```
! ---------------------------------------------------------------------
!  parametros y constantes
module m_ctes
  implicit none
  public
  !
  !parametros de precision usuales
  integer, parameter :: iin = kind (1)
  integer, parameter :: isp = kind (1.0e0)
  integer, parameter :: idp=kind(1.0d0)
  integer, private, parameter :: q=selected_real_kind(2*precision(1.0_idp))
  integer, parameter :: iqp = (1+sign(1,q))/2*q+(1-sign(1,q))/2*idp
  !
  !parametros de precision para el Numerical Recipes (NR)
  integer, parameter :: j4b = selected_int_kind (9)
  integer, parameter :: j2b = selected_int_kind (4)
  integer, parameter :: j1b = selected_int_kind (2)
  !
  !constantes varias
  real (idp), parameter :: cero = 0.0d0
  real (idp), parameter :: uno  = 1.0d0
  real (idp), parameter :: dos  = 2.0d0
  real (idp), parameter :: tres = 3.0d0
  real (idp), parameter :: four = 4.0d0
  real (idp), parameter :: half = 0.5d0
  real (idp), parameter :: tol  = 1.0d-14
  real (idp), parameter :: tol2 = 1.0d-12
  real (idp), parameter :: eps  = 1.0d-6
  real (idp), parameter :: eps2 = 1.0d-3
  real (idp), parameter :: cono = 1.0d-3
  real (idp), parameter :: big  = 1.0d+6
  real (idp), parameter :: rmax = 1.0d+8
  real (idp), parameter ::                                          &
     tercio = 0.33333333333333333333333333333333333333333333333_idp &
   , sexto  = 0.66666666666666666666666666666666666666666666666_idp
  real (idp), parameter ::                                          &
      pi    = 3.14159265358979323846264338327950288419716_idp &
    , pi050 = 1.57079632679489661923132169163975144209858_idp &
    , pi2   = 6.28318530717958647692528676655900576839400_idp &
    , pi025 = 0.7853981633974828d0                            &
    , cpi2  = 0.6366197723675813d0                            &
    , rpi1  = 0.3183098861837910d0                            &
    , rpi2  = 0.1591549430918950d0                            &
    , rpi4  = 0.0795774715459477d0                            &
    , rpi8  = 0.0397887357729738d0                            &
    , cte_euler = 0.57721566490153286060651 2d0
   complex (idp), parameter ::                                &
      znulo = (0.0d0,0.0d0)                                   &
    , zreal = (1.0d0,0.0d0)                                   &
    , zimag = (0.0d0,1.0d0)                                   &
    , zpi2  = (0.0d0,6.28318530717958647692528676655900576d0)
  !
end module
! ---------------------------------------------------------------------
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!   Gauss con PIVOTAJE parcial con (A, B), donde
!   A ; matriz de coeficientes
!   B ; matriz de cargas
! ---------------------------------------------------------------------
!$Id: 2006/09/25 jdelia Exp $
module  m_gauss0      ! without Any Mapping
  use   m_ctes
  use   m_temps
  use   m_tools0
  implicit none
  private             ! por omision todo es privado
  !
  interface solve_b1
    module procedure dsolve_b1, zsolve_b1         & !precision doble
                   , qsolve_b1, wsolve_b1           !precision cuadruple
  end interface
  interface solve_bm
    module procedure dsolve_bm, zsolve_bm         & !precision doble
                   , qsolve_bm, wsolve_bm           !precision cuadruple
  end interface
  interface lu_factor
    module procedure dlu_factor, zlu_factor       & !precision doble
                   , qlu_factor, wlu_factor         !precision cuadruple
  end interface
  interface forward_sol
    module procedure dforward_sol, zforward_sol   & !precision doble
                   , qforward_sol, wforward_sol     !precision cuadruple
  end interface
  interface backward_sol
    module procedure dbackward_sol, zbackward_sol & !precision doble
                   , qbackward_sol, wbackward_sol   !precision cuadruple
  end interface
  !
  public solve_b1, solve_bm, lu_factor
  public forward_sol, backward_sol
  !
  !variables internas
  character (11), parameter :: c = "m_gauss0 > "
  !para medir tiempos
  real    (idp), dimension (3) :: time
  integer (iin) :: h1, h2
  contains
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
```



```
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     dlu_factor (a, d, indx)
    call   dforward_sol (a, b, y, indx)
    call  dbackward_sol (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  dlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   dforward_sol (a, v, y, indx)
      call  dbackward_sol (a, y, x)
      b (1:n,p) = x (1:n)
    end do
    !
```

```
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    !
    !valida tamanios
    n = siza (size (a,1), size (a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs ( a (k:n,k) ) )
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (a (k,k) == 0.0_idp)  a (k,k) = tenue
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                       &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    real    (idp), dimension (:,:), intent (in)    :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    real    (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    real    (idp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
```



```
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s = 0.0_idp
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_idp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    real    (idp), dimension (:,:), intent (in)  :: a
    real    (idp), dimension (:)  , intent (in)  :: y
    real    (idp), dimension (:)  , intent (out) :: x
    real    (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2),e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
```

```
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     zlu_factor (a, d, indx)
    call   zforward_sol (a, b, y, indx)
    call  zbackward_sol (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier (1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  zlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   zforward_sol (a, v, y, indx)
      call  zbackward_sol (a, y, x)
      b (1:n,p) = x (1:n)
    end do
```



```
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    complex (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    !
