```
#include "numbat_decl.h"

!  Solves the electromagnetic FEM problem defined in
 !  Dossou & Fontaine, Comp Meth. App. Mech. Eng, 194, 837 (2005).

 !  The weak formulation of Maxwell wave equation is in Eqs 14, 15.
 !  \langle 1/\mu (\nabla_t \times E_t), (\nabla_t \times F_t) \rangle
 !  - \omega^2 \langle (\epsilon E_t, F_t)
 !  = \beta^2 \langle 1/\mu (\nabla_t hE_z -E_t, F_t), \rangle

 !  \langle 1/\mu E_t, \nabla_t F_z \rangle
 !  - \langle 1/\mu\nabla_t hE_z, \nabla_t F_z \rangle
 !  + \omega^2 \langle\eps hE_z, F_z\rangle =0

 !  where \hE_z = -1/\beta E_z

 !  The fields are expanded in in-plane vector and longitudinal scalar elements
 !  \vecphi_h and \psi_h:
 !  E = E_{t,h} \vecphi_h + \unitz hE_{z,h} \psi_h = [E_{t,h} \vecphi_h, hE_{z,h
} \psi_h ]
 !  F = F_{t,h} \vecphi_h + \unitz F_{z,h} \psi_h   (note F, not hF)

 !  Then  inner product (L_1 E, L_2 F) is evaluated:
 !  (E,F) = \int dx dy   (L_2 F)^* \cdot (L_1 E)
 !  = \int dx dy    ((L_2 F)_t)^* \cdot ((L_1 E)_t)
 !  +  ((L_2 F)_z)^* . ((L_1 E)_z)

 !  = \int dx dy    ((L_2 F)_t)^* \cdot ((L_1 E)_t)
 !  +  ((L_2 F)_z)^* . ((L_1 E)_z)

 !  This translates to the geneig problem (eq 40)

 !  [ K_tt    0 ] [ E_t,h]  = \beta^2  [M_tt    (K_zt)^T] [E_t,h]
 !  [ 0       0 ] [ hE_z,h]            [K_zt     K_zz   ] [hE_z,h]



 !  lambda - free space wavelength in m
 !  n_modes - desired number of eigenvectors
 !  n_msh_pts - number of FEM mesh points
 !  n_msh_el  - number of FEM (triang) elements
 !  n_elt_mats  - number of types of elements (and therefore elements)
 !  v_refindex_n - array of effective index of materials
 !  bloch_vec - in-plane k-vector (normally tiny just to avoid degeneracies)
 !  shift_ksqr   - k_est^2 = n^2 vacwavenum_k0^2  : estimate of eigenvalue k^2
 !  bnd_cnd_i - bnd conditions (Dirichlet = 0, Neumann = 1, Periodic = 2)
 !  v_evals_beta  - array of eigenvalues kz
 !  femsol_evecs   - 4-dim array of solutions [field comp, node of element (1..1
3)?!, eigvalue, element number] (strange ordering)
 !  poln_fracs  - unknown - never used in python
 !  elnd_to_mshpt - 2D array [node_on_elt-1..6][n_msh_el] giving the mesh point
mp of each node
 !  Points where v_el_material[mp] is not the same for all 6 nodes must be inter
face points
 !  v_el_material    - n_msh_el array: material index for each element
 !  v_nd_physindex  - is boundary node?
 !  v_nd_xy  - (2 , n_msh_pts)  x,y coords?
