```fortran
c
c--------------------------------------------------------------------------
c
      subroutine esffrc(dum,grav0,grav1)
c
c     Ensemble local Source Function (ESF) method for radiation force.
c
c     Version with "Piecewise Linear Source Function" (PLSF)
c
c     Operation controlled by ifrc switch as follows:
c
c     ifrc.le.0 =>  grad = gcak
c         .eq.1 =>  grad = gabs
c         .eq.2 =>  grad = gabs,       gscat=gssf
c         .eq.3 =>  grad = gabs+gscat, gscat=gssf
c         .eq.4 =>  grad = gabs+gscat, gscat=gesf w/ S(rp)=So(r)=const (~SSF har
d way)
c         .eq.5 =>  grad = gabs+gscat, gscat=gesf w/ S(rp)=So(rp)=Smooth VARIABL
E source funct
c         .eq.6 =>  grad = gabs+gscat, gscat=gesf w/ S(rp)=So(rp)*betac(rp)/beta
(rp)+tauc=0.
c         .ge.7 =>  grad = gabs+gscat, gscat=gesf w/ S(rp)=So(rp)*betac(rp)/beta
(rp)+tauc.ne.0.
c
c     Revision history:
c    11/24/97: made irssfmax max grid point for S=So in EISF model
c    11/24/97: set ifrc=7 for tauc.ne.0., retained ifrc=6 w/ tauc=0.
c     9/10/96: add tauc to eisf
c      2/7/95: tests of varying LBC vth (fvthlbc)
c     10/1/94:  initial simulations using PLSF
c     9/25/94:  initial simulations using PCSF
c     9/19/94:  adapted from Smooth Source Function method routine ssffrc.
c     4/20/94:  corrected to take account Feldmeier erratum treatment of vth(T(r
))
c     7/15/91:  limit frequency integration to local CMF frequency interval
c
c     Method:
c     Calculate ensemble absorption line RADiation FORce at nr radial grid pts,
c     by integrating absorption in z along nray rays with impact parameters p,
c     where 0 < p < R*, and z = sqrt(r**2-p**2).
c     One-sided version of RADFRC for staggered mesh.
c     Assumes linear variation of velocity and density between zone INTERFACES;
c     Changes in optical depth deta are then at ZONE-CENTERS, while eta,
c     the optical depth itself, is at zone INTERFACES.
c     The interface value the radiative acceleration is then computed from
c     grad = SUM(iray) SUM(ix) phi(x-vntf/vth)/eta**alpha.
c     Note boundary condition on radiation force is at r=max(rstar,r0).
c     Also,
c     if (ifrc.gt.1), also computes SSF diffuse force,
c
c     CMF frequency integrations limited to ixc+-nxcb2, where
c        ixc(ir)=ifix((v(r)/vth-x(1))/delx)+1
c        nxcb2=nxc/2=32
c     Storage requirements minimized by computing inward ray optical
c     depth eminr from detacmf, which spans only CMF x-range.
c
      include 'global.h'
      include 'sweep.h'
      include 'zone.h'
      integer stderr
      parameter (adumin=1.e-4,stderr=6)
      parameter (spiinv=.5641895835)
```



```fortran
c     parameter (elkap = 0.5*sigel*(1.+xhabun)/xmpro)
      parameter (xhabun=0.73)
      parameter (elkap = 0.2*(1.+xhabun))
      parameter (xmo = 1./elkap)
      parameter (fvthlbc=1.0)
      parameter (ekpuls=1.23e2,vthpuls=5.88e5,denpuls=1.67266e-12) !Puls epsp
      parameter (epscl=ekpuls*vthpuls*elkap/denpuls)
      parameter (nrmax=maxsweep+3,ngrmax=nrmax,fuz=1.e-30)
c     parameter (nxcc=nxmax,nxcmax=nxmax,nxcdop=20) !Disables CMF Limit
c     parameter (nxcc=64,nxcmax=128,nxcdop=20)      ! Enables CMF Limit
c     parameter (nxcc=16,nxcmax= 64,nxcdop= 8)      ! Enables CMF Limit
      parameter (nxcc=32,nxcmax= 64,nxcdop=16)      ! Enables CMF Limit
c
      dimension dum(maxsweep)
      dimension rntf(0:nrmax),vntf(0:nrmax),roi(0:nrmax),asndz(nrmax)
      dimension radz(nrmax),sobicrr(nrmax),tauc(nrmax)
      dimension prad(0:nrmax),grad(0:nrmax),gscat(0:nrmax),gssf(0:nrmax)
      dimension grav0(maxsweep),grav1(maxsweep)
c
      dimension c1(0:nrmax),c2(0:nrmax),c3(0:nrmax)
      dimension c4(0:nrmax),c5(0:nrmax), b_mu(0:nrmax),f_ld(0:nrmax)
      dimension velm(0:nrmax),zray(0:nrmax)
      dimension ixcvec(0:nrmax),ixcmnv(0:nrmax),ixcmxv(0:nrmax)
      dimension xmu (nxmax),ex (nxmax),fc (nxmax)
      dimension xmuo(nxmax),exo(nxmax),fco(nxmax)
      dimension elbc(nxmax),erbc(nxmax)
      dimension eminr(nxmax)  ! xvec,esumr in zone.h, to allow print out.
