```
c
c Subroutine for the TVD to capture the shock
c-------------------------------------------------------------------------------
c NAME: tvdmod
c
c RELEASED: Zhi Shang released the SBLI3.7 original version
c
c REVISED: Adriano Cerminara
c
c PURPOSE: define the arrays and arguments for TVD
c
c MOUDLE USING: incl3d in incl3d.F
c mppvars in mppvars.F
c mbvars in mbvars.F
c
c SUBROUTINES CALLED: rmnx() in tvd3d.F
c updis24() in tvd3d.F
c
c***5***10********20********30********40********50********60********70********80
module tvdmod
use const, only : rk
c
c real(kind=rk) alam(-1:ijkm,5)
c & ,al(-1:ijkm,5),gt(-1:ijkm,5)

real(kind=rk), allocatable, dimension(:,:) :: al,alam,gt,th
end module tvdmod
c***5***10********20********30********40********50********60********70********80
subroutine tvdx
c
c evaluation of tvd terms in x direction
c
use incl3d
use mppvars
use mbvars

implicit none
integer :: i,ijk,j,k,n
real(kind=rk) :: rxm,rym,rzm,rxp,ryp,rzp,dr, fact
real(kind=rk),dimension(-1:ijkm,nvar) :: eff
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
logical :: xm_bound_jk, xp_bound_jk

c xm_bound_jk = xm_bound
eff = 0.0d0
do k=1,nzp
do j=1,nyp
call rmnx(j,k,q,r,tvdsmu1)
if(xm_bound_block2(blockid).ge.100) then
xm_bound_jk = (.not.xm_bc_intf(j,k)).and.xm_bound
else
xm_bound_jk = xm_bound
endif
if(xp_bound_block2(blockid).ge.100) then
xp_bound_jk = (.not.xp_bc_intf(j,k)).and.xp_bound
else
xp_bound_jk = xp_bound
endif
call updis24(nxp,eff,r,x,nxp,xperi,j,k,1,
& xm_bound_jk,xp_bound_jk,nprocx,tvdsmu1)
do i = 1,nxp
! dx=0.5d0*(x(i+1,j,k)-x(i-1,j,k))
! fact=0.5d0*dt/dx
cnn use euclidean distance to account for curvilinear grids
rxm=x(i,j,k)-x(i-1,j,k)
rym=y(i,j,k)-y(i-1,j,k)
```

```
rxp=x(i+1,j,k)-x(i,j,k)
ryp=y(i+1,j,k)-y(i,j,k)
rzm=z(i,j,k)-z(i-1,j,k)
rzp=z(i+1,j,k)-z(i,j,k)
dr = 0.5d0*(sqrt((rxm+rxp)**2 +(rym+ryp)**2
& +(rzm+rzp)**2))
fact=0.5d0*dt/dr
s(i,j,k,1) = s(i,j,k,1)-fact*(eff(i,1)-eff(i-1,1))
s(i,j,k,2) = s(i,j,k,2)-fact*(eff(i,2)-eff(i-1,2))
s(i,j,k,3) = s(i,j,k,3)-fact*(eff(i,3)-eff(i-1,3))
s(i,j,k,4) = s(i,j,k,4)-fact*(eff(i,4)-eff(i-1,4))
s(i,j,k,5) = s(i,j,k,5)-fact*(eff(i,5)-eff(i-1,5))
cnn...
s(i,j,k,6) = s(i,j,k,6)-fact*(eff(i,6)-eff(i-1,6))
end do
end do
end do

return
end
c***5***10********20********30********40********50********60********70********80
subroutine tvdy
c
c evaluation of tvd terms in y direction
c
use incl3d
use mppvars
use mbvars

implicit none
integer :: i,ijk,j,k,n
real(kind=rk) :: rxm,rym,rzm,rxp,ryp,rzp,dr,fact
real(kind=rk),dimension(-1:ijkm,nvar) :: eff
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
logical :: ym_bound_ik, yp_bound_ik

eff = 0.0d0

do k=1,nzp
do i=1,nxp
call rmny(i,k,q,r,tvdsmu1)
if(ym_bound_block2(blockid).ge.100) then
ym_bound_ik = (.not.ym_bc_intf(i,k)).and.ym_bound
else
ym_bound_ik = ym_bound
endif
if(yp_bound_block2(blockid).ge.100) then
yp_bound_ik = (.not.yp_bc_intf(i,k)).and.yp_bound
else
yp_bound_ik = yp_bound
endif
call updis24(nyp,eff,r,y,nyp,yperi,i,k,2,
& ym_bound_ik,yp_bound_ik,nprocy,tvdsmu1)
do j = 1,nyp
! dy=0.5d0*(y(i,j+1,k)-y(i,j-1,k))
! fact=0.5d0*dt/dy
cnn use euclidean distance to account for curvilinear grids
rxm=x(i,j,k)-x(i,j-1,k)
rym=y(i,j,k)-y(i,j-1,k)
rxp=x(i,j+1,k)-x(i,j,k)
ryp=y(i,j+1,k)-y(i,j,k)
rzm=z(i,j,k)-z(i,j-1,k)
rzp=z(i,j+1,k)-z(i,j,k)
dr = 0.5d0*(sqrt((rxm+rxp)**2 +(rym+ryp)**2
& +(rzm+rzp)**2))
fact=0.5d0*dt/dr
```

```
s(i,j,k,1) = s(i,j,k,1)-fact*(eff(j,1)-eff(j-1,1))
s(i,j,k,2) = s(i,j,k,2)-fact*(eff(j,2)-eff(j-1,2))
s(i,j,k,3) = s(i,j,k,3)-fact*(eff(j,3)-eff(j-1,3))
s(i,j,k,4) = s(i,j,k,4)-fact*(eff(j,4)-eff(j-1,4))
s(i,j,k,5) = s(i,j,k,5)-fact*(eff(j,5)-eff(j-1,5))
cnn...