 !  ls_material  - (1, N_DOF_PER_EL, n_msh_el)

module calc_em_impl
```

```
   use numbatmod
   use alloc

   use class_stopwatch
   use class_MeshRaw
   use class_SparseCSC
   use class_PeriodicBCs



contains

   subroutine calc_em_modes_impl(n_modes, lambda, dimscale_in_m, bloch_vec, shif
t_ksqr, &
      E_H_field, bdy_cdn, itermax, debug, &
      mesh_file, n_msh_pts, n_msh_el, n_elt_mats, v_refindex_n, shortrun, &
      v_evals_beta, femsol_evecs, poln_fracs, &
      elnd_to_mshpt, v_el_material, v_nd_physindex, v_nd_xy, ls_material, nberr)

      integer(8), intent(in) :: n_modes
      double precision, intent(in) :: lambda, dimscale_in_m, bloch_vec(2)
      complex(8), intent(in) :: shift_ksqr

      integer(8), intent(in) :: E_H_field, bdy_cdn, itermax, debug
      character(len=*), intent(in) :: mesh_file
      integer(8), intent(in) :: n_msh_pts,  n_msh_el, n_elt_mats

      complex(8), intent(in) ::  v_refindex_n(n_elt_mats)
      integer(8) :: shortrun

      complex(8), target, intent(out) :: v_evals_beta(n_modes)
      complex(8), target, intent(out) :: femsol_evecs(3,N_DOF_PER_EL,n_modes,n_m
sh_el)

      complex(8), intent(out) :: poln_fracs(4,n_modes)

      integer(8), intent(out) :: v_el_material(n_msh_el)
      integer(8), intent(out) :: v_nd_physindex(n_msh_pts)
      integer(8), intent(out) :: elnd_to_mshpt(P2_NODES_PER_EL, n_msh_el)
      double precision, intent(out) :: v_nd_xy(2,n_msh_pts)

      complex(8), intent(out) :: ls_material(1,N_DOF_PER_EL,n_msh_el)
      type(NBError) nberr

      ! locals

      type(MeshRaw) :: mesh_raw
      type(MeshEntities) :: entities
      type(SparseCSC) :: cscmat
      type(PeriodicBCs) :: pbcs

      integer(8), dimension(:), allocatable :: v_eig_index
      complex(8), dimension(:,:), allocatable :: overlap_L

      complex(8), dimension(:,:), allocatable :: arp_evecs

      ! Should these be dynamic?
      complex(8) pp(n_elt_mats), qq(n_elt_mats)
      complex(8) eps_eff(n_elt_mats)

      integer(8) ui_out
```

```fortran
      !  Variable used by valpr
      integer(8) dim_krylov
      integer(8) i_base
      double precision arp_tol

      integer(8) n_core(2)  !  index of highest epsilon material, seems funky
      double precision vacwavenum_k0

      type(Stopwatch) :: clock_main, clock_spare


      ui_out = stdout

      arp_tol = 1.0d-12 ! TODO: ARPACK_ stopping precision,  connect  to user sw
itch


      call clock_main%reset()

      !TODO: move pp,qq to elsewhere. SparseCSC?
      vacwavenum_k0 = 2.0d0*D_PI/lambda
      call  check_materials_and_fem_formulation(E_H_field, n_elt_mats, &
      vacwavenum_k0, v_refindex_n, eps_eff, n_core, pp, qq, debug, ui_out, nberr
)
      RET_ON_NBERR(nberr)

      !  ----------------------------------------------------------------

      call mesh_raw%allocate(n_msh_pts, n_msh_el, n_elt_mats, nberr)
      RET_ON_NBERR(nberr)

      call entities%allocate(n_msh_el, nberr)
      RET_ON_NBERR(nberr)


      ! These are never actually used for now so could disable
      call pbcs%allocate(mesh_raw, entities, nberr);
      RET_ON_NBERR(nberr)

      !  Fills:  MeshRaw: v_nd_xy, v_nd_physindex, v_el_material, elnd_to_mshpt
      ! This knows the position and material of each elt and mesh point but not
their connectedness or edge/face nature
      call mesh_raw%construct_node_tables(mesh_file, dimscale_in_m, nberr);
      RET_ON_NBERR(nberr)

      ! Fills entities
      call entities%build_mesh_tables(mesh_raw, nberr);
      RET_ON_NBERR(nberr)

      ! Builds the m_eqs table which maps element DOFs to the equation handling
them, according to the BC (Dirichlet/Neumann)
      call cscmat%set_boundary_conditions(bdy_cdn, mesh_raw, entities, pbcs, nbe
rr);
      RET_ON_NBERR(nberr)


      ! Build sparse matrix index arrays
      call cscmat%make_csc_arrays(mesh_raw, entities, nberr); RET_ON_NBERR(nberr
)

      i_base = 0
```

```fortran

      write(ui_out,*)
      write(ui_out,*) "-----------------------------------------------"

      !  Main eigensolver
      write(ui_out,*) "EM FEM: "

      !  Assemble the coefficient matrix A and the right-hand side F of the
      !  finite element equations

      write(ui_out,'(A,A)') "  - assembling linear system:"
      call clock_spare%reset()






      !  Build the actual matrices A (cscmat%mOp_stiff) and M(cscmat%mOp_mass) f
or the arpack solving.

