```
#include "numbat_decl.h"

C--------------------------------------------------------------------------------
---------

c     subroutine prepare_workspaces( n_msh_pts, n_msh_el, n_modes,
c    *     int_max, cmplx_max, real_max, awk, bwk, cwk, overlap_L,
c    *     iindex, errco, emsg)
c
c     integer*8 int_max, cmplx_max, real_max
c     integer*8 n_msh_el, n_msh_pts, n_modes
c     integer :: stat=0
c
c     integer*8, dimension(:), allocatable :: awk
c     complex*16, dimension(:), allocatable :: bwk
c     double precision, dimension(:), allocatable :: cwk
c     integer*8, dimension(:), allocatable :: iindex
c     complex*16, dimension(:,:), allocatable :: overlap_L
c     integer*8 errco
c     character*2048 emsg
c
c     call array_size(n_msh_pts, n_msh_el, n_modes,
c    *     int_max, cmplx_max, real_max, errco, emsg)
c     RETONERROR(errco)
c
c     allocate(bwk(cmplx_max), STAT=stat)
c     call check_alloc(stat, cmplx_max, "b", -1, errco, emsg)
c     RETONERROR(errco)
c
c     allocate(cwk(real_max), STAT=stat)
c     call check_alloc(stat, real_max, "c", -1, errco, emsg)
c     RETONERROR(errco)
c
c     allocate(awk(int_max), STAT=stat)
c     call check_alloc(stat, int_max, "a", -1, errco, emsg)
c     RETONERROR(errco)
c
c     allocate(overlap_L(n_modes,n_modes), STAT=stat)
c     call check_alloc(stat, n_modes*n_modes,
c    *   "overlap_L", -1, errco, emsg)
c     RETONERROR(errco)
c
c     allocate(iindex(n_modes), STAT=stat)
c     call check_alloc(stat, n_modes, "iindex", -1, errco, emsg)
c     RETONERROR(errco)
c
c     end subroutine prepare_workspaces
C--------------------------------------------------------------------------------
---------



c   lambda       - free space wavelength in m
c   n_modes    - desired number of eigenvectors
c   n_msh_pts    - number of FEM mesh points
c   n_msh_el     - number of FEM (triang) elements
c   n_typ_el     - number of types of elements (and therefore elements)
c   v_refindex_n - array of effective index of materials
c   bloch_vec    - in-plane k-vector (normally tiny just to avoid degeneracies)
c   shift_ksqr   - k_est^2 = n^2 k_0^2  : estimate of eigenvalue k^2
c   bnd_cnd_i    - bnd conditions (Dirichlet = 0, Neumann = 1, Periodic = 2)
c   beta1        - array of eigenvalues kz
```



```
c   sol1         - 4-dim array of solutions [field comp, node of element (1..13)
?!, eigvalue, element number] (strange ordering)
c   mode_pol     - unknown - never used in python
c   table_nod    - 2D array [node_on_elt-1..6][n_msh_el] giving the mesh point m
p of each node
c                  Points where type_el[mp] is not the same for all 6 nodes must
 be interface points
c   type_el      - n_msh_el array: material index for each element
c   type_nod     - is boundary node?
c   mesh_xy        - (2 , n_msh_pts)  x,y coords?
c   ls_material  - (1, nodes_per_el+7, n_msh_el)

      subroutine calc_EM_modes(
c     Inputs
     *    n_modes, lambda, dimscale_in_m, bloch_vec, shift_ksqr,
     *    E_H_field, bnd_cdn_i, itermax, debug,
     *    mesh_file, n_msh_pts, n_msh_el, n_typ_el, v_refindex_n,
c     Outputs
     *    beta1, sol1, mode_pol,
     *    table_nod, type_el, type_nod, mesh_xy, ls_material,
     *    errco, emsg)

C**********************************************************************
C
C  Program:
C    FEM solver of Electromagnetic waveguide problems.
C    This subroutine is compiled by f2py & called in mode_calcs.py
C
C  Authors:
C    Bjorn Sturmberg & Kokou B. Dossou
C
C**********************************************************************
C
      implicit none

C  Local parameters:
      integer*8 int_max, cmplx_max, int_used, cmplx_used
      integer*8 real_max, real_used
C      parameter (int_max=2**22, cmplx_max=2**26)
C      parameter (real_max=2**21)
C     !    a(int_max)
      integer*8, dimension(:), allocatable :: awk
C     !  b(cmplx_max)
      complex*16, dimension(:), allocatable :: bwk
C     !  c(real_max)
      double precision, dimension(:), allocatable :: cwk
      integer :: stat=0
      integer*8 errco
      character*2048 emsg
C
C  Declare the pointers of the integer super-vector
      integer*8 ip_table_E, ip_table_N_E_F, ip_visite
      integer*8 ip_type_N_E_F, ip_eq
      integer*8 ip_period_N, ip_nperiod_N
      integer*8 ip_period_N_E_F, ip_nperiod_N_E_F
C      integer*8 ip_col_ptr, ip_bandw
C  Declare the pointers of the real super-vector
      integer*8 jp_x_N_E_F
C      integer*8 jp_matD, jp_matL, jp_matU
C      integer*8 jp_matD2, jp_matL2, jp_matU2
      integer*8 jp_vect1, jp_vect2, jp_workd, jp_resid, jp_vschur
      integer*8 jp_trav, jp_vp
```

```
      integer*8 n_typ_el
      complex*16 pp(n_typ_el), qq(n_typ_el)
      complex*16 eps_eff(n_typ_el), v_refindex_n(n_typ_el)
c     bnd_cdn_i = 0 => Dirichlet boundary condition
c     bnd_cdn_i = 1 => Neumann boundary condition
c     bnd_cdn_i = 2 => Periodic boundary condition
      integer*8 n_msh_el, n_msh_pts, nodes_per_el, ui, bnd_cdn_i
C     ! Number of nodes per element
      parameter(nodes_per_el=6)

      ! type_nod: interior (=0) or boundary (!=0) point?
