```
subroutine moving_boundary (nval_EM_p, nval_EM_S, nval_AC, ival1,&
   ival2, ival3, nel, npt, nnodes, table_nod, type_el, x,&
   nb_typ_el, typ_select_in, typ_select_out,&
   soln_EM_p, soln_EM_S,&
   soln_AC, eps_lst, debug, overlap)

   use numbatmod
   integer(8) nel, npt, nnodes, nb_typ_el
   integer(8) type_el(nel)
   integer(8) table_nod(6,nel)
   double precision x(2,npt)
   integer(8) nval_EM_p, nval_EM_S, nval_AC, ival1, ival2, ival3
   integer(8) ival3s, ival2s, ival1s
   integer(8) typ_select_in, typ_select_out
   complex(8) soln_EM_p(3,nnodes,nval_EM_p,nel)
   complex(8) soln_EM_S(3,nnodes,nval_EM_S,nel)
   complex(8) soln_AC(3,nnodes,nval_AC,nel)
   complex(8) eps_lst(nb_typ_el)
   complex(8) overlap(nval_EM_S, nval_EM_p, nval_AC)

   !     Local variables
   integer(8) debug
   integer(8) nb_visited(npt)
   integer(8) ls_edge_endpoint(2,npt)
   integer(8) edge_direction(npt)
   integer(8) iel, inod, typ_e
   integer(8) inod_1, inod_2, inod_3, ls_inod(3)
   integer(8) j, j_1, j_2, j_3, i, k
   integer(8) nb_edges, nb_interface_edges
   integer(8) edge_endpoints(2,3), opposite_node(3)
   double precision xy_1(2), xy_2(2), xy_3(2), ls_xy(2,3)
   double precision edge_vec(2), edge_perp(2), vec_0(2)
   double precision edge_length, r_tmp
   complex(8) ls_n_dot(3), ls_n_cross(3,3)
   complex(8) vec(3,3)
   complex(8) n_dot_d(2)
   complex(8) eps_a, eps_b, tmp1, tmp2
   double precision p2_p2_p2_1d(3,3,3)

   !
   !
   !f2py intent(in) nval_EM_p, nval_EM_S, nval_AC
   !f2py intent(in) ival1, ival2, ival3, nb_typ_el
   !f2py intent(in) nel, npt, nnodes, table_nod, debug
   !f2py intent(in) type_el, x, soln_EM_p, soln_EM_S, soln_AC
   !f2py intent(in) typ_select_in, typ_select_out, eps_lst, debug
   !
   !f2py depend(table_nod) nnodes, nel
   !f2py depend(type_el) npt
   !f2py depend(x) npt
   !f2py depend(soln_EM_p) nnodes, nval_EM_p, nel
   !f2py depend(soln_EM_S) nnodes, nval_EM_S, nel
   !f2py depend(soln_AC) nnodes, nval_AC, nel
   !f2py depend(eps_lst) nb_typ_el
   !
   !f2py intent(out) overlap
   !
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !
   !     typ_select_in: Only the elements iel with type_el(iel)=typ_select_in wi
ll be analysed
   !     When nb_visited(j) is not zero: nb_visited(j) indicates the number of e
```



```
lement the edge j belongs
   !
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !



   nb_visited = 0
   ls_edge_endpoint = 0
   ls_edge_endpoint = 0
   edge_direction = 0

   edge_endpoints(1,1) = 1
   edge_endpoints(2,1) = 2
   edge_endpoints(1,2) = 2
   edge_endpoints(2,2) = 3
   edge_endpoints(1,3) = 3
   edge_endpoints(2,3) = 1

   !