      call assembly_em (bdy_cdn, i_base, shift_ksqr, bloch_vec, pp, qq, &
      mesh_raw, entities, cscmat, pbcs, nberr)
      RET_ON_NBERR(nberr)

      dim_krylov = 2*n_modes + n_modes/2 +3


      write(ui_out,'(A,i9,A)') '    ', n_msh_el, ' mesh elements'
      write(ui_out,'(A,i9,A)') '    ', n_msh_pts, ' mesh nodes'
      write(ui_out,'(A,i9,A)') '    ', cscmat%n_dof, ' linear equations (cscmat%n_dof)'
      write(ui_out,'(A,i9,A)') '    ', cscmat%n_nonz, ' nonzero elements  (cscmat%n_nonz)'
      write(ui_out,'(A,f9.3,A)') '    ', cscmat%n_nonz/(1.d0*cscmat%n_dof*cscmat%n_d
of)*100.d0, ' % sparsity'
      write(ui_out,'(A,i9,A)') '    ', cscmat%n_dof*(dim_krylov+6)*16/2**20, ' MB est.
 working memory '

      write(ui_out,'(/,A,A)') '    ', clock_spare%to_string()


      !  This is the main solver.
      !  On completion:
      !  unshifted unsorted eigenvalues are in v_evals_beta[1..n_modes]
      !  eigvectors are in arp arp_evecs

      write(ui_out,'(/,A)') "  - solving linear system: "

      write(ui_out,'(/,A)') "      solving eigensystem"
      call clock_spare%reset()


      call integer_nalloc_1d(v_eig_index, n_modes, 'v_eig_index', nberr); RET_ON_N
BERR(nberr)

      call complex_nalloc_2d(overlap_L, n_modes, n_modes, 'overlap_L', nberr); RET
_ON_NBERR(nberr)

      call complex_nalloc_2d(arp_evecs, cscmat%n_dof, n_modes, 'arp_evecs', nberr)
; RET_ON_NBERR(nberr)

      call valpr_64( i_base, dim_krylov, n_modes, itermax, arp_tol, cscmat, &
      v_evals_beta, arp_evecs, nberr, shortrun); RET_ON_NBERR(nberr)
      if (shortrun .ne. 0) then
         write(*,*) 'Exiting with shortrun in py_calc_modes.f'
         return
      endif

      write(ui_out,'(A,A)') '    ', clock_spare%to_string()



      write(ui_out,'(/,A)') "   assembling modes"
      call clock_spare%reset()


      !  The eigenvectors will be stored in the array femsol_evecs
      !  The eigenvalues and eigenvectors are renumbered according to evalue sor
ting
      call construct_solution_fields_em(bdy_cdn, shift_ksqr, n_modes, mesh_raw,
&
      entities, cscmat, pbcs, bloch_vec, v_evals_beta, arp_evecs, &
      femsol_evecs, poln_fracs, nberr)
      RET_ON_NBERR(nberr)


      !TODO: does this serve any purpose any more? Just poln_fracs?
      call mode_energy (n_modes, n_msh_el, n_core, mesh_raw, &
      n_elt_mats, eps_eff, femsol_evecs, poln_fracs)



      ! prepare to return data to python end
      call array_material_EM (n_msh_el, n_elt_mats, v_refindex_n, mesh_raw%el_ma
terial, ls_material)
      call mesh_raw%fill_python_arrays(v_el_material, v_nd_physindex, elnd_to_ms
hpt, v_nd_xy)


      deallocate(v_eig_index, overlap_L, arp_evecs)

      write(ui_out,'(A,A)') '    ', clock_spare%to_string()
      write(ui_out,*) "-----------------------------------------------"

   end subroutine calc_em_modes_impl






   subroutine check_materials_and_fem_formulation(E_H_field,n_elt_mats, &
      vacwavenum_k0, v_refindex_n, eps_eff, n_core, pp, qq, debug, ui_out, nberr
)

      integer(8), intent(in) :: E_H_field, debug
      integer(8), intent(in) :: n_elt_mats, ui_out
      double precision, intent(in):: vacwavenum_k0
      complex(8), intent(in) :: v_refindex_n(n_elt_mats)
      complex(8), intent(out) :: eps_eff(n_elt_mats)

      integer(8), intent(out) :: n_core(2)
      complex(8), intent(out) :: pp(n_elt_mats), qq(n_elt_mats)
```

```

      type(NBError) nberr

      integer(8) i
      logical is_homogeneous

      eps_eff = v_refindex_n**2

      !  what actually even is this?