      ! type_el:  material index of element
      integer*8 type_nod(n_msh_pts), type_el(n_msh_el)


      integer*8 table_nod(nodes_per_el, n_msh_el)

C, len_skyl, nsym
c     E_H_field = 1 => Electric field formulation (E-Field)
c     E_H_field = 2 => Magnetic field formulation (H-Field)
      integer*8 E_H_field
      integer*8 neq, debug
      integer*8 n_msh_pts_p3

C  Variable used by valpr
      integer*8 n_modes, nvect, itermax, ltrav
      integer*8 n_conv, i_base
      double precision ls_data(10)
c      integer*8 pointer_int(20), pointer_cmplx(20)
C      integer*8 iindex(2000), n_core(2)
      integer*8, dimension(:), allocatable :: iindex
      integer*8 n_core(2)
      complex*16 z_beta, z_tmp, z_tmp0
      integer*8 n_edge, n_face, n_ddl, n_ddl_max, n_k
c     variable used by UMFPACK
      double precision control (20), info_umf (90)
      integer*8 numeric
C  Renumbering
c      integer*8 ip_row_ptr, ip_bandw_1, ip_adjncy
c      integer*8 len_adj, len_adj_max, len_0_adj_max
c, iout, nonz_1, nonz_2
      integer*8 i, j
      integer*8 ival, iel, inod
c     Wavelength lambda in units of m
      double precision lambda, dimscale_in_m
      double precision freq, lat_vecs(2,2), tol
      double precision k_0, pi, dim_x, dim_y
      double precision bloch_vec(2), bloch_vec_k(2)
      complex*16 shift_ksqr
C  Timing variables
      double precision time1, time2
      double precision stime1, stime2
      double precision time1_fact, time2_fact
C     double precision time1_asmbl, time2_asmbl
      double precision time1_postp
      double precision stime1_postp
      double precision time1_arpack, time2_arpack
      double precision time1_J, time2_J
      double precision stime1_J, stime2_J
C     character*(8) start_date, end_date
      character*(10) start_time, end_time
C  Names and Controls
```



```
      character mesh_file*1000, gmsh_file*1000, log_file*1000
      character gmsh_file_pos*1000
      character overlap_file*1000, dir_name*1000, msg*20
      character*1000 tchar
      integer*8 namelength, PrintAll
      integer*8 plot_modes
      integer*8 pair_warning, homogeneous_check
      integer*8 q_average, plot_real, plot_imag, plot_abs
      complex*16 ii

c     Declare the pointers of the real super-vector
      integer*8 kp_rhs_re, kp_rhs_im, kp_lhs_re, kp_lhs_im
      integer*8 kp_mat1_re, kp_mat1_im

c     Declare the pointers of for sparse matrix storage
      integer*8 ip_col_ptr, ip_row
      integer*8 jp_mat2
      integer*8 ip_work, ip_work_sort, ip_work_sort2
      integer*8 nonz, nonz_max, max_row_len

      integer*8 ip
      integer i_32

c     new breed of variables to prise out of awk, bwk and cwk
      double precision mesh_xy(2,n_msh_pts)
      complex*16, target :: sol1(3,nodes_per_el+7,n_modes,n_msh_el)
      complex*16, target :: sol2(3,nodes_per_el+7,n_modes,n_msh_el)
      complex*16, pointer :: sol(:,:,:,:)
      complex*16, dimension(:,:), allocatable :: overlap_L

      complex*16, target :: beta1(n_modes), beta2(n_modes)
      complex*16, pointer :: beta(:)
      complex*16 mode_pol(4,n_modes)

      complex*16 ls_material(1,nodes_per_el+7,n_msh_el)

Cf2py intent(in) lambda, n_modes
Cf2py intent(in) debug, mesh_file, n_msh_pts, n_msh_el
Cf2py intent(in) v_refindex_n, bloch_vec, dimscale_in_m, shift_ksqr
Cf2py intent(in) E_H_field, bnd_cdn_i, itermax
Cf2py intent(in) plot_modes, plot_real, plot_imag, plot_abs
Cf2py intent(in) cmplx_max, real_max, int_max, n_typ_el

Cf2py depend(v_refindex_n) n_typ_el

Cf2py intent(out) beta1, type_nod, ls_material
Cf2py intent(out) sol1, mode_pol, table_nod, type_el, mesh_xy

Cf2py intent(out) errco
Cf2py intent(out) emsg


C      n_64 = 2
C     !n_64**28 on Vayu, **27 before
C      cmplx_max=n_64**25
C      real_max = n_64**23
C      int_max  = n_64**22
c      3*n_msh_pts+n_msh_el+nodes_per_el*n_msh_el

C      write(*,*) "cmplx_max = ", cmplx_max
C      write(*,*) "real_max = ", real_max
C      write(*,*) "int_max = ", int_max
```

```
c     ii = sqrt(-1)
      ii = cmplx(0.0d0, 1.0d0, 8)

C     Old inputs now internal to here and commented out by default.