c
      parameter (nray=1)
c      parameter (nray=100)
      dimension yray(nray),wyray(nray)
c TEST TO INCLUDE MORE RAYS, with and without limb-darkening
c JS 0412 ------------------
c      data yray  /0.5/
c      data wyray /1./
c      dimension detacmf(nxcmax,ngrmax,nray) !not yet implemented.
c      dimension exc    (nxcmax,ngrmax,nray)
      dimension detacmf(nxcmax,ngrmax)
      dimension exc    (nxcmax,ngrmax)
      logical notinit
      flr(x) = sign(max(abs(x),fuz),x)
      ih(i)   = max(min(i,1),0) !Integer Heaviside function
c
      data notinit /.false./
      data xo,xf,delx /-25.,-5.,0.3333333333333/
      real delv,red_fac,sobi_ld
c      parameter (neg_frad=0)
c      parameter (delv_cut=)
c     data xo,xf,delx /-25.,-5.,0.1/
c     data issfrbc /1/ ! now in global.h
c
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      if(sweep.eq.'x') then
      if(ifrc.le.0) then
         call cakfrc(dum,grav0,grav1)
      return
      endif
c NEW -- SOME TESTS TO USE MORE THAN ONE RAY IN ANGLE INTEGRATION
c JS-0412
      if (nray.gt.1) then
         call angle_weights(yray,wyray,nray)
```

```
      else
         yray=0.5
         wyray=1.
      endif
c
c First, convert VH-1 to TDSW variables,
c taking TDSW interface = VH-1 zonal centers
c
      alpha = abbott
      nr    = nmax
      irad1 = nmin
      iradf = nmax
      rsscl = rstar
      oma   = 1.-alpha
      opa   = 1.+alpha
      alpham= -alpha
      obkapm= 1./xkapm
      obkmtoma=obkapm**oma
c     write(stderr,*) 'obkapm=',obkapm
c
      nrp1   = nr+1
      irad0  = irad1-1
      iradfp1= iradf+1
      iprnt=irad0+50
c
      do ir=irad1,iradf
         radz(ir)=xa0(ir)
      enddo
      do ir=irad0,iradf
         grad(ir) = fuz
         gscat(ir)= fuz
         gssf(ir) = fuz
         tauc(ir) = fuz
         vntf(ir) = u(ir)
         roi(ir)  = rho(ir)
         rntf(ir) = xa0(ir)+0.5*dx0(ir)
         asndz(ir)= sqrt(p(ir)/rho(ir))
      enddo
      rntf(irad0)=xa0(irad1)-0.5*dx0(irad1)
      radz(irad0)=xa0(irad1)-dx0(irad1)
c     write(stderr,*)
c    $ irad0,vntf(irad0),roi(irad0),rntf(irad0),asndz(irad0)
      asndre = sqrt(boltzman*tempw/avgmass)
      vth    = vthba*asndre
      bvth   = cak*vth**alpha
      frnorm = spiinv*delx*oma
c
c Set range of freq grid.
c
      xvmin = xo
      xvrng = xf-xo
      if (xf.lt.0.) then
         vmax = -1.e20
         vmin =  1.e20
         do i = 1 , nr
            vmax = max(vmax,vntf(i))
            vmin = min(vmin,vntf(i))
         enddo
         if (xf.lt.xo) then
            xvrng = vmax/vth - 2.*xo
            xvmin = xo
         else
```



```
            xvmax = vmax/vth-xf
            xvmin = max(xo,vmin/vth+xf)
            xvmin = delx*(int(xvmin/delx))
            xvrng = xvmax-xvmin
         endif
       endif
1111   nx = 64*(int((xvrng)/(delx*64.))+1)
c555   nx =    (int((xvrng)/(delx    ))+1)
       nx = min(nxmax,nx)
       x = xvmin
       do  ix=1,nx
          xvec(ix) = x
          x = x+delx
       enddo
c
c Begin RAY LOOP:
c
      do iy=1,nray
c
c Store coefficients for x-integration:
c
         do ir=irad0,iradf
            cost     = sqrt(1.-yray(iy)*(rsscl/rntf(ir))**2)
            irm1     = max(irad0,ir-1)
            vthz     = vthba*asndz(ir)
            velm(ir) = cost*vntf(ir)/vthz
            zray(ir) = cost*rntf(ir)
            c4(ir)   = vth/vthz
            c5(ir)   = c4(ir)*exp(min(0.,1.-(asndz(ir)/asndre)**2))
c           c4(ir)   = 1.   ! No therm speed correction
c           c5(ir)   = 1.   ! No kappa temp. correction
            prad(ir)= wyray(iy)*c5(ir)/c4(ir)
c Compute index for local CMF frequency,
c  and use to set x-integration range
            ixcvec(ir)=ifix((velm(ir)/c4(ir)-xvec(1))/delx)+1
         enddo
         do ir=irad1,iradf
            irm1    = ir-1
            rhoo    = roi(irm1)
            dz      = zray(ir)-zray(irm1)
            drho    = roi(ir)-rhoo
            c1(ir)  = rhoo*dz
            c2(ir)  = drho*dz
            c3(ir)  = spiinv*c2(ir)/2.