s(i,j,k,6) = s(i,j,k,6)-fact*(eff(j,6)-eff(j-1,6))
end do
end do
end do
c
return
end
c***5***10********20********30********40********50********60********70********80
subroutine tvdz
c
c evaluation of tvd terms in z direction
c
use incl3d
use mppvars
use mbvars

implicit none
integer :: i,ijk,j,k,n
real(kind=rk) :: rxm,rym,rzm,rxp,ryp,rzp,dr,fact
real(kind=rk),dimension(-1:ijkm,nvar) :: eff
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
logical :: zm_bound_ij, zp_bound_ij

eff = 0.0d0

do j=1,nyp
do i=1,nxp
call rmnz(i,j,q,r,tvdsmu1)
if(zm_bound_block2(blockid).ge.100) then
zm_bound_ij = (.not.zm_bc_intf(i,j)).and.zm_bound
else
zm_bound_ij = zm_bound
endif
if(zp_bound_block2(blockid).ge.100) then
zp_bound_ij = (.not.zp_bc_intf(i,j)).and.zp_bound
else
zp_bound_ij = zp_bound
endif
call updis24(nzp,eff,r,z,nzp,zperi,i,j,3,
& zm_bound_ij,zp_bound_ij,nprocz,tvdsmu1)
do k = 1,nzp
! dz=0.5d0*(z(i,j,k+1)-z(i,j,k-1))
! fact=0.5d0*dt/dz
cnn use euclidean distance to account for curvilinear grids
rxm=x(i,j,k)-x(i,j,k-1)
rym=y(i,j,k)-y(i,j,k-1)
rxp=x(i,j,k+1)-x(i,j,k)
ryp=y(i,j,k+1)-y(i,j,k)
rzm=z(i,j,k)-z(i,j,k-1)
rzp=z(i,j,k+1)-z(i,j,k)
dr = 0.5d0*(sqrt((rxm+rxp)**2 +(rym+ryp)**2
& +(rzm+rzp)**2))
fact=0.5d0*dt/dr
s(i,j,k,1) = s(i,j,k,1)-fact*(eff(k,1)-eff(k-1,1))
s(i,j,k,2) = s(i,j,k,2)-fact*(eff(k,2)-eff(k-1,2))
s(i,j,k,3) = s(i,j,k,3)-fact*(eff(k,3)-eff(k-1,3))
s(i,j,k,4) = s(i,j,k,4)-fact*(eff(k,4)-eff(k-1,4))
s(i,j,k,5) = s(i,j,k,5)-fact*(eff(k,5)-eff(k-1,5))
cnn...
s(i,j,k,6) = s(i,j,k,6)-fact*(eff(k,6)-eff(k-1,6))
```

```
end do
end do
end do

return
end
c***5***10********20********30********40********50********60********70********80
subroutine rmnx(jj,kk,qq,r,tvdsmu1)
use tvdmod
use incl3d

implicit none
integer :: jj,kk
real(kind=rk), dimension(1-xhalo:nxd+xhalo,1-yhalo:nyd+yhalo,
& 1-zhalo:nzd+zhalo,nvar) :: qq

integer :: i,m,n,l
real(kind=rk) :: c,cinv,csq,csqinv,delpc2,delr,g,h,p,qsq,sspsq
& ,rdeluc,rdelvc,rdelwc,rho,rhoi,sqrho,trho,u,v,w
& ,ssp_re
real(kind=rk) ri(-1:ijkm,5,5)
cnn..