C      mesh_format = 1
C      Checks = 0 ! check completeness, energy conservation
C       ! only need to print when debugging J overlap, orthogonal
      PrintAll = debug
C       ! ARPACK accuracy (0.0 for machine precision)
C       lx=1.0 ! Diameter of unit cell. Default, lx = 1.0.
C       ly=1.0 ! NOTE: currently requires ly=lx, ie rectangular unit cell.

      tol = 0.0

      errco= 0
      emsg = ""

c     Declare work space arrays

c     call prepare_workspaces( n_msh_pts, n_msh_el, n_modes,
c    *     int_max, cmplx_max, real_max, awk, bwk, cwk, overlap_L,
c    *     iindex, errco, emsg)
c     RETONERROR(errco)

      call array_size(n_msh_pts, n_msh_el, n_modes,
     *     int_max, cmplx_max, real_max, errco, emsg)
      RETONERROR(errco)

      allocate(bwk(cmplx_max), STAT=stat)
      call check_alloc(stat, cmplx_max, "b", -1, errco, emsg)
      RETONERROR(errco)

      allocate(cwk(real_max), STAT=stat)
      call check_alloc(stat, real_max, "c", -1, errco, emsg)
      RETONERROR(errco)

      allocate(awk(int_max), STAT=stat)
      call check_alloc(stat, int_max, "a", -1, errco, emsg)
      RETONERROR(errco)

      allocate(overlap_L(n_modes,n_modes), STAT=stat)
      call check_alloc(stat, n_modes*n_modes,
     *   "overlap_L", -1, errco, emsg)
      RETONERROR(errco)

      allocate(iindex(n_modes), STAT=stat)
      call check_alloc(stat, n_modes, "iindex", -1, errco, emsg)
      RETONERROR(errco)


CCCCCCCCCCCCCCCCC POST F2PY CCCCCCCCCCCCCCCCCCCCCCCCC

C     clean mesh_format
C     TODO: Drop these debug files?
      namelength = len_trim(mesh_file)
      gmsh_file = mesh_file(1:namelength-5)//'.msh'
      gmsh_file_pos = mesh_file(1:namelength)
      log_file = mesh_file(1:namelength-5)//'.log'
      if (debug .eq. 1) then
        write(*,*) "mesh_file = ", mesh_file
        write(*,*) "gmsh_file = ", gmsh_file
```



```
      endif

c     Calculate effective permittivity for each type of material
      do i_32 = 1, int(n_typ_el)
        eps_eff(i_32) = v_refindex_n(i_32)**2
      end do

C     !ui = Unite dImpression
      ui = 6
      pi = 3.141592653589793d0
C      nsym = 1 ! nsym = 0 => symmetric or hermitian matrices
C

      nvect = 2*n_modes + n_modes/2 +3

CCCCCCCCCCCCCCCCC END POST F2PY CCCCCCCCCCCCCCCCCCCCC

C     ! initial time  in unit = sec.
      call get_clocks(stime1, time1)
C
C####################  Start FEM PRE-PROCESSING  ########################
C

      dim_x = dimscale_in_m
      dim_y = dimscale_in_m

      ! fills table_nod and mesh_xy
      call geometry (n_msh_el, n_msh_pts, nodes_per_el, n_typ_el,
     *     dim_x, dim_y, type_nod, type_el, table_nod,
     *     mesh_xy, mesh_file, errco, emsg)
      RETONERROR(errco)



C
      call lattice_vec (n_msh_pts, mesh_xy, lat_vecs, debug)
C
C      V = number of vertices
C      E = number of edges
C      F =  number of faces
C      C =  number of cells (3D, tetrahedron)
C
C     From Euler's theorem on 3D graphs: V-E+F-C = 1 - (number of holes)
C     n_msh_pts = (number of vertices) + (number of mid-edge point) = V + E;
C
C     ! each element is a face
      n_face = n_msh_el
      ip_table_N_E_F = 1
      call list_face (n_msh_el, awk(ip_table_N_E_F))

C     n_ddl_max = max(N_Vertices) + max(N_Edge) + max(N_Face)
C     For P2 FEM n_msh_pts=N_Vertices+N_Edge
C     note: each element has 1 face, 3 edges and 10 P3 nodes
      n_ddl_max = n_msh_pts + n_face
      ip_visite =  ip_table_N_E_F  + 14*n_msh_el
      ip_table_E = ip_visite + n_ddl_max
C
      call list_edge (n_msh_el, n_msh_pts, nodes_per_el, n_edge,
     *    type_nod, table_nod,
     *    awk(ip_table_N_E_F), awk(ip_table_E), awk(ip_visite))
      call list_node_P3 (n_msh_el, n_msh_pts, nodes_per_el, n_edge,
     *    n_msh_pts_p3, table_nod, awk(ip_table_N_E_F), awk(ip_visite))
      n_ddl = n_edge + n_face + n_msh_pts_p3
```

```
C
       if (debug .eq. 1) then
         write(ui,*) "py_calc_modes.f: n_msh_pts, n_msh_el = ",
     *      n_msh_pts, n_msh_el
         write(ui,*) "py_calc_modes.f: n_msh_pts_p3 = ", n_msh_pts_p3
         write(ui,*) "py_calc_modes.f: n_vertex, n_edge, n_face,",
     *      " n_msh_el = ",