    !valida tamanios
    n = siza (size(a,1), size(a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs (a(k:n,k)))
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (abs (a(k,k)) == 0.0_idp) a (k,k) = cmplx (tenue, 0.0_idp)
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                          &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    complex (idp), dimension (:,:), intent (in)    :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    complex (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    complex (idp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
```

```
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s     = cmplx (0.0_idp, 0.0_idp)
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_idp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    complex (idp), dimension (:,:), intent (in)  :: a
    complex (idp), dimension (:)  , intent (in)  :: y
    complex (idp), dimension (:)  , intent (out) :: x
    complex (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2), e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  qsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (iqp), dimension (:,:), intent (inout) :: a
    real    (iqp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (iqp), dimension (:)  , allocatable :: y
    real    (iqp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
```



```
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     qlu_factor (a, d, indx)
    call   qforward_sol (a, b, y, indx)
    call  qbackward_sol (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  qsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (iqp), dimension (:,:), intent (inout) :: a
    real    (iqp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (iqp), dimension   (:), allocatable :: x, y, v
    real    (iqp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  qlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   qforward_sol (a, v, y, indx)
      call  qbackward_sol (a, y, x)
      b (1:n,p) = x (1:n)
```

```
    end do
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  qlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    real    (iqp), dimension (:,:), intent (inout) :: a
    real    (iqp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (iqp), parameter :: tenue = 1.0_iqp * 1.0d-20
    !
    !valida tamanios
    n = siza (size (a,1), size (a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_iqp) ) stop" singular matrix ... "
    d = 1.0_iqp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs ( a (k:n,k) ) )
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (a (k,k) == 0.0_iqp)  a (k,k) = tenue
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                    &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  qforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    real    (iqp), dimension (:,:), intent (in)    :: a
    real    (iqp), dimension (:)  , intent (inout) :: b
    real    (iqp), dimension (:)  , intent (inout) :: y
    integer (iin), dimension (:)  , intent (in)    :: indx
    real    (iqp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
```



```
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s = 0.0_iqp
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_iqp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  qbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    real    (iqp), dimension (:,:), intent (in)  :: a
    real    (iqp), dimension (:)  , intent (in)  :: y
    real    (iqp), dimension (:)  , intent (out) :: x
    real    (iqp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2),e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  wsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (iqp), dimension (:,:), intent (inout) :: a
    complex (iqp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (iqp), dimension (:)  , allocatable :: y
    real    (iqp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
```

```
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     wlu_factor (a, d, indx)
    call   wforward_sol (a, b, y, indx)
    call  wbackward_sol (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier (1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  wsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (iqp), dimension (:,:), intent (inout) :: a
    complex (iqp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (iqp), dimension   (:), allocatable :: x, y, v
    real    (iqp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  wlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   wforward_sol (a, v, y, indx)
      call  wbackward_sol (a, y, x)
```



```
      b (1:n,p) = x (1:n)
    end do
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  wlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    complex (iqp), dimension (:,:), intent (inout) :: a
    real    (iqp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (iqp), parameter :: tenue = 1.0_iqp * 1.0d-20
    !
    !valida tamanios
    n = siza (size(a,1), size(a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_iqp) ) stop" singular matrix ... "
    d = 1.0_iqp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs (a(k:n,k)))
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (abs (a(k,k)) == 0.0_iqp) a (k,k) = cmplx (tenue, 0.0_iqp)
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                        &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  wforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    complex (iqp), dimension (:,:), intent (in)    :: a
    complex (iqp), dimension (:)  , intent (inout) :: b
    complex (iqp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    complex (iqp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
```

```
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s = cmplx (0.0_iqp, 0.0_iqp)
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_iqp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  wbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    complex (iqp), dimension (:,:), intent (in)  :: a
    complex (iqp), dimension (:)  , intent (in)  :: y
    complex (iqp), dimension (:)  , intent (out) :: x
    complex (iqp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2), e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  mflops_g (time, n, m, raiz)
    real    (idp), dimension (:), intent (inout) :: time
    integer (iin), intent (in) :: n     ! orden de la matriz
    integer (iin), intent (in) :: m     ! nro de cargas
    character (*), intent (in) :: raiz  ! raiz archivo
    character (len(raiz)+4) :: arch     ! extension prefijada = 4
    character (4) :: exte = ".tim"      ! aqui la exten. es cte
    character (44):: s1, s2
    integer (iin) :: l, p
    integer (iin) :: ntime, h3
    real    (idp) :: total, ops, mflops
    !
    !nro de procesadores
    !p = 1                            ! secuencial
    p = number_of_processors ()       ! solo en HPF
    !
    !control
    ntime = size (time)
    if (ntime < 3) stop "error (mflops_g): size (time) < 3 "
    !
```



```
    !pasa a segundos
    call system_clock (count_rate = h3)  ! cte para pasar a seg
    if (h3 < 1) h3 = 1                   ! control
    time = time / dble (h3)              ! pasa de ciclos a seg
    !
    !tiempo total
    total = sum (time)
    !
    !estadistica
    ops = (2.0/3.0) * dble (n) ** 3 + 2.0 * dble (n) ** 2
    mflops = ops / (1.0e6 * total)
    !
    !define archivo
    l = len_trim (raiz)        ! longitud omitiendo blancos
    arch = raiz (1:l) // exte  ! apendiza en ese orden sin blancos
    !
    !resumen a disco
    s1 ="  p        n   m   twall_factorLU   twall_"
    s2 ="forward  twall_back            Mflops     "
    print *
    print 100, " archivo timer: " // arch
    print 100, s1 // s2
    !
    open (1, file = arch         , &
             status = "unknown"  , &
             position = "append")
    write (*,110) p, n, m, time (1:3), mflops
    write (1,110) p, n, m, time (1:3), mflops
    close (1, status = 'keep')
    !
    print *
    print 120, total, mflops
    !
