```
#include "numbat_decl.h"

 !  q_ac :   acoustic wave number (q_ac)
 !  n_modes:  desired number of solved acoustic modes
 !  n_msh_pts:  number of nodes in mesh
 !  n_msh_el:   number of (triang) elements in mesh
 !  n_v_el_material:  number of types of material
 !  v_nd_physindex:    ??
 !  elnd_to_mshpt:
 !  v_el_material:
 !  v_nd_xy
 !  v_eigs_nu:  eigen frequencies nu=omega/(2D_PI) for each mode
 !  femsol_ac:
 !  poln_fracs:


module calc_ac_impl

   use numbatmod
   use alloc

   use class_stopwatch
   use class_MeshRaw
   use class_SparseCSC_AC

contains

   subroutine calc_ac_modes_impl(n_modes, q_ac, dimscale_in_m, shift_nu, &
      bdy_cdn, itermax, tol, debug, show_mem_est, &
      symmetry_flag, n_elt_mats, c_tensor, rho, supplied_geo_flag, &
      mesh_file, n_msh_pts, n_msh_el, &
      v_nd_physindex, &
      elnd_to_mshpt, v_el_material, v_nd_xy, &
      v_eigs_nu, femsol_ac, poln_fracs, nberr)



      integer(8), intent(in) :: n_modes

      complex(8), intent(in) :: q_ac
      double precision, intent(in) :: dimscale_in_m
      complex(8), intent(in) :: shift_nu
      integer(8), intent(in) :: bdy_cdn, itermax, debug, show_mem_est
      double precision, intent(in) :: tol
      integer(8), intent(in) :: symmetry_flag, supplied_geo_flag
      integer(8), intent(in) :: n_elt_mats

      complex(8), intent(in) :: c_tensor(6,6,n_elt_mats)
      complex(8), intent(in) :: rho(n_elt_mats)

      character(len=FNAME_LENGTH), intent(in)  :: mesh_file
      integer(8), intent(in) :: n_msh_pts, n_msh_el

      integer(8), intent(in) :: v_nd_physindex(n_msh_pts)

      integer(8), intent(inout) :: v_el_material(n_msh_el)
      integer(8), intent(inout) :: elnd_to_mshpt(P2_NODES_PER_EL, n_msh_el)

      double precision, intent(inout) ::  v_nd_xy(2,n_msh_pts)

      complex(8), intent(out), target :: v_eigs_nu(n_modes)
      complex(8), intent(out), target :: femsol_ac(3,P2_NODES_PER_EL,n_modes,n_m
```



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


      type(NBError) nberr

      ! locals

      type(MeshRawAC) mesh_raw
      type(MeshEntitiesAC) entities
      type(SparseCSC_AC) cscmat


      integer(8) :: errco
      character(len=EMSG_LENGTH) :: emsg


      integer(8) int_max, cmplx_max, int_used, cmplx_used
      integer(8) real_max, real_used, n_ddl
      integer(8) n_dof

      integer(8) :: alloc_stat

      integer(8), dimension(:), allocatable :: a_iwork
      complex(8), dimension(:), allocatable :: b_zwork
      double precision, dimension(:), allocatable :: c_dwork

      !double precision, dimension(:,:), allocatable :: d_dwork
      !complex(8), dimension(:,:), allocatable :: dummy_overlap_L  !  not actual
ly used.

      !  Declare the pointers of the integer(8) super-vector
      integer(8) ip_eq
      integer(8) ip_visited

      !  Declare the pointers of the real super-vector
      integer(8) jp_x, jp_mat2
      integer(8) jp_vect1, jp_vect2, jp_workd, jp_resid, jp_vschur
      integer(8) jp_trav, jp_vp, jp_rhs
      integer(8) jp_eigenum_modes_tmp, jp_eigen_pol

      !  Declare the pointers of the real super-vector
      integer(8) kp_mat1_re, kp_mat1_im
      integer(8) kp_rhs_re, kp_rhs_im, kp_lhs_re, kp_lhs_im