     *     (n_msh_pts - n_edge), n_edge, n_face, n_msh_el
         write(ui,*) "py_calc_modes.f: 2D case of the Euler &
  & characteristic: V-E+F=1-(number of holes)"
         write(ui,*) "py_calc_modes.f: Euler characteristic: V - E + F &
  &= ", (n_msh_pts - n_edge) - n_edge + n_face
       endif
cC
cC-----------
cC
cC      overwriting pointers ip_row_ptr, ..., ip_adjncy
c
       ip_type_N_E_F = ip_table_E + 4*n_edge
C
       jp_x_N_E_F = 1
       call type_node_edge_face (n_msh_el, n_msh_pts, nodes_per_el,
     *       n_ddl, type_nod, table_nod, awk(ip_table_N_E_F),
     *       awk(ip_visite), awk(ip_type_N_E_F),
     *       mesh_xy, bwk(jp_x_N_E_F))
C
       call get_coord_p3 (n_msh_el, n_msh_pts, nodes_per_el, n_ddl,
     *       table_nod, type_nod, awk(ip_table_N_E_F),
     *       awk(ip_type_N_E_F), mesh_xy, bwk(jp_x_N_E_F),
     *       awk(ip_visite))
C
         ip_period_N = ip_type_N_E_F + 2*n_ddl
         ip_nperiod_N = ip_period_N + n_msh_pts
         ip_period_N_E_F = ip_nperiod_N + n_msh_pts
         ip_nperiod_N_E_F = ip_period_N_E_F + n_ddl
         ip_eq = ip_nperiod_N_E_F + n_ddl
C
C      Dirichlet or Neumann conditions
       if (bnd_cdn_i .eq. 0 .or. bnd_cdn_i .eq. 1) then
         call bound_cond (bnd_cdn_i, n_ddl, neq, awk(ip_type_N_E_F),
     *     awk(ip_eq))

C      Periodic  conditions (never in NumBAT)
       elseif(bnd_cdn_i .eq. 2) then
         if (debug .eq. 1) then
           write(ui,*) "###### periodic_node"
         endif

         call periodic_node(n_msh_el, n_msh_pts, nodes_per_el, type_nod,
     *       mesh_xy, awk(ip_period_N), awk(ip_nperiod_N),
     *       table_nod, lat_vecs)
         if (debug .eq. 1) then
           write(ui,*) "py_calc_modes.f: ###### periodic_N_E_F"
         endif

         call periodic_N_E_F (n_ddl, awk(ip_type_N_E_F),
     *       bwk(jp_x_N_E_F), awk(ip_period_N_E_F),
     *       awk(ip_nperiod_N_E_F), lat_vecs)

         call periodic_cond (bnd_cdn_i, n_ddl, neq, awk(ip_type_N_E_F),
     *        awk(ip_period_N_E_F), awk(ip_eq), debug)
```



```
       else
C TODO: this will never happen because of python checks
         write(ui,*) "py_calc_modes.f: bnd_cdn_i has value : ",
     *      bnd_cdn_i
         write(ui,*) "py_calc_modes.f: Aborting..."
         errco = -10
         return
       endif
C
       if (debug .eq. 1) then
         write(ui,*) "py_calc_modes.f: neq, n_ddl = ", neq, n_ddl
       endif
C
C=====calcul du vecteur de localisation des colonnes
C      pour le traitement skyline de la matrice globale
C      Type of sparse storage of the global matrice:
C                                 Symmetric Sparse Skyline format
C      Determine the pointer for the Symmetric Sparse Skyline format
c
c       ip_col_ptr = ip_eq + 3*n_ddl
c       ip_bandw  = ip_col_ptr + neq + 1
c       int_used = ip_bandw + neq + 1
cC
c       if (int_max .lt. int_used) then
c         write(ui,*)
c         write(ui,*) 'The size of the integer supervector is too small'
c         write(ui,*) 'integer super-vec: int_max  = ', int_max
c         write(ui,*) 'integer super-vec: int_used = ', int_used
c         write(ui,*) 'Aborting...'
c         stop
c       endif
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
c
c      Sparse matrix storage
c
       ip_col_ptr = ip_eq + 3*n_ddl

       call csr_max_length (n_msh_el, n_ddl, neq, nodes_per_el,
     *   awk(ip_table_N_E_F), awk(ip_eq), awk(ip_col_ptr), nonz_max)
c
c       ip = ip_col_ptr + neq + 1 + nonz_max
       ip = ip_col_ptr + neq + 1
       if (ip .gt. int_max) then
          write(emsg,*) "py_calc_modes.f: ip > int_max : ",
     *     ip, int_max, "py_calc_modes.f: nonz_max = ", nonz_max,
     *    "py_calc_modes.f: increase the size of int_max"
         errco = -11
         return
       endif
c
       ip_row = ip_col_ptr + neq + 1

       call csr_length (n_msh_el, n_ddl, neq, nodes_per_el,
     *    awk(ip_table_N_E_F), awk(ip_eq), awk(ip_row), awk(ip_col_ptr),
     *   nonz_max, nonz, max_row_len, ip, int_max, debug)

       ip_work = ip_row + nonz
       ip_work_sort = ip_work + 3*n_ddl
       ip_work_sort2 = ip_work_sort + max_row_len

c      sorting csr ...