   !     opposite_node(i): Node which is opposite to the edge i
   !     i = 1 is inod = 4 etc

   opposite_node(1) = 3
   opposite_node(2) = 1
   opposite_node(3) = 2

   overlap = D_ZERO

   do iel=1,nel
      typ_e = type_el(iel)
      if(typ_e == typ_select_in) then
         !          !  Scan the edges
         do inod=4,6
            j = table_nod(inod,iel)
            !             !  Will indicate the number of
            nb_visited(j) = nb_visited(j) + 1
         enddo
      endif
   enddo

   nb_edges = 0
   nb_interface_edges = 0
   do inod=1,npt
      if (nb_visited(inod) >= 1) then
         nb_edges = nb_edges + 1
      endif
      if (nb_visited(inod) == 1) then
         nb_interface_edges = nb_interface_edges + 1
      endif
   enddo

   if (debug .eq. 1) then
      write(*,*)
      write(*,*) "edge_orientation: npt, nel = ", npt, nel
      write(*,*) "edge_orientation: nb_edges = ", nb_edges
      write(*,*) "nb_interface_edges = ", nb_interface_edges
   endif

   !     Outward pointing normal vector to the interface edges
   do iel=1,nel
      typ_e = type_el(iel)
```

```
      if(typ_e == typ_select_in) then
         !           !  Scan the edges
         do inod=4,6
            j = table_nod(inod,iel)
            if (nb_visited(j) == 1) then
               inod_1 = edge_endpoints(1,inod-3)
               inod_2 = edge_endpoints(2,inod-3)
               ls_edge_endpoint(1,j) = table_nod(inod_1,iel)
               ls_edge_endpoint(2,j) = table_nod(inod_2,iel)
               xy_1(1) = x(1,table_nod(inod_1,iel))
               xy_1(2) = x(2,table_nod(inod_1,iel))
               xy_2(1) = x(1,table_nod(inod_2,iel))
               xy_2(2) = x(2,table_nod(inod_2,iel))
               !             edge_vec: vector parallel to the edge
               edge_vec(1) = xy_2(1) - xy_1(1)
               edge_vec(2) = xy_2(2) - xy_1(2)
               !             Normalisation of edge_vec
               r_tmp = sqrt(edge_vec(1)**2+edge_vec(2)**2)
               edge_vec(1) = edge_vec(1) / r_tmp
               edge_vec(2) = edge_vec(2) / r_tmp
               !             edge_vec: vector perpendicular to the edge (rotatio
n of edge_vec by -pi/2)
               edge_perp(1) = edge_vec(2)
               edge_perp(2) = -edge_vec(1)
               !             Node opposite to the edge inod
               inod_3 = opposite_node(inod-3)
               xy_3(1) = x(1,table_nod(inod_3,iel))
               xy_3(2) = x(2,table_nod(inod_3,iel))
               vec_0(1) = xy_3(1) - xy_1(1)
               vec_0(2) = xy_3(2) - xy_1(2)
               !             Scalar product of edge_perp and vec_0:
               r_tmp = edge_perp(1)*vec_0(1)+edge_perp(2)*vec_0(2)
               !             if r_tmp < 0: then edge_perp is oriented in the out
ward direction
               if( r_tmp < 0) then
                  edge_direction(j) = 1
               elseif( r_tmp > 0) then
                  edge_direction(j) = -1
               else
                  write(*,*) "edge_orientation: illegal:"
                  write(*,*) "edge_perp is perpendicular to vec_0"
                  write(*,*) "edge_orientation: Aborting..."
                  stop
               endif
            endif
         enddo
      endif
   enddo



   !     Numerical integration
   do iel=1,nel
      typ_e = type_el(iel)
      if(typ_e == typ_select_in) then
         eps_a = eps_lst(typ_e)
         if (typ_select_out .eq. -1) then
            eps_b = 1.0d0
         else
            eps_b = eps_lst(typ_select_out)
         endif
         !           !  Scan the edges
```



```
         do inod=4,6
            j = table_nod(inod,iel)
            xy_3(1) = x(1,j)
            xy_3(2) = x(2,j)
            if (ls_edge_endpoint(1,j) .ne. 0) then
               !               write(*,*) "an edge"
               inod_1 = ls_edge_endpoint(1,j)
               inod_2 = ls_edge_endpoint(2,j)
               xy_1(1) = x(1,inod_1)
               xy_1(2) = x(2,inod_1)
               xy_2(1) = x(1,inod_2)
               xy_2(2) = x(2,inod_2)
               !             List of the nodes coordinates
               ls_xy(1,1) = xy_1(1) !                 !  x-coord. of node 1
               ls_xy(2,1) = xy_1(2) !                 !  y-coord. of node 1
               ls_xy(1,2) = xy_2(1) !                 !  x-coord. of node 2
               ls_xy(2,2) = xy_2(2) !                 !  y-coord. of node 2
               ls_xy(1,3) = xy_3(1) !                 !  x-coord. of mid-edge node