c        if((ir.eq.iprnt).or.(ir.eq.iprnt+1))
c     $ write(stderr,*) ir,rhoo,dz,drho,c1(ir),c2(ir)
            ixcavg  = (ixcvec(ir)+ixcvec(irm1))/2
            ixcdel  = abs(ixcvec(ir)-ixcvec(irm1))
c           ixcfac  = ifix(float(ixcdel)/nxcc+1.3)
            nxc     = nxcc
c           if (ixcdel.gt.(0.7*nxcc)) nxc=nxcc2
            nxc     = ((nxcdop+ixcdel)/nxcc+1)*nxcc
            nxc     = min(nx,nxc,nxcmax)
            ixcmnv(ir)=min(max(ixcavg-nxc/2,0),nx-nxc)+1
            ixcmxv(ir)=ixcmnv(ir)+nxc-1
c           if (nxc.gt.nxcc)
c        write(stderr,*)ir,nxc,ixcmin,ixcmax,ixc,xvec(ixcvec(ir))-velm(ir)
         enddo
c
c Now initialize esumr at LBC
c as Schuster-Schwarzschild reversing layer (for gabs):
c
```

```
        do ix=1,nx
           xmu(ix)  = xvec(ix)*fvthlbc*c4(irad0) - velm(irad0)
           ex(ix)   = 0.
c          fc(ix)   = cvmgp(0.,1.,xmu(ix))
           fc(ix)   = cvmgp(0.,1.,float(ix-ixcvec(irad0)))
           esumr(ix)= obkapm
        enddo
        ixcmin = ixcmnv(irad1)
        ixcmax = ixcmxv(irad1)
        ixcmnv(irad0) = ixcmin
        ixcmxv(irad0) = ixcmax
        nxc    = ixcmax-ixcmin+1
c        write(stderr,*) irad0,ixcmin,ixcmax,ixc,xmu(ixc)
c       call fcaphi(nxc,xmu(ixcmin),ex(ixcmin),fc(ixcmin))
        call fcaphi(ixcmax,xmu,ex,fc,ixcmin)
        fsum=0.
        do ix=ixcmin,ixcmax
           esumr(ix) = xmo*spiinv*ex(ix) + obkapm
           tmp=ex(ix)/esumr(ix)**alpha
           fsum=fsum+tmp
        enddo
        grad(irad0) = grad(irad0)+prad(irad0)*fsum
        do ix=1,nx
           elbc(ix)=esumr(ix)
        enddo
c
c Begin RADIUS LOOP:
c
        do ir=irad1,iradf
c
c Initialize x arrays over the full x-range by presetting
c the exponential and error function arrays
c to their limiting forms:
c
           do ix=1,nx
              exo(ix)  = ex(ix)
              fco(ix)  = fc(ix)
              ex(ix)   = 0.
c             fc(ix)   = cvmgp(0.,1.,ix-ixcvec(ir))
              fc(ix)   = cvmgp(0.,1.,float(ix-ixcvec(ir)))
           enddo
c
c Now over the limited CMF x-range, perform the following:
c First, compute exponential & error functions ex(ix) & fc(ix),
c and use these to obtain esumr(ix) increments deta.
c Then compute force contribution by summing
c deta/esumr(ix)**alpha
c
c Note that because of possible x-range differences between
c zones ir and ir-1, it is necessary to recompute xmuo(ix).
c However, exo(ix) and fco(ix) are known over the full x-range
c by virtue of the preset to their asymptotic values.
c
c
           irm1   = ir-1
           ixcmin = ixcmnv(ir)
           ixcmax = ixcmxv(ir)
           nxc    = ixcmax-ixcmin+1
           do ix=ixcmin,ixcmax
              xmu (ix)  = xvec(ix)*c4(ir)   - velm(ir)
              xmuo(ix)  = xvec(ix)*c4(irm1) - velm(irm1)
           enddo
```



```
c          call fcaphi(nxc,xmu(ixcmin),ex(ixcmin),fc(ixcmin))
           call fcaphi(ixcmax,xmu,ex,fc,ixcmin)
           fsum  = 0.