```

```
      call sort_csr (neq, nonz, max_row_len, awk(ip_row),
     *  awk(ip_col_ptr), awk(ip_work_sort), awk(ip_work),
     *  awk(ip_work_sort2))

      if (debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: nonz_max = ", nonz_max
        write(ui,*) "py_calc_modes.f: nonz = ", nonz
        write(ui,*) "py_calc_modes.f: cmplx_max/nonz = ",
     *    dble(cmplx_max)/dble(nonz)
      endif

      int_used = ip_work_sort2 + max_row_len

      if (int_max .lt. int_used) then
        write(emsg,*)'The size of the integer supervector is too small',
     *   'integer super-vec: int_max  =', int_max,
     *   'integer super-vec: int_used =', int_used
        errco = -12
        return
      endif
c
ccccccccccccccccccccccccccccccccccccccccccccccccc
c
cC
      jp_mat2 = jp_x_N_E_F + 3*n_ddl

      jp_vect1 = jp_mat2 + nonz
      jp_vect2 = jp_vect1 + neq
      jp_workd = jp_vect2 + neq
      jp_resid = jp_workd + 3*neq

C     ! Eigenvectors
      jp_vschur = jp_resid + neq
      jp_trav = jp_vschur + neq*nvect

      ltrav = 3*nvect*(nvect+2)
      jp_vp = jp_trav + ltrav

      cmplx_used = jp_vp + neq*n_modes
C
      if (cmplx_max .lt. cmplx_used)  then
        write(emsg,*)'The size of the complex supervector is too small',
     *   'complex super-vec: int_max  =', cmplx_max,
     *   'complex super-vec: int_used =', cmplx_used
        errco = -13
        return
      endif
c
      kp_rhs_re = 1
      kp_rhs_im = kp_rhs_re + neq
      kp_lhs_re = kp_rhs_im + neq
      kp_lhs_im = kp_lhs_re + neq
      kp_mat1_re = kp_lhs_im + neq
      kp_mat1_im = kp_mat1_re + nonz
      real_used = kp_mat1_im + nonz

      if (real_max .lt. real_used) then
        write(emsg,*) 'The size of the real supervector is too small',
     *  '2*nonz =', 2*nonz,
     *  'real super-vec: real_max  =', real_max,
     *  'real super-vec: real_used =', real_used
        errco = -14
```



```
        return
      endif
c
c
c##############################################
c
c       ------------------------------------------------------------
c       convert from 1-based to 0-based
c       ------------------------------------------------------------
c
      do j = 1, neq+1
         awk(j+ip_col_ptr-1) = awk(j+ip_col_ptr-1) - 1
      end do
      do  j = 1, nonz
         awk(j+ip_row-1) = awk(j+ip_row-1) - 1
      end do
c
c
c     The CSC indexing, i.e., ip_col_ptr, is 1-based
c       (but valpr.f will change the CSC indexing to 0-based indexing)
      i_base = 0
c
C
C#####################  End FEM PRE-PROCESSING  ##########################
C
C
C
      write(ui,*)
      write(ui,*) "------------------------------------------------"
C       write(ui,*) " EM FEM, lambda : ", lambda*1.0d9, "nm"
C       write(ui,*) "------------------------------------------------"
C       write(ui,*)
C C
      freq = 1.0d0/lambda
      k_0 = 2.0d0*pi*freq
C
C  Index number of the core materials (material with highest Re(eps_eff))
      if(dble(eps_eff(1)) .gt. dble(eps_eff(2))) then
         n_core(1) = 1
      else
         n_core(1) = 2
      endif
      n_core(2) = n_core(1)
C  Check that the layer is not in fact homogeneous
        homogeneous_check = 0
        do i=1,n_typ_el-1
          if(dble(eps_eff(i)) .ne. dble(eps_eff(i+1))) then
            homogeneous_check = 1
          elseif(dimag(eps_eff(i)) .ne. dimag(eps_eff(i+1))) then
            homogeneous_check = 1
          endif
        enddo
        if(homogeneous_check .eq. 0) then
          write(emsg,*) "py_calc_modes_1d.f: ",
     *              "FEM routine cannot handle homogeneous layers.",
     *     "Define layer as object.ThinFilm"
         errco = -17
         return
        endif
C Parameter for shift-and-invert method - now given as input from python
C      shift_ksqr = 1.01d0*Dble(v_refindex_n(n_core(1)))**2 * k_0**2
C     *    - bloch_vec(1)**2 - bloch_vec(2)**2
```

```
      if(debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: n_core = ", n_core
        if(E_H_field .eq. 1) then
          write(ui,*) "py_calc_modes.f: E-Field formulation"
        else
          write(ui,*) "py_calc_modes.f: H-Field formulation"
        endif
      endif
C
C
      if(E_H_field .eq. 1) then
        do i=1,n_typ_el
          qq(i) = eps_eff(i)*k_0**2
          pp(i) = 1.0d0
        enddo
      elseif(E_H_field .eq. 2) then
        do i=1,n_typ_el
          qq(i) = k_0**2
          pp(i) = 1.0d0/eps_eff(i)
        enddo
      else
C     Can't happen
        write(ui,*) "py_calc_modes.f: action indef. avec E_H_field = ",
     *                E_H_field
        write(ui,*) "Aborting..."
