```
      subroutine asmbly  (i_cond, i_base, nel, npt, n_ddl, neq, nnodes,
     *  shift, bloch_vec, nb_typ_el, pp, qq, table_nod,
     *  table_N_E_F, type_el, ineq, ip_period_N,
     *  ip_period_E_F, x, x_N_E_F, nonz, row_ind, col_ptr,
     *  mat1_re, mat1_im, mat2, i_work)

c     NQUAD: The number of quadrature points used in each element.

      implicit none
      integer*8 nel, npt, n_ddl, neq, nnodes
      integer*8 i_cond, i_base, i_base2, nb_typ_el, nonz
      integer*8 ip_period_N(npt), ip_period_E_F(n_ddl)
      integer*8 row_ind(nonz), col_ptr(neq+1)
      integer*8 type_el(nel)
      integer*8 table_nod(nnodes,nel), ineq(3,n_ddl)
      integer*8 table_N_E_F(14,nel)
      integer*8 i_work(3*n_ddl)
      double precision x(2,npt), x_N_E_F(2,n_ddl)
      complex*16 pp(nb_typ_el), qq(nb_typ_el), shift
      complex*16 mat2(nonz)
      double precision mat1_re(nonz), mat1_im(nonz)

      integer*8 nquad, nquad_max
      parameter (nquad_max = 25)
      double precision wq(nquad_max)
      double precision xq(nquad_max), yq(nquad_max)
      double precision xx(2), xx_g(2), ww, det
      double precision mat_B(2,2), mat_T(2,2)
      double precision grad_i(2), grad_j(2)
      double precision phi_z_i, phi_z_j

      integer*8 nnodes_0, nddl_0, nddl_t, ui
      parameter (nnodes_0 = 6)
      parameter (nddl_0 = 14)
      parameter (nddl_t=4)

      integer*8 nod_el_p(nnodes_0), basis_list(4,3,nddl_t)
      double precision xel(2,nnodes_0)

      double precision phi1_list(3), grad1_mat0(2,3), grad1_mat(2,3)

      double precision phi2_list(6), grad2_mat0(2,6)
      double precision grad2_mat(2,6)

      double precision phi3_list(10), grad3_mat0(2,10)
      double precision grad3_mat(2,10)

      double precision vec_phi_j(2), curl_phi_j
      double precision vec_phi_i(2), curl_phi_i
      complex*16 val_exp(nddl_0), z_phase_fact

      integer*8 i, j, k, j1, iel, iq, typ_e
      integer*8 jtest, jp, ind_jp, j_eq
      integer*8 itrial, ip, ind_ip, i_eq
      integer*8 info_curved, n_curved, debug, col_start, col_end
      complex*16 z_tmp1, z_tmp2
      double precision ZERO, ONE
      parameter (ZERO = 0.0D0)
      parameter (ONE = 1.0D0)
      double precision bloch_vec(2), r_tmp1, r_tmp2
```



```
      double precision delta_xx(2)
      double precision ddot
      complex*16 M_tt, M_zz, M_tz, M_zt
      complex*16 K_tt, K_zz, K_tz, K_zt
      complex*16 ii
c
ccccccccccccccccccccccccccccccccccccccc
c
      ui = 6
      debug = 0
c
c     The CSC indexing, i.e., col_ptr, is 1-based
c      But valpr.f may have changed the CSC indexing to 0-based indexing)
      if (i_base .eq. 0) then
        i_base2 = 1
      else
        i_base2 = 0
      endif
c
      if ( nnodes .ne. 6 ) then
        write(ui,*) "asmbly: problem nnodes = ", nnodes
        write(ui,*) "asmbly: nnodes should be equal to 14 !"
        write(ui,*) "asmbly: Aborting..."
