```
module comm_lowl_mod
  use comm_like_utils
  use comm_proc_utils
  implicit none

! *********************************************************************
!*  comm_lowl_mod -- An F90 module for computing a Gaussian          *
!*                   low-l likelihood by brute force                 *
!*                                                                   *
!*            H. K. Eriksen, E. Gjerløw (University of Oslo)         *
!*                                                                   *
!*                                and                                *
!*                                                                   *
!*                           I. K. Wehus (JPL)                       *
!*                                                                   *
!*  Please cite the following papers when using this code:           *
!*                                                                   *
!*  - Gjerløw et al., 2014, ApJS, in preparation                     *
!*                                                                   *
!*  History:                                                         *
!*      January 10th, 2014 -- First fully functional version         *
!*                                                                   *
! *********************************************************************

! ======================================================================
! User routines:
!
!  ** subroutine comm_lowl_initialize_object(paramfile, handle)
!
!     Input parameters:
!       paramfile   (character*) :: Low-l parameter file
!
!     Output parameters:
!       handle (i4b)  :: this code supports multiple objects (optional; default=1),
!                        and the handle identifies specific data sets
!
!
!  ** function comm_lowl_compute_lnL(cls, ierr, handle)
!
!     Input parameters:
!       cls    (dp)  :: array containing at least cls(2:lmax) in units of l(l+1)/2pi
!       ierr   (i4b) :: error flag
!       handle  (i4b) :: handle selecting which data set to use (optional; default = 1)
!
!
!  ** subroutine comm_lowl_deallocate_object(handle)
!
!     Input parameters:
!       handle  (i4b) :: handle of object to deallocate (optional; default=remove all)
!
! ======================================================================

  integer(i4b), parameter :: MAX_N_LOWL = 5

  type comm_lowl_data
    logical(lgt) :: initialized=.false.
    integer(i4b) :: n, n_h, nmaps, lowl, lhigh, lmax
    real(dp)     :: loglike_weight
    real(dp), allocatable, dimension(:)     :: d
    real(dp), allocatable, dimension(:,:)   :: cl_fid
    real(dp), allocatable, dimension(:,:)   :: N_cov, P_harm
    real(dp), allocatable, dimension(:,:,:) :: W
    real(dp), allocatable, dimension(:,:)   :: beam
  end type comm_lowl_data

  type(comm_lowl_data), dimension(MAX_N_LOWL) :: comm_lowl


contains

  ! Initialization routines
  subroutine comm_lowl_initialize_object(paramfile, handle)
    implicit none

    character(len=*),   intent(in)  :: paramfile
    integer(i4b),       intent(out), optional :: handle

    integer(i4b)       :: firstchain, lastchain, firstsample, lastsample, thinstep
    integer(i4b)       :: b, i, j, k, l, m, n, q, c1, c2, f, ind, d, n_h
    integer(i4b)       :: unit, numsamples, numchains, lmax_chain, id, bin(2), n_p, n_g, nmode
    integer(i4b)       :: col, p, nmaps, nspec, nsamp, lmax_cl, lmin_bin, lmax_bin, ncomp
    logical(lgt)       :: pattern(3,3), polarization, exist
    real(dp)           :: lnL, t1, t2
    character(len=512) :: line, s1, s2, datafile
    character(len=128) :: sigmafile, clfile
    real(dp),     allocatable, dimension(:,:)     :: cls
    integer(i4b), allocatable, dimension(:)       :: i2p
    real(dp),     allocatable, dimension(:,:,:,:) :: sigma
    real(dp),     allocatable, dimension(:,:,:)   :: sigma_1D
    real(dp),     allocatable, dimension(:,:)     :: S, sqrtS_P

    id = 1
    unit = comm_getlun()
    if (present(handle)) then
       do while (comm_lowl(id)%initialized .and. id < MAX_N_LOWL)
          id = id+1
       end do
       handle = id
    end if
    if (id < 1 .or. id > MAX_N_LOWL) stop 'Error -- planck_br_mod: lowl id out of range'

    ! Initialize all distributions
    comm_lowl(id)%initialized = .true.