      !  Declare the pointers of for sparse matrix storage
      integer(8) ip_col_ptr, ip_row
      integer(8) ip_work, ip_work_sort, ip_work_sort2
      integer(8) nonz, nonz_max, max_row_len



      integer(8) i, j, ip
      integer(8) ui_out,  namelength


      !double precision lat_vecs(2,2)
      double precision dim_x, dim_y

      complex(8) shift_omsq
      integer(8)  i_base
```

```
      !  Variable used by valpr
      integer(8) ltrav, n_conv
      complex(8) z_beta, z_tmp, z_tmp0
      integer(8), dimension(:), allocatable :: iindex
      !  variable used by UMFPACK

      !character*(8) start_date, end_date
      !character*(10) start_time, end_time

      !  Variable used by valpr
      integer(8)  nvect

      !  Names and Controls

      character(len=FNAME_LENGTH)  gmsh_file, log_file, gmsh_file_pos


      type(Stopwatch) :: clock_main, clock_spare

      integer(8) :: is_em



      errco = 0

      ui_out = stdout


      !  nvect = 2*n_modes + n_modes/2 +3
      nvect = 3*n_modes + 3


      errco= 0
      emsg = ""

      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)

      call array_size(n_msh_pts, n_msh_el, n_modes, int_max, cmplx_max, real_max
, &
         n_ddl, errco, emsg)
      call nberr%set(errco, emsg); RET_ON_NBERR(nberr)

      allocate(a_iwork(int_max), STAT=alloc_stat)
      call check_alloc(alloc_stat, int_max, "a", -1, nberr);
      RET_ON_NBERR(nberr)

      allocate(b_zwork(cmplx_max), STAT=alloc_stat)
      call check_alloc(alloc_stat, cmplx_max, "b", -1, nberr); RET_ON_NBERR(nber
r)

      allocate(c_dwork(real_max), STAT=alloc_stat)
      call check_alloc(alloc_stat, real_max, "c", -1,  nberr); RET_ON_NBERR(nber
r)

      allocate(iindex(n_modes), STAT=alloc_stat)
      call check_alloc(alloc_stat, n_modes, "iindex", -1, nberr); RET_ON_NBERR(nb
err)
```



```
      is_em = 0


      !  clean mesh_format
      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)//'-AC.log'
      if (debug .eq. 1) then
         write(*,*) "mesh_file = ", mesh_file
         write(*,*) "gmsh_file = ", gmsh_file
      endif


      dim_x = dimscale_in_m
      dim_y = dimscale_in_m
      shift_omsq= (2*D_PI*shift_nu)**2


      !  pointer to FEM connectivity table
      ip_visited= 1
      ip_eq = ip_visited+ n_msh_pts
      jp_x = 1
      !


      call clock_main%reset()


      if (supplied_geo_flag .eq. 0) then
         call construct_fem_node_tables_ac (mesh_file, dim_x, dim_y, n_msh_el, n
_msh_pts, &
            P2_NODES_PER_EL, n_elt_mats, v_nd_xy, v_nd_physindex, v_el_material,
 elnd_to_mshpt, errco, emsg)
         call nberr%set(errco, emsg); RET_ON_NBERR(nberr)

      endif


      !  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

      if (supplied_geo_flag .eq. 0) then

         call mesh_raw%construct_node_tables(mesh_file, dimscale_in_m, nberr);
         RET_ON_NBERR(nberr)
      else
         call mesh_raw%load_node_tables_from_py(v_nd_xy, v_nd_physindex, &
            v_el_material, elnd_to_mshpt, nberr);
         RET_ON_NBERR(nberr)
      endif


      !call periodic_lattice_vec (n_msh_pts, v_nd_xy, lat_vecs, debug)


      !  Determine number of boundary conditions (n_dof) and 2D index array
      !  a_iwork(ip_eq)
      !call bound_cond_AC (bdy_cdn, n_msh_pts, n_dof, v_nd_physindex, a_iwork(ip
_eq))
```

```
      call bound_cond_AC (bdy_cdn, mesh_raw, n_dof,  a_iwork(ip_eq))

      call cscmat%set_bound_cond(bdy_cdn, mesh_raw, nberr)
      RET_ON_NBERR(nberr)