      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)

      !  Check that the structure is not entirely homogeneous (TODO: does this a
ctually matter?)
      is_homogeneous = .true.
      do i=1,n_elt_mats-1

         if (.not. almost_equal(dble(eps_eff(i)), dble(eps_eff(i+1)))) then
            is_homogeneous = .false.
         elseif (.not. almost_equal(dimag(eps_eff(i)), dimag(eps_eff(i+1)))) the
n
            is_homogeneous = .false.
         endif

      enddo

      if (is_homogeneous) then
         call nberr%set(-17_8, &
            "py_calc_modes.f: FEM routine cannot adjacent identical layers. Define layer as object.ThinFilm.")
         return
      endif


      if(debug .eq. 1) then
         write(ui_out,*) "py_calc_modes.f: n_core = ", n_core
         if(E_H_field .eq. FEM_FORMULATION_E) then
            write(ui_out,*) "py_calc_modes.f: E-Field formulation"
         else
            write(ui_out,*) "py_calc_modes.f: H-Field formulation"
         endif
      endif


      !  set up some kind of mass vectors for the FEM
      !  weird place but ok.
      if(E_H_field .eq. FEM_FORMULATION_E) then
         qq = eps_eff*vacwavenum_k0**2
         pp = 1.0d0
      elseif(E_H_field .eq. FEM_FORMULATION_H) then
         qq = vacwavenum_k0**2
         pp = 1.0d0/eps_eff
      endif

   end subroutine

   subroutine check_orthogonality_of_em_sol(n_modes, n_msh_el, n_msh_pts, &
      n_elt_mats, pp, elnd_to_mshpt, &
      v_el_material, v_nd_xy, v_evals_beta, femsol_evecs, &
   !v_evals_beta_pri, femsol_evecs_pri, &
      overlap_L, overlap_file, debug, ui_out, pair_warning, vacwavenum_k0, errco
, emsg)

      use numbatmod
      logical pair_warning

      integer(8), intent(in) :: n_modes, debug, ui_out
      integer(8), intent(in) :: n_msh_pts,  n_msh_el, n_elt_mats
      complex(8) pp(n_elt_mats)

      integer(8), intent(out) :: elnd_to_mshpt(P2_NODES_PER_EL, n_msh_el)
      integer(8), intent(out) :: v_el_material(n_msh_el)
      double precision, intent(out) :: v_nd_xy(2,n_msh_pts)
      double precision vacwavenum_k0

      complex(8), target, intent(out) :: v_evals_beta(n_modes)
      complex(8), target, intent(out) :: femsol_evecs(3,N_DOF_PER_EL,n_modes,n_m
sh_el)

      complex(8), dimension(:,:) :: overlap_L


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

      character(len=FNAME_LENGTH)  overlap_file

      !complex(8)  :: v_evals_beta_pri(n_modes)
      !complex(8)  :: femsol_evecs_pri(3,N_DOF_PER_EL,n_modes,n_msh_el)

      !  Orthogonal integral
      pair_warning = .false.

      if (debug .eq. 1) then
         write(ui_out,*) "py_calc_modes.f: Field product"
      endif

      overlap_file = "Orthogonal.txt"

      call orthogonal (n_modes, n_msh_el, n_msh_pts, P2_NODES_PER_EL, n_elt_mats
, pp, elnd_to_mshpt, &
         v_el_material, v_nd_xy, v_evals_beta, femsol_evecs, &
      !v_evals_beta_pri, femsol_evecs_pri,
         overlap_L, overlap_file, debug, pair_warning, vacwavenum_k0)

      if (pair_warning .and. n_modes .le. 20) then
         emsg = "py_calc_modes.f: Warning found 1 BM of cmplx conj pair, increase num_BMs to include the
other."
