif3
use biblio, only : rk4
use fonctions
implicit none

real :: xi,yi,xf,yf,eps
int   r :: itmax,n,i
real, pointer, dimension(:) :: x,y

write (*,"(/////////////////////////)")
write (*,*) "eps = "
read (*,*) eps
write (*,*) "itmax = "
read (*,*) itmax

xi = 0.
yi = -1.

xf = 3.
call rk4(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
write (*,*) yf
open (unit = 1, file = '6eqdif3a.txt')
  do i = 1,n+1
    write (1,*) x(i),y(i)
  end do
dea   ate(x,y)

xf = 6.
call rk4(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
write (*,*) yf
open (unit = 1, file = '6eqdif3b.txt')
  do i = 1,n+1
    write (1,*) x(i),y(i)
  end do
deallocate(x,y)

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

stop 'termine'
end program eqdif3
```

accède au disque dur : '1' , file = 'nom du fichier sur le disque'

accède au fichier : write (1, *) (le rapporte au fichier)

puis avec GNUPLOT : plot [0:6] exp(x) - (x**2 + 2*x + 2) , "6eqdif3a.txt" , "6eqdif3b.txt"

6eqdif5.f90 : exemple d'utilisation de rksys

```
module fonctions
IMPLICIT NONE
integer, parameter :: nbeq = 2
contains

function fct(x,y)
  real, intent(in) :: x
  real, intent(in), dimension(:) :: y
  real, dimension(size(y)) :: fct
  fct(1) = y(2) - 3*x*y(1)
  fct(2) = (x-2*x**2)*y(1)-y(2)
  return
end function fct

end module fonctions

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

program eqdif5
use biblio
us   onctions
implicit none

real, pointer, dimension(:) :: x
real, pointer, dimension(:,:) :: y
real :: xi,xf
integer :: n,itmax
real, dimension(nbeq) :: eps,yi,yf
integer :: i

write (*,"(//////////////////////////)")
do i=1,nbeq
  write (*,'(" eps ",I3," = ? ")') i
  read (*,*) eps(i)
end do
write (*,*) " itmax = "
read (*,*) itmax

xi = 0.
yi(1) = -1.
yi(2) = 1.

xf = 1.5
call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
write (*,*) n
write (*,'(f15.8)') yf
deallocate(x,y)

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

stop 'termine'
end program eqdif5
```

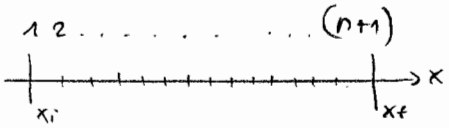
constante qui donne le # d'éqn : permet d'éviter de changer à d'autres emplacements.

↳ cet emplacement est possible : sera utilisé partout

c'est le système qui va de 1 à nbeq = 2 (ici) i.e. on a :

$$\begin{pmatrix} y_1'(x) \\ y_2'(x) \end{pmatrix} = \begin{pmatrix} y_2(x) - 3.x.y_1(x) \\ (x-2x^2).y_1(x) - y_2(x) \end{pmatrix}$$

contient les pas d'évaluation :



à tester t.q.

real, pointer, dimension(:,:) :: y → contient la valeurs des fonctions aux points donnés par X : $Y(i,j) = y_i(x_j)$

⇒ pour i fixé, i.e. $Y(i,:)$, on a la fct $y_i(x)$ évaluée en x_j $\forall j$, i.e. on a toute l'info. sur y_i. Dimension : $Y(nbeq., n+1)$

real, dimension(nbeq) :: eps,yi,yf → contient les valeurs des nbeq équations en x_f, i.e.

$yf(i) = y_i(x_f) = Y(i, n+1)$

début de l'intégration

C.I. $y_i = y_{initial}$, $y_i(j) = y_{j\,initial}$ = c.i. de la fct. j

fin de l'intégration

⇒ tout ce qui est écrit en bleu doit être modifié pour avoir un autre cas

6eqdif6.f90 : exemple de transformation d'une éqn. diff. du 2nd ordre en une éqn. diff. système de 1er ordre

```
!d^2y/dx^2 = -sin(y) -> dy/dx = z
!                    -> dz/dx = -sin(y)
!-> y(1) = y
!-> y(2) = z = y'

module fonctions
IMPLICIT NONE
integer, parameter :: nbeq = 2

contains

function fct(x,y)
  real, intent(in) :: x
  real, intent(in), dimension(:) :: y
  real, dimension(size(y)) :: fct
  fct(1) = y(2)+x-x            !pour utiliser x, on est obligé; dy/dx=z        z=y'   } ce qui n'a aucun intérêt, c'est fct(4)
  fct(2) = -sin(y(1))          !dz/dx = -sin(y)
  return
end function fct

end module fonctions

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

program eqdif6
use biblio
use fonctions
implicit none

real, pointer, dimension(:) :: x
real, pointer, dimension(:,:) :: y
real :: xi,xf
integer :: n,itmax
real, dimension(nbeq) :: eps,yi,yf
!spécifiquement pour cet exercice:
real, parameter :: g = 9.8
real, dimension(3) :: solution,exact
integer :: i
real :: unitetemps,borne,dx

wr (*,"(/////////////////////////)")
write (*,*) " itmax = "
read (*,*) itmax
write (*,*) "eps1 = ? "        !sur y(x)
read (*,*) eps(1)
write (*,*) "eps2 = ? "        !sur y'(x)
read (*,*) eps(2)

unitetemps = sqrt(1/g)
xi = 0.0/unitetemps
yi(1) = 0.5                    !C.I. sur y(x)
yi(2) = 0.                     !C.I. sur y'(x)

xf = 3.0/unitetemps
call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
solution(1) = yf(1)
exact(1) = yi(1)*cos(xf+xi)
deallocate(x,y)
xf = 3.5/unitetemps
```