    call comm_get_parameter(paramfile, 'DATAFILE',           par_string=datafile)
    call comm_get_parameter(paramfile, 'FIDUCIAL_CL_FILE',   par_string=clfile)
    call comm_get_parameter(paramfile, 'LOGLIKE_WEIGHT',     par_dp=comm_lowl(id)%loglike_weight)
    call comm_get_parameter(paramfile, 'LMIN',               par_int=comm_lowl(id)%lowl)
    call comm_get_parameter(paramfile, 'LMAX',               par_int=comm_lowl(id)%lhigh)

    ! Read data file
    inquire(file=trim(datafile), exist=exist)
    if (.not. exist) then
       write(*,*) 'Error -- low-l datafile = ', trim(datafile), ' does not exist'
       stop
    end if

    open(unit,file=trim(datafile),form='unformatted')
    read(unit) comm_lowl(id)%lmax, nmaps
    read(unit) n, n_h, nmode
    ncomp          = (comm_lowl(id)%lmax+1)**2
    comm_lowl(id)%n = nmode
    comm_lowl(id)%nmaps = nmaps
    comm_lowl(id)%n_h   = n_h
    allocate(comm_lowl(id)%w(0:comm_lowl(id)%lmax,nmaps,nmaps))
    allocate(comm_lowl(id)%beam(0:comm_lowl(id)%lmax,nmaps))
    allocate(comm_lowl(id)%d(nmode), comm_lowl(id)%N_cov(nmode,nmode))
    allocate(comm_lowl(id)%P_harm(n_h,nmode))
    read(unit) comm_lowl(id)%d
    read(unit) comm_lowl(id)%N_cov
    read(unit) comm_lowl(id)%W
    read(unit) comm_lowl(id)%beam
    read(unit) comm_lowl(id)%P_harm
    close(unit)

    ! Read in fiducial spectrum, to be conditioned upon outside range of interest
    call read_fiducial_spectrum(clfile, comm_lowl(id)%cl_fid)

    ! Add basis vectors with fixed C_l's directly into noise covariance
    allocate(sqrtS_P(n_h,nmode), S(nmaps,nmaps))
    sqrtS_P = 0.d0
    ind     = 1
    do l = 0, comm_lowl(id)%lmax
       if (l >= comm_lowl(id)%llow .and. l <= comm_lowl(id)%lhigh) then
          ind = ind + 2*l+1
          cycle
       end if
       if (l < 2) then
          ! Allow a large value for the CMB monopole/dipole (ie., "marginalize")
          S     = 0.d0
          S(1,1) = 1.d6
       else
          call cl2s(comm_lowl(id)%cl_fid(l,:), S) ! Fix to fiducial
       end if
       if (l >= 2) S = S * comm_lowl(id)%w(l,:,:) / (l*(l+1)/(2.d0*pi)) ! w = (b_l*p_l)^2
       call cholesky_decompose_with_mask_dp(S)
       do m = -l, l
          call dgemm('N', 'N', nmaps, nmode, nmaps, 1.d0, S, nmaps, &
               & comm_lowl(id)%P_harm(ind:n_h:ncomp,:), &
               & nmaps, 0.d0, sqrtS_P(ind:n_h:ncomp,:), nmaps)
          ind = ind+1
       end do
    end do
    call dsyrk('L','T',nmode,n_h,1.d0,sqrtS_P,n_h,1.d0,comm_lowl(id)%N_cov,nmode)
    deallocate(sqrtS_P, S)

  end subroutine comm_lowl_initialize_object


  subroutine comm_lowl_deallocate_object(handle)
    implicit none

    integer(i4b), optional :: handle

    integer(i4b) :: i, j, k, id, id_min, id_max

    id_min = 1; id_max = MAX_N_LOWL;
    if (present(handle)) then
       id_min = handle
       id_max = handle
    end if

    do id = id_min, id_max
       if (comm_lowl(id)%initialized) then
          deallocate(comm_lowl(id)%d, comm_lowl(id)%cl_fid, comm_lowl(id)%N_cov)
          deallocate(comm_lowl(id)%P_harm, comm_lowl(id)%W)
       end if
    end do

  end subroutine comm_lowl_deallocate_object


  ! Base computation routine
  function comm_lowl_compute_lnL(cls, sqrt_S, chisq, red_chisq, handle, ierr)
    implicit none

    real(dp),     dimension(0:,1:), intent(in),  optional :: cls
    integer(i4b),                   intent(out), optional :: ierr
    integer(i4b),                   intent(in),  optional :: handle
    real(dp),     dimension(1:,1:), intent(in),  optional :: sqrt_S
    real(dp),                       intent(out), optional :: chisq, red_chisq
    real(dp)                                              :: comm_lowl_compute_lnL