         errco = -57
      endif


   end subroutine

   subroutine report_results_em(debug, ui_out, &
      n_msh_pts, n_msh_el, &
      time1, time2, time_fact, time_arpack, time1_postp, &
      lambda, e_h_field, bloch_vec, bdy_cdn,  &
      int_max, cmplx_max, cmplx_used,  n_core, n_conv, n_modes, &
      n_elt_mats, n_dof, dim_krylov, &
      shift_ksqr, v_evals_beta, eps_eff, v_refindex_n)
```

```


      use numbatmod

      integer(8) debug, ui_out, e_h_field, bdy_cdn
      integer(8) int_max, cmplx_max, cmplx_used, int_used, real_max,  n_msh_pts,
 n_msh_el
      double precision bloch_vec(2), lambda
      double precision time1, time2, start_time, end_time, time_fact, time_arpac
k, time1_postp
      integer(8) n_conv, n_modes, n_elt_mats, nonz,  n_core(2), n_dof, dim_krylo
v
      character(len=FNAME_LENGTH)  log_file
      complex(8), intent(in) :: shift_ksqr
      complex(8), target, intent(out) :: v_evals_beta(n_modes)
      complex(8) eps_eff(n_elt_mats)

      complex(8), intent(in) ::  v_refindex_n(n_elt_mats)

      complex(8) z_tmp
      integer(8) i

      !  TODO: hook these up if needed
      cmplx_max = 0
      int_max = 0
      int_used =0
      nonz = 0
      cmplx_used = 0
      real_max = 0


      if (debug .eq. 1) then
         write(ui_out,*)
         write(ui_out,*) '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 =", time_fact, &
            100*(time_fact)/(time2-time1),"%"
         write(26,*) "ARPACK : CPU time and % Total time =", time_arpack, &
            100*(time_arpack)/(time2-time1),"%"
         !  write(26,*) "Assembly : CPU time and % Total time = ",
         !  *    (time2_asmbl-time1_asmbl),
         !  *   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),"%"
         !  write(26,*) "Pre-Assembly : CPU time and % Total time = ",
         !  *    (time1_asmbl-time1),
         !  *   100*(time1_asmbl-time1)/(time2-time1),"%"
         write(26,*)
         write(26,*) "lambda =", lambda
         write(26,*) "n_msh_pts, n_msh_el =", n_msh_pts, n_msh_el
         write(26,*) "n_dof, bdy_cdn =", n_dof, bdy_cdn
         if ( E_H_field .eq. FEM_FORMULATION_E) then
            write(26,*) "E_H_field  =", E_H_field, " (E-Field formulation)"
         elseif ( E_H_field .eq. FEM_FORMULATION_H) then
            write(26,*) "E_H_field  =", E_H_field, " (H-Field formulation)"
         endif
         write(26,*) " bloch_vec = ", bloch_vec
         write(26,*) "bloch_vec/pi = ", (bloch_vec(i)/D_PI,i=1,2)
         z_tmp = sqrt(shift_ksqr)/(2.0d0*D_PI)
         write(26,*) "shift_ksqr = ", shift_ksqr, z_tmp
         !  write(26,*) "integer(8) super-vector :"
         !  write(26,*) "int_used, int_max, int_used/int_max   = ", int_used , i
nt_max, dble(int_used)/dble(int_max)
         !write(26,*) "cmplx super-vector : "
         !write(26,*) "cmplx_used, cmplx_max, cmplx_used/cmplx_max = ", cmplx_us
ed, cmplx_max, dble(cmplx_used)/dble(cmplx_max)

         write(26,*)
         write(26,*) "n_modes, dim_krylov, n_conv = ", n_modes, dim_krylov, 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,*) "len_skyl, n_msh_pts*n_modes, len_skyl/(n_msh_pts*n_mode
s) = ",
         !  *    len_skyl, n_msh_pts*n_modes, dble(len_skyl)/dble(n_msh_pts*n_mod
es)

         write(26,*)
         do i=1,n_modes
            write(26,"(i4,2(g22.14),g18.10)") i, v_evals_beta(i)
         enddo
         write(26,*)
         write(26,*) "n_core = ", n_core
         write(26,*) "eps_eff = ", (eps_eff(i),i=1,n_elt_mats)
         write(26,*) "v_refindex_n = ", (v_refindex_n(i),i=1,n_elt_mats)
         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)

      endif

      write(ui_out,*) "-----------------------------------------------"
      write(ui_out,*)

   end subroutine





end module calc_em_impl
```