      !
      !  Sparse matrix storage
      ip_col_ptr = ip_eq + 3*n_msh_pts

      call csr_make_col_ptr_loose_AC (n_msh_el, n_msh_pts, n_dof, P2_NODES_PER_E
L, &
         elnd_to_mshpt, a_iwork(ip_eq), a_iwork(ip_col_ptr), nonz_max)

      ip = ip_col_ptr + n_dof + 1
      if (ip .gt. int_max) then
         write(emsg,*) "py_calc_modes_AC: ip > int_max : ", &
            ip, int_max, &
            "py_calc_modes_AC: nonz_max = ", nonz_max, &
            "py_calc_modes_AC: increase the size of int_max"
         errco = -3
         return
      endif
      !
      ip_row = ip_col_ptr + n_dof + 1

      call csr_length_AC (n_msh_el, n_msh_pts, n_dof, P2_NODES_PER_EL, &
         elnd_to_mshpt, a_iwork(ip_eq), a_iwork(ip_row), a_iwork(ip_col_ptr), no
nz_max, &
         nonz, max_row_len, ip, int_max, debug)

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

      !  sorting csr ...
      call sort_csr (n_dof, nonz, max_row_len, a_iwork(ip_row), &
         a_iwork(ip_col_ptr), &
      !a_iwork(ip_work_sort),
         a_iwork(ip_work))
      !a_iwork(ip_work_sort2)


      if (debug .eq. 1) then
         write(ui_out,*) "py_calc_modes_AC: nonz_max = ", nonz_max
         write(ui_out,*) "py_calc_modes_AC: nonz = ", nonz
         write(ui_out,*) "py_calc_modes_AC: 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(8) supervector is too small", &
            "integer(8) super-vec: int_max  = ", int_max, &
            "integer(8) super-vec: int_used = ", int_used
         errco = -4
         call nberr%set(errco, emsg); RET_ON_NBERR(nberr)

      endif

      jp_rhs = jp_x + 2*n_msh_pts
      !  jp_rhs will also be used (in gmsh_post_process) to store a solution
```



```
      jp_mat2 = jp_rhs + max(n_dof, 3*n_msh_pts)
      jp_vect1 = jp_mat2 + nonz
      jp_vect2 = jp_vect1 + n_dof
      jp_workd = jp_vect2 + n_dof
      jp_resid = jp_workd + 3*n_dof
      jp_eigenum_modes_tmp = jp_resid+3*P2_NODES_PER_EL*n_modes*n_msh_el
      !  Eigenvectors
      jp_vschur = jp_eigenum_modes_tmp + n_modes + 1
      jp_eigen_pol = jp_vschur + n_dof*nvect
      jp_trav = jp_eigen_pol + n_modes*4

      ltrav = 3*nvect*(nvect+2)
      jp_vp = jp_trav + ltrav
      cmplx_used = jp_vp + n_dof*n_modes

      if (cmplx_max .lt. cmplx_used)  then
         write(emsg,*)"The size of the complex supervector is too small", &
            "complex super-vec: cmplx_max  = ", cmplx_max, &
            "complex super-vec: cmplx_used = ", cmplx_used
         errco = -5
         call nberr%set(errco, emsg); RET_ON_NBERR(nberr)
      endif



      kp_rhs_re = 1
      kp_rhs_im = kp_rhs_re + n_dof
      kp_lhs_re = kp_rhs_im + n_dof
      kp_lhs_im = kp_lhs_re + n_dof
      kp_mat1_re = kp_lhs_im + n_dof
      kp_mat1_im = kp_mat1_re + nonz
      real_used = kp_mat1_im + nonz

      if (real_max .lt. real_used) then
         write(ui_out,*)
         write(ui_out,*) "The size of the real supervector is too small"
         write(ui_out,*) "2*nonz  = ", 2*nonz
         write(ui_out,*) "real super-vec: real_max  = ", real_max
         write(ui_out,*) "real super-vec: real_used = ", real_used

         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 = -6
         call nberr%set(errco, emsg); RET_ON_NBERR(nberr)
         return
      endif