    integer(i4b) :: i, j, l, m, n, n_h, id, nmaps, k, ncomp, stat, ind, lmax, ind1, ind2
    integer(i4b) :: ind_min, ind_max, nmode
    logical(lgt) :: posdef
    real(dp)     :: chi2, logdet, t1, t2
    real(dp), allocatable, dimension(:)   :: W
    real(dp), allocatable, dimension(:,:) :: S, C, sqrtS_P, invC_d, V, P_sub

    if (present(ierr)) ierr = 0
    id = 1; if (present(handle)) id = handle

    ! Check that likelihood structure is initialized
    if (.not. comm_lowl(id)%initialized) then
       write(*,*) 'Error -- comm_lowl_mod: Requested handle ', id, ' is not initialized'
       stop
    end if

    n_h     = comm_lowl(id)%n_h
    n       = comm_lowl(id)%n
    nmaps   = comm_lowl(id)%nmaps
    lmax    = comm_lowl(id)%lmax
    ncomp   = comm_lowl(id)%n_h / comm_lowl(id)%nmaps
    ind_min = comm_lowl(id)%llow**2+1
    ind_max = (comm_lowl(id)%lhigh+1)**2
    nmode   = (ind_max-ind_min+1)*nmaps

    allocate(C(n,n), sqrtS_P(nmode,n), S(nmaps,nmaps), invC_d(n,1))
    if (present(sqrt_S)) then
       ! Use user-supplied harmonic space covariance

       ! Extract vectors to be multiplied with sqrt_S, and multiply with beam
       allocate(P_sub(nmode,n))
       ind2 = 1
       do i = 1, nmaps
          ind1 = (i-1)*ncomp + comm_lowl(id)%llow**2 + 1
          do l = comm_lowl(id)%llow, comm_lowl(id)%lhigh
             do m = -l, l
                P_sub(ind2,:) = comm_lowl(id)%beam(l,i)*comm_lowl(id)%P_harm(ind1,:)
                ind2 = ind2+1
                ind1 = ind1+1
             end do
          end do
       end do

       ! Multiply basis vectors with sqrt(S), ie., Cholesky factor of S
       call dgemm('T', 'N', nmode, n, nmode, 1.d0, sqrt_S, nmode, &
            & P_sub, nmode, 0.d0, sqrtS_P, nmode)

       deallocate(P_sub)

    else
       ! Use user-supplied power spectrum for S

       ! Check that spectrum is positive definite
       if (.not. comm_cls_posdef(cls(2:lmax,:))) then
          comm_lowl_compute_lnL = -1.d30
          return
       end if

       ! Compute signal covariance matrix
       sqrtS_P         = 0.d0
       ind             = comm_lowl(id)%llow**2+1
       ind2            = 1
       do l = comm_lowl(id)%llow, comm_lowl(id)%lhigh
          call cl2x(cls(l,:), S)
          S = S * comm_lowl(id)%w(l,:,:) / (l*(l+1)/(2.d0*pi)) ! w = (b_l*p_l)^2
          call cholesky_decompose_with_mask_dp(S)
          do m = -l, l
             call dgemm('N', 'N', nmaps, n, nmaps, 1.d0, S, nmaps, &
                  & comm_lowl(id)%P_harm(ind:n_h:ncomp,:), &
                  & nmaps, 0.d0, sqrtS_P(ind2:ind2+nmaps-1,:), nmaps)
             ind = ind  + 1
             ind2 = ind2 + nmaps
          end do
       end do
    end if
    call dsyrk('L','T',n,nmode,1.d0,sqrtS_P,nmode,0.d0,C,n)

    ! Add noise
    C = C + comm_lowl(id)%N_cov

    ! Cholesky decompose matrix
    call dpotrf('L', n, C, n, stat)
    if (stat /= 0) then
       if (present(ierr)) ierr = ierr + 1
       deallocate(C, sqrtS_P, S, invC_d)
       return
    end if

    ! Compute log-determinant
    logdet = 0.d0
    do i = 1, n
       logdet = logdet + 2.d0 * log(C(i,i))
    end do

    ! Compute chi-square term
    invC_d(:,1) = comm_lowl(id)%d
    call dpotrs('L', n, 1, C, n, invC_d, n, stat)
    chi2 = sum(comm_lowl(id)%d*invC_d(:,1))
    if (present(chisq))     chisq     = chi2
    if (present(red_chisq)) red_chisq = chi2 / n

    ! Return log-like value
    if (stat == 0) then
       comm_lowl_compute_lnL = -0.5d0 * (chi2 + logdet)
    else
       comm_lowl_compute_lnL = -1.d30
    end if

    deallocate(C, sqrtS_P, S, invC_d)

  end function comm_lowl_compute_lnL

end module comm_lowl_mod
```