```
!----------------------------------------------------------------------------
! This file is part of UCLALES.
!
! UCLALES is free software; you can redistribute it and/or modify
! it under the terms of the GNU General Public License as published by
! the Free Software Foundation; either version 3 of the License, or
! (at your option) any later version.
!
! UCLALES is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
! GNU General Public License for more details.
!
! You should have received a copy of the GNU General Public License
! along with this program.  If not, see <http://www.gnu.org/licenses/>.
!
! Copyright 1999-2007, Bjorn B. Stevens, Dep't Atmos and Ocean Sci, UCLA
!----------------------------------------------------------------------------
!
! PRSS: Pressure Solver:  Solves the anelastic or bousinessq system
! for the pressure using a fractional step method, which is implemented
! using fft's and a tri-diagonal solver in the vertical
!
module prss

  use defs, only: pi
  implicit none

contains
!
!----------------------------------------------------------------------
! subroutine poisson: called by timesteping driver to invert the
! poisson equation for pressure and apply the velocity tendencies.
!
  subroutine poisson

    use grid, only : nxp, nyp, nzp, dtlt, dxi, dyi, dzm, dzt, a_up,       &
         a_uc, a_ut, a_vp, a_vc, a_vt, a_wp, a_wc, a_wt, a_press, a_pexnr,&
         th00, dn0, wsavex, wsavey
    use stat, only : fill_scalar, sflg
    use util, only : ae1mm

    complex, allocatable     :: s1(:,:,:)
    real    :: mxdiv, awpbar(nzp)
    integer :: ix,iy

    ix=max(1,nxp-4)
    iy=max(1,nyp-4)

    allocate (s1(ix,iy,nzp))
    s1=0.0
    !
    ! -------
    ! Do first step of asselin filter, first saving corrlations of
    ! tendencies for TKE budget on statistics timestep
    !
    call asselin(1)

    call apl_tnd(nzp,nxp,nyp,a_up,a_vp,a_wp,a_ut,a_vt,a_wt,dtlt)
    !
    ! ------
    ! Pressure Solve
    !
    call poiss(nzp,nxp,nyp,ix,iy,a_up,a_vp,a_wp,a_pexnr,a_press,dn0,th00,dzt &
         ,dzm,dxi,dyi,dtlt,s1,wsavex,wsavey)

    call ae1mm(nzp,nxp,nyp,a_wp,awpbar)
    !
    ! -------
    ! Do second step of asselin filter, first saving corrlations of
    ! tendencies for TKE budget on statistics timestep, note that the
    ! old centered velocity resides in up,vp,wp after the second step
    ! of the asselin filter, hence the pressure correlation terms in the
    ! tke budget include the effects of time filtering
    !
    call asselin(2)

    call velocity_bcs

    if (sflg) then
       call get_diverg(nzp,nxp,nyp,ix,iy,s1,a_up,a_vp,a_wp,dn0,dzt,dxi,dyi,  &
            dtlt,mxdiv)
       call fill_scalar(2,mxdiv)
       call prs_cor(nzp,nxp,nyp,a_pexnr,a_up,a_vp,a_wp,dzm,dxi,dyi,th00)
       call chk_tsplt(nzp,nxp,nyp,a_up,a_vp,a_wp,a_uc,a_vc,a_wc)
    end if
    deallocate (s1)

  end subroutine poisson

  !
  ! ----------------------------------------------------------------------
  ! subroutine apl_tnd: applies tendencies to velocity field
  !
  subroutine apl_tnd(n1,n2,n3,u,v,w,ut,vt,wt,dtlt)

    use util, only : velset

    integer :: n1,n2,n3,i,k,j
    real :: u(n1,n2,n3),v(n1,n2,n3),w(n1,n2,n3)
    real :: ut(n1,n2,n3),vt(n1,n2,n3),wt(n1,n2,n3),dtlt,dt

    dt=dtlt*2.

    do j=1,n3
       do i=1,n2
          do k=2,n1-1
             w(k,i,j)=w(k,i,j)+wt(k,i,j)*dt
             u(k,i,j)=u(k,i,j)+ut(k,i,j)*dt
             v(k,i,j)=v(k,i,j)+vt(k,i,j)*dt
          end do
       end do
    end do

    call velset(n1,n2,n3,u,v,w)

  end subroutine apl_tnd
  !
  ! ----------------------------------------------------------------------
  ! subroutine poiss: called each timestep to evaluate the pressure
  ! in accordance with the anelastic continuity equation, and then apply
  ! the pressure to the velocity terms for three dimensional flow,
  ! cyclic in x and y.  pp and pc are used as scratch arrays in the
  ! call to trdprs.  pp is filled with its diagnostic value in fll_prs
  !
  subroutine poiss(n1,n2,n3,ix,iy,u,v,w,pp,pc,dn0,th00,dzt,dzm,dx,dy,  &
       dtlt,s1,wsvx,wsvy)

    use util, only  : velset, get_fft_twodim
```

```
    integer :: n1,n2,n3,ix,iy
    real    :: pp(n1,n2,n3),pc(n1,n2,n3),dmy
    real    :: u(n1,n2,n3),v(n1,n2,n3),w(n1,n2,n3)
    real    :: wsvx(1:),wsvy(1:),dn0(n1),dzt(n1),dzm(n1),dx,dy,dtlt,th00
    complex :: s1(ix,iy,n1)

    call get_diverg(n1,n2,n3,ix,iy,s1,u,v,w,dn0,dzt,dx,dy,dtlt,dmy)

    call get_fft_twodim(ix,iy,n1,s1,wsvx,wsvy,-1)

    call trdprs(n1,ix,iy,s1,dn0,dzt,dzm,dx,dy)

    call get_fft_twodim(ix,iy,n1,s1,wsvx,wsvy,+1)

    call fll_prs(n1,n2,n3,ix,iy,pp,s1)
    call prs_grd(n1,n2,n3,pp,u,v,w,dzm,dx,dy,dtlt)

    call velset(n1,n2,n3,u,v,w)

    pp(:,:,:) = pp(:,:,:)/th00
    pc(:,:,:) = pp(:,:,:)

  end subroutine poiss
  !
  ! ------------------------------------------------------------------
  ! subroutine get_diverg: gets velocity divergence and puts it into
  ! a complex value array for use in pressure calculation
  !
  subroutine get_diverg(n1,n2,n3,ix,iy,s1,u,v,w,dn0,dz,dx,dy,dt,mxdiv)

    integer, intent (in)  :: n1,n2,n3,ix,iy
    real, intent (in)     :: dz(n1),dn0(n1),dx,dy,dt
    real, intent (in)     :: u(n1,n2,n3),v(n1,n2,n3),w(n1,n2,n3)
    real, intent (out)    :: mxdiv
    complex, intent (out) :: s1(ix,iy,n1)

    integer :: k,i,j,l,m
    real    :: xf,yf,zf,wf1,wf2,dtv,dti

    s1(:,:,:) = (0.0,0.0)
    dtv=dt*2.
    dti=1./dtv
    m=0
    do j=3,n3-2
       m=m+1
       l=0
       do i=3,n2-2
          l=l+1
          do k=2,n1-1
             wf1=0.5*(dn0(k+1)+dn0(k))
             wf2=0.5*(dn0(k)+dn0(k-1))
             if (k == 2 )   wf2=0.
             if (k == n1-1) wf1=0.
             xf=dn0(k)*dx*dti
             yf=dn0(k)*dy*dti
             zf=dti*dz(k)
             s1(l,m,k)=(wf1*w(k,i,j)-wf2*w(k-1,i,j))*zf                   &
                  +(v(k,i,j)-v(k,i,j-1))*yf+(u(k,i,j)-u(k,i-1,j))*xf
          enddo
       enddo
    enddo
!
! save mxdiv to a statistics array, no reduction necessary as this is done
! in post processing
!
    mxdiv = maxval(real(s1))

  end subroutine get_diverg
  !
  !----------------------------------------------------------------------
  ! subroutine fll_prs: writes the pressure to the appropriate array
  !
  subroutine fll_prs(n1,n2,n3,ix,iy,pp,s1)

    use mpi_interface, only : cyclics,cyclicc

    integer :: n1,n2,n3,ix,iy,k,i,j,l,m,req(16)
    real :: pp(n1,n2,n3)
    complex :: s1(ix,iy,n1)

    pp(:,:,:)=0.0
    do k=2,n1-1
       l=0
       do i=3,n2-2
          l=l+1
          m=0
          do j=3,n3-2
             m=m+1
             pp(k,i,j) =real(s1(l,m,k))
          enddo
       enddo
    enddo
    call cyclics(n1,n2,n3,pp,req)
    call cyclicc(n1,n2,n3,pp,req)

  end subroutine fll_prs
  !
  !---------------------------------------------------------------------
  ! TRDPRS: solves for the wave number (l,m) component of
  ! pressure in a vertical column using a tri-diagonal solver.
  !
  subroutine trdprs(n1,ix,iy,s1,dn0,dzt,dzm,dx,dy)

    use mpi_interface, only : yoffset, nypg, xoffset, wrxid, wryid, nxpg
    use util, only          : tridiff

    integer, intent (in)    :: n1,ix,iy
    real, intent (in)       :: dn0(n1),dzt(n1),dzm(n1),dx,dy
    complex, intent (inout) :: s1(ix,iy,n1)

    real    :: ak(ix,n1),dk(ix,n1),bk(ix,n1),ck(ix,n1)
    real    :: xk(ix,n1),yk(ix,n1),wv(ix,iy)

    integer :: k,l,m
    real    :: fctl,fctm,xl,xm,af,cf
    fctl=2.*pi/float(nxpg-4)
    fctm=2.*pi/float(nypg-4)

    do l=1,ix
          if(l+xoffset(wrxid) .le. (nxpg-4)/2+1) then
            xl=float(l-1+xoffset(wrxid))
          else
             xl=float(l-(nxpg-4)-1+xoffset(wrxid))
          endif

       do m=1,iy
          if(m+yoffset(wryid) .le. (nypg-4)/2+1) then
             xm=float(m-1+yoffset(wryid))
          else
             xm=float(m-(nypg-4)-1+yoffset(wryid))
```

```
        endif
        wv(l,m)=2.*((cos(fctl*xl)-1.)*dx*dx + (cos(fctm*xm)-1.)*dy*dy)
     enddo
  enddo


  if(wrxid.eq.0 .and. wryid .eq.0 ) then
     wv(1,1)=0.
  endif
  !
  ! configure vectors for tri-diagonal solver
  !
  do m=1,iy
     do k=2,n1-1
        af=(dn0(k)+dn0(k-1))*.5
        cf=(dn0(k+1)+dn0(k))*.5
        if (k == 2   )af=0.
        if (k == n1-1)cf=0.
        do l=1,ix
           ak(l,k)=dzt(k)*dzm(k-1)*af
           bk(l,k)=s1(l,m,k)
           ck(l,k)=dzt(k)*dzm(k)*cf
           dk(l,k)=dn0(k)*wv(l,m)-(ak(l,k)+ck(l,k))
        enddo
     enddo
     !
     ! solve for fourier components, x_k, given a tri-diagonal matrix of the
     ! form a_k x_k-1 + d_k x_k + c_k x_k+1 = b_k.  y_k is a scratch array.
     !
     call tridiff(ix,n1-1,ix,ak,dk,ck,bk,xk,yk)

     do k=2,n1-1
        do l=1,ix
           if (m+yoffset(wryid)+l+xoffset(wrxid)>2) bk(l,k)=aimag(s1(l,m,k))
           if (m+yoffset(wryid)+l+xoffset(wrxid)>2) s1(l,m,k)=xk(l,k)
        enddo
     enddo

     call tridiff(ix,n1-1,ix,ak,dk,ck,bk,xk,yk)

     do k=2,n1-1
        do l=1,ix
           if (m+yoffset(wryid)+l+xoffset(wrxid) > 2)                &
                s1(l,m,k)=cmplx(real(s1(l,m,k)),xk(l,k))
        enddo
     enddo

  enddo

end subroutine trdprs
!
!----------------------------------------------------------------------
! Subroutine Prs_grd: apply the pressure gradient term
!
subroutine prs_grd(n1,n2,n3,p,u,v,w,dz,dx,dy,dtlt)

  integer, intent (in) :: n1,n2,n3
  real, intent (in)    :: p(n1,n2,n3),dz(n1),dx,dy,dtlt
  real, intent (inout) :: u(n1,n2,n3),v(n1,n2,n3),w(n1,n2,n3)

  integer :: i,j,k

  do j=1,n3-1
     do i=1,n2-1
        do k=2,n1-1
           if(k /= n1-1)w(k,i,j)=w(k,i,j)-dz(k)*2.*dtlt*(p(k+1,i,j)-p(k,i,j))
           u(k,i,j)=u(k,i,j)-dx*2.*dtlt*(p(k,i+1,j)-p(k,i,j))
           v(k,i,j)=v(k,i,j)-dy*2.*dtlt*(p(k,i,j+1)-p(k,i,j))
        enddo
     enddo
  enddo

end subroutine prs_grd
!
!----------------------------------------------------------------------
! Subroutine Prs_cor: correlate the pressure tendency with velocity
! field for TKE budget
!
subroutine prs_cor(n1,n2,n3,p,u,v,w,dz,dx,dy,th00)

  use stat, only : updtst
  use util, only : get_cor

  integer, intent (in) :: n1,n2,n3
  real, intent (in)    :: p(n1,n2,n3),dz(n1),dx,dy,th00
  real, intent (in)    :: u(n1,n2,n3),v(n1,n2,n3),w(n1,n2,n3)

  real, dimension (n2,n3) :: pgx, pgy, pgz, ufld, vfld, wfld
  real    :: fx, fy, fz, v1da(n1), v1db(n1), v1dc(n1)
  integer :: i,j,k,ip1,jp1

  v1da = 0.0
  v1db = 0.0
  v1dc = 0.0

  do k=2,n1-1
     fx=dx*th00
     fy=dy*th00
     fz=dz(k)*th00
     do j=1,n3
        do i=1,n2
           ip1 = min(n2,i+1)
           jp1 = min(n3,j+1)
           pgx(i,j) = -fx*(p(k,ip1,j)-p(k,i,j))
           pgy(i,j) = -fy*(p(k,i,jp1)-p(k,i,j))
           pgz(i,j) = -fz*(p(k+1,i,j)-p(k,i,j))
           ufld(i,j) = u(k,i,j)
           vfld(i,j) = v(k,i,j)
           wfld(i,j) = w(k,i,j)
        end do
     end do
     v1da(k) = get_cor(1,n2,n3,1,ufld,pgx)
     v1db(k) = get_cor(1,n2,n3,1,vfld,pgy)
     v1dc(k) = get_cor(1,n2,n3,1,wfld,pgz)
  enddo
  call updtst(n1,'prs',1,v1da,1)
  call updtst(n1,'prs',2,v1db,1)
  call updtst(n1,'prs',3,v1dc,1)

end subroutine prs_cor
!
! ----------------------------------------------------------------------
! subroutine chk_tsplt
!
subroutine chk_tsplt(n1,n2,n3,up,vp,wp,uc,vc,wc)

  integer, intent (in) :: n1,n2,n3
  real, intent (in)    :: up(n1,n2,n3),vp(n1,n2,n3),wp(n1,n2,n3)
  real, intent (in)    :: uc(n1,n2,n3),vc(n1,n2,n3),wc(n1,n2,n3)
```

```
  real :: wmx,umx,vmx

  wmx = maxval((wp-wc)/(wp+wc+1.e-5))
  umx = maxval((up-uc)/(up+uc+1.e-5))
  vmx = maxval((vp-vc)/(vp+vc+1.e-5))

end subroutine chk_tsplt
!
!----------------------------------------------------------------------
! Subroutine Asselin:  Applies the asselin filter in two stages
! depending on the value of iac
!
subroutine asselin(iac)

  use grid, only : a_up,a_vp,a_wp,a_uc,a_vc,a_wc,a_scr1, nxyzp, runtype

  integer :: iac
  integer, save ::ncall=0

  if (runtype == 'HISTORY') ncall=1

  call predict(nxyzp,a_uc,a_up,a_scr1,iac,ncall)
  call predict(nxyzp,a_vc,a_vp,a_scr1,iac,ncall)
  call predict(nxyzp,a_wc,a_wp,a_scr1,iac,ncall)
  if (iac == 2) ncall=ncall+1

end subroutine asselin
!
!----------------------------------------------------------------------
! Subroutine predict:  This subroutine advances the leapfrog terms
! in two stages.  It applies the filter equation:
!
!        a(n) = a(n) + eps * (a(n-1) - 2*a(n) + a(n+1))
!
! the first stage of the filter applies all but the a(n+2) term.
! the second stage renames the variables and applies this term, i.e.,
! a(n+1) -> a(n), a(n+2) -> a(n+1).  Note that for iac=2 ap=a(n+2)
! because the tendencies have been updated in pressure solver.  Durran,
! in his text cites values of eps of 0.2 for convective cloud models,
! we seem to get by with eps=0.1, perhaps because of the coupling
! provided by the staggered forward step.
!
subroutine predict(npts,ac,ap,af,iac,iflag)

  integer :: m,npts,iac,iflag
  real :: ac(npts),ap(npts),af(npts),epsu
  real, parameter :: eps=0.1

  epsu=eps
  if (iflag == 0) epsu=0.5
  if (iac == 1) then
     do m=1,npts
        ac(m)=ac(m)+epsu*(ap(m)-2.*ac(m))
     enddo
  else if (iac == 2) then
     do m=1,npts
        af(m)=ap(m)
        ap(m)=ac(m)+epsu*af(m)
        ac(m)=af(m)
     enddo
  endif

end subroutine predict
!
!----------------------------------------------------------------------
  ! Subroutine Velocity_bcs: Applies boundary conditions on veolicities
  !
  subroutine velocity_bcs

    use grid, only : a_up,a_vp,a_wp,a_uc,a_vc,a_wc,a_pexnr,a_press,           &
         nxp, nyp, nzp, dzm
    use util, only : velset, sclrset

    call velset(nzp,nxp,nyp,a_up,a_vp,a_wp)
    call velset(nzp,nxp,nyp,a_uc,a_vc,a_wc)
    call sclrset('grad',nzp,nxp,nyp,a_pexnr,dzm)
    call sclrset('grad',nzp,nxp,nyp,a_press,dzm)

  end subroutine velocity_bcs

end module prss
```