    ! formatos
    100 format (a)
    110 format (1x, i2, 1x, i8, 1x, i3, 3(1x,e16.8), 1x, f12.3 )
    120 format (" elapsed time = ",e16.8," seconds ;",f12.3," Mflops")
    !
  end subroutine
end module
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!   Gauss con PIVOTAJE parcial con (A, B), donde
!   A ; matriz de coeficientes
!   B ; matriz de cargas
! ---------------------------------------------------------------------
!$Id: 2006/09/25 jdelia Exp $
module  m_gauss1       ! with or without mapping
  use   m_ctes
  use   m_temps
  use   m_tools1
  implicit none
  private              ! por omision todo es privado
  !precision doble     ! without any mapping
  public  dsolve_b1    , zsolve_b1
  public  dsolve_bm    , zsolve_bm
  public  dlu_factor   , zlu_factor
  public  dforward_sol , zforward_sol
  public  dbackward_sol, zbackward_sol
  !precision double    ! with descriptive Mapping
  public  dsolve_b1_adp    , zsolve_b1_adp
  public  dsolve_bm_adp    , zsolve_bm_adp
  public  dlu_factor_adp   , zlu_factor_adp
  public  dforward_sol_adp , zforward_sol_adp
  public  dbackward_sol_adp, zbackward_sol_adp
  !precision cuadruple ! it's not possible due to ADAPTOR restrictions
  !variables internas
  character (11), parameter :: c = "m_gauss1 > "
  !para medir tiempos
  real    (idp), dimension (3) :: time
  integer (iin) :: h1, h2
  contains
  !
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     dlu_factor (a, d, indx)
    call   dforward_sol (a, b, y, indx)
    call  dbackward_sol (a, y, b)
    !
    !estadistica
```



```
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  dlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   dforward_sol (a, v, y, indx)
      call  dbackward_sol (a, y, x)
      b (1:n,p) = x (1:n)
    end do
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
```

```
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    !
    !valida tamanios
    n = siza (size (a,1), size (a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs ( a (k:n,k) ) )
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (a (k,k) == 0.0_idp)  a (k,k) = tenue
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                        &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    real    (idp), dimension (:,:), intent (in)    :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    real    (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    real    (idp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s = 0.0_idp
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_idp) then
```



```
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    real    (idp), dimension (:,:), intent (in)  :: a
    real    (idp), dimension (:)  , intent (in)  :: y
    real    (idp), dimension (:)  , intent (out) :: x
    real    (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2),e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  zsolve_b1 (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     zlu_factor (a, d, indx)
    call   zforward_sol (a, b, y, indx)
    call  zbackward_sol (a, y, b)
    !
```

```
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier (1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zsolve_bm (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  zlu_factor (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   zforward_sol (a, v, y, indx)
      call  zbackward_sol (a, y, x)
      b (1:n,p) = x (1:n)
    end do
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
```



```
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zlu_factor (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    complex (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    !
    !valida tamanios
    n = siza (size(a,1), size(a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs (a(k:n,k)))
      if (kmax .ne. k) then
        call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (abs (a(k,k)) == 0.0_idp) a (k,k) = cmplx (tenue, 0.0_idp)
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                        &
              spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
              spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zforward_sol (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    complex (idp), dimension (:,:), intent (in)    :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    complex (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    complex (idp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s     = cmplx (0.0_idp, 0.0_idp)
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
```

```
     elseif (prima .ne. 0.0_idp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zbackward_sol (a, y, x)
    integer (iin) :: i, j, k, n
    complex (idp), dimension (:,:), intent (in)  :: a
    complex (idp), dimension (:)  , intent (in)  :: y
    complex (idp), dimension (:)  , intent (out) :: x
    complex (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2), e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_b1_adp (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: b
    !hpf$ align (i)   with   a (i,*) :: y
    !hpf$ align (i)   with   a (i,*) :: d, indx
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
```



```
    !
    !tareas
    call     dlu_factor_adp (a, d, indx)
    call   dforward_sol_adp (a, b, y, indx)
    call  dbackward_sol_adp (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_bm_adp (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    real    (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i,*) with * a (i,*) :: b
    !hpf$ align (i)   with   a (i,*) :: x, y, v
    !hpf$ align (i)   with   a (i,*) :: d, indx
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  dlu_factor_adp (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   dforward_sol_adp (a, v, y, indx)
      call  dbackward_sol_adp (a, y, x)
      b (1:n,p) = x (1:n)
    end do
    !
```

```
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dlu_factor_adp (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    real    (idp), dimension (size(a,1)) :: t
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: d, indx
    !hpf$ align (i)   with   a (i,*) :: t
    !
    !valida tamanios
    n = siza (size (a,1), size (a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs ( a (k:n,k) ) )
      if (kmax .ne. k) then
        t (1:n)      = a (kmax,1:n)
        a (kmax,1:n) = a (k,1:n)
        a (k,1:n)    = t (1:n)
!       call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (a (k,k) == 0.0_idp)  a (k,k) = tenue
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                       &
               spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
               spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dforward_sol_adp (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    real    (idp), dimension (:,:), intent (in)    :: a
    real    (idp), dimension (:)  , intent (inout) :: b
    real    (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    real    (idp) :: prima, s
```



```
    character (22) :: e = "size (a) | length (b,y)"
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: b, y, indx
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s = 0.0_idp
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_idp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  dbackward_sol_adp (a, y, x)
    integer (iin) :: i, j, k, n
    real    (idp), dimension (:,:), intent (in)  :: a
    real    (idp), dimension (:)  , intent (in)  :: y
    real    (idp), dimension (:)  , intent (out) :: x
    real    (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: y, x
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2),e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  zsolve_b1_adp (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension (:)  , allocatable :: y
    real    (idp), dimension (:)  , allocatable :: d
    integer (iin), dimension (:)  , allocatable :: indx
    integer (iin) :: n, m
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: b
    !hpf$ align (i)   with   a (i,*) :: y
```

```
    !hpf$ align (i)   with   a (i,*) :: d, indx
    !
    !valida tamanios
    n = siza (size (b), size (a,1), size (a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (y    (1:n), stat = ier (1) )
    allocate (d    (1:n), stat = ier (2) )
    allocate (indx (1:n), stat = ier (3) )
    if (any (ier .ne. 0)) call errata (ier(1:3), "solve: aloca")
    !