               ls_xy(2,3) = xy_3(2) !                 !  y-coord. of mid-edge node


               edge_vec(1) = ls_xy(1,2) - ls_xy(1,1)
               edge_vec(2) = ls_xy(2,2) - ls_xy(2,1)

               !             Normalisation of edge_vec
               r_tmp = sqrt(edge_vec(1)**2+edge_vec(2)**2)
               edge_vec(1) = -1*edge_direction(j)*edge_vec(1) / r_tmp
               edge_vec(2) = -1*edge_direction(j)*edge_vec(2) / r_tmp

               !             edge_vec: vector perpendicular to the edge (rotatio
n of edge_vec by -pi/2)
               edge_perp(1) = -1*edge_vec(2)
               edge_perp(2) = edge_vec(1)


               r_tmp = (ls_xy(1,2) - ls_xy(1,1))**2 + (ls_xy(2,2) - ls_xy(2,1))*
*2
               edge_length = sqrt(r_tmp)
               call mat_p2_p2_p2_1d (p2_p2_p2_1d, edge_length)

               !             Identification number of the two end-points and mid
-edge point
               ls_inod(1) = edge_endpoints(1,inod-3)
               ls_inod(2) = edge_endpoints(2,inod-3)
               ls_inod(3) = inod


               ! If only want overlap of one given combination of EM modes and A
C mode.
               if (ival1 .ge. 0 .and. ival2 .ge. 0 .and. ival3 .ge. 0) then
                  !             Nodes of the edge
                  do j_1=1,3
                     !             (x,y,z)-components of the electric field
                     vec(1,1) = soln_EM_p(1,ls_inod(j_1),ival1,iel)
                     vec(2,1) = soln_EM_p(2,ls_inod(j_1),ival1,iel)
                     vec(3,1) = soln_EM_p(3,ls_inod(j_1),ival1,iel)

                     !             ls_n_dot(1): Normal component of vec(:,1)
                     ls_n_dot(1) = vec(1,1) * edge_perp(1) + vec(2,1) * edge_per
p(2)
                     ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                     ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
```

```
                        ls_n_cross(3,1) = vec(2,1) * edge_perp(1) - vec(1,1) * edge
_perp(2)

                        do j_2=1,3
                           !                   (x,y,z)-components of the electric fie
ld
                           vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2,iel)
                           vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2,iel)
                           vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2,iel)

                           !                  ls_n_dot(2): Normal component of vec(:
,2)
                           ls_n_dot(2) = vec(1,2) * edge_perp(1) + vec(2,2) * edge_
perp(2)

                           ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                           ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                           ls_n_cross(3,2) = vec(2,2) * edge_perp(1) - vec(1,2) * e
dge_perp(2)

                           do j_3=1,3
                              !                   (x,y,z)-components of the acousti
c field
                              vec(1,3) = soln_AC(1,ls_inod(j_3),ival3,iel)
                              vec(2,3) = soln_AC(2,ls_inod(j_3),ival3,iel)
                              vec(3,3) = soln_AC(3,ls_inod(j_3),ival3,iel)

                              !                  ls_n_dot(3): scalar product of ve
c(:,3) and normal vector edge_perp
                              ls_n_dot(3) = vec(1,3) * edge_perp(1) + vec(2,3) * ed
ge_perp(2)

                              tmp1 = (eps_a - eps_b)*SI_EPS_0
                              tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,2)&
                              &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                              &+ (ls_n_cross(3,1))*ls_n_cross(3,2))

                              n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                              n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)

                              tmp2 = (1.0d0/eps_b - 1.0d0/eps_a)*(1.0d0/SI_EPS_0)
                              tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                              r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)
                              overlap(ival1,ival2,ival3) = overlap(ival1,ival2,ival
3) +&
                              &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                           enddo
                        enddo
                     enddo

                     ! If want overlap of given EM mode 1 and 2 and all AC modes.