```
call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
solution(2) = yf(1)
exact(2) = yi(1)*cos(xf+xi)
deallocate(x,y)
xf = 4.0/unitetemps
call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
solution(3) = yf(1)
exact(3) = yi(1)*cos(xf+xi)
deallocate(x,y)
write (*,*) n

do i=1,3
  write (*,'("   ",f12.8,"          ",f12.8)') solution(i),exact(i)
end do

write (*,*)
write (*,*) "Appuyer sur ENTER pour passer à la partie 6c"
read (*,*)

write (*,*) "Nombre de subdivisions de l'intervale [0,4] ?"
read (*,*) n
borne = 1e-8
dx = (4./n)/unitetemps
do i=1,n
  xf = xi + i*dx
  call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
  solution(1) = yf(1)
  exact(1) = yi(1)*cos(xf+xi)
  deallocate(x,y)
  if (abs(solution(1) - exact(1)) > borne) then
    borne = abs(solution(1) - exact(1))
  end if
end do

write (*,"(////////////////////////////////)")
write (*,'(" Borne = ",f10.6)') borne


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

stop 'termine'
end program eqdif6
```

6eq dif 13 . f90 : exemple d'utilisation de la subroutine SIMPTAB

```
module fonctions
IMPLICIT NONE
integer, parameter :: nbeq = 2

contains

function fct(x,y)
  real, intent(in) :: x
  real, intent(in), dimension(:) :: y
  real, dimension(size(y)) :: fct
  fct(1) = y(2)+x-x
  fct(2) = -sin(y(1))
  return
end function fct

end module fonctions

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

program eqdif13
use biblio
use fonctions
implicit none

real, pointer, dimension(:) :: x
real, pointer, dimension(:,:) :: y
real :: xi,xf
integer :: n,itmax
real, dimension(nbeq) :: eps,yi,yf
real, parameter :: g = 9.8
integer :: i
real :: unitetemps,periode,angle_initial,borne,h,Pi
real, dimension(5) :: somme                          ! → normalement : real
real, dimension(:), allocatable :: y_fourier         ! → par l'intégrale du développement de Fourier

write (*,"(//////////////////////)")
write (*,*) " itmax = "
read (*,*) itmax
write (*,*) "eps = ? "
read (*,*) eps(1)
eps(2) = eps(1)

angle_initial = 1.8686          } déjà connu
periode = 2.56

!write (*,*) " angle initial = "
!read (*,*) angle_initial
!write (*,*) " periode = "
!read (*,*) periode

Pi = 4*Atan(1.)

unitetemps = sqrt(1/g)
xi = 0.0/unitetemps
xf = periode/unitetemps          !ATTENTION: fourier sur 1 periode seulement

yi(1) = angle_initial                    !C.I. sur y(x)
yi(2) = 0.                               !C.I. sur y'(x)
call rk4sys(fct,xi,yi,xf,itmax,eps,yf,n,x,y)
```

→ doit faire ça pour avoir Y(xi) et pouvoir connaître la fct. qui permet de faire le D.L. de Fourier

```
write (*,*) yf                          !Sinon yf non utilise

x = x*unitetemps                        !conversion en temps réel
allocate(y_fourier(n+1))
h = (xf-xi)*unitetemps/n
```

← une fois que on a les points d'évaluation et la fonction à ces points, on peut allouer le tableau de dimension (n+1) contenant les valeurs $Y(x_i)$

```
do i=1,5,2
  y_fourier = (2./periode)*y(1,:)*Cos(i*2*Pi*x/periode)
  call simptab(y_fourier,h,n,somme(i))
end do
```

→ veut dire : i varie de 1 à 5 par incrément de 2 : $i = 1, 3, 5$

→ calcule l'intégrale

$$a_n = \frac{2}{T} \int_{x_0}^{x_0+T} f(x) \cdot \cos\left(n \frac{2\pi}{T} x\right)$$

$f(x)$ → $Y(1,:)$; x → x

cas général : a_n, b_n → à modifier ici ; (*)

```
write (*,*) "Coefficients a1, a3, a5 du développement de fourier"
write (*,*) somme(1), somme(3), somme(5)

y_fourier = somme(1)*Cos(1*2*Pi*x/periode) + somme(3)*Cos(3*2*Pi*x/periode)&
&+somme(5)*Cos(5*2*Pi*x/periode)
borne = maxval(abs(y(1,:)-y_fourier(:)))

write (*,*) "Borne supérieure de l'erreur avec les 3 termes de fourier"
write (*,'(F15.8)') borne

deallocate (x,y,y_fourier)
wr_   (*,*)
write (*,*) "Appuyer sur ENTER pour terminer le programme"
read (*,*)

stop 'termine'
end program eqdif13
```

$$f(x) = \frac{1}{2} a_0 + \sum_{n \geq 1} a_n \cdot \cos(n\omega x) + b_n \cdot \sin(n\omega x)$$

avec ici : $f(x) \simeq a_1 \cdot \cos(\omega x) + a_3 \cdot \cos(3\omega x) + a_5 \cdot \cos(5\omega x)$

(*) Cas général : 6eqdif140.f90