           do ix=ixcmin,ixcmax
              dxmu  = xmuo(ix)-xmu(ix)
              dxmusq= dxmu*dxmu
              if (dxmusq.lt.fuz) then
                 deta = c1(ir)*(ex(ix)+exo(ix))
              else
                 deta = ((c1(ir)*dxmu+c2(ir)*xmuo(ix))*
     $                                      (fc(ix)-fco(ix))
     $                               - c3(ir)*(ex(ix)-exo(ix)))/dxmusq
              endif
c          if(((ir.eq.iprnt).or.(ir.eq.iprnt+1)).and.(ix.eq.9))
c    $  write(stderr,*) ix,ir,
c    $   (c1(ir)*dxmu+c2(ir)*xmuo(ix))*(fc(ix)-fco(ix)),
c    $    c1(ir)*dxmu,c2(ir)*xmuo(ix),fc(ix)-fco(ix),
c    $   -c3(ir)*(ex(ix)-exo(ix)),dxmusq,deta
              deta      = c5(ir)*max(deta,fuz) !kap temp corr
              esumr(ix) = esumr(ix) + deta
              ixc       = ix-ixcmin+1
              detacmf(ixc,ir) = deta
              exc    (ixc,ir) = ex(ix)
              tmp  = ex(ix)/esumr(ix)**alpha
              if ((ix.eq.ixcmin).or.(ix.eq.ixcmax)) tmp=0.5*tmp
              fsum = fsum + tmp
           enddo
           c2(ir) = fsum !Store to vectorize normalization
c -- NEW~~~~~~~~~ JS -- Simplity later!
           rsbr      = rsscl/radz(ir) !make So ZONE-centered
           rsbrsq    = rsbr*rsbr
           xmustar   = sqrt(max((1.-rsbrsq),0.))
           cost      = sqrt(1.-yray(iy)*rsbrsq)
           f_ld(ir)  = (cost**2-xmustar**2)/(1.-xmustar**2)
           if (f_ld(ir).lt.0.0) stop'ld corr, dir < 0'
           b_mu(ir) = fsum*sqrt(f_ld(ir))
c           write(*,*) cost,xmustar,sqrt(f_ld(ir))
         enddo ! End Main Radius loop for direct term
        do ir=irad1,iradf
           if (epsabs.le.-1.0) then
c ld correction, JS0412
              grad(ir) = grad(ir) + 0.5*prad(ir)*c2(ir) + (3./4.)*prad(ir)*b_mu
(ir)
c              write(*,44) ir,radz(ir)/rsscl,c2(ir),grad(ir),b_mu(ir),sqrt(f_ld
(ir))
c OBS! This is actually a *constant* as it is now -- increase force by ~3% for y
=0.5 ray
           else
              grad(ir) = grad(ir) + prad(ir)*c2(ir)
           endif
        enddo
 44      FORMAT(i10,5e15.5)
c
c Now conditionally compute SSF approx diffuse force.
c
        if (ifrc.gt.1) then
c
c First implement eta_minus RBC.
c
c  For ISSFRBC =<0, use SPO RBC:
c   Assume eta_minus(RBC) is ~ rho*r
c   (i.e., velocity constant, density ~1/r^2) ---
```

```
c   this zeroes out ftot.
c
            fsum=0.
            do ix=1,nx
               xmu(ix) = xvec(ix)*c4(iradf) - velm(iradf)
            enddo
            if (issfrbc.le.0) then
               do ix=1,nx
                 tmp  = xmu(ix)
                 tmp  = spiinv*exp(-tmp*tmp)
                 eminr(ix) = roi(iradf)*rntf(iradf)*tmp+obkapm
               enddo
c
c  For ISSFRBC >=1, use JIC RBC:
c   Assume eta_minus is eta_plus reflected about CMF line center.
c   (i.e., Odd symmetry of v wrt rmax, and even symmetry of rho) ---
c   this zeroes out gscat.
c
            else
               do ix =1,nx
                  xref = -xmu(ix)
                  uref = (xref-xmu(1))/delx+10001.
                  ixr = int(uref) - 10000.
                  uref = uref - 10000. - float(ixr)
                  if ((ixr.le.0).or.(ixr.ge.nx)) then
                     eminr(ix) = obkapm
                  else
                     eminr(ix) = esumr(ixr)+
     $                              uref*(esumr(ixr+1)-esumr(ixr))
                  endif
c     write(stderr,375) ix,ixr,xref,uref,esumr(ix),esumr(ixr),
c    $             eminr(ix)
 375  format(' RBC:',2i4,1p6e10.3)
               enddo
            endif
           do ix=1,nx
              erbc(ix)=eminr(ix)  ! save rbc for later
           enddo
c
c Now do BACKward spatial integral for the
c inward contribution to diffuse force.