real(kind=rk) :: phs,ephi
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
real(kind=rk),dimension(nvar) :: dw
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
real(kind=rk),dimension(-1:ijkm) :: ssp
c
if ( .not.allocated(alam) ) then
allocate (alam(-1:ijkm,nvar))
allocate (al (-1:ijkm,nvar))
allocate (gt (-1:ijkm,nvar))
endif

c
c roe's formula
c
do i=-1,nxp+2
rho=qq(i,jj,kk,1)
rhoi=1.0d0/rho
u=qq(i,jj,kk,2)*rhoi
v=qq(i,jj,kk,3)*rhoi
w=qq(i,jj,kk,4)*rhoi
p=gami*(qq(i,jj,kk,5)-0.5d0*rho*(u*u+v*v+w*w))
ssp(i)=1.4d0*0.4d0*(qq(i,jj,kk,5)*rhoi-0.5d0*(u*u+v*v+w*w))
sqrho=sqrt(rho)
wm(i,1)=sqrho
wm(i,2)=sqrho*u
wm(i,3)=sqrho*v
wm(i,4)=sqrho*w
wm(i,5)=(qq(i,jj,kk,5)+p)/sqrho
phs = qq(i,jj,kk,6)*rhoi
wm(i,6)=sqrho*phs
enddo
c
c do i=0,nxp+1
do i=-1,nxp+1
trho=wm(i,1)+wm(i+1,1)
u=(wm(i,2)+wm(i+1,2))/trho
v=(wm(i,3)+wm(i+1,3))/trho
w=(wm(i,4)+wm(i+1,4))/trho
h=(wm(i,5)+wm(i+1,5))/trho
qsq=u*u+v*v+w*w

cc sound speed correction in Roe's average state
sspsq=gami*(h-0.5d0*qsq)
```

```
ssp_re=min(ssp(i),ssp(i+1))
csq=max(sspsq,ssp_re)
c=sqrt(csq)
cc
g=gami/csq
csqinv=1.0d0/csq
cinv=1.0d0/c

cc local threshold value for the entropy correction term
tvdsmu1(i)=tvdsmu*(abs(u)+abs(v)+c)
cc

phs = (wm(i,6)+wm(i+1,6))/trho
cnn... choice of last element of the 2nd and 6th projected eigenvectors
cnn... suitable choices 1.0d0 or c (c seems to give better results)
ephi = 1.0d0
c
c ******* eigenvalues *******
c
alam(i,1)=u-c
alam(i,2)=u
alam(i,3)=u+c
alam(i,4)=u
alam(i,5)=u
alam(i,6)=u
c
c ***** eigenvectors ****
c
dw(1)=qq(i+1,jj,kk,1)-qq(i,jj,kk,1)
dw(2)=qq(i+1,jj,kk,2)-qq(i,jj,kk,2)
dw(3)=qq(i+1,jj,kk,3)-qq(i,jj,kk,3)
dw(4)=qq(i+1,jj,kk,4)-qq(i,jj,kk,4)
dw(5)=qq(i+1,jj,kk,5)-qq(i,jj,kk,5)
dw(6)=qq(i+1,jj,kk,6)-qq(i,jj,kk,6)
c
r(i,1,1)=1.0d0
r(i,1,2)=1.0d0
r(i,1,3)=1.0d0
r(i,1,4)=0.0d0
r(i,1,5)=0.0d0
r(i,2,1)=u-c
r(i,2,2)=u
r(i,2,3)=u+c
r(i,2,4)=0.0d0
r(i,2,5)=0.0d0
r(i,3,1)=v
r(i,3,2)=v
r(i,3,3)=v
r(i,3,4)=1.0d0
c ac r(i,3,4)=c
r(i,3,5)=0.0d0
r(i,4,1)=w
r(i,4,2)=w
r(i,4,3)=w
r(i,4,4) =0.0d0
r(i,4,5)=1.0d0
c ac r(i,4,5)=c
r(i,5,1)=h-u*c
r(i,5,2)=0.5d0*qsq
r(i,5,3)=h+u*c
r(i,5,4)=v
r(i,5,5)=w
c ac r(i,5,4)=v*c
c ac r(i,5,5)=w*c
r(i,1,6)=0.0d0
r(i,2,6)=0.0d0
r(i,3,6)=0.0d0
```

```
r(i,4,6)=0.0d0
r(i,5,6)=0.0d0
c...