        stop
      endif
c
      call quad_triangle (nquad, nquad_max, wq, xq, yq)
c
      if (debug .eq. 1) then
        write(ui,*) "asmbly: bloch_vec = ", bloch_vec
        write(ui,*) "asmbly: nquad, nquad_max = ",
     *              nquad, nquad_max
        write(ui,*) "asmbly: i_cond = ", i_cond
      endif
c
ccccccccccccccccccccccccccccccccccccccc
c
c  ii = sqrt(-1)
      ii = dcmplx(0.0d0, 1.0d0)

      do i=1,nonz
        mat1_re(i) = 0.d0
        mat1_im(i) = 0.d0
        mat2(i) = 0.d0
      enddo
c
      n_curved = 0
      do iel=1,nel
        typ_e = type_el(iel)
        do j=1,nnodes
          j1 = table_nod(j,iel)
          nod_el_p(j) = j1
          xel(1,j) = x(1,j1)
          xel(2,j) = x(2,j1)
          val_exp(j) = 1.0d0
        enddo
        call curved_elem_tri (nnodes, xel, info_curved, r_tmp1)
        if (info_curved .eq. 1) then
          n_curved = n_curved + 1
        endif

        if (i_cond .eq. 2) then
```

```
c         Periodic boundary condition
          do j=1,nnodes
            j1 = ip_period_N(nod_el_p(j))
            if (j1 .ne. 0) nod_el_p(j) = j1
          enddo
        endif
        call basis_ls(nod_el_p, basis_list)
        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 destinat
ion point
c         For a pair of periodic points, one is chosen as origin and the other i
s the destination
          do j=1,nddl_0
            ip = table_N_E_F(j,iel)
            j1 = ip_period_E_F(ip)
            if (j1 .ne. 0) then
              do k=1,2
                delta_xx(k) = x_N_E_F(k,ip) - x_N_E_F(k,j1)
              enddo
              r_tmp1 = ddot(2, bloch_vec, 1, delta_xx, 1)
              val_exp(j) = exp(ii*r_tmp1)
            endif
          enddo
        endif
c
        do iq=1,nquad
          xx(1) = xq(iq)
          xx(2) = yq(iq)
          ww = wq(iq)
c         xx   = coordinate on the reference triangle
c         xx_g = coordinate on the actual triangle
c
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 triangle (P3 element)
           call phi3_2d_mat(xx, phi3_list, grad3_mat0)
c
          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 .and. iq .eq. 1) then
              write(ui,*) " !!!"
              write(ui,*) "asmbly: det <= 0: iel, det ", iel, det
              write(ui,*) "x : ", (nod_el_p(j),j=1,nnodes)
              write(ui,*) "x : ", (xel(1,j),j=1,3)
              write(ui,*) "y : ", (xel(2,j),j=1,3)
              write(ui,*)
            endif
          elseif (info_curved .eq. 1) then
c           Isoparametric element
            call jacobian_p2_2d(xx, xel, nnodes, phi2_list,
     *                 grad2_mat0, xx_g, det, mat_B, mat_T)
          else
            write(ui,*) "asmbly: info_curved has an invalid value : ",
     *                 info_curved
            write(ui,*) "asmbly: Aborting..."
```



```
            stop
          endif
c           write(ui,*) "asmbly: info_curved = ", info_curved
C           if(abs(det) .lt. 1.0d-10) then
           if(abs(det) .lt. 1.0d-20) then
            write(ui,*)
            write(ui,*) " ???"
            write(ui,*) "asmbly: det = 0 : iel, det = ", iel, det
            write(ui,*) "asmbly: Aborting..."
            stop
           endif
c
c          grad_i  = gradient on the actual triangle
c          grad_i  = Transpose(mat_T)*grad_i0
c          Calculation of the matrix-matrix product:
c
          call DGEMM('Transpose','N', 2, 3, 2, ONE, mat_T, 2,
     *      grad1_mat0, 2, ZERO, grad1_mat, 2)

          call DGEMM('Transpose','N', 2, 6, 2, ONE, mat_T, 2,
     *      grad2_mat0, 2, ZERO, grad2_mat, 2)

          call DGEMM('Transpose','N', 2, 10, 2, ONE, mat_T, 2,
     *      grad3_mat0, 2, ZERO, grad3_mat, 2)

          do jtest=1,nddl_0
            jp = table_N_E_F(jtest,iel)
            do j_eq=1,3
c             jp = table_N_E_F(jtest,iel)
              ind_jp = ineq(j_eq,jp)
              if (ind_jp .gt. 0) then
                col_start = col_ptr(ind_jp) + i_base2
                col_end = col_ptr(ind_jp+1) - 1 + i_base2
c               unpack row into i_work
                do i=col_start,col_end
                  i_work(row_ind(i) + i_base2) = i
                enddo
C                 ! edge or face element
                if (jtest .le. nddl_t) then
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)
                  grad_j(1) = 0.0d0
                  grad_j(2) = 0.0d0
                  phi_z_j = 0.0d0
                else
                  vec_phi_j(1) = 0.0d0
                  vec_phi_j(2) = 0.0d0
                  curl_phi_j = 0.0d0
                  grad_j(1) = grad3_mat(1,jtest-nddl_t)
                  grad_j(2) = grad3_mat(2,jtest-nddl_t)
                  phi_z_j = phi3_list(jtest-nddl_t)
                endif
                do itrial=1,nddl_0
                  z_phase_fact = val_exp(jtest) * conjg(val_exp(itrial))
                  do i_eq=1,3
                    ip = table_N_E_F(itrial,iel)
                    ind_ip = ineq(i_eq,ip)
                    if (ind_ip .gt. 0) then
                      if (ind_jp .eq. ind_ip .and.