c
            xkapc = elkap
c           xkapc = xkapc+1./c1(irad1)
c **BUG**   tauc(iradf) = xkapc*rho(iradf)*radz(iradf)
            tauc(iradfp1) = 0.
            do ir=iradf,irad1,-1
               tauc(ir) = tauc(ir+1)+xkapc*c1(ir)
c              write(stderr,*) ir,c1(ir),xkapc,tauc(ir)
               ixcmin = ixcmnv(ir) !ir-1 better???
               ixcmax = ixcmxv(ir) ! "     "
               fsum   = 0.
               do ix=ixcmin,ixcmax
                 ixc  = ix-ixcmin+1
                 tmp  = exc(ixc,ir)
                 tmp  = tmp/eminr(ix)**alpha
                 if ((ix.eq.ixcmin).or.(ix.eq.ixcmax)) tmp=0.5*tmp
                 fsum = fsum + tmp
                 eminr(ix) = eminr(ix)+detacmf(ixc,ir)
               enddo
               c2(ir) = fsum
            enddo ! end SSF radius loop
```



```
               do ir=irad1,iradf
                  gscat(ir) = gscat(ir) + prad(ir)*c2(ir)
               enddo
             endif ! SSF
         enddo   !end of ray loop
c
c Next, conditionally compute smooth source function, sobicrr.
c
         if (ifrc.gt.1) then
           do ir=irad0,iradf
c           rsbr      = rsscl/rntf(ir) !make So INTF-centered
            rsbr      = rsscl/radz(ir) !make So ZONE-centered
            rsbrsq    = rsbr*rsbr
            xmustar   = sqrt(max((1.-rsbrsq),0.))
            thinfac   = 0.5/(1.+xmustar)
            sobicrr(ir) = zso(max(ir-2,1),1,1)
            if (sobicrr(ir).le.0.) sobicrr(ir)=thinfac  !make sure So defined.
c Compute Source function correction for given eps assuming Bplanck/Ic = 1.
            if (epsabs.lt.0.) then
               tmp = thinfac-epsabs/rsbrsq
               sobicrr(ir) = tmp/(1.-epsabs)                  !set So= opt. THIN form
               if (epsabs.le.-1.) then
                  call s_limb_dark(sobi_ld,xmustar)
                  sobicrr(ir) = sobi_ld/rsbrsq
c HAVE ADDED TEST_CASES FOR LIMB-DARK HERE
c -- Don't forget the r^2 correction-factor!!
               endif
c               write(*,*) 1./rsbr, sobicrr(ir),thinfac
               if (epsabs.le.-10.) sobicrr(ir) = -epsabs-1. !set So= const.
            else if (epsabs.gt.0) then                        !set So= opt. THICK form
 (OR-II modified)
c              dvbdr= (vzone(ir)-vzone(ir-1))/(radn(ir)-radn(ir-1))
               dvbdr= (vntf(ir)-vntf(ir-1))/(rntf(ir)-rntf(ir-1))
c Abs value is poor man's way of dealing with nonmonotoncity; improve later.
c              dvbdr= abs(dvbdr)
               vbr = abs(vntf(ir)/rntf(ir))
               sig = dvbdr/flr(vbr)-1.
c              vbr = vinfscl*(1.-rsbr)/rntf(ir) ! use smooth beta=1 vel. law
c              sig = (2.*rsbr-1.)/flr(1.-rsbr)
               e2  = 1.+xmustar
               e3  = 1.+e2*xmustar
               e4  = 1.+e3*xmustar
               e5  =(1.+e4*xmustar)/5.
               e4  = e4/4.
               e3  = e3/3.
               e2  = e2/2.
               qc  = 1.+sig*e3
               q   = 1.+sig/3.
               if (sig.lt.-1.) then             !Do proper correction for dv/dr<0.
                  xo = sqrt(-1./sig)
                  eo3= (1.+xo*(1.+xo))/3.
                  cf = (1.-xo)*(1.+sig*eo3)
                  q  = q -cf
                  xo = max(xo,xmustar)
                  eo3= (1.+xo*(1.+xo))/3.
                  cf = (1.-xo)*(1.+sig*eo3)
                  cf = cf*2./(1.-xmustar)
                  qc = qc-cf
               endif
               et  = (epsabs*epscl*roi(ir)**2)/vbr  !eps'*tauo
               tmp = thinfac*qc + et/rsbrsq
               sobs= tmp/(q+et)
```

```
            sobicrr(ir) = sobs
c Now correct to get right asymptotic growth rates.
c           sig= abs(dvbdr)/flr(vbr)-1.  !revert to poor man's method here.