r(i,6,1)=phs
r(i,6,2)=0.5*(phs+ephi)
r(i,6,3)=phs
r(i,6,4)=0.0d0
r(i,6,5)=0.0d0
r(i,6,6)=ephi
c
c --- scaling with csq
c
do n=1,nvar
do m=1,nvar
r(i,m,n)=r(i,m,n)*csqinv
end do
end do
c
c --- alpha coeeficients (rinv*delw)
c
delpc2=gami*(dw(5)+0.5d0*qsq*dw(1)
& -u*dw(2)-v*dw(3)-w*dw(4))*csqinv
delr=dw(1)
rdeluc=(dw(2)-u*dw(1))*cinv
rdelvc=(dw(3)-v*dw(1))*cinv
rdelwc=(dw(4)-w*dw(1))*cinv
c al(i,1)=0.5d0*(delpc2-rdeluc)
c al(i,2)=delr-delpc2
c al(i,3)=0.5d0*(delpc2+rdeluc)
c al(i,4)=rdelvc
c al(i,5)=rdelwc
!cnn
!#ifdef 1
! cphi=0.5d0*gami*(phs-c)*cinv*csqinv
! al(i,6)=-dw(1)*(0.5d0*(phs+c)*cinv+0.5*cphi*qsq)+
! & dw(2)*cphi*u+dw(3)*cphi*v+
! & dw(4)*cphi*w-dw(5)*cphi+dw(6)*cinv
!#endif
c#ifdef 1
c al(i,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi
c#endif
c
c --- scaling with csq
c
c...
c do m=1,nvar
c al(i,m)=al(i,m)*csq
c end do

c end do
c
ri(i,1,1)=0.5d0*(0.5d0*g*qsq+u/c)
ri(i,1,2)=-0.5d0*(g*u+1.0d0/c)
ri(i,1,3)=-0.5d0*g*v
ri(i,1,4)=-0.5d0*g*w
ri(i,1,5)=0.5d0*g
ri(i,2,1)=1.0d0-0.5d0*g*qsq
ri(i,2,2)=g*u
ri(i,2,3)=g*v
ri(i,2,4)=g*w
ri(i,2,5)=-g
ri(i,3,1)=0.5d0*(0.5d0*g*qsq-u/c)
ri(i,3,2)=-0.5d0*(g*u-1.0d0/c)
ri(i,3,3)=-0.5d0*g*v
ri(i,3,4)=-0.5d0*g*w
ri(i,3,5)=0.5d0*g
```

```
ri(i,4,1)=-v
ri(i,4,2)=0.0d0
ri(i,4,3)=1.0d0
ri(i,4,4)=0.0d0
ri(i,4,5)=0.0d0
ri(i,5,1)=-w
ri(i,5,2)=0.0d0
ri(i,5,3)=0.0d0
ri(i,5,4)=1.0d0
ri(i,5,5)=0.0d0
c
c ***** coefficients (alpha) - characteristic jumps *****
c
do l=1,5
al(i,l)=0.0d0
do m=1,5
al(i,l)=al(i,l)+ri(i,l,m)*dw(m)
end do
end do

al(i,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi

do m=1,nvar
al(i,m)=al(i,m)*csq
end do

enddo
return
end
c***5***10********20********30********40********50********60********70********80
subroutine rmny(ii,kk,qq,r,tvdsmu1)
use tvdmod
use incl3d

implicit none
integer ii,kk
real(kind=rk), dimension(1-xhalo:nxd+xhalo,1-yhalo:nyd+yhalo,
& 1-zhalo:nzd+zhalo,nvar) :: qq
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
integer :: j,m,n,l
real(kind=rk) :: c,cinv,csq,csqinv,delpc2,delr,g,h,p,qsq,sspsq
& ,rdeluc,rdelvc,rdelwc,rho,rhoi,sqrho,trho,u,v,w
& ,ssp_re
real(kind=rk) ri(-1:ijkm,5,5)
real(kind=rk) :: phs, ephi
real(kind=rk),dimension(nvar) :: dw
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
real(kind=rk),dimension(-1:ijkm) :: ssp
c
if ( .not.allocated(alam) ) then
allocate (alam(-1:ijkm,nvar))
allocate (al (-1:ijkm,nvar))
allocate (gt (-1:ijkm,nvar))
endif
c
c roe's formula