                  else if (ival1 .ge. 0 .and. ival2 .ge. 0 .and. ival3 .eq. -1) the
n
                     !             Nodes of the edge
                     do j_1=1,3

                        !                (x,y,z)-components of the electric field
                        vec(1,1) = conjg(soln_EM_S(1,ls_inod(j_1),ival1,iel))
                        vec(2,1) = conjg(soln_EM_S(2,ls_inod(j_1),ival1,iel))
                        vec(3,1) = conjg(soln_EM_S(3,ls_inod(j_1),ival1,iel))

                        !                ls_n_dot(1): Normal component of vec(:,1)
                        ls_n_dot(1) = vec(1,1) * edge_perp(1) + vec(2,1) * edge_per
p(2)
```



```
                        ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                        ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
                        ls_n_cross(3,1) = vec(2,1) * edge_perp(1) - vec(1,1) * edge
_perp(2)

                        do j_2=1,3
                           !                   (x,y,z)-components of the electric fie
ld
                           vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2,iel)
                           vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2,iel)
                           vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2,iel)

                           !                  ls_n_dot(2): Normal component of vec(:
,2)
                           ls_n_dot(2) = vec(1,2) * edge_perp(1) + vec(2,2) * edge_
perp(2)

                           ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                           ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                           ls_n_cross(3,2) = vec(2,2) * edge_perp(1) - vec(1,2) * e
dge_perp(2)

                           do ival3s = 1,nval_AC
                              do j_3=1,3
                                 !                   (x,y,z)-components of the ac
oustic field
                                 vec(1,3) = soln_AC(1,ls_inod(j_3),ival3s,iel)
                                 vec(2,3) = soln_AC(2,ls_inod(j_3),ival3s,iel)
                                 vec(3,3) = soln_AC(3,ls_inod(j_3),ival3s,iel)

                                 !                  ls_n_dot(3): scalar product
of vec(:,3) and normal vector edge_perp
                                 ls_n_dot(3) = vec(1,3) * edge_perp(1) + vec(2,3) *
 edge_perp(2)

                                 tmp1 = (eps_a - eps_b)*SI_EPS_0
                                 tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,2)&
                                 &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                                 &+ (ls_n_cross(3,1))*ls_n_cross(3,2))

                                 n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                                 n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)

                                 tmp2 = (1.0d0/eps_b - 1.0d0/eps_a)*(1.0d0/SI_EPS_0
)

                                 tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                                 r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)

                                 overlap(ival1,ival2,ival3s) = overlap(ival1,ival2,
ival3s) +&
                                 &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                              enddo
                           enddo
                        enddo
                     enddo
                     !

                     ! If want overlap of given EM mode 1 and all EM modes 2 and al
l AC modes.
                  else if (ival1 .ge. 0 .and. ival2 .eq. -1 .and. ival3 .eq. -1) th
en
                     !             Nodes of the edge
                     do j_1=1,3
                        !                (x,y,z)-components of the electric field
```

```
                     vec(1,1) = conjg(soln_EM_S(1,ls_inod(j_1),ival1,iel))
                     vec(2,1) = conjg(soln_EM_S(2,ls_inod(j_1),ival1,iel))
                     vec(3,1) = conjg(soln_EM_S(3,ls_inod(j_1),ival1,iel))

                     !           ls_n_dot(1): Normal component of vec(:,1)
                     ls_n_dot(1) = vec(1,1) * edge_perp(1) + vec(2,1) * edge_per
p(2)
                     ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                     ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
                     ls_n_cross(3,1) = vec(2,1) * edge_perp(1) - vec(1,1) * edge
_perp(2)
                     do ival2s = 1,nval_EM_p
                        do j_2=1,3
                           !                 (x,y,z)-components of the electri
c field
                           vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2s,iel)
                           vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2s,iel)
                           vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2s,iel)

                           !                 ls_n_dot(2): Normal component of
vec(:,2)
                           ls_n_dot(2) = vec(1,2) * edge_perp(1)+ vec(2,2) * edg
e_perp(2)
                           ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                           ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                           ls_n_cross(3,2) = vec(2,2) * edge_perp(1)- vec(1,2) *
 edge_perp(2)

                           do ival3s = 1,nval_AC
                              do j_3=1,3
                                 !                 (x,y,z)-components of t
he acoustic field
                                 vec(1,3) = soln_AC(1,ls_inod(j_3),ival3s,iel)
                                 vec(2,3) = soln_AC(2,ls_inod(j_3),ival3s,iel)
                                 vec(3,3) = soln_AC(3,ls_inod(j_3),ival3s,iel)

                                 !                 ls_n_dot(3): scalar pro
duct of vec(:,3) and normal vector edge_perp
                                 ls_n_dot(3) = vec(1,3) * edge_perp(1)+ vec(2,3)
 * edge_perp(2)
                                 tmp1 = (eps_a - eps_b)*SI_EPS_0
                                 tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,2)&
                                 &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                                 &+ (ls_n_cross(3,1))*ls_n_cross(3,2))
                                 n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                                 n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)

                                 tmp2 = (1.0d0/eps_b - 1.0d0/eps_a)*(1.0d0/SI_EP
S_0)
                                 tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                                 r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)
                                 overlap(ival1,ival2s,ival3s) =overlap(ival1,iva
l2s,ival3s)+&
                                 &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                              enddo
                           enddo
                        enddo
                     enddo
                  enddo
                  !