         errco = -18
         return
      endif
C
CCCCCCCCCCCCCCCCCCCC  Loop over Adjoint and Prime  CCCCCCCCCCCCCCCCCCCCC
C
      do n_k = 1,2
C
      if (n_k .eq. 1) then
        sol => sol1
        beta => beta1
        bloch_vec_k = bloch_vec
        msg = "adjoint solution"
      else
        sol => sol2
        beta => beta2
        bloch_vec_k = -bloch_vec
        msg = "prime solution"
      endif
C
C     Assemble the coefficient matrix A and the right-hand side F of the
C     finite element equations
C       if (debug .eq. 1) then
C         write(ui,*) "py_calc_modes.f: Asmbly: call to asmbly"
C       endif
      write(ui,*) "EM FEM: "
      write(ui,'(A,A)') "    - assembling linear system for ", msg

      call get_clocks(stime1, time1)

      call asmbly (bnd_cdn_i, i_base, n_msh_el, n_msh_pts, n_ddl, neq,
     *  nodes_per_el, shift_ksqr, bloch_vec_k, n_typ_el, pp, qq,
     * table_nod, awk(ip_table_N_E_F), type_el, awk(ip_eq),
     *  awk(ip_period_N), awk(ip_period_N_E_F), mesh_xy,
     *  bwk(jp_x_N_E_F), nonz, awk(ip_row), awk(ip_col_ptr),
```



```
     *  cwk(kp_mat1_re), cwk(kp_mat1_im), bwk(jp_mat2), awk(ip_work))

      call get_clocks(stime2, time2)
      write(ui,'(A,F6.2,A)') '     cpu time = ', (time2-time1),
     *   ' secs.'
      write(ui,'(A,F6.2,A)') '     wall time = ', (stime2-stime1),
     *   ' secs.'
C
C     factorization of the globale matrice
C     ------------------------------------
C
      if (debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: Adjoint(1) / Prime(2)", n_k
c       write(ui,*) "py_calc_modes.f: factorisation: call to znsy"
      endif
C
C       if (debug .eq. 1) then
C         write(ui,*) "py_calc_modes.f: call to valpr"
C       endif
      write(ui,*) "   - solving linear system"

      call get_clocks(stime1, time1)

      call valpr_64 (i_base, nvect, n_modes, neq, itermax, ltrav,
     *  tol, nonz, awk(ip_row), awk(ip_col_ptr), cwk(kp_mat1_re),
     *  cwk(kp_mat1_im), bwk(jp_mat2), bwk(jp_vect1), bwk(jp_vect2),
     *  bwk(jp_workd), bwk(jp_resid), bwk(jp_vschur), beta,
     *  bwk(jp_trav), bwk(jp_vp), cwk(kp_rhs_re), cwk(kp_rhs_im),
     *  cwk(kp_lhs_re), cwk(kp_lhs_im), n_conv, ls_data,
     *  numeric, control, info_umf, debug, errco, emsg)
      call get_clocks(stime2, time2)
      write(ui,'(A,F6.2,A)') '     cpu time = ', (time2-time1),
     *   ' secs.'
      write(ui,'(A,F6.2,A)') '     wall time = ', (stime2-stime1),
     *   ' secs.'
c
      if (errco .ne. 0) then
         return
      endif

      if (n_conv .ne. n_modes) then
         write(ui,*) "py_calc_modes.f: convergence problem in valpr_64"
c         write(ui,*) "You should probably increase resolution of mesh!"
         write(ui,*) "py_calc_modes.f: n_conv != n_modes : ",
     *    n_conv, n_modes
c        write(ui,*) "n_core(1), v_refindex_n(n_core(1)) = ",
c    *                n_core(1), v_refindex_n(n_core(1))
         write(ui,*) "py_calc_modes.f: Aborting..."
