```
!-------------------------------------------------------------------------
! Maxime ESCOURBIAC
! G1
! 16/01/2010
! TP4 Methode du gradient conjugue, de Fletcher-Reeves
!       Et Polak-Ribiere
!-------------------------------------------------------------------------
!


module gc

   use fct

   implicit none

   contains

   !methode de  flectcher reeves
   subroutine fletcher_reeves(x,n,ichoix,itermax)

      implicit none

      !declaration des variables
      real(8),dimension(:),intent(inout)  :: x                  !x0 en entree, x* en sortie
      integer              ,intent(in)    :: n                  !rang de la fonction
      integer              ,intent(in)    :: ichoix,itermax     !pour le choix de la fonction a utiliser
      real(8),dimension(:),allocatable    :: d,g1,g2            !les differents vecteur utilises
      real(8)                             :: beta,t,ng1,ng2
      real(8)                             :: eps                !variables de precision
          integer                             :: iter,itertot   !nombre d'iteration effectue

      eps = 1.d-6
      iter = 0
      itertot = 0
      allocate(d(n))
      allocate(g1(n))
      allocate(g2(n))

      call fonc(x,g1,n,ichoix)
      d = -g1
      ng1 = dot_product(g1,g1)
      alex: do
             call rech_lin(x,d,t,n,ichoix)      !recherche de tk
             x = x+t*d                          !nouvelle aproximation de la solution
             call fonc(x,g2,n,ichoix)           !calcul du gradient de x1
             iter = iter+1
             itertot = itertot+1
             if(iter .eq. n) then               ! test pour garantir la convergence en n etapes
                iter = 0
                d = -g2
                ng2 = dot_product(g2,g2)
             else
```

```
            ng2 = dot_product(g2,g2)
            beta = ng2/ng1
            d = -g2+beta*d                    !calcul de d(k+1)
         end if

            g1 = g2
            ng1 = ng2
         if((ng1 .lt. eps).or.(itertot .gt. itermax)) then
            exit alex
         end if



     end do alex
     print *, "nombre d'iteration",itertot
end subroutine fletcher_reeves

!methode de Polak-ribiere

subroutine polak_ribiere(x,n,ichoix,itermax)

   implicit none

   !declaration des variables
        real(8),dimension(:) ,intent(inout) :: x              !x0 en entree, x* en sortie
        integer              ,intent(in)    :: n              !rang de la fonction
        integer              ,intent(in)    :: ichoix,itermax !pour le choix de la fonction a utiliser
        real(8),dimension(:) ,allocatable   :: d,g1,g2        !les differents vecteur utilises
        real(8)                             :: beta,t,ng1,ng2
        real(8)                             :: eps            !variables de precision
        integer                             :: iter,itertot   !nombre d'iteration effectue

   eps = 1.d-6
   iter = 0
   itertot=0
   allocate(d(n))
   allocate(g1(n))
   allocate(g2(n))

   call fonc(x,g1,n,ichoix)
   d = -g1
   ng1 = dot_product(g1,g1)
   celia: do

           call rech_lin(x,d,t,n,ichoix)      !recherche de tk
           x = x+t*d                          !nouvelle aproximation de la solution
           call fonc(x,g2,n,ichoix)           !calcul du gradient de x1
           iter = iter + 1
           itertot = itertot+1
           if(iter .eq. n) then               ! test pour garantir la convergence en n etapes
              iter = 0
              d = -g2
           else
```

```
            ng2 = dot_product(g2,(g2-g1))   !calcul du beta
            beta = ng2/ng1
            d = -g2+beta*d                  !calcul de d(k+1)
             end if

          g1 = g2
          ng1= dot_product(g1,g1)
          if((ng1 .lt. eps).or.(itertot .gt. itermax)) then
             exit celia
          end if

     end do celia
     print *, "nombre d'iteration", itertot
end subroutine polak_ribiere

!subroutine de la recherche lineaire qui permet de calculer tk

subroutine rech_lin(x,d,t,n,ichoix)

    implicit none

    !declaration des variables
    real(8),dimension(:),intent(in)  :: x         !vecteur xk
    real(8),dimension(:),intent(in)  :: d         !vecteur directeur en kx
    integer             ,intent(in)  :: n         !pour l'allocation de g et de y
    integer             ,intent(in)  :: ichoix    !pour le choix de la fonction a utiliser
    real(8)             ,intent(out) :: t         !reel pour le calcul de x(k+1)
    real(8),dimension(:),allocatable :: g,y       !variable pour le calcul temporaire de phip
    real(8)                          :: phip      !(D(Phi))(t) = Nabla(f(x[k]+td[k])^T)*t[k]
    real(8)                          :: delta     !reel pour la recherche d'intervalle
    real(8)                          :: t0,t1     !intervalle pour la dichotomie
    real(8)                          :: eps       !variable pour verifier la précision
    real(8)                          :: tm        !variable temporaire

    allocate(y(n))
    allocate(g(n))

    delta = 0.1d0
    eps   = 1.d-8
    t0 = 0.d0
    t1 = delta
    !recherche de tk par la methode de la dichotomie

    !recherche de l'intervalle [t0,t1]  tel que phip(t0) <0 et phip(t1)>0

    anna: do
            !calcul de phip(t1)
            y = x + t1*d
            call fonc(y,g,n,ichoix)
            phip = dot_product(d,g)
            if (phip .lt. 0.d0) then
               t0 = t1
               t1 = 2.d0*t1
```

```
          else
             exit anna
          end if
       end do anna

      !application de la dichotomie pour la recherche de tk

      do while ((t1 - t0) .gt. (2.d0*eps))

               tm = (t0 + t1)/2.d0
               y = x + tm*d
               call fonc(y,g,n,ichoix)
               phip = dot_product(d,g)
               if(abs(phip) .lt. eps) then
                  exit
               else
                   if(phip .gt. 0.d0) then
                      t1 = tm
                   else
                      t0 = tm
                   end if
               end if
      end do
      t = tm
   end subroutine rech_lin

end module gc
```