      !
      !###############################################
      !
      !  ---------------------------------------------------------------
      !  convert from 1-based to 0-based
      !  ---------------------------------------------------------------
      !
      do 60 j = 1, n_dof+1
         a_iwork(j+ip_col_ptr-1) = a_iwork(j+ip_col_ptr-1) - 1
60    continue
      do 70 j = 1, nonz
         a_iwork(j+ip_row-1) = a_iwork(j+ip_row-1) - 1
```

```
70     continue
       !
       !
       !  The CSC iindexing, i.e., ip_col_ptr, is 1-based
       !  (but valpr.f will change the CSC iindexing to 0-based iindexing)
       i_base = 0

       !####################  End FEM PRE-PROCESSING  ########################
       !
       write(ui_out,*)
       write(ui_out,*) "-----------------------------------------------"
       !  write(ui_out,*) " AC FEM, k_AC : ", real(q_ac), " 1/m"
       !  write(ui_out,*) "-----------------------------------------------"
       !  write(ui_out,*)

       !  if (debug .eq. 1) then
       !  write(ui_out,*) "py_calc_modes_AC: call to asmbly"
       !  endif
       write(ui_out,*) "AC FEM: "

       !  Assemble the coefficient matrix K and M of the finite element equations

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

       call asmbly_AC (i_base, n_msh_el, n_msh_pts, n_dof, P2_NODES_PER_EL, &
          shift_omsq, q_ac, n_elt_mats, rho, c_tensor, &
          elnd_to_mshpt, v_el_material, a_iwork(ip_eq), &
          v_nd_xy, nonz, a_iwork(ip_row), a_iwork(ip_col_ptr), &
          c_dwork(kp_mat1_re), c_dwork(kp_mat1_im), b_zwork(jp_mat2), a_iwork(ip_
work), &
          symmetry_flag, debug)


       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)') '   ', n_dof, ' linear equations'
       write(ui_out,'(A,i9,A)') '   ', nonz, ' nonzero elements'
       write(ui_out,'(A,f9.3,A)') '   ', nonz/(1.d0*n_dof*n_dof)*100.d0, ' % sparsity'
       write(ui_out,'(A,i9,A)') '   ', n_dof*(nvect+6)*16/2**20, ' MB est. working memory
,

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


       write(ui_out,'(/,A)') "  – solving linear system: "
       write(ui_out,'(/,A)') "    solving eigensystem"
       call clock_spare%reset()

       call valpr_64_AC (i_base, nvect, n_modes, n_dof, itermax, ltrav, &
          tol, nonz, a_iwork(ip_row), a_iwork(ip_col_ptr), c_dwork(kp_mat1_re), &
          c_dwork(kp_mat1_im), b_zwork(jp_mat2), &
          b_zwork(jp_vect1), b_zwork(jp_vect2), b_zwork(jp_workd), b_zwork(jp_res
id), &
          b_zwork(jp_vschur), v_eigs_nu, b_zwork(jp_trav), b_zwork(jp_vp), &
          c_dwork(kp_rhs_re), c_dwork(kp_rhs_im), c_dwork(kp_lhs_re), c_dwork(kp_
lhs_im), n_conv, &
          debug, show_mem_est, errco, emsg)

       call nberr%set(errco, emsg); RET_ON_NBERR(nberr)
```



```
       if (n_conv .ne. n_modes) then
          write(emsg, '(A,I5,I5)') &
             "py_calc_modes_AC: convergence problem " // &
             "in valpr_64: n_conv != n_modes ", n_conv, n_modes
          errco = -7
          call nberr%set(errco, emsg); RET_ON_NBERR(nberr)

       endif

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


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

       !
       do i=1,n_modes
          z_tmp0 = v_eigs_nu(i)
          z_tmp = 1.0d0/z_tmp0+shift_omsq
          z_beta = sqrt(z_tmp) / (2.0d0 * D_PI)
          !  Frequency (z_beta) should always be positive.
          if (dble(z_beta) .lt. 0) z_beta = -z_beta
          v_eigs_nu(i) = z_beta
       enddo
       !
       call z_indexx_AC (n_modes, v_eigs_nu, iindex)
       !
       !  The eigenvectors will be stored in the array femsol_ac
       !  The eigenum_modesues and eigenvectors will be renumbered
       !  using the permutation vector iindex
       if (debug .eq. 1) then
          write(ui_out,*) "py_calc_modes_AC: call to array_sol"
       endif
       call array_sol_AC (n_modes, n_msh_el, n_msh_pts, n_dof, &
          P2_NODES_PER_EL, iindex, elnd_to_mshpt, v_el_material, a_iwork(ip_eq),
v_nd_xy, &
          v_eigs_nu,  b_zwork(jp_eigenum_modes_tmp), poln_fracs, b_zwork(jp_vp),
femsol_ac)