         errco = -19
         return
      endif
c
      time1_fact = ls_data(1)
      time2_fact = ls_data(2)
c
      time1_arpack = ls_data(3)
      time2_arpack = ls_data(4)
C
      do i=1,n_modes
        z_tmp0 = beta(i)
        z_tmp = 1.0d0/z_tmp0+shift_ksqr
        z_beta = sqrt(z_tmp)
```

```
C       Mode classification - we want the forward propagating mode
        if (abs(imag(z_beta)/z_beta) .lt. 1.0d-8) then
C         re(z_beta) > 0 for forward propagating mode
          if (dble(z_beta) .lt. 0) z_beta = -z_beta
        else
C         im(z_beta) > 0 for forward decaying evanescent mode
          if (imag(z_beta) .lt. 0) z_beta = -z_beta
        endif
C     !  Effective iindex
C        z_beta = sqrt(z_tmp)/k_0
        beta(i) = z_beta
      enddo
c
      call get_clocks(stime1_postp, time1_postp)
C
      call z_indexx (n_modes, beta, iindex)
C
C       The eigenvectors will be stored in the array sol
C       The eigen_modesues and eigenvectors will be renumbered
C                 using the permutation vector iindex
        call array_sol (bnd_cdn_i, n_modes, n_msh_el, n_msh_pts,
     *  n_ddl, neq, nodes_per_el, n_core, bloch_vec_k, iindex,
     *   table_nod, awk(ip_table_N_E_F), type_el, awk(ip_eq),
     *   awk(ip_period_N), awk(ip_period_N_E_F), mesh_xy,
     *   bwk(jp_x_N_E_F), beta, mode_pol, bwk(jp_vp), sol)
C
      if(debug .eq. 1) then
        write(ui,*) 'iindex = ', (iindex(i), i=1,n_modes)
      endif
      if(debug .eq. 1) then
        write(ui,*)
        write(ui,*) "lambda, 1/lambda = ", lambda, 1.0d0/lambda
        write(ui,*) (bloch_vec_k(i)/(2.0d0*pi),i=1,2)
        write(ui,*) "sqrt(shift_ksqr) = ", sqrt(shift_ksqr)
        write(ui,*) "n_modess = "
        do i=1,n_modes
          write(ui,"(i4,2(g22.14),2(g18.10))") i,
     *       beta(i)
        enddo
      endif
C
C  Calculate energy in each medium (typ_el)
      if (n_k .eq. 2) then
        call mode_energy (n_modes, n_msh_el, n_msh_pts, nodes_per_el,
     *     n_core, table_nod, type_el, n_typ_el, eps_eff,
     *     mesh_xy, sol, beta, mode_pol)
      endif
C
      enddo
C
CCCCCCCCCCCCCCCCCCCCCCCC  End Prime, Adjoint Loop  CCCCCCCCCCCCCCCCCCCCC
C
C  Orthogonal integral
      pair_warning = 0
      if (debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: Field product"
      endif
      overlap_file = "Normed/Orthogonal.txt"
      call get_clocks(stime1_J, time1_J)
      call orthogonal (n_modes, n_msh_el, n_msh_pts, nodes_per_el,
     *  n_typ_el, pp, table_nod,
     *  type_el, mesh_xy, beta1, beta2,
```



```
     *  sol1, sol2, overlap_L,
     *  overlap_file, PrintAll, pair_warning, k_0)

      if (pair_warning .ne. 0 .and. n_modes .le. 20) then
        write(ui,*) "py_calc_modes.f: Warning found 1 BM of cmplx conj"
        write(ui,*) "pair, increase num_BMs to include the other."
      endif

      call get_clocks(stime2_J, time2_J)
      if (debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: CPU time for orthogonal :",
     *  (time2_J-time1_J)
      endif

c     The z-component must be multiplied by -ii*beta in order to
C     get the physical, un-normalised z-component
C     (see Eq. (25) of the JOSAA 2012 paper)
      do ival=1,n_modes
        do iel=1,n_msh_el
          do inod=1,nodes_per_el+7
            sol1(3,inod,ival,iel)
     *        = ii * beta(ival) * sol1(3,inod,ival,iel)
          enddo
        enddo
      enddo

      call array_material_EM (n_msh_el,
     *  n_typ_el, v_refindex_n, type_el, ls_material)

C
C     Save Original solution
c       if (plot_modes .eq. 1) then
c         dir_name = "Bloch_fields"
c         q_average = 0
cC          call write_sol (n_modes, n_msh_el, nodes_per_el, E_H_field, lambda,
cC     *       beta1, sol1, mesh_file, dir_name)
cC          call write_param (E_H_field, lambda, n_msh_pts, n_msh_el, bnd_cdn_i,
cC     *       n_modes, nvect, itermax, tol, shift_ksqr, dim_x, dim_y,
cC     *       mesh_file, mesh_format, n_conv, n_typ_el, eps_eff,
cC     *       bloch_vec, dir_name)
c         tchar = "Bloch_fields/PDF/All_plots_pdf.geo"
c         open (unit=34,file=tchar)
c           do i=1,n_modes
c             call gmsh_post_process (i, E_H_field, n_modes, n_msh_el,
c     *         n_msh_pts, nodes_per_el, table_nod, type_el, n_typ_el,
c     *         v_refindex_n, mesh_xy, beta1, sol1,
c     *         awk(ip_visite), gmsh_file_pos, dir_name,
c     *         q_average, plot_real, plot_imag, plot_abs)
c           enddo
c         close (unit=34)
c       endif
cC

c Normalisation
      if(debug .eq. 1) then
        write(ui,*) "py_calc_modes.f: Field  Normalisation"
      endif
      call get_clocks(stime1_J, time1_J)
      call normalisation (n_modes, n_msh_el, nodes_per_el, sol1, sol2,
     *    overlap_L)
      call get_clocks(stime2_J, time2_J)
      if (debug .eq. 1) then
```

```
        write(ui,*) "py_calc_modes.f: CPU time for normalisation :",
     *  (time2_J-time1_J)
       endif