c
do j=-1,nyp+2
rho=qq(ii,j,kk,1)
rhoi=1.0d0/rho
u=qq(ii,j,kk,2)*rhoi
v=qq(ii,j,kk,3)*rhoi
w=qq(ii,j,kk,4)*rhoi
p=gami*(qq(ii,j,kk,5)-0.5d0*rho*(u*u+v*v+w*w))
ssp(j)=1.4d0*0.4d0*(qq(ii,j,kk,5)*rhoi-0.5d0*(u*u+v*v+w*w))
sqrho=sqrt(rho)
```

```
wm(j,1)=sqrho
wm(j,2)=sqrho*u
wm(j,3)=sqrho*v
wm(j,4)=sqrho*w
wm(j,5)=(qq(ii,j,kk,5)+p)/sqrho
phs = qq(ii,j,kk,6)*rhoi
wm(j,6)=sqrho*phs
enddo
c
c do j=0,nyp+1
do j=-1,nyp+1
trho=wm(j,1)+wm(j+1,1)
u=(wm(j,2)+wm(j+1,2))/trho
v=(wm(j,3)+wm(j+1,3))/trho
w=(wm(j,4)+wm(j+1,4))/trho
h=(wm(j,5)+wm(j+1,5))/trho
qsq=u*u+v*v+w*w

cc sound speed correction in Roe's average state
sspsq=gami*(h-0.5d0*qsq)
ssp_re=min(ssp(j),ssp(j+1))
csq=max(sspsq,ssp_re)
c=sqrt(csq)
cc
g=gami/csq
csqinv=1.0d0/csq
cinv=1.0d0/c

cc local threshold value for the entropy correction term
tvdsmu1(j)=tvdsmu*(abs(u)+abs(v)+c)
cc

phs = (wm(j,6)+wm(j+1,6))/trho
cnn... choice of last element of the 2nd and 6th projected eigenvectors
cnn... suitable choices 1.0d0 or c (c seems to give better results)
ephi = 1.0d0
c
c ******* eigenvalues *******
c
alam(j,1)=v-c
alam(j,2)=v
alam(j,3)=v+c
alam(j,4)=v
alam(j,5)=v
alam(j,6)=v
c
c ***** eigenvectors ****
c
dw(1)=qq(ii,j+1,kk,1)-qq(ii,j,kk,1)
dw(2)=qq(ii,j+1,kk,2)-qq(ii,j,kk,2)
dw(3)=qq(ii,j+1,kk,3)-qq(ii,j,kk,3)
dw(4)=qq(ii,j+1,kk,4)-qq(ii,j,kk,4)
dw(5)=qq(ii,j+1,kk,5)-qq(ii,j,kk,5)
dw(6)=qq(ii,j+1,kk,6)-qq(ii,j,kk,6)
c
r(j,1,1)=1.0d0
r(j,1,2)=1.0d0
r(j,1,3)=1.0d0
r(j,1,4)=0.0d0
r(j,1,5)=0.0d0
r(j,2,1)=u
r(j,2,2)=u
r(j,2,3)=u
r(j,2,4)=1.0d0
c ac r(j,2,4)=c
r(j,2,5)=0.0d0
r(j,3,1)=v-c
```

```
r(j,3,2)=v
r(j,3,3)=v+c
r(j,3,4)=0.0d0
r(j,3,5)=0.0d0
r(j,4,1)=w
r(j,4,2)=w
r(j,4,3)=w
r(j,4,4)=0.0d0
r(j,4,5)=1.0d0
c ac r(j,4,5)=c
r(j,5,1)=h-v*c
r(j,5,2)=0.5d0*qsq
r(j,5,3)=h+v*c
r(j,5,4)=u
r(j,5,5)=w
c ac r(j,5,4)=u*c
c ac r(j,5,5)=w*c
r(j,1,6)=0.0d0
r(j,2,6)=0.0d0
r(j,3,6)=0.0d0
r(j,4,6)=0.0d0
r(j,5,6)=0.0d0
c...
r(j,6,1)=phs
r(j,6,2)=0.5*(phs+ephi)
r(j,6,3)=phs
r(j,6,4)=0.0d0
r(j,6,5)=0.0d0
r(j,6,6)=ephi
c
c --- scaling with csq
c
do n=1,nvar
do m=1,nvar
r(j,m,n)=r(j,m,n)*csqinv
end do
end do
c
c --- alpha coeeficients (rinv*delw)
c
delpc2=gami*(dw(5)+0.5d0*qsq*dw(1)
& -u*dw(2)-v*dw(3)-w*dw(4))*csqinv
delr=dw(1)
rdeluc=(dw(2)-u*dw(1))*cinv
rdelvc=(dw(3)-v*dw(1))*cinv
rdelwc=(dw(4)-w*dw(1))*cinv
c al(j,1)=0.5d0*(delpc2-rdelvc)
c al(j,2)=delr-delpc2
c al(j,3)=0.5d0*(delpc2+rdelvc)
c al(j,4)=rdeluc
c al(j,5)=rdelwc
!!cnn...