     *                  abs(imag(z_phase_fact)) .gt. 1.0d-15) then
                        write(ui,*) "phase_fact: ", ind_jp, ind_ip,
```

```
     *                    z_phase_fact, val_exp(jtest), val_exp(itrial)
                          endif
C                           ! edge or face element
                          if (itrial .le. nddl_t) then
                            call basis_vec (i_eq, itrial, basis_list,
     *                     phi2_list, grad1_mat, grad2_mat, vec_phi_i,
     *                     curl_phi_i)
                              grad_i(1) = 0.0d0
                              grad_i(2) = 0.0d0
                              phi_z_i = 0.0d0
                            else
                              vec_phi_i(1) = 0.0d0
                              vec_phi_i(2) = 0.0d0
                              curl_phi_i = 0.0d0
                              grad_i(1) = grad3_mat(1,itrial-nddl_t)
                              grad_i(2) = grad3_mat(2,itrial-nddl_t)
                              phi_z_i = phi3_list(itrial-nddl_t)
                          endif
ccccccccccccccccccccc
c                         Reference; see Eq. (40) of the FEM paper:
c                         K. Dossou and M. Fontaine
c                         "A high order isoparametric finite element method for the
computation of waveguide modes"
c                         Computer Methods in Applied Mechanics and Engineering, vol
. 194, no. 6-8, pp. 837-858, 2005.
ccccccccccccccccccccc
                          if (itrial .le. nddl_t .and.
     *                        jtest .le. nddl_t) then
                            r_tmp1 = curl_phi_j * curl_phi_i
                            r_tmp2 = ddot(2, vec_phi_j, 1, vec_phi_i, 1)
                            K_tt = r_tmp1 * pp(typ_e) - r_tmp2 * qq(typ_e)
                            M_tt = - r_tmp2 * pp(typ_e)
                            z_tmp1 = K_tt * ww * abs(det) * z_phase_fact
                            z_tmp2 = M_tt * ww * abs(det) * z_phase_fact
                          elseif (itrial .le. nddl_t .and.
     *                        jtest .gt. nddl_t) then
                            r_tmp1 = ddot(2, grad_j, 1, vec_phi_i, 1)
                            K_tz = 0.0d0
                            M_tz = r_tmp1 * pp(typ_e)
                            z_tmp1 = K_tz * ww * abs(det) * z_phase_fact
                            z_tmp2 = M_tz * ww * abs(det) * z_phase_fact
                          elseif (itrial .gt. nddl_t .and.
     *                        jtest .le. nddl_t) then
                            r_tmp1 = ddot(2, vec_phi_j, 1, grad_i, 1)
                            K_zt = r_tmp1 * pp(typ_e)
                            M_zt = 0.0d0
                            z_tmp1 = K_zt * ww * abs(det) * z_phase_fact
                            z_tmp2 = M_zt * ww * abs(det) * z_phase_fact
                          elseif (itrial .gt. nddl_t .and.
     *                        jtest .gt. nddl_t) then
                            r_tmp1 = ddot(2, grad_j, 1, grad_i, 1)
                            r_tmp2 = phi_z_j * phi_z_i
                            K_zz = - r_tmp1 * pp(typ_e) + r_tmp2 * qq(typ_e)
                            M_zz = 0.0d0
                            z_tmp1 = K_zz * ww * abs(det) * z_phase_fact
                            z_tmp2 = M_zz * ww * abs(det) * z_phase_fact
                          else
                            write(ui,*) "itrial or jtest has an ",
     *                        "invalid value"
                            write(ui,*) "itrial jtest, = ", itrial, jtest
                            write(ui,*) "asmbly: Aborting..."
                            stop
```



```
                          endif
                          z_tmp1 = z_tmp1 - shift*z_tmp2

                          k = i_work(ind_ip)
                          if (k .gt. 0 .and. k .le. nonz) then
                            mat1_re(k) = mat1_re(k) + dble(z_tmp1)
                            mat1_im(k) = mat1_im(k) + imag(z_tmp1)
                            mat2(k) = mat2(k) + z_tmp2
                          else
                            write(ui,*) "asmbly: problem with row_ind !!"
                            write(ui,*) "asmbly: k, nonz = ", k, nonz
                            write(ui,*) "asmbly: Aborting..."
                            stop
                          endif
                        endif
                      enddo
                    enddo
                  endif
                enddo
              enddo
            enddo
          enddo
c
          if (debug .eq. 1) then
            write(ui,*) "asmbly: shift = ", shift
            write(ui,*) "asmbly: number of curved elements = ", n_curved
            write(ui,*) "asmbly: nel, (nel-n_curved) = ", nel,
     *                  (nel-n_curved)
          endif
c
          if (debug .eq. 1) then
            write(ui,*)
            write(ui,*) " Re pp = ", dble(pp)
            write(ui,*) "imag pp = ", imag(pp)
            write(ui,*)
            write(ui,*) " Re qq = ", dble(qq)
            write(ui,*) "imag qq = ", imag(qq)
          endif
c
          return
          end
```