```fortran
#include "numbat_decl.h"

! Calculate the EM mode power using numerical quadrature for the basis functions
.
!
! Sturmberg: Eq. (6)
!
!   P_z = 2 Re[\zhat \dot \int dx dy E^* \cross H] =  2 Re[ zhat \dot  \int dx d
y E_t^* \cross H_t]
!

subroutine em_mode_power_sz_quadrature (k_0, n_modes, n_msh_el, n_msh_pts, &
   elnd_to_mesh, v_nd_xy, v_beta, soln_em_e, m_power, errco, emsg)


!     k_0 = 2 pi / lambda, where lambda in meters.

   use numbatmod
   use class_TriangleIntegrators

   double precision k_0

   integer(8) n_modes, n_msh_el, n_msh_pts
   integer(8) elnd_to_mesh(P2_NODES_PER_EL, n_msh_el)
   double precision v_nd_xy(2, n_msh_pts)
   complex(8) soln_em_e(3, P2_NODES_PER_EL+7, n_modes, n_msh_el)
   complex(8) beta1
   complex(8) v_beta(n_modes)
   complex(8), dimension(n_modes) :: m_power

   integer(8), intent(out) :: errco
   character(len=EMSG_LENGTH), intent(out) ::  emsg

!     Local variables
   integer(8) nod_el_p(P2_NODES_PER_EL)
   complex(8) sol_el_1(2*P2_NODES_PER_EL+10), sol_el_2(2*P2_NODES_PER_EL)
   complex(8) vec_1(2*P2_NODES_PER_EL)

   complex(8) bas_ovrlp(2*P2_NODES_PER_EL,2*P2_NODES_PER_EL+10)

   integer(8) i, j, iq
   integer(8) i_el, ival
   integer(8) nd_j, ind_j, xy_j
   integer(8) nd_i, ind_i, xy_i
   integer(8)  n_curved, debug, ui
   logical is_curved
   double precision nds_xy(2,P2_NODES_PER_EL)
   double precision vec_phi_j(2), vec_phi_i(2)
   complex(8) z_tmp1

   type(QuadIntegrator) quadint
   logical do_P3

   double precision t_quadwt

!f2py intent(in) k_0, n_modes, n_msh_el, n_msh_pts
!f2py intent(in) P2_NODES_PER_EL, elnd_to_mesh
!f2py intent(in) x, v_beta, soln_em_e
!
!f2py depend(elnd_to_mesh) P2_NODES_PER_EL, n_msh_el
!f2py depend(x) n_msh_pts
!f2py depend(v_beta) n_modes
```



```fortran
!f2py depend(soln_em_e) P2_NODES_PER_EL, n_modes, n_msh_el
!
!f2py intent(out) m_power
!



   ui = stdout

   debug = 0

   errco = 0
   emsg = ""

   call quadint%setup_reference_quadratures()


   m_power = D_ZERO

   n_curved = 0
   do_P3 = .true.
   do i_el=1,n_msh_el

      do j=1,P2_NODES_PER_EL
         nds_xy(:, j) = v_nd_xy(:,  elnd_to_mesh(j,i_el))
      enddo

      is_curved = log_is_curved_elem_tri (P2_NODES_PER_EL, nds_xy)
      if (is_curved) then
         n_curved = n_curved + 1
      endif

      bas_ovrlp = D_ZERO

      do iq=1,quadint%n_quad

         call quadint%build_transforms_at(iq, nds_xy, is_curved, do_P3, errco, e
msg)
         RETONERROR(errco)

         ! transformed weighting of this quadrature point including triangle are
a transform
         t_quadwt = quadint%wt_quad(iq) * abs(quadint%det)


         do nd_i=1,P2_NODES_PER_EL
            do xy_i=1,2
               ind_i = xy_i + 2*(nd_i-1)

               ! Determine the basis vector
               vec_phi_i = D_ZERO
               vec_phi_i(xy_i) = quadint%phi_P2_ref(nd_i)

               do nd_j=1,P2_NODES_PER_EL
                  do xy_j=1,2
                     ind_j = xy_j + 2*(nd_j-1)

                     ! Determine the basis vector
                     vec_phi_j = D_ZERO
                     vec_phi_j(xy_j) = quadint%phi_P2_ref(nd_j)

                     z_tmp1 = vec_phi_i(1)*vec_phi_j(1) + vec_phi_i(2)*vec_phi_j
```

```
(2)
                     z_tmp1 = t_quadwt * z_tmp1 / (k_0 * SI_C_SPEED * SI_MU_0)
                     bas_ovrlp(ind_i,ind_j) = bas_ovrlp(ind_i,ind_j) + z_tmp1
                  enddo
               enddo

               do nd_j=1,10
                  xy_j = 3
                  ind_j = nd_j + 2*P2_NODES_PER_EL

                  ! Determine the basis vector
                  vec_phi_j = -quadint%gradt_P3_act(:, nd_j)

                  z_tmp1 = vec_phi_i(1)*vec_phi_j(1) + vec_phi_i(2)*vec_phi_j(2)
                  z_tmp1 = t_quadwt * z_tmp1 / (k_0 * SI_C_SPEED * SI_MU_0)
                  bas_ovrlp(ind_i,ind_j) = bas_ovrlp(ind_i,ind_j) + z_tmp1
               enddo

            enddo
         enddo
      enddo

      do ival=1,n_modes
         beta1 = v_beta(ival)
         do i=1,P2_NODES_PER_EL
            do j=1,2
               ! The 2 transverse components of the mode ival
               ind_i = j + 2*(i-1)
               ! sol_el_2 : E-field
               sol_el_2(ind_i) = soln_em_e(j,i,ival,i_el)
            enddo
         enddo

         do i=1,P2_NODES_PER_EL
            do j=1,2
               !  The 2 transverse components of the mode jval
               ind_j = j + 2*(i-1)
               ! sol_el_1 : H-field
               sol_el_1(ind_j) = soln_em_e(j,i,ival,i_el) * beta1
            enddo
         enddo

         do i=1,3
            ! The longitudinal component at the vertices (P3 elements)
            ind_j = i + 2*P2_NODES_PER_EL
            !   sol_el_1 : H-field
            sol_el_1(ind_j) = - soln_em_e(3,i,ival,i_el) * C_IM_ONE
            !  sol_el_1(ind_j) = z_tmp1 * beta1
         enddo

         do i=P2_NODES_PER_EL+1,13

            !  The longitudinal component at the edge nodes and interior node (P
3 elements)
            ind_j = i + 2*P2_NODES_PER_EL - P2_NODES_PER_EL + 3
            sol_el_1(ind_j) = - soln_em_e(3,i,ival,i_el) * C_IM_ONE
            !  sol_el_1(ind_j) = z_tmp1 * beta1
         enddo

         !   Matrix-Vector product
         do i=1,2*P2_NODES_PER_EL
            vec_1(i) = 0.0d0
```



```
               do j=1,2*P2_NODES_PER_EL+10
                  vec_1(i) = vec_1(i) + sol_el_1(j) * bas_ovrlp(i,j)
               enddo
            enddo

            ! Scalar product
            z_tmp1 = 0.0d0
            do i=1,2*P2_NODES_PER_EL
               m_power(ival) = m_power(ival) + vec_1(i) * sol_el_2(i)
            enddo
         enddo
      enddo


end subroutine em_mode_power_sz_quadrature
```