!#ifdef 1
! cphi=0.5d0*gami*(phs-c)*cinv*csqinv
! al(j,6)=-dw(1)*(0.5d0*(phs+c)*cinv+0.5*cphi*qsq)+
! & dw(2)*cphi*u+dw(3)*cphi*v+
! & dw(4)*cphi*w-dw(5)*cphi+dw(6)*cinv
!#endif
!cnn
c#ifdef 1
c al(j,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi
c#endif
c
c --- scaling with csq
c
c do m=1,nvar
```

```
c al(j,m)=al(j,m)*csq
c end do
c end do
c
ri(j,1,1)=0.5d0*(0.5d0*g*qsq+v/c)
ri(j,1,2)=-0.5d0*g*u
ri(j,1,3)=-0.5d0*(g*v+1.0d0/c)
ri(j,1,4)=-0.5d0*g*w
ri(j,1,5)=0.5d0*g
ri(j,2,1)=1.0d0-0.5d0*g*qsq
ri(j,2,2)=g*u
ri(j,2,3)=g*v
ri(j,2,4)=g*w
ri(j,2,5)=-g
ri(j,3,1)=0.5d0*(0.5d0*g*qsq-v/c)
ri(j,3,2)=-0.5d0*g*u
ri(j,3,3)=-0.5d0*(g*v-1.0d0/c)
ri(j,3,4)=-0.5d0*g*w
ri(j,3,5)=0.5d0*g
ri(j,4,1)=-u
ri(j,4,2)=1.0d0
ri(j,4,3)=0.0d0
ri(j,4,4)=0.0d0
ri(j,4,5)=0.0d0
ri(j,5,1)=-w
ri(j,5,2)=0.0d0
ri(j,5,3)=0.0d0
ri(j,5,4)=1.0d0
ri(j,5,5)=0.0d0
c
C ***** coefficients (alpha) *****
c
do l=1,5
al(j,l)=0.0d0
do m=1,5
al(j,l)=al(j,l)+ri(j,l,m)*dw(m)
end do
end do

al(j,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi


do m=1,nvar
al(j,m)=al(j,m)*csq
end do

enddo
return
end
c***5***10********20********30********40********50********60********70********80
subroutine rmnz(ii,jj,qq,r,tvdsmu1)
use tvdmod
use incl3d

implicit none
integer :: ii,jj
c
cyyf if no scalar we should change array of Q to (*,*,*,5)
c
real(kind=rk), dimension(1-xhalo:nxd+xhalo,1-yhalo:nyd+yhalo,
& 1-zhalo:nzd+zhalo,nvar) :: qq
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
c
c Local variables.
c
real(kind=rk) :: phs,ephi
integer :: k,m,n,l
```

```
real(kind=rk) :: c,cinv,csq,csqinv,delpc2,delr,g,h,p,qsq,sspsq
& ,rdeluc,rdelvc,rdelwc,rho,rhoi,sqrho,trho,u,v,w
& ,ssp_re
real(kind=rk) ri(-1:ijkm,5,5)
real(kind=rk),dimension(nvar) :: dw
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
real(kind=rk),dimension(-1:ijkm) :: tvdsmu1
real(kind=rk),dimension(-1:ijkm) :: ssp
c
if ( .not.allocated(alam) ) then
allocate (alam(-1:ijkm,nvar))
allocate (al (-1:ijkm,nvar))
allocate (gt (-1:ijkm,nvar))
endif
c
c roe's formula
c
do k=-1,nzp+2
rho=qq(ii,jj,k,1)
rhoi=1.0d0/rho
u=qq(ii,jj,k,2)*rhoi
v=qq(ii,jj,k,3)*rhoi
w=qq(ii,jj,k,4)*rhoi
p=gami*(qq(ii,jj,k,5)-0.5d0*rho*(u*u+v*v+w*w))
ssp(k)=1.4d0*0.4d0*(qq(ii,jj,k,5)*rhoi-0.5d0*(u*u+v*v+w*w))
sqrho=sqrt(rho)
wm(k,1)=sqrho
wm(k,2)=sqrho*u
wm(k,3)=sqrho*v
wm(k,4)=sqrho*w
wm(k,5)=(qq(ii,jj,k,5)+p)/sqrho
phs = qq(ii,jj,k,6)*rhoi
wm(k,6)=sqrho*phs
enddo
c
ceg do k=0,nzp+1
do k=-1,nzp+1
trho=wm(k,1)+wm(k+1,1)
u=(wm(k,2)+wm(k+1,2))/trho
v=(wm(k,3)+wm(k+1,3))/trho
w=(wm(k,4)+wm(k+1,4))/trho
h=(wm(k,5)+wm(k+1,5))/trho
qsq=u*u+v*v+w*w

cc sound speed correction in Roe's average state
sspsq=gami*(h-0.5d0*qsq)
ssp_re=min(ssp(k),ssp(k+1))
csq=max(sspsq,ssp_re)
c=sqrt(csq)
cc
g=gami/csq
csqinv=1.0d0/csq
cinv=1.0d0/c

cc local threshold value for the entropy correction term
tvdsmu1(k)=tvdsmu*(abs(u)+abs(v)+c)

phs = (wm(k,6)+wm(k+1,6))/trho
cnn... choice of last element of the 2nd and 6th projected eigenvectors
cnn... suitable choices 1.0d0 or c (c seems to give better results)
ephi = 1.0d0
c