            if (sig.gt.-1.) then
               tmp = e3+sig*e5-4.*e2*(1./3.+sig/5.)*sobs
               tmp = 0.5*(1.-tmp/(e2+sig*e4))
               sobicrr(ir) = tmp
            endif !** endif sig.gt.-1
         endif  !** endif epsabs.gt.0
        enddo
      endif  !** endif ifrc.gt.1 => Compute sobicrr
c
c If not ESF, normalize now.
c
      if (ifrc.lt.4) then
      do ir=irad1,iradf
           tmp  = frnorm*bvth/(rntf(ir)*rntf(ir))
           gscat(ir) = tmp*(gscat(ir)-grad(ir))*sobicrr(ir)
           gssf(ir)  = gscat(ir)
           grad(ir)  = tmp*grad(ir)
           if (ifrc.eq.3) grad(ir)=grad(ir)+gscat(ir)
c     write(stderr,*) ir,gscat(ir),grad(ir)
        enddo
c
       else
c
c local, Ensemble Source Function (ESF) force option:
c (*** NOTE: Currently assumes only a single ray ***)
c
c
c First, build ensemble source function into c3(ir).
c
         do ir=irad1,iradf
            tmp=2./(1.+(gscat(ir)+gscat(ir-1))/(grad(ir)+grad(ir-1)))
            if (ifrc.eq.5) tmp=1.                             ! Smooth VARIABLE
Source Func.
c           if ((ir.lt.irad0+10).and.notinit) tmp=zesf(ir-2,1,1)  ! Use init S n
ear LBC.
c           if ((ir.lt.irad0+irssfmax)) tmp=1.                ! Test to keep LBC
 smooth.
            if ((ir.lt.irad0+irssfmax)) tmp=0.                ! Test to destab.
base
             sobic=(sobicrr(ir)+sobicrr(ir-1))/(2.*radz(ir)*radz(ir)) ! zone-cen
ter So
            c3(ir)=(tmp+2.*ih(ifrc-6)*tauc(ir))*sobic         ! S(r), including
tauc for ifrc>6
            if (ifrc.eq.4) c3(ir)=1.                          ! SSF the hard way
, for testing.
c      write(stderr,990)ir,sobicrr(ir),c3(ir),tmp,gscat(ir),grad(ir)
         enddo
       notinit=.true.
c
c For both forward and backward rays, compute nested rp sum of intensity differe
ntial,
c and then frequency integrate for forward-backward stream intensity.
c
         do ir=irad0,iradf
            do ix=1,nx                  ! Reinitialize for calculation of eta(r)-
eta(rp)
               esumr(ix)= elbc(ix)
               eminr(ix)= erbc(ix)
c              esumr(ix)= obkapm
```



```
c              eminr(ix)= obkapm
               fc(ix)   = 0.
               xmu(ix)  = 0.
               xmuo(ix) = xmu(ix)
            enddo
            if (ir.gt.irad0) then
              do irp=ir,irad1,-1       !forward ray
                ixmin = max(ixcmnv(ir),ixcmnv(irp))
                ixmax = min(ixcmxv(ir),ixcmxv(irp))
                do ix = ixmin,ixmax
                  ixcrp = ix-ixcmnv(irp)+1
                  esumr(ix) = esumr(ix)+detacmf(ixcrp,irp)
                  etma = esumr(ix)**alpham
                  tmp=c3(irp)*(xmu(ix)-etma)
                  fc(ix)=fc(ix)+tmp
                  xmu(ix)=etma
c               if ((ir.eq.ipnrt).and.(ixcrp.ge.29).and.(ixcrp.le.35)
c    $           .and.(irp.ge.40))
c               if ((ir.eq.iprnt).and.(ix.eq.ixcvec(iprnt))) then
c       write(stderr,980)
c    $ ix,ixcrp,irp,c3(irp),detacmf(ixcrp,irp),esumr(ix),tmp,fc(ix)
c           endif
980   format(1x,3i5,1p6e11.4)
                enddo
              enddo
            endif
            if (ir.lt.iradf) then
               do irp=ir+1,iradf        !backward ray
                ixmin = max(ixcmnv(ir),ixcmnv(irp))
                ixmax = min(ixcmxv(ir),ixcmxv(irp))
                do ix = ixmin,ixmax
                  ixcrp = ix-ixcmnv(irp)+1
                  eminr(ix) = eminr(ix)+detacmf(ixcrp,irp)
                  etma = eminr(ix)**alpham
                  tmp=c3(irp)*(xmuo(ix)-etma)
                  fc(ix)=fc(ix)-tmp
                  xmuo(ix)=etma
c            if ((ir.eq.iprnt).and.(ixcrp.ge.29).and.(ixcrp.le.35)
c    $          .and.(irp.le.61))
c            if ((ir.eq.iprnt).and.(ix.eq.ixcvec(iprnt))) then
c      write(stderr,980)
c    $ ix,ixcrp,irp,c3(irp),detacmf(ixcrp,irp),eminr(ix),tmp,fc(ix)
c      endif
                enddo
              enddo
            endif
            fsum = 0.
            irp1=min(ir+1,iradf)
            do ix =ixcmnv(ir),ixcmxv(ir)
               ixcr = ix - ixcmnv(ir)  +1
               ixcr1= ix - ixcmnv(irp1)+1
c **BUG**      ixcr1= min(max(1,ixcr1),ixcmxv(irp1))
c              tmp = flr(0.25*(detacmf(ixcr,ir)+detacmf(ixcr1,irp1)))
               tmp = detacmf(ixcr,ir)
               if((ix.ge.ixcmnv(irp1)).and.(ix.le.ixcmxv(irp1))) then
     $          tmp=tmp+detacmf(ixcr1,irp1)
               tmp=flr(0.25*tmp)
c              tmp2= 1.                                      !PCSF (much too stro
ng)
               tmp2= ((obkapm+tmp)**oma-obkmtoma)/(oma*tmp) !PLSF
               fc(ix) = fc(ix)+tmp2*(c3(ir)-c3(ir+1))       !local S' term corre
ction
```

```
              tmp  = exc(ixcr,ir)*fc(ix)
              if ((ix.eq.ixcmnv(ir)).or.(ix.eq.ixcmxv(ir))) tmp=tmp/2.
              fsum = fsum + tmp
c             if ((ir.eq.iprnt).and.(abs(ix-ixcvec(ir)).le.5)) then
c             tmp3=1./eminr(ix)**alpha-1./esumr(ix)**alpha
c             if (ifrc.ne.4) tmp3=tmp3*sobicrr(ir)/(rntf(ir)*rntf(ir))
c     write(stderr,990)
c    $ ix,eminr(ix),esumr(ix),fc(ix),tmp2,tmp,fsum,exc(ixcr,ir)
c             endif
           enddo
           c2(ir) = fsum        ! ~ Ibar^+(r)-Ibar^-(r)
        enddo
        tmp1=grad (irad1)
        tmp2=gscat(iradf)
        if (ifrc.ne.4) then
           tmp1=tmp1*sobicrr(irad0)/(rntf(irad0)*rntf(irad0))
           tmp2=tmp2*sobicrr(iradf)/(rntf(iradf)*rntf(iradf))
        endif
        c2(irad0) = c2(irad0)+tmp1
        c2(iradf) = c2(iradf)-tmp2
c
c Normalize to forces.
c
        do ir=irad0,iradf
           rsq  = rntf(ir)*rntf(ir)
           sobic= sobicrr(ir)/rsq
           if (ifrc.eq.4) c2(ir)= c2(ir)*sobic
           tmp  = frnorm*bvth
           gssf(ir)  = tmp*(gscat(ir)-grad(ir))*sobic
           if(ifrc.gt.6)
     $       grad(ir)=(tauc(ir)+1.)*grad(ir)-tauc(ir)*gscat(ir) !  tauc correcti
on.
           grad(ir)  = tmp*grad(ir)/rsq
           gscat(ir) = tmp*c2(ir)
c          if(ir.le.irad0+10) gscat(ir)=gssf(ir)     !*** TEST using SSF near L
BC
c     write(stderr,990)ir,sobic,c3(ir)/sobic,grad(ir),gssf(ir),gscat(ir)
990        format(1x,i5,1p7e12.4)
        enddo
        gscat(irad0)=gssf(irad0)
        gscat(iradf)=gssf(iradf) ! Revert to SSF for RBC.