C
C  Orthonormal integral
      if (PrintAll .eq. 1) then
        write(ui,*) "py_calc_modes.f: Product of normalised field"
        overlap_file = "Normed/Orthogonal_n.txt"
        call get_clocks(stime1_J, time1_J)
        call orthogonal (n_modes, n_msh_el, n_msh_pts, nodes_per_el,
     *    n_typ_el, pp, table_nod,
     *    type_el, mesh_xy, beta1, beta2,
     *    sol1, sol2, overlap_L,
     *    overlap_file, PrintAll, pair_warning, k_0)
        call get_clocks(stime2_J, time2_J)
        write(ui,*) "py_calc_modes.f: CPU time for orthogonal :",
     *    (time2_J-time1_J)
       endif
C
C########################  End Calculations  ############################
C
      call get_clocks(stime2, time2)
C
      if (debug .eq. 1) then
        write(ui,*)
        write(ui,*) 'Total CPU time (sec.) =', (time2-time1)

        open (unit=26,file=log_file)
        write(26,*)
        write(26,*) "Date and time formats = ccyymmdd ; hhmmss.sss"
        write(26,*) "Start time  = ",  start_time
        write(26,*) "End time    = ",  end_time
        write(26,*) "Total CPU time (sec.) = ",  (time2-time1)
        write(26,*) "LU factorisation : CPU time and % Total time = ",
     *          (time2_fact-time1_fact),
     *          100*(time2_fact-time1_fact)/(time2-time1),"%"
        write(26,*) "ARPACK : CPU time and % Total time = ",
     *          (time2_arpack-time1_arpack),
     *          100*(time2_arpack-time1_arpack)/(time2-time1),"%"
C       write(26,*) "Assembly : CPU time and % Total time = ",
C    *          (time2_asmbl-time1_asmbl),
C    *          100*(time2_asmbl-time1_asmbl)/(time2-time1),"%"
        write(26,*) "Post-processsing : CPU time and % Total time = ",
     *          (time2-time1_postp),
     *          100*(time2-time1_postp)/(time2-time1),"%"
C       write(26,*) "Pre-Assembly : CPU time and % Total time = ",
C    *          (time1_asmbl-time1),
C    *          100*(time1_asmbl-time1)/(time2-time1),"%"
        write(26,*)
        write(26,*) "lambda =", lambda
        write(26,*) "n_msh_pts, n_msh_el, nodes_per_el =", n_msh_pts,
     *    n_msh_el, nodes_per_el
        write(26,*) "neq, bnd_cdn_i =", neq, bnd_cdn_i
        if ( E_H_field .eq. 1) then
          write(26,*) "E_H_field     =", E_H_field,
     *                " (E-Field formulation)"
        elseif ( E_H_field .eq. 2) then
          write(26,*) "E_H_field     =", E_H_field,
     *                " (H-Field formulation)"
      else
          write(ui,*) "MAIN (B): action indef. avec E_H_field =",
     *                E_H_field
```



```
          write(ui,*) "Aborting..."
         errco = -20
         return
        endif
        write(26,*) "  bloch_vec =", bloch_vec
        write(26,*) "bloch_vec/pi =",  (bloch_vec(i)/pi,i=1,2)
        z_tmp = sqrt(shift_ksqr)/(2.0d0*pi)
        write(26,*) "shift_ksqr       =", shift_ksqr, z_tmp
        write(26,*) "integer super-vector :"
        write(26,*) "int_used, int_max, int_used/int_max      =",
     *    int_used , int_max, dble(int_used)/dble(int_max)
        write(26,*) "cmplx super-vector : "
        write(26,*) "cmplx_used, cmplx_max, cmplx_used/cmplx_max =",
     *    cmplx_used, cmplx_max, dble(cmplx_used)/dble(cmplx_max)
        write(26,*) "Real super-vector : "
        write(26,*) "real_used, real_max, real_max/real_used =",
     *    real_used, real_max, dble(real_max)/dble(real_used)
        write(26,*)
        write(26,*) "n_modes, nvect, n_conv =", n_modes, nvect,
     *    n_conv
        write(26,*) "nonz, n_msh_pts*n_modes, ",
     *    "nonz/(n_msh_pts*n_modes) = ",
     *  nonz, n_msh_pts*n_modes, dble(nonz)/dble(n_msh_pts*n_modes)
        write(26,*) "nonz, nonz_max, nonz_max/nonz = ",
     *  nonz, nonz_max, dble(nonz_max)/dble(nonz)
        write(26,*) "nonz, int_used, int_used/nonz = ",
     *  nonz, int_used, dble(int_used)/dble(nonz)
c
c         write(26,*) "len_skyl, n_msh_pts*n_modes, len_skyl/(n_msh_pts*n_modes)
 = ",
c     *   len_skyl, n_msh_pts*n_modes, dble(len_skyl)/dble(n_msh_pts*n_modes)
c
        write(26,*)
        do i=1,n_modes
          write(26,"(i4,2(g22.14),g18.10)") i,
     *      beta1(i)
        enddo
        write(26,*)
        write(26,*) "n_core =", n_core
        write(26,*) "eps_eff =", (eps_eff(i),i=1,n_typ_el)
        write(26,*) "v_refindex_n =", (v_refindex_n(i),i=1,n_typ_el)
        write(26,*)
        write(26,*) "conjugate pair problem", pair_warning, "times"
        write(26,*)
        write(26,*) "mesh_file =", mesh_file
        write(26,*) "gmsh_file =", gmsh_file
        write(26,*) "log_file  =", log_file
        close(26)
C
        write(ui,*) " .  .  ."
        write(ui,*) " .  .  ."
        write(ui,*) " .  .  ."
        write(ui,*) " and  we're  done!"
      endif

      write(ui,*) "-----------------------------------------------"
      write(ui,*)
C
      deallocate(awk, bwk, cwk , iindex, overlap_L)

      end subroutine calc_EM_modes
```