                  ! If want overlap of given EM mode 2 and all EM modes 1 and al
l AC modes.
```



```
               else if (ival1 .eq. -1 .and. ival2 .ge. 0 .and. ival3 .eq. -1) th
en
                  !              Nodes of the edge
                  do ival1s = 1,nval_EM_S
                     do j_1=1,3
                        !                 (x,y,z)-components of the electric fie
ld
                        vec(1,1) = conjg(soln_EM_S(1,ls_inod(j_1),ival1s,iel))
                        vec(2,1) = conjg(soln_EM_S(2,ls_inod(j_1),ival1s,iel))
                        vec(3,1) = conjg(soln_EM_S(3,ls_inod(j_1),ival1s,iel))

                        !                 ls_n_dot(1): Normal component of vec(:
,1)
                        ls_n_dot(1) = vec(1,1) * edge_perp(1) + vec(2,1) * edge_
perp(2)

                        ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                        ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
                        ls_n_cross(3,1) = vec(2,1) * edge_perp(1)- vec(1,1) * ed
ge_perp(2)
                        do j_2=1,3
                           !                 (x,y,z)-components of the electri
c field
                           vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2,iel)
                           vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2,iel)
                           vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2,iel)

                           !                 ls_n_dot(2): Normal component of
vec(:,2)
                           ls_n_dot(2) = vec(1,2) * edge_perp(1) + vec(2,2) * ed
ge_perp(2)

                           ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                           ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                           ls_n_cross(3,2) = vec(2,2) * edge_perp(1)- vec(1,2) *
 edge_perp(2)

                           do ival3s = 1,nval_AC
                              do j_3=1,3
                                 !                 (x,y,z)-components of t
he acoustic field
                                 vec(1,3) = soln_AC(1,ls_inod(j_3),ival3s,iel)
                                 vec(2,3) = soln_AC(2,ls_inod(j_3),ival3s,iel)
                                 vec(3,3) = soln_AC(3,ls_inod(j_3),ival3s,iel)

                                 !                 ls_n_dot(3): scalar pro
duct of vec(:,3) and normal vector edge_perp
                                 ls_n_dot(3) = vec(1,3) * edge_perp(1)+ vec(2,3)
 * edge_perp(2)

                                 tmp1 = (eps_a - eps_b)*SI_EPS_0
                                 tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,2)&
                                 &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                                 &+ (ls_n_cross(3,1))*ls_n_cross(3,2))
                                 n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                                 n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)
                                 tmp2 = (1.0d0/eps_b - 1.0d0/eps_a)*(1.0d0/SI_EP
S_0)

                                 tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                                 r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)
                                 overlap(ival1s,ival2,ival3s) =&
                                 &overlap(ival1s,ival2,ival3s)+&
                                 &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                              enddo
                           enddo
```

```
                        enddo
                     enddo
                  enddo
                  !
                  ! If want overlap of all EM mode 1, all EM modes 2 and all AC
modes.
               else if (ival1 .eq. -1 .and. ival2 .eq. -1 .and. ival3 .eq. -1) t
hen
                  !           Nodes of the edge
                  do ival1s = 1,nval_EM_S
                     do j_1=1,3
                        !                 (x,y,z)-components of the electric fie
ld
                        vec(1,1) = conjg(soln_EM_S(1,ls_inod(j_1),ival1s,iel))
                        vec(2,1) = conjg(soln_EM_S(2,ls_inod(j_1),ival1s,iel))
                        vec(3,1) = conjg(soln_EM_S(3,ls_inod(j_1),ival1s,iel))
                        !                 ls_n_dot(1): Normal component of vec(:
,1)
                        ls_n_dot(1) = vec(1,1) * edge_perp(1)+ vec(2,1) * edge_p
erp(2)
                        ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                        ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
                        ls_n_cross(3,1) = vec(2,1) * edge_perp(1)- vec(1,1) * ed
ge_perp(2)
                        do ival2s = 1,nval_EM_p
                           do j_2=1,3
                              !                    (x,y,z)-components of the el
ectric field
                              vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2s,iel)
                              vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2s,iel)
                              vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2s,iel)

                              !                    ls_n_dot(2): Normal componen
t of vec(:,2)
                              ls_n_dot(2) = vec(1,2) * edge_perp(1)+ vec(2,2) *
edge_perp(2)

                              ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                              ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                              ls_n_cross(3,2) = vec(2,2) * edge_perp(1)- vec(1,2
) * edge_perp(2)
                              do ival3s = 1,nval_AC
                                 do j_3=1,3
                                    !                          (x,y,z)-components
 of the acoustic field
                                    vec(1,3) = soln_AC(1,ls_inod(j_3),ival3s,iel
)
                                    vec(2,3) = soln_AC(2,ls_inod(j_3),ival3s,iel
)
                                    vec(3,3) = soln_AC(3,ls_inod(j_3),ival3s,iel
)

                                    !                          ls_n_dot(3): scala
r product of vec(:,3) and normal vector edge_perp
                                    ls_n_dot(3) = vec(1,3) * edge_perp(1)+ vec(2
,3) * edge_perp(2)

                                    tmp1 = (eps_a - eps_b)*SI_EPS_0

                                    tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,
2)&

                                    &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                                    &+ (ls_n_cross(3,1))*ls_n_cross(3,2))
```



```
                                    n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                                    n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)
                                    tmp2 = (1.0d0/eps_b-1.0d0/eps_a)*(1.0d0/SI_E
PS_0)

                                    tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                                    r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)
                                    overlap(ival1s,ival2s,ival3s) =overlap(ival1
s,ival2s,ival3s)+&

                                    &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                                 enddo
                              enddo
                           enddo
                        enddo
                     enddo
                  enddo
                  !
                  ! If want overlap of all EM mode 1, all EM modes 2 and one AC
mode.
               else if (ival1 .eq. -1 .and. ival2 .eq. -1 .and. ival3 .ge. 0) th
en
                  !           Nodes of the edge
                  do ival1s = 1,nval_EM_S
                     do j_1=1,3
                        !                 (x,y,z)-components of the electric fie
ld
                        vec(1,1) = conjg(soln_EM_S(1,ls_inod(j_1),ival1s,iel))
                        vec(2,1) = conjg(soln_EM_S(2,ls_inod(j_1),ival1s,iel))
                        vec(3,1) = conjg(soln_EM_S(3,ls_inod(j_1),ival1s,iel))

                        !                 ls_n_dot(1): Normal component of vec(:
,1)
                        ls_n_dot(1) = vec(1,1) * edge_perp(1)+ vec(2,1) * edge_p
erp(2)
                        ls_n_cross(1,1) = vec(3,1) * edge_perp(2)
                        ls_n_cross(2,1) = -1*vec(3,1) * edge_perp(1)
                        ls_n_cross(3,1) = vec(2,1) * edge_perp(1)- vec(1,1) * ed
ge_perp(2)
                        do ival2s = 1,nval_EM_p
                           do j_2=1,3
                              !                    (x,y,z)-components of the el
ectric field
                              vec(1,2)=soln_EM_p(1,ls_inod(j_2),ival2s,iel)
                              vec(2,2)=soln_EM_p(2,ls_inod(j_2),ival2s,iel)
                              vec(3,2)=soln_EM_p(3,ls_inod(j_2),ival2s,iel)

                              !                    ls_n_dot(2): Normal componen
t of vec(:,2)
                              ls_n_dot(2) = vec(1,2) * edge_perp(1)+ vec(2,2) *
edge_perp(2)
                              ls_n_cross(1,2) = vec(3,2) * edge_perp(2)
                              ls_n_cross(2,2) = -1*vec(3,2) * edge_perp(1)
                              ls_n_cross(3,2) = vec(2,2) * edge_perp(1)- vec(1,2
) * edge_perp(2)
                              do j_3=1,3
                                 !                       (x,y,z)-components of t
he acoustic field
                                 vec(1,3) = soln_AC(1,ls_inod(j_3),ival3,iel)
                                 vec(2,3) = soln_AC(2,ls_inod(j_3),ival3,iel)
                                 vec(3,3) = soln_AC(3,ls_inod(j_3),ival3,iel)

                                 !                       ls_n_dot(3): scalar pro
duct of vec(:,3) and normal vector edge_perp
```

```
                                    ls_n_dot(3) = vec(1,3) * edge_perp(1)+ vec(2,3)
 * edge_perp(2)
                                    tmp1 = (eps_a - eps_b)*SI_EPS_0
                                    tmp1 = tmp1*((ls_n_cross(1,1))*ls_n_cross(1,2)&
                                    &+ (ls_n_cross(2,1))*ls_n_cross(2,2)&
                                    &+ (ls_n_cross(3,1))*ls_n_cross(3,2))
                                    n_dot_d(1) = SI_EPS_0*eps_a * ls_n_dot(1)
                                    n_dot_d(2) = SI_EPS_0*eps_a * ls_n_dot(2)
                                    tmp2 = (1.0d0/eps_b-1.0d0/eps_a)*(1.0d0/SI_EPS_
0)
                                    tmp2 = tmp2*(n_dot_d(1))*n_dot_d(2)
                                    r_tmp = p2_p2_p2_1d(j_1, j_2, j_3)
                                    overlap(ival1s,ival2s,ival3) = overlap(ival1s,i
val2s,ival3) +&
                                    &r_tmp*conjg(ls_n_dot(3))*(tmp1 + tmp2)
                                 enddo
                              enddo
                           enddo
                        enddo
                     enddo
                  endif
               endif
            enddo
         endif
      enddo


   !
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !
   !       open (unit=26,file="Output/edge_data.txt")
   !       write(26,*)
   !       write(26,*) "typ_select_in = ", typ_select_in
   !       write(26,*) "npt, nel = ", npt, nel
   !       write(26,*) "nb_edges = ", nb_edges
   !       write(26,*) "nb_interface_edges = ", nb_interface_edges
   !       write(26,*) "nb_interface_edges: z_integral = ", z_integral
   !       j = 0
   !       do inod=1,npt
   !         if (ls_edge_endpoint(1,inod) .ne. 0) then
   !           j = j + 1
   !           write(26,*) j, inod, ls_edge_endpoint(1,inod),
   !      *              ls_edge_endpoint(2,inod),
   !      *              edge_direction(inod)
   !         endif
   !       enddo
   !       close(26)
   !
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   !
   !       debug = 1
   !       if (debug .eq. 1) then
   !         version_number = 2.2
   ! !  An integer(8) equal to 0 in the ASCII file format
   !         file_type = 0
   ! !  An integer(8) equal to the size of the floating point numbers used in th
e file
   !         data_size = 8
   !         open (unit=27,file="../Output/edge_data.msh")
   !         write(27,'(a11)') "$MeshFormat"
   !         write(27,'((f4.1,1x,I1,1x,I1,1x))') version_number,
   !      *            file_type, data_size
```



```
   !         write(27,'(a14)') "$EndMeshFormat"
   !         write(27,'(a6)') "$Nodes"
   !         write(27,'(I0.1)') nb_interface_edges
   !         zz = 0.0d0
   !         j = 0
   !         do inod=1,npt
   !           if (ls_edge_endpoint(1,inod) .ne. 0) then
   !               xy_1(1) = 100*x(1,inod)
   !               xy_1(2) = 100*x(2,inod)
   !             j = j + 1
   !             write(27,*) j, xy_1(1), xy_1(2), zz
   !           endif
   !         enddo
   !         write(27,'(a9)') "$EndNodes"
   !         write(27,'(a9)') "$Elements"
   !         write(27,'(I0.1)') nb_interface_edges
   ! !  1-node point
   !         element_type = 15
   !         number_of_tags = 2
   !         j = 0
   !         do inod=1,npt
   !           if (ls_edge_endpoint(1,inod) .ne. 0) then
   !             j = j + 1
   !             physical_tag = j
   !             elementary_tag = j
   !             write(27,'(100(I0.1,2x))') j, element_type,
   !      *        number_of_tags, physical_tag, elementary_tag,
   !      *        j
   !           endif
   !         enddo
   !         write(27,'(a12)') "$EndElements"
   !         number_of_string_tags = 1
   !         number_of_real_tags = 1
   !         number_of_integer_tags = 3
   !         write(27,'(a9)') "$NodeData"
   !         write(27,*) number_of_string_tags
   !         write(27,*) " ""View of tangential vector"" "
   !         write(27,*) number_of_real_tags
   !         write(27,*) 0.0
   !         write(27,*) number_of_integer_tags
   ! !  the time step (0; time steps always start at 0)
   !         write(27,*) 0
   ! !  3-component (vector) field
   !         write(27,*) 3
   ! !  Number of associated nodal values
   !         write(27,*) nb_interface_edges
   ! c        node-number value
   !         zz = 0.0d0
   !         j = 0
   !         do inod=1,npt
   !           if (ls_edge_endpoint(1,inod) .ne. 0) then
   !             inod_1 = ls_edge_endpoint(1,inod)
   !             inod_2 = ls_edge_endpoint(2,inod)
   !             xy_1(1) = x(1,inod_1)
   !             xy_1(2) = x(2,inod_1)
   !             xy_2(1) = x(1,inod_2)
   !             xy_2(2) = x(2,inod_2)
   !             edge_vec(1) = xy_2(1) - xy_1(1)
   !             edge_vec(2) = xy_2(2) - xy_1(2)
   ! c            Normalisation of edge_vec
   !             r_tmp = sqrt(edge_vec(1)**2+edge_vec(2)**2)
   !             edge_vec(1) = -1*edge_direction(inod)*edge_vec(1) / r_tmp
```

```
    !             edge_vec(2) = -1*edge_direction(inod)*edge_vec(2) / r_tmp
    !             j = j + 1
    !             write(27,*) j, edge_vec(1), edge_vec(2), zz
    !           endif
    !         enddo
    !         write(27,'(a12)') "$EndNodeData"
    ! c
    !  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    ! c
    !         write(27,'(a9)') "$NodeData"
    !         write(27,*) number_of_string_tags
    !         write(27,*) " ""View of the normal vector"" "
    !         write(27,*) number_of_real_tags
    !         write(27,*) 0.0
    !         write(27,*) number_of_integer_tags
    ! !  the time step (0; time steps always start at 0)
    !         write(27,*) 0
    ! !  3-component (vector) field
    !         write(27,*) 3
    ! !  Number of associated nodal values
    !         write(27,*) nb_interface_edges
    ! c        node-number value
    !         zz = 0.0d0
    !         j = 0
    !         do inod=1,npt
    !           if (ls_edge_endpoint(1,inod) .ne. 0) then
    !             inod_1 = ls_edge_endpoint(1,inod)
    !             inod_2 = ls_edge_endpoint(2,inod)
    !             xy_1(1) = x(1,inod_1)
    !             xy_1(2) = x(2,inod_1)
    !             xy_2(1) = x(1,inod_2)
    !             xy_2(2) = x(2,inod_2)
    !             edge_vec(1) = xy_2(1) - xy_1(1)
    !             edge_vec(2) = xy_2(2) - xy_1(2)
    ! c            Normalisation of edge_vec
    !             r_tmp = sqrt(edge_vec(1)**2+edge_vec(2)**2)
    !             edge_vec(1) = -1*edge_direction(inod)*edge_vec(1) / r_tmp
    !             edge_vec(2) = -1*edge_direction(inod)*edge_vec(2) / r_tmp
    ! c            edge_vec: vector perpendicular to the edge (rotation of edge_v
ec by -pi/2)
    !             edge_perp(1) = -edge_vec(2)
    !             edge_perp(2) = edge_vec(1)
    ! C             edge_perp(1) = edge_perp(1) * edge_direction(inod)
    ! C             edge_perp(2) = edge_perp(2) * edge_direction(inod)
    !             j = j + 1
    !             write(27,*) j, edge_perp(1), edge_perp(2), zz
    !           endif
    !         enddo
    !         write(27,'(a12)') "$EndNodeData"
    !         close(27)
    !       endif
    !
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !
end subroutine moving_boundary
```