c ******* eigenvalues *******
c
alam(k,1)=w-c
alam(k,2)=w
alam(k,3)=w+c
```

```
alam(k,4)=w
alam(k,5)=w
alam(k,6)=w
c
c ***** eigenvectors ****
c
dw(1)=qq(ii,jj,k+1,1)-qq(ii,jj,k,1)
dw(2)=qq(ii,jj,k+1,2)-qq(ii,jj,k,2)
dw(3)=qq(ii,jj,k+1,3)-qq(ii,jj,k,3)
dw(4)=qq(ii,jj,k+1,4)-qq(ii,jj,k,4)
dw(5)=qq(ii,jj,k+1,5)-qq(ii,jj,k,5)
dw(6)=qq(ii,jj,k+1,6)-qq(ii,jj,k,6)
c
r(k,1,1)=1.0d0
r(k,1,2)=1.0d0
r(k,1,3)=1.0d0
r(k,1,4)=0.0d0
r(k,1,5)=0.0d0
r(k,2,1)=u
r(k,2,2)=u
r(k,2,3)=u
r(k,2,4)=1.0d0
c ac r(k,2,4)=c
r(k,2,5)=0.0d0
r(k,3,1)=v
r(k,3,2)=v
r(k,3,3)=v
r(k,3,4)=0.0d0
r(k,3,5)=1.0d0
c ac r(k,3,5)=c
r(k,4,1)=w-c
r(k,4,2)=w
r(k,4,3)=w+c
r(k,4,4)=0.0d0
r(k,4,5)=0.0d0
r(k,5,1)=h-w*c
r(k,5,2)=0.5d0*qsq
r(k,5,3)=h+w*c
r(k,5,4)=u
r(k,5,5)=v
c ac r(k,5,4)=u*c
c ac r(k,5,5)=v*c
r(k,1,6)=0.0d0
r(k,2,6)=0.0d0
r(k,3,6)=0.0d0
r(k,4,6)=0.0d0
r(k,5,6)=0.0d0
c...
r(k,6,1)=phs
r(k,6,2)=0.5*(phs+1.0d0)
r(k,6,3)=phs
r(k,6,4)=0.0d0
r(k,6,5)=0.0d0
r(k,6,6)=1.0d0
c
c --- scaling with csq
c
do n=1,nvar
do m=1,nvar
r(k,m,n)=r(k,m,n)*csqinv
end do
end do
c
c --- alpha coeeficients (rinv*delw)
c
delpc2=gami*(dw(5)+0.5d0*qsq*dw(1)
```

```
& -u*dw(2)-v*dw(3)-w*dw(4))*csqinv
delr=dw(1)
rdeluc=(dw(2)-u*dw(1))*cinv
rdelvc=(dw(3)-v*dw(1))*cinv
rdelwc=(dw(4)-w*dw(1))*cinv
c al(k,1)=0.5d0*(delpc2-rdelwc)
c al(k,2)=delr-delpc2
c al(k,3)=0.5d0*(delpc2+rdelwc)
c al(k,4)=rdeluc
c al(k,5)=rdelvc
!cnn...
!#ifdef 1
! cphi=0.5d0*gami*(phs-c)*cinv*csqinv
! al(k,6)=-dw(1)*(0.5d0*(phs+c)*cinv+0.5*cphi*qsq)+
! & dw(2)*cphi*u+dw(3)*cphi*v+
! & dw(4)*cphi*w-dw(5)*cphi+dw(6)*cinv
!#endif
!cnn...
c#ifdef 1
c al(k,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi
c#endif
c
c --- scaling with csq
c
c do m=1,nvar
c al(k,m)=al(k,m)*csq
c end do
c end do
c
ri(k,1,1)=0.5d0*(0.5d0*g*qsq+w/c)
ri(k,1,2)=-0.5d0*g*u
ri(k,1,3)=-0.5d0*g*v
ri(k,1,4)=-0.5d0*(g*w+1.0d0/c)
ri(k,1,5)=0.5d0*g
ri(k,2,1)=1.0d0-0.5d0*g*qsq
ri(k,2,2)=g*u
ri(k,2,3)=g*v
ri(k,2,4)=g*w
ri(k,2,5)=-g
ri(k,3,1)=0.5d0*(0.5d0*g*qsq-w/c)
ri(k,3,2)=-0.5d0*g*u
ri(k,3,3)=-0.5d0*g*v
ri(k,3,4)=-0.5d0*(g*w-1.0d0/c)
ri(k,3,5)=0.5d0*g
ri(k,4,1)=-u
ri(k,4,2)=1.0d0
ri(k,4,3)=0.0d0
ri(k,4,4)=0.0d0
ri(k,4,5)=0.0d0
ri(k,5,1)=-v
ri(k,5,2)=0.0d0
ri(k,5,3)=1.0d0
ri(k,5,4)=0.0d0
ri(k,5,5)=0.0d0
c
c ***** coefficients (alpha) *****
c
do l=1,5
al(k,l)=0.0d0
do m=1,5
al(k,l)=al(k,l)+ri(k,l,m)*dw(m)
end do
end do

al(k,6)=0.5d0*(delpc2*(ephi-phs)-(phs+ephi)*dw(1)+2*dw(6))/ephi
```

```
do m=1,nvar
al(k,m)=al(k,m)*csq
end do


enddo
c
c print *, (al(-1,l),l=1,5)
c print *, (al(0,l),l=1,5)
c print *, (al(imax,l),l=1,5)
c print *, (al(imax+1,l),l=1,5)
return
end
c***5***10********20********30********40********50********60********70********80
subroutine dis24(n,per,ep2,ep4,xyzm_bound,xyzp_bound,nproc)
use tvdmod
use incl3d

implicit none
integer :: n
integer :: nproc
logical :: per,logical,xyzm_bound,xyzp_bound

integer :: i,m

real(kind=rk),dimension(-1:ijkm,nvar) :: wm
real(kind=rk),dimension(-1:ijkm,nvar) :: ep2, ep4

c
c computes weights of 2nd-order TVD dissipation & 4th-order dissipation
c

do m=1,nvar
c...