    !decide si imprimir
    if  ( present (imprime) ) then
      if (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !tareas
    call     zlu_factor_adp (a, d, indx)
    call   zforward_sol_adp (a, b, y, indx)
    call  zbackward_sol_adp (a, y, b)
    !
    !estadistica
    if  (eco == 1) then
      m = 1
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (3) )
    deallocate (d   , stat = ier (2) )
    deallocate (y   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier (1:3), "solve: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zsolve_bm_adp (a, b, imprime)
    integer (iin) :: i, j, k, p, eco
    complex (idp), dimension (:,:), intent (inout) :: a
    complex (idp), dimension (:,:), intent (inout) :: b
    logical      , optional       , intent (in)    :: imprime
    complex (idp), dimension   (:), allocatable :: x, y, v
    real    (idp), dimension   (:), allocatable :: d
    integer (iin), dimension   (:), allocatable :: indx
    integer (iin) :: n, m
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i,*) with * a (i,*) :: b
    !hpf$ align (i)   with   a (i,*) :: x, y, v
    !hpf$ align (i)   with   a (i,*) :: d, indx
    !
    !valida tamanios
    n = siza (size (b,1), size (a,1), size (a,2), "size (a)" )
    m =       size (b,2)
    !
    !aloca
    ier = 0
    allocate (x    (1:n), stat = ier (1) )
    allocate (y    (1:n), stat = ier (2) )
    allocate (v    (1:n), stat = ier (3) )
    allocate (d    (1:n), stat = ier (4) )
    allocate (indx (1:n), stat = ier (5) )
    if (any (ier .ne. 0)) call errata (ier(1:5), "solve: aloca")
    !
```



```
    !decide si imprimir
    if  ( present (imprime) ) then
      if  (imprime) then
        eco = 1
      else
        eco = 0
      end if
    end if
    !
    !re-inicia cronometro:
    time = 0.0
    !
    call  zlu_factor_adp (a, d, indx)
    do  p = 1, m
      v (1:n) = b (1:n,p)
      call   zforward_sol_adp (a, v, y, indx)
      call  zbackward_sol_adp (a, y, x)
      b (1:n,p) = x (1:n)
    end do
    !
    !estadistica
    if  (eco == 1) then
      call  mflops_g (time, n, m, "gauss_pivo")
    end if
    !
    !dloca
    ier = 0
    deallocate (indx, stat = ier (5) )
    deallocate (d   , stat = ier (4) )
    deallocate (v   , stat = ier (3) )
    deallocate (y   , stat = ier (2) )
    deallocate (x   , stat = ier (1) )
    if (any(ier.ne.0)) call errata (ier(1:5), "solve: dloca")
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zlu_factor_adp (a, d, indx)
    integer (iin) :: i, j, k, n,kmax
    complex (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension (:)  , intent (out)   :: d
    integer (iin), dimension (:)  , intent (out)   :: indx
    real    (idp), parameter :: tenue = 1.0d-20
    complex (idp), dimension (size(a,1)) :: t
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: d, indx
    !hpf$ align (i)   with   a (*,i) :: t
    !
    !valida tamanios
    n = siza (size(a,1), size(a,2), "size (a)" )
    !
    !
    call system_clock (count = h1)
    !scaling
    indx = 0
    d = maxval (abs (a), dim = 2) ! loop on rows
    if (any (d == 0.0_idp) ) stop" singular matrix ... "
    d = 1.0_idp / d
    do  k = 1, n
      !partial pivoting
      kmax = (k - 1) + maxlocat (d (k:n) * abs (a(k:n,k)))
      if (kmax .ne. k) then
        t (1:n)      = a (kmax,1:n)
        a (kmax,1:n) = a (k,1:n)
        a (k,1:n)    = t (1:n)
!       call  swap (a (kmax,:), a (k,:))
        d (kmax) = d (k)
      end if
      indx (k) = kmax
      if  (abs (a(k,k)) == 0.0_idp) a (k,k) = cmplx (tenue, 0.0_idp)
```

```
      !reduction with outer product
      a (k+1:n,k:k)   = a (k+1:n,k:k) / a (k,k)
      a (k+1:n,k+1:n) = a (k+1:n,k+1:n) -                        &
             spread (a (k+1:n,k), dim = 2, ncopies = n-k) *   &
             spread (a (k,k+1:n), dim = 1, ncopies = n-k)
    end do
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zforward_sol_adp (a, b, y, indx)
    integer (iin) :: i, j, k, l, n
    complex (idp), dimension (:,:), intent (in)    :: a
    complex (idp), dimension (:)  , intent (inout) :: b
    complex (idp), dimension (:)  , intent (out)   :: y
    integer (iin), dimension (:)  , intent (inout) :: indx
    complex (idp) :: prima, s
    character (22) :: e = "size (a) | length (b,y)"
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: b, y, indx
    !
    !valida tamanios
    n = siza (size (b), size (y), size (a,1), size (a,2), e)
    if (any (indx < 1)) stop "error (forward_sol): index  < 1 "
    if (any (indx > n)) stop "error (forward_sol): index  > n "
    !
    !
    call system_clock (count = h1)
    k = 0
    do  i = 1, n
     l     = indx (i)
     prima = b (l)
     b (l) = b (i)
     s     = cmplx (0.0_idp, 0.0_idp)
     if     (k .ne. 0) then
       s = sum ( a (i,k:i-1) * y (k:i-1) )
     elseif (prima .ne. 0.0_idp) then
       k = i
     end if
     y (i) = prima - s
    end do
    call system_clock (count = h2)
    time (2) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  zbackward_sol_adp (a, y, x)
    integer (iin) :: i, j, k, n
    complex (idp), dimension (:,:), intent (in)  :: a
    complex (idp), dimension (:)  , intent (in)  :: y
    complex (idp), dimension (:)  , intent (out) :: x
    complex (idp) :: s
    character (22) :: e = "size (a) | length (x,y)"
    !hpf$ distribute * (*,block) :: a
    !hpf$ align (i)   with * a (i,*) :: y, x
    !
    !valida tamanios
    n = siza (size (x), size (y), size (a,1), size (a,2), e)
    !
    !
    call system_clock (count = h1)
    do  i = n, 1, -1
      s = sum ( a (i,i+1:n) * x (i+1:n) )
      x (i) = (y (i) - s) / a (i,i)
```



```
    end do
    call system_clock (count = h2)
    time (3) = dble (h2 - h1)
    !
    !
  end subroutine
  !
  !---------------------------------------------------------------------
  subroutine  mflops_g (time, n, m, raiz)
    real    (idp), dimension (:), intent (inout) :: time
    integer (iin), intent (in) :: n     ! orden de la matriz
    integer (iin), intent (in) :: m     ! nro de cargas
    character (*), intent (in) :: raiz  ! raiz archivo
    character (len(raiz)+4) :: arch     ! extension prefijada = 4
    character (4) :: exte = ".tim"      ! aqui la exten. es cte
    character (44):: s1, s2
    integer (iin) :: l, p
    integer (iin) :: ntime, h3
    real    (idp) :: total, ops, mflops
    !
    !nro de procesadores
    !p = 1                              ! secuencial
    p = number_of_processors ()         ! solo en HPF
    !
    !control
    ntime = size (time)
    if (ntime < 3) stop "error (mflops_g): size (time) < 3 "
    !
    !pasa a segundos
    call system_clock (count_rate = h3)  ! cte para pasar a seg
    if (h3 < 1) h3 = 1                   ! control
    time = time / dble (h3)              ! pasa de ciclos a seg
    !
    !tiempo total
    total = sum (time)
    !
    !estadistica
    ops = (2.0/3.0) * dble (n) ** 3 + 2.0 * dble (n) ** 2
    mflops = ops / (1.0e6 * total)
    !
    !define archivo
    l = len_trim (raiz)        ! longitud omitiendo blancos
    arch = raiz (1:l) // exte  ! apendiza en ese orden sin blancos
    !
    !resumen a disco
    s1 ="  p        n   m   twall_factorLU   twall_"
    s2 ="forward  twall_back            Mflops     "
    print *
    print 100, " archivo timer: " // arch
    print 100, s1 // s2
    !
    open (1, file = arch         , &
             status = "unknown"  , &
             position = "append")
    write (*,110) p, n, m, time (1:3), mflops
    write (1,110) p, n, m, time (1:3), mflops
    close (1, status = 'keep')
    !
    print *
    print 120, total, mflops
    !
    ! formatos
    100 format (a)
    110 format (1x, i2, 1x, i8, 1x, i3, 3(1x,e16.8), 1x, f12.3 )
    120 format (" elapsed time = ",e16.8," seconds ;",f12.3," Mflops")
    !
  end subroutine
end module
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!   Gauss con (A, B), donde
!   A ; matriz de coeficientes
!   B ; matriz de cargas
! ---------------------------------------------------------------------
module  m_krylov0     ! without mapping
  use   m_ctes
  use   m_temps
  use   m_tools0
  use   m_gauss0
  implicit none
  private              ! por omision todo es privado
  !precision doble     ! without mapping
  public  dsolve_fom
  !precision cuadruple ! without mapping due to ADAPTOR restrictions
  public  qsolve_fom
  !
  character (12), parameter :: c = "m_krylov0 > "
  !para medir tiempos
  real    (idp), dimension (1) :: time
  integer (iin) :: h1, h2
  contains
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_fom (a, b, m)
    integer (iin) :: i, j, k, p
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension   (:), intent (inout) :: b
    integer (iin)                 , intent (inout) :: m
    real    (idp), dimension (:,:), allocatable :: h, v
    real    (idp), dimension (:)  , allocatable :: e1, y
    real    (idp), dimension (:)  , allocatable :: x0, r0, x, r
    !
    real    (idp), dimension (:,:), allocatable :: e
    real    (idp), dimension (:)  , allocatable :: q, t, u
    real    (idp) :: h0
    !
    real    (idp) :: beta0, beta1
    integer (iin) :: n
    !
    !valia tamanios
    n = siza (size(b), size(a,1), size(a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (x0 (n)    , stat = ier  (1) )
    allocate (r0 (n)    , stat = ier  (2) )
    allocate (x  (n)    , stat = ier  (3) )
    allocate (r  (n)    , stat = ier  (4) )
    allocate (v  (n,m+1), stat = ier  (5) )
    allocate (h  (m+1,m), stat = ier  (6) )
    allocate (e1 (m)    , stat = ier  (7) )
    allocate (y  (m)    , stat = ier  (8) )
    allocate (q (n)     , stat = ier  (9) )
    allocate (t (n)     , stat = ier (10) )
    allocate (u (n)     , stat = ier (11) )
    allocate (e (m,m+1) , stat = ier (12) )
    if (any (ier .ne. 0)) call errata (ier(1:12), "dsolve_fom: aloca")
    !
    !cronometra
    time = cero
    call system_clock (count = h1)
    !
    !solucion inicial
    x0 = 0.0_idp
    !
    !Vector residuo inicial "r0" y su norma L2 en "z0"
    r0 = b - matmul (a, x0)
    beta0 = sqrt (sum (r0 * r0) )  ! residuo inicial
```



```
    !
    if (beta0 > tol) then
      !
      !Algoritmo de Arnoldi: devuelve H (m+1,m) y V (n,m), con m < n
      !
      !1ra columna con el residuo inicial normalizado
      v (:,1) = r0 / beta0
      !
      !Lazo iterativo: llena columnas V_m = { v_1, v_2, ..., v_{m+1} } y
      !las entradas en la matriz F (m+1,m)
      p = 0
      do  j = 1, m
        t = v (:,j)
        u = matmul (a, v (:,j))
        do  i = 1, j
          h (i,j) = sum (u * v (:,i)) ! f_ij = (Av_j, v_i) ; i=1,2,...,j
        end do
        q = 0.0_idp
        do   i = 1, j
          q = q + h (i,j) * v (:,i)
        end do
        t = u - q
        h0 = sqrt ( sum (t * t) )
        h (j+1,j) = h0
        if  (h0 > tol) then
          p = j
          v (:,j+1) = t / h0
        else
          exit
        end if
      end do
      !actualiza "m"
      m = p
      if (m < 1) stop "error (solve_fom): m < 1 "
      !
      !solucion auxiliar por ortogonalizacion completa del residuo inicial
      !a partir del sistema cuadrado: Hm (m,m) * y (m,1) = e1 (m,1)
      e1 = 0.0_idp  ;  e1 (1) = beta0  ;  y = e1
      call  solve_b1 (h(1:m,1:m), y(1:m))
      !
      !solucion iterada m-veces x^{(m)} del SEL
      x = x0 + matmul (v(1:n,1:m), y(1:m))
    else
      x = x0
    end if
    !
    r = b - matmul (a,x)
    beta1 = sqrt (sum (r * r) )
    !cronometrea
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !estadistica
    call  mflops_k (time, n, m, "krylov")
    !
    print *
    print *, "unknows number          ;     n = ", n
    print *, "efective Krylov subspace;     m = ", m
    print *, "initial || r ||_2       ; beta0 = ", beta0
    print *, "last    || r ||_2       ; beta1 = ", beta1
    !
    !dloca
    ier = 0
    deallocate (e , stat = ier (12) )
    deallocate (u , stat = ier (11) )
    deallocate (t , stat = ier (10) )
    deallocate (q , stat = ier  (9) )
    deallocate (y , stat = ier  (8) )
    deallocate (e1, stat = ier  (7) )
```

```
    deallocate (h , stat = ier  (6) )
    deallocate (v , stat = ier  (5) )
    deallocate (r , stat = ier  (4) )
    deallocate (x , stat = ier  (3) )
    deallocate (r0, stat = ier  (2) )
    deallocate (x0, stat = ier  (1) )
    if (any(ier.ne.0)) call errata (ier(1:12), "dsolve_fom: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------
  subroutine  qsolve_fom (a, b, m)
    integer (iin) :: i, j, k, p
    real    (iqp), dimension (:,:), intent (inout) :: a
    real    (iqp), dimension   (:), intent (inout) :: b
    integer (iin)                 , intent (inout) :: m
    real    (iqp), dimension (:,:), allocatable :: h, v
    real    (iqp), dimension (:)  , allocatable :: e1, y
    real    (iqp), dimension (:)  , allocatable :: x0, r0, x, r
    !
    real    (iqp), dimension (:,:), allocatable :: e
    real    (iqp), dimension (:)  , allocatable :: q, t, u
    real    (iqp) :: h0
    !
    real    (iqp) :: beta0, beta1
    integer (iin) :: n
    !
    !valia tamanios
    n = siza (size(b), size(a,1), size(a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (x0 (n)    , stat = ier  (1) )
    allocate (r0 (n)    , stat = ier  (2) )
    allocate (x  (n)    , stat = ier  (3) )
    allocate (r  (n)    , stat = ier  (4) )
    allocate (v  (n,m+1), stat = ier  (5) )
    allocate (h  (m+1,m), stat = ier  (6) )
    allocate (e1 (m)    , stat = ier  (7) )
    allocate (y  (m)    , stat = ier  (8) )
    allocate (q (n)     , stat = ier  (9) )
    allocate (t (n)     , stat = ier (10) )
    allocate (u (n)     , stat = ier (11) )
    allocate (e (m,m+1) , stat = ier (12) )
    if (any (ier .ne. 0)) call errata (ier(1:12), "qsolve_fom: aloca")
    !
    !cronometra
    time = cero
    call system_clock (count = h1)
    !
    !solucion inicial
    x0 = 0.0_iqp
    !
    !Vector residuo inicial "r0" y su norma L2 en "z0"
    r0 = b - matmul (a, x0)
    beta0 = sqrt (sum (r0 * r0) )  ! residuo inicial
    !
    if (beta0 > tol) then
      !
      !Algoritmo de Arnoldi: devuelve H (m+1,m) y V (n,m), con m < n
      !
      !1ra columna con el residuo inicial normalizado
      v (:,1) = r0 / beta0
      !
      !Lazo iterativo: llena columnas V_m = { v_1, v_2, ..., v_{m+1} } y
      !las entradas en la matriz F (m+1,m)
      p = 0
      do  j = 1, m
        t = v (:,j)
```



```
        u = matmul (a, v (:,j))
        do  i = 1, j
          h (i,j) = sum (u * v (:,i)) ! f_ij = (Av_j, v_i) ; i=1,2,...,j
        end do
        q = 0.0_iqp
        do   i = 1, j
          q = q + h (i,j) * v (:,i)
        end do
        t = u - q
        h0 = sqrt ( sum (t * t) )
        h (j+1,j) = h0
        if  (h0 > tol) then
          p = j
          v (:,j+1) = t / h0
        else
          exit
        end if
      end do
      !actualiza "m"
      m = p
      if (m < 1) stop "error (solve_fom): m < 1 "
      !
      !solucion auxiliar por ortogonalizacion completa del residuo inicial
      !a partir del sistema cuadrado: Hm (m,m) * y (m,1) = e1 (m,1)
      e1 = 0.0_iqp  ;  e1 (1) = beta0  ;  y = e1
      call  solve_b1 (h(1:m,1:m), y(1:m))
      !
      !solucion iterada m-veces x^{(m)} del SEL
      x = x0 + matmul (v(1:n,1:m), y(1:m))
    else
      x = x0
    end if
    !
    r = b - matmul (a,x)
    beta1 = sqrt (sum (r * r) )
    !cronometrea
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !estadistica
    call  mflops_k (time, n, m, "krylov")
    !
    print *
    print *, "unknows number          ;     n = ", n
    print *, "efective Krylov subspace;     m = ", m
    print *, "initial || r ||_2       ; beta0 = ", beta0
    print *, "last    || r ||_2       ; beta1 = ", beta1
    !
    !dloca
    ier = 0
    deallocate (e , stat = ier (12) )
    deallocate (u , stat = ier (11) )
    deallocate (t , stat = ier (10) )
    deallocate (q , stat = ier  (9) )
    deallocate (y , stat = ier  (8) )
    deallocate (e1, stat = ier  (7) )
    deallocate (h , stat = ier  (6) )
    deallocate (v , stat = ier  (5) )
    deallocate (r , stat = ier  (4) )
    deallocate (x , stat = ier  (3) )
    deallocate (r0, stat = ier  (2) )
    deallocate (x0, stat = ier  (1) )
    if (any(ier.ne.0)) call errata (ier(1:12), "qsolve_fom: dloca")
    !
  end subroutine
  !
  !---------------------------------------------------------------
  subroutine  mflops_k (time, n, m, raiz)
    real    (idp), dimension (:), intent (inout) :: time
```

```
    integer (iin), intent (in) :: n     ! orden de la matriz
    integer (iin), intent (in) :: m     ! subespacio de Krylov
    character (*), intent (in) :: raiz  ! raiz archivo
    character (len(raiz)+4) :: arch     ! extension prefijada = 4
    character (4) :: exte = ".tim"      ! aqui la exten. es cte
    character (44):: s1
    integer (iin) :: l, p
    integer (iin) :: ntime, h3
    real    (idp) :: total, ops, mflops, c1, c2
    !
    !nro de procesadores
    !p = 1                              ! secuencial
    p = number_of_processors ()         ! solo en HPF
    !
    !control
    ntime = size (time)
    if (ntime < 1) stop "error (mflops_k): size (time) < 1 "
    !
    !pasa a segundos
    call system_clock (count_rate = h3)  ! cte para pasar a seg
    if (h3 < 1) h3 = 1                   ! control
    time = time / dble (h3)              ! pasa de ciclos a seg
    !
    !tiempo total
    total = sum (time)
    !
    !estadistica
    c1 = (2.0/3.0) * dble (m) ** 3 + 2.0 * dble (m) ** 2
    c2 = dble (m + 2) * (n ** 2)
    ops = c1 + c2
    mflops = ops / (1.0e6 * total)
    !
    !define archivo
    l = len_trim (raiz)        ! longitud omitiendo blancos
    arch = raiz (1:l) // exte  ! apendiza en ese orden sin blancos
    !
    !resumen a disco
    s1 ="  p        n   m   total_wall_time  Mflops   "
    print *
    print 100, " archivo timer: " // arch
    print 100, s1
    !
    open (1, file = arch         , &
             status = "unknown"  , &
             position = "append")
    write (*,110)  p, n, m, total, mflops
    write (1,110)  p, n, m, total, mflops
    close (1, status = 'keep')
    !
    print *
    print 120, total, mflops
    !
    ! formatos
    100 format (a)
    110 format (1x, i2, 1x, i8, 1x, i3, 1 (1x,e16.8), 1x, f12.3 )
    120 format (" elapsed time = ",e16.8," seconds ;",f12.3," Mflops")
    !
  end subroutine
  !
end module
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!   Gauss con (A, B), donde
!   A ; matriz de coeficientes
!   B ; matriz de cargas
! ---------------------------------------------------------------------
module  m_krylov1     ! con Descriptive Mapping
  use   m_ctes
  use   m_temps
  use   m_tools1
  use   m_gauss1
  implicit none
  private             ! por omision todo es privado
  !precision doble    ! with Descriptive Mapping
  public  dsolve_fom
  !
  character (11), parameter :: c = "m_krylov > "
  !para medir tiempos
  real    (idp), dimension (1) :: time
  integer (iin) :: h1, h2
  contains
  !
  !---------------------------------------------------------------------
  subroutine  dsolve_fom (a, b, m)
    integer (iin) :: i, j, k, p
    real    (idp), dimension (:,:), intent (inout) :: a
    real    (idp), dimension   (:), intent (inout) :: b
    integer (iin)                 , intent (inout) :: m
    real    (idp), dimension (:,:), allocatable :: h, v
    real    (idp), dimension (:)  , allocatable :: e1, y
    real    (idp), dimension (:)  , allocatable :: x0, r0, x, r
    !hpf$ distribute * (block,*) :: a
    !hpf$ align (i)   with * a (i,*) :: b
    !hpf$ align (i,*) with   a (i,*) :: v
    !hpf$ align (i)   with   a (i,*) :: x0, r0, x, r
    !hpf$ align (i)   with   a (i,*) :: q, t, u
    !
    real    (idp), dimension (:,:), allocatable :: e
    real    (idp), dimension (:)  , allocatable :: q, t, u
    real    (idp) :: h0
    !
    real    (idp) :: beta0, beta1
    integer (iin) :: n
    !
    !valia tamanios
    n = siza (size(b), size(a,1), size(a,2), "size (a)" )
    !
    !aloca
    ier = 0
    allocate (x0 (n)    , stat = ier  (1) )
    allocate (r0 (n)    , stat = ier  (2) )
    allocate (x  (n)    , stat = ier  (3) )
    allocate (r  (n)    , stat = ier  (4) )
    allocate (v  (n,m+1), stat = ier  (5) )
    allocate (h  (m+1,m), stat = ier  (6) )
    allocate (e1 (m)    , stat = ier  (7) )
    allocate (y  (m)    , stat = ier  (8) )
    allocate (q (n)     , stat = ier  (9) )
    allocate (t (n)     , stat = ier (10) )
    allocate (u (n)     , stat = ier (11) )
    allocate (e (m,m+1) , stat = ier (12) )
    if (any (ier .ne. 0)) call errata (ier(1:12), "dsolve_fom: aloca")
    !
    !cronometra
    time = cero
    call system_clock (count = h1)
    !
    !solucion inicial
    x0 = 0.0_idp
    !
```



```
    !Vector residuo inicial "r0" y su norma L2 en "z0"
    r0 = b - matmul (a, x0)
    beta0 = sqrt (sum (r0 * r0) )  ! residuo inicial
    !
    if (beta0 > tol) then
      !
      !Algoritmo de Arnoldi: devuelve H (m+1,m) y V (n,m), con m < n
      !
      !1ra columna con el residuo inicial normalizado
      v (:,1) = r0 / beta0
      !
      !Lazo iterativo: llena columnas V_m = { v_1, v_2, ..., v_{m+1} } y
      !las entradas en la matriz F (m+1,m)
      p = 0
      do  j = 1, m
        t = v (:,j)
        u = matmul (a, v (:,j))
        do  i = 1, j
          h (i,j) = sum (u * v (:,i)) ! f_ij = (Av_j, v_i) ; i=1,2,...,j
        end do
        q = 0.0_idp
        do   i = 1, j
          q = q + h (i,j) * v (:,i)
        end do
        t = u - q
        h0 = sqrt ( sum (t * t) )
        h (j+1,j) = h0
        if  (h0 > tol) then
          p = j
          v (:,j+1) = t / h0
        else
          exit
        end if
      end do
      !actualiza "m"
      m = p
      if (m < 1) stop "error (solve_fom): m < 1 "
      !
      !solucion auxiliar por ortogonalizacion completa del residuo inicial
      !a partir del sistema cuadrado: Hm (m,m) * y (m,1) = e1 (m,1)
      e1 = 0.0_idp  ;  e1 (1) = beta0  ;  y = e1
      call  dsolve_b1 (h(1:m,1:m), y(1:m))
      !
      !solucion iterada m-veces x^{(m)} del SEL
      x = x0 + matmul (v(1:n,1:m), y(1:m))
    else
      x = x0
    end if
    !
    r = b - matmul (a,x)
    beta1 = sqrt (sum (r * r) )
    !cronometrea
    call system_clock (count = h2)
    time (1) = dble (h2 - h1)
    !
    !estadistica
    call  mflops_k (time, n, m, "krylov")
    !
    print *
    print *, "unknows number          ;     n = ", n
    print *, "efective Krylov subspace;     m = ", m
    print *, "initial || r ||_2       ; beta0 = ", beta0
    print *, "last    || r ||_2       ; beta1 = ", beta1
    !
    !dloca
    ier = 0
    deallocate (e , stat = ier (12) )
    deallocate (u , stat = ier (11) )
    deallocate (t , stat = ier (10) )
```

```
    deallocate (q , stat = ier  (9) )
    deallocate (y , stat = ier  (8) )
    deallocate (e1, stat = ier  (7) )
    deallocate (h , stat = ier  (6) )
    deallocate (v , stat = ier  (5) )
    deallocate (r , stat = ier  (4) )
    deallocate (x , stat = ier  (3) )
    deallocate (r0, stat = ier  (2) )
    deallocate (x0, stat = ier  (1) )
    if (any(ier.ne.0)) call errata (ier(1:12), "dsolve_fom: dloca")
    !
  end subroutine
  !
  !-----------------------------------------------------------------
  subroutine  mflops_k (time, n, m, raiz)
    real    (idp), dimension (:), intent (inout) :: time
    integer (iin), intent (in) :: n     ! orden de la matriz
    integer (iin), intent (in) :: m     ! subespacio de Krylov
    character (*), intent (in) :: raiz  ! raiz archivo
    character (len(raiz)+4) :: arch     ! extension prefijada = 4
    character (4) :: exte = ".tim"      ! aqui la exten. es cte
    character (44):: s1
    integer (iin) :: l, p
    integer (iin) :: ntime, h3
    real    (idp) :: total, ops, mflops, c1, c2
    !
    !nro de procesadores
    !p = 1                              ! secuencial
    p = number_of_processors ()         ! solo en HPF
    !
    !control
    ntime = size (time)
    if (ntime < 1) stop "error (mflops_k): size (time) < 1 "
    !
    !pasa a segundos
    call system_clock (count_rate = h3)  ! cte para pasar a seg
    if (h3 < 1) h3 = 1                   ! control
    time = time / dble (h3)              ! pasa de ciclos a seg
    !
    !tiempo total
    total = sum (time)
    !
    !estadistica
    c1 = (2.0/3.0) * dble (m) ** 3 + 2.0 * dble (m) ** 2
    c2 = dble (m + 2) * (n ** 2)
    ops = c1 + c2
    mflops = ops / (1.0e6 * total)
    !
    !define archivo
    l = len_trim (raiz)        ! longitud omitiendo blancos
    arch = raiz (1:l) // exte  ! apendiza en ese orden sin blancos
    !
    !resumen a disco
    s1 ="  p        n   m   total_wall_time  Mflops   "
    print *
    print 100, " archivo timer: " // arch
    print 100, s1
    !
    open (1, file = arch         , &
             status = "unknown"  , &
             position = "append")
    write (*,110)  p, n, m, total, mflops
    write (1,110)  p, n, m, total, mflops
    close (1, status = 'keep')
    !
    print *
    print 120, total, mflops
    !
    ! formatos
```



```
    100 format (a)
    110 format (1x, i2, 1x, i8, 1x, i3, 1 (1x,e16.8), 1x, f12.3 )
    120 format (" elapsed time = ",e16.8," seconds ;",f12.3," Mflops")
    !
  end subroutine
  !
end module
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```