        do ir=irad0,iradf
           grad(ir) = grad(ir)+gscat(ir)
        enddo
      endif  !** endif ifrc.lt.4 , else

c
c Store results for VH-1
c
1000  continue
      do n = nmin, nmax
c Option to turn off rad force if v-del below some negative cut-off
c JS 1204
        if (neg_frad.eq.1) then
           delv = u(n)-u(n-1)
           red_fac = exp(Min(0.0,delv)/delv_cut)
           grav0(n) = grav0(n) + grad(n)*red_fac
           grav1(n) = grav1(n) + grad(n)*red_fac
c           if (red_fac.lt.1.0) write(*,*) n,delv,grav0(n),grad(n),exp(Min(0.0,
delv)/delv_cut),delv_cut
        else
c Here comes standard option
```



```
           grav0(n) = grav0(n) + grad(n)
           grav1(n) = grav1(n) + grad(n)
        endif
        zgr(n-2,1,1) = flr(grad(n))
        zgs(n-2,1,1) = flr(gscat(n))
        zso(n-2,1,1) = sobicrr(n)
        zesf(n-2,1,1)=c3(n)*radz(n)*radz(n)*2./flr(sobicrr(n)+sobicrr(n-1))
        zgssf(n-2,1,1)=flr(gssf(n))
        ztauc(n-2,1,1)=tauc(n-1)
c       write(stderr,*) n-2,ztauc(n-2,1,1)
      enddo
C
C      do n = nmin-2, nmax
C       write(15,1500) xa0(n)/xa0(nmin),rho(n),p(n),u(n),
C     &                 soblev(n)/GM* (xa0(n)+0.5*dx0(n))**2
C      enddo
C1500   format(1x,1pe11.3,4e11.3)
      endif
      return
      end
c
c-----------------------------------------------------------------------
c
      subroutine fcaphi(nx,x,ex,fc,ix1)
c
c     include 'params.cli'
c     Common block for Voigt profile parameters:
c     include 'cvoi.cli'
      dimension x(nx),ex(nx),fc(nx)
c
c Fast evaluation of
c     caphi(x) == int from x to inf phi(x) dx ,
c
c where
c     phi(x)   == exp(-x*x)/sqrt(pi) ; |x| < Xv
c              == a/(pi*x**2)        ;     > Xv
c and a==Voigt parameter = sqrt(pi)*(Xv**2)*exp(-Xv**2)
c
c Algorithm: in Doppler core, use adaptation of formula 7.1.25 A&S p. 299,
c  rational approximation to erf,
c  with |eps(x)| =< 2.5e-5.
c
      parameter (a1=.1740121,a2=-.0479399,a3=.3739278,p=.47047)
c
c      option assert (nohazard)
c     flr(xarg) = sign(max(abs(xarg),fuz),xarg)
c                [xarg is scalar dummy argument for function flr]
c
c     parameter (spiinv=.5641895835)
c     data xvoigt /0./
c     data avoigt,bvoigt,cvoigt,dvoigt /0.,0.,0.,0./
c     if (xvoi.gt.0.)  call fcaphi(1,xvoi,ev,fv)
c     xvoigt = max(xvoi,0.)
c     avoigt = ev*xvoigt**2
c     cvoigt = spiinv*avoigt
c     dvoigt = spiinv*ev*xvoigt-fv
c     bvoigt = 1./(1.+2.*dvoigt)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      do ix = ix1 , nx
        ex(ix) = exp(-x(ix)*x(ix))
```

```
         t = 1./(1.+p*abs(x(ix)))
         fc(ix) = t*(a1+t*(a2+t*a3))*ex(ix)
      enddo
c      if (xvoigt.gt.0.) then
c         do ix = 1 , nx
c           ax     = flr(abs(x(ix)))
c           xvmx   = xvoigt-ax
c           ev     = avoigt/(ax**2)
c           ex(ix) = cvmgp(ex(ix),ev,xvmx)
c           ex(ix) = ex(ix)*bvoigt
c           fv     = cvoigt/ax
c           fc(ix) = fc(ix) + dvoigt
c           fc(ix) = cvmgp(fc(ix),fv,xvmx)
c           fc(ix) = fc(ix)*bvoigt
c         enddo
c      endif
      do ix = 1 , nx
         fc(ix) = cvmgp(fc(ix),1.-fc(ix),x(ix))
      enddo
c     write(stderr,10) x,ex,t,fc
c10   format(' fcaphi:'1p5e10.3)
      return
      end
c
      subroutine s_limb_dark(s_ld,mustar)
c
c     Include Eddingtion limb-darkening
c     in the optically thin scattering source function
c
c     Use approximation below mustar=1e-3
      real s_ld,mustar,sinmustar
      if (mustar.gt.1.0.or.mustar.lt.0.0) then
         write(*,*),'mustar:',mustar
         stop'0>mustar or 1<mustar'
      else if (mustar.gt.1.e-3) then
         sinmustar = sqrt(1.-mustar**2.)
         s_ld = (1./16.)*(7.-4.*mustar+3.*mustar**2.*alog(mustar/(1.+sinmustar))
/sinmustar)
      else
         s_ld = (1./16.)*(7.-4.*mustar-3.*mustar**2.)
      endif

      return
      end

      subroutine angle_weights(yray,wyray,nray)

c Set up angle integration steps (y=[0,1]) and weights
c Use simple Trapez for testing for now
      integer nray, i
      dimension yray(nray),wyray(nray)
      real dyray

      dyray = 1./(nray-1)
      do i=1,nray
         yray(i) = (i-1)*dyray
         if (i.eq.1.or.i.eq.nray) then
            wyray(i)=0.5*dyray
         else
            wyray(i)=1.0*dyray
         endif
c         write(*,*),i,yray(i),wyray(i)
```



```
      enddo
c      stop
      return
      end
```