       if (debug .eq. 1) then
          write(ui_out,*) "py_calc_modes_AC: array_sol returns call"
       endif
       !
       if(debug .eq. 1) then
          write(ui_out,*) 'iindex = ', (iindex(i), i=1,n_modes)
       endif
       if(debug .eq. 1) then
          write(ui_out,*)
          !  write(ui_out,*) "lambda, 1/lambda = ", lambda, 1.0d0/lambda
          !  write(ui_out,*) "sqrt(shift_omsq)/(2*D_PI) = ", sqrt(omsq) / (2.0d0
* D_PI)
          do i=1,n_modes
             write(ui_out,"(i4,2(g22.14),2(g18.10))") i, v_eigs_nu(i)
          enddo
       endif

       !C    Save Original solution
       !  if (plot_modes .eq. 1) then
       !  dir_name = "AC_fields"
       !C        call write_sol_AC (n_modes, n_msh_el, P2_NODES_PER_EL, lambda,
```

```
      !C      *       v_eigs_nu, femsol_ac, mesh_file, dir_name)
      !C        call write_param (lambda, n_msh_pts, n_msh_el, bdy_cdn,
      !C     *       n_modes, nvect, itermax, tol, shift_omsq, lx, ly,
      !C     *       mesh_file, n_conv, dir_name)
      !  tchar = "AC_fields/All_plots_png_abs2_eE.geo"
      !  open (unit=34,file=tchar)
      !  do i=1,n_modes
      !  call gmsh_post_process_AC (i, n_modes, n_msh_el,
      !  *         n_msh_pts, P2_NODES_PER_EL, elnd_to_mshpt, v_el_material,
      !  *         v_nd_xy, v_eigs_nu, femsol_ac, b_zwork(jp_rhs), a_iwork(ip_vi
site),
      !  *         gmsh_file_pos, dir_name, dimscale_in_m, debug)
      !  enddo
      !  close (unit=34)
      !  endif
      !C
      !#########################  End Calculations  ############################
      !
      !  call date_and_time ( end_date, end_time )
      !  call cpu_time(time2)
      !
      !  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 date and time   = ", start_date, &
      !  " ; ", start_time
      !  write(26,*) "End date and time     = ", end_date, &
      !  " ; ", end_time
      !  write(26,*) "Total CPU time (sec.) = ",  (time2-time1)
      !  write(26,*)
      !  write(26,*) "q_ac = ", q_ac
      !  write(26,*) "shift_omsq= ", shift_omsq
      !  write(26,*)
      !  write(26,*) "n_msh_pts, n_msh_el, P2_NODES_PER_EL  = ", n_msh_pts, &
      !  n_msh_el, P2_NODES_PER_EL
      !  write(26,*) "n_dof, bdy_cdn = ", n_dof, bdy_cdn
      !  write(26,*) " lat_vecs:  = "
      !  write(26,"(2(f18.10))") lat_vecs
      !  write(26,*) "mesh_file = ", mesh_file
      !  write(26,*) "gmsh_file = ", gmsh_file
      !  write(26,*) "log_file  = ", log_file
      !  close(26)
      !
      !  write(ui_out,*) "   .      .      ."
      !  write(ui_out,*) "   .      .      ."
      !  write(ui_out,*) "   .      . (d=",dimscale_in_m,")"
      !  write(ui_out,*) "  and   we're   done!"
      !  endif

      write(ui_out,'(A,A)') '    ', clock_spare%to_string()
      write(ui_out,*) "-------------------------------------------------"
      write(ui_out,*)
      !
      deallocate(a_iwork, b_zwork, c_dwork, iindex)

   end subroutine calc_ac_modes_impl
```



```
end module calc_ac_impl
```