do i=0,n
ep2(i,m) = akap2*max(th(i,m),th(i+1,m))
end do
do i=0,n
ep4(i,m) = al(i+1,m)-2.0d0*al(i,m)+al(i-1,m)
ep4(i,m) = ep4(i,m)* max( 0.0d0, (akap4-ep2(i,m)) )
end do
c
c if physical bc at two ends, linear extrapolation (uniform grids assumed)
c
if(xyzm_bound) then
if(per) then
if(nproc.eq.1) then
c
ceg periodic
ep4(1,m) = al(2,m)-2.0d0*al(1,m)+al(0,m)
c
c ep4(1,m) = al(2,m)-2.0d0*al(1,m)+al(n,m)
ep4(1,m) = ep4(1,m)* max( 0.0d0, (akap4-ep2(1,m)) )
endif
else
c
ceg non-periodic
c
ep4(1,m) = 2.0d0*al(2,m)-al(3,m)
ep4(1,m) = ep4(1,m)* max( 0.0d0, (akap4-ep2(1,m)) )
end if
ep4(0,m) = ep4(1,m)
else
c
ceg internal face
c
ep4(1,m) = al(2,m)-2.0d0*al(1,m)+al(0,m)
```

```
ep4(1,m) = ep4(1,m)* max( 0.0d0, (akap4-ep2(1,m)) )
end if
if(xyzp_bound) then
if(per) then
if(nproc.eq.1) then
c
ceg periodic
ep4(n,m) = al(n+1,m)-2.0d0*al(n,m)+al(n-1,m)
c
c ep4(n,m) = al(1,m)-2.0d0*al(n,m)+al(n-1,m)
ep4(n,m) = ep4(n,m)* max( 0.0d0, (akap4-ep2(n,m)) )
endif
else
c
ceg non-periodic
c
ep4(n,m) = 2.0d0*al(n-1,m)-al(n-2,m)
ep4(n,m) = ep4(n,m)* max( 0.0d0, (akap4-ep2(n,m)) )
end if
else
c
ceg internal face
c
ep4(n,m) = al(n+1,m)-2.0d0*al(n,m)+al(n-1,m)
ep4(n,m) = ep4(n,m)* max( 0.0d0, (akap4-ep2(n,m)) )
end if
end do

return
end
c***5***10********20********30********40********50********60********70********80
subroutine updis24(n,eff,r,xyz,ijkdim,per,mm,nn,l,
& xyzm_bound,xyzp_bound,nproc,tvdsmu1)
use tvdmod
use incl3d

implicit none
c
c Subroutine arguments.
c
integer :: ijkdim,n,mm,nn,l
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc

real(kind=rk),dimension(1-xhalo:nxd+xhalo,1-yhalo:nyd+yhalo,
& 1-zhalo:nzd+zhalo) :: xyz
c
integer :: i,ialloc,m
cnn...
real(kind=rk),dimension(-1:ijkm,nvar,nvar) :: r
real(kind=rk),dimension(-1:ijkm,nvar) :: eff
real(kind=rk) :: aaa,anu,con,e1,e2,e3,e4,e5,eps,gama,qf,ww
real(kind=rk),dimension(-1:ijkm,nvar)::wm,cf,cmp,ep2,ep4
real(kind=rk),dimension(-1:ijkm):: tvdsmu1
cnn..
real(kind=rk) :: e6

c
eps=1.d-7

c con=tvdsmu**2
c
c Allocate th, its values are set in uplim1-5
c
if ( .not.allocated(th) ) then
allocate (th(-1:ijkm,nvar),stat=ialloc)
endif
```

```
c
if(limiter.eq.1) call uplim1(n,per,xyzