```
c   sol_0(*,i) : contains the imaginary and real parts of the solution for point
s such that ineq(i) != 0
c   sol(i) : contains solution for all points
c   The dimension of the geometric domain is : dim_32 = 2
c   The dimension of the vector field is : dim2 = 3
c

      subroutine array_sol (i_cond, num_modes, n_msh_el, n_msh_pts,
     *     n_ddl, neq, nnodes, n_core, bloch_vec, iindex, table_nod,
     *     table_N_E_F, type_el, ineq,
     *     ip_period_N, ip_period_N_E_F, x, x_N_E_F,
     *     v_cmplx, mode_pol, sol_0, sol)

      implicit none
      integer*8 i_cond, num_modes, n_msh_el, n_msh_pts, n_ddl
      integer*8 neq, nnodes
      integer*8 n_core(2), type_el(n_msh_el)
      integer*8 ineq(3,n_ddl), iindex(*)
      integer*8 ip_period_N(n_msh_pts), ip_period_N_E_F(n_ddl)
      integer*8 table_nod(nnodes,n_msh_el), table_N_E_F(14,n_msh_el)
      double precision bloch_vec(2), x(2,n_msh_pts), x_N_E_F(2,n_ddl)
      complex*16 sol_0(neq,num_modes)
c     sol(3, 1..nnodes,num_modes, n_msh_el)          contains the values of the
3 components at P2 interpolation nodes
c     sol(3, nnodes+1..nnodes+7,num_modes, n_msh_el) contains the values of Ez c
omponent at P3 interpolation nodes (per element: 6 edge-nodes and 1 interior nod
e)
      complex*16 sol(3,nnodes+7,num_modes,n_msh_el)
      complex*16 v_cmplx(num_modes), v_tmp(num_modes)
      complex*16 mode_pol(4,num_modes)


c     Local variables
      integer*8 nnodes_0, nddl_0, nddl_t
c     32-but integers for BLAS and LAPACK
      integer*4 dim_32
      parameter (nnodes_0 = 6)
      parameter (nddl_0 = 14)
      parameter (nddl_t=4)
      parameter (dim_32=2)
c
      double precision mode_comp(4)
      integer*8 nod_el_p(nnodes_0), basis_list(4,3,nddl_t)
      double precision xn(dim_32,nnodes_0), xel(dim_32,nnodes_0)
      complex*16 sol_el(3,nnodes_0+7)

      double precision phi1_list(3), grad1_mat0(dim_32,3)
      double precision grad1_mat(dim_32,3)

      double precision phi2_list(6), grad2_mat0(dim_32,6)
      double precision grad2_mat(dim_32,6)

      double precision phi3_list(10), grad3_mat0(dim_32,10)
      double precision grad3_mat(dim_32,10)

      double precision vec_phi_j(dim_32), curl_phi_j, phi_z_j
      double complex val_exp(nddl_0)

      integer*8 info_curved
      double precision xx(dim_32), xx_g(dim_32), det, r_tmp1
```



```
      double precision delta_xx(dim_32)
      double precision mat_B(dim_32,dim_32)
      double precision mat_T(dim_32,dim_32)

      double precision ZERO, ONE
      parameter (ZERO = 0.0D0)
      parameter (ONE = 1.0D0)
c
      integer*8 j, k, i1, j1, m, inod, typ_e
      integer*8 debug, i_sol_max
      integer*8 iel, ival, ival2, jtest, jp, ind_jp, j_eq
      double precision ddot
      complex*16 ii, z_tmp1, z_tmp2, z_sol_max
c
c  ii = sqrt(-1)
      ii = cmplx(0.0d0, 1.0d0, 8)
c
      debug = 0
c
      if ( nnodes .ne. 6 ) then
        write(*,*) "array_sol: problem nnodes = ", nnodes
        write(*,*) "array_sol: nnodes should be equal to 6 !"
        write(*,*) "array_sol: Aborting..."
        stop
      endif
c
      do j=1,num_modes
        j1=iindex(j)
        v_tmp(j) = v_cmplx(j1)
      enddo
      do j=1,num_modes
        v_cmplx(j) = v_tmp(j)
      enddo
c
c     Coordinates of the interpolation nodes
      call interp_nod_2d (nnodes, xn)
      do ival=1,num_modes
        ival2 = iindex(ival)
        do j=1,4
          mode_pol(j,ival) = 0.0d0
        enddo
        z_sol_max = 0.0d0
        i_sol_max = 0
        do iel=1,n_msh_el
          typ_e = type_el(iel)
          do j=1,4
            mode_comp(j) = 0.0d0
          enddo
          do inod=1,nnodes
            j = table_nod(inod,iel)
            nod_el_p(inod) = j
            xel(1,inod) = x(1,j)
            xel(2,inod) = x(2,j)
          enddo
          if (i_cond .eq. 2) then
c           Periodic boundary condition
            do inod=1,nnodes
              j = table_nod(inod,iel)
              k = ip_period_N(j)
              if (k .ne. 0) j=k
              nod_el_p(inod) = j
            enddo
```

```
          endif
          do j=1,nddl_0
            val_exp(j) = 1.0d0
          enddo
          if (i_cond .eq. 2) then
c           val_exp: Bloch mod ephase factor between the origin point and destin
ation point
c           For a pair of periodic points, one is chosen as origin and the other
 is the destination
            do j=1,nddl_0
              jp = table_N_E_F(j,iel)
              j1 = ip_period_N_E_F(jp)
              if (j1 .ne. 0) then
                do k=1,dim_32
                  delta_xx(k) = x_N_E_F(k,jp) - x_N_E_F(k,j1)
                enddo
                r_tmp1 = ddot(dim_32, bloch_vec, 1, delta_xx, 1)
                val_exp(j) = exp(ii*r_tmp1)
              endif
            enddo
          endif
          call basis_ls (nod_el_p, basis_list)
          call curved_elem_tri (nnodes, xel, info_curved, r_tmp1)
c         P2 Lagrange Interpolation nodes for the unit triangle
c         xn    = coordinate on the reference triangle
c          do inod=1,nnodes+7
c            do j=1,3
c              sol_el(j,inod) = 0.00
c            enddo
c          enddo
          do inod=1,nnodes
            do j=1,dim_32
              xx(j) = xn(j,inod)
            enddo
            do j=1,3
              sol_el(j,inod) = 0.00
            enddo
c           We will also need the gradients of the P1 element
            call phi1_2d_mat (xx, phi1_list, grad1_mat0)
c           grad2_mat0 = gradient on the reference triangle (P2 element)
            call phi2_2d_mat (xx, phi2_list, grad2_mat0)
c           grad3_mat0 = gradient on the reference tetrahedron (P3 element)
            call phi3_2d_mat (xx, phi3_list, grad3_mat0)

            if (info_curved .eq. 0) then
c             Rectilinear element
              call jacobian_p1_2d (xx, xel, nnodes,
     *                   xx_g, det, mat_B, mat_T)
              if (det .le. 0 .and. debug .eq. 1) then
                write(*,*) " !!!"
                write(*,*) "array_sol: det <= 0: iel, det ", iel, det
              endif
            else
c             Isoparametric element
              call jacobian_p2_2d (xx, xel, nnodes, phi2_list,
     *                     grad2_mat0, xx_g, det, mat_B, mat_T)
            endif
C           if(abs(det) .lt. 1.0d-10) then
            if(abs(det) .lt. 1.0d-20) then
              write(*,*)
              write(*,*) " ???"
              write(*,*) "array_sol: det = 0 : iel, det = ", iel, det
```



```
              write(*,*) "array_sol: Aborting..."
              stop
            endif
c           grad_i  = gradient on the actual triangle
c           grad_i  = Transpose(mat_T)*grad_i0
c           Calculation of the matrix-matrix product:
            call DGEMM('Transpose','N', dim_32, 3, dim_32, ONE, mat_T,
     *        dim_32, grad1_mat0, dim_32, ZERO, grad1_mat, dim_32)
            call DGEMM('Transpose','N', dim_32, 6, dim_32, ONE, mat_T,
     *        dim_32, grad2_mat0, dim_32, ZERO, grad2_mat, dim_32)
            call DGEMM('Transpose','N', dim_32, 10, dim_32, ONE, mat_T,
     *        dim_32, grad3_mat0, dim_32, ZERO, grad3_mat, dim_32)
c
c           Contribution to the transverse component
            do jtest=1,nddl_t
              do j_eq=1,3
                jp = table_N_E_F(jtest,iel)
                ind_jp = ineq(j_eq,jp)
                if (ind_jp .gt. 0) then
                  m  = basis_list(2, j_eq, jtest)
                  if (m .eq. inod) then
C                   !  inod correspond to a P2 interpolation node
c                                              The contribution is nonzero only whe
n m=inod.
c                   Determine the basis vector
                    call basis_vec (j_eq, jtest, basis_list, phi2_list,
     *                grad1_mat, grad2_mat, vec_phi_j, curl_phi_j)
                    z_tmp1 = sol_0(ind_jp, ival2)
                    z_tmp1 = z_tmp1 * val_exp(jtest)
                    do j=1,dim_32
                      z_tmp2 = z_tmp1 * vec_phi_j(j)
                      sol_el(j,inod) = sol_el(j,inod) + z_tmp2
                      if (m .ne. inod .and. abs(z_tmp2) .gt. 1.0d-7) then
                        write(*,*)
                        write(*,*) iel, inod, m, abs(z_tmp2)
                        write(*,*) "vec_phi_j = ", vec_phi_j
                        write(*,*) "xx = ", xx
                        write(*,*) "xn = ", (xn(k,inod),k=1,dim_32)
                        write(*,*) "phi2_list = ", phi2_list
                      endif
                    enddo
                  endif
                endif
              enddo
            enddo
c           Contribution to the longitudinal component
C             !  The initial P3 value of Ez isinterpolated over P2 nodes
            do jtest=nddl_t+1,nddl_0
C               ! 3
              do j_eq=1,1
                jp = table_N_E_F(jtest,iel)
                ind_jp = ineq(j_eq,jp)
                if (ind_jp .gt. 0) then
                  z_tmp1 = sol_0(ind_jp, ival2)
                  m  = jtest-nddl_t
                  phi_z_j = phi3_list(m)
                  z_tmp1 = z_tmp1 * val_exp(jtest)
                  z_tmp2 = z_tmp1 * phi_z_j
                  sol_el(3,inod) = sol_el(3,inod) + z_tmp2
                endif
              enddo
            enddo
```

```
            do j=1,3
              z_tmp2 = sol_el(j,inod)
              sol(j,inod,ival,iel) = z_tmp2
              if (abs(z_sol_max) .lt. abs(z_tmp2)) then
                z_sol_max = z_tmp2
                i_sol_max = table_nod(inod,iel)
              endif
            enddo
c           Contribution of the element iel to the mode component
            do j=1,3
              mode_comp(j) = mode_comp(j) + abs(sol_el(j,inod))**2
            enddo
          enddo
ccccccccccc
c         Saving the P3 values of Ez at: the 6 edge nodes and the interior node
          do inod=nnodes+1,nnodes+7
            do j=1,3
              sol_el(j,inod) = 0.00
            enddo
            jtest = nddl_t+inod-nnodes+3
            j_eq = 1
            jp = table_N_E_F(jtest,iel)
            ind_jp = ineq(j_eq,jp)
            if (ind_jp .gt. 0) then
              z_tmp1 = sol_0(ind_jp, ival2)
              z_tmp1 = z_tmp1 * val_exp(jtest)
              sol_el(3,inod) = z_tmp1
            endif
            do j=1,3
              z_tmp2 = sol_el(j,inod)
              sol(j,inod,ival,iel) = z_tmp2
            enddo
          enddo
ccccccccccc
c         Avarage values
          do j=1,3
            mode_comp(j) = abs(det)*mode_comp(j)/dble(nnodes)
          enddo
c         Add the contribution of the element iel to the mode component
          do j=1,3
            mode_pol(j,ival) = mode_pol(j,ival) + mode_comp(j)
          enddo
          if (typ_e .eq. n_core(1) .or. typ_e .eq. n_core(2)) then
            z_tmp2 = mode_comp(1) + mode_comp(2)
     *        + mode_comp(3)
            mode_pol(4,ival) = mode_pol(4,ival) + z_tmp2
          endif
        enddo
c       Total energy and normalization
        z_tmp2 = mode_pol(1,ival) + mode_pol(2,ival)
     *       + mode_pol(3,ival)
        if (abs(z_tmp2) .lt. 1.0d-10) then
          write(*,*) "array_sol: the total energy ",
     *       "is too small : ", z_tmp2
          write(*,*) "array_sol: ival ival2 = ", ival, ival2
          write(*,*) "array_sol: zero eigenvector; aborting..."
          stop
        endif
        do j=1,3
          mode_pol(j,ival) = mode_pol(j,ival) / z_tmp2
        enddo
        j=4
```



```
          mode_pol(j,ival) = mode_pol(j,ival) / z_tmp2
c       Check if the eigenvector is nonzero
        if (abs(z_sol_max) .lt. 1.0d-10) then
          z_sol_max = z_tmp2
          write(*,*) "array_sol: z_sol_max is too small"
          write(*,*) "array_sol: z_sol_max = ", z_sol_max
          write(*,*) "ival, ival2, num_modes = ", ival, ival2, num_modes
          write(*,*) "array_sol: zero eigenvector; aborting..."
          stop
        endif
c       Normalization so that the maximum field component is 1
        do iel=1,n_msh_el
          do inod=1,nnodes
            i1 = table_nod(inod,iel)
            do j=1,3
              z_tmp1 = sol(j,inod,ival,iel)/z_sol_max
              sol(j,inod,ival,iel) = z_tmp1
            enddo
            i1 = table_nod(inod,iel)
            if (i1 .eq. i_sol_max .and. debug .eq. 1) then
              write(*,*) "array_sol:"
              write(*,*) "ival, i1, iel = ", ival, i1, iel
              write(*,*) "array_sol: Field normalisaion point:"
              write(*,*) "x = ", dble(x(1,i1))
              write(*,*) "y = ", dble(x(2,i1))
              write(*,*) "i_sol_max = ", i_sol_max
              write(*,*) ival, i1, iel,
     *                   (dble(sol(j,inod,ival,iel)),j=1,3)
              write(*,*) ival, i1, iel,
     *                   (imag(sol(j,inod,ival,iel)),j=1,3)
            endif
          enddo
ccccccccccc
          do inod=nnodes+1,nnodes+7
            do j=1,3
              z_tmp1 = sol(j,inod,ival,iel)/z_sol_max
              sol(j,inod,ival,iel) = z_tmp1
            enddo
          enddo
ccccccccccc
        enddo
        do j=1,neq
          z_tmp1 = sol_0(j,ival2)/z_sol_max
          sol_0(j,ival2) = z_tmp1
        enddo

        if (debug .eq. 1) then
          write(*,*)
        endif
      enddo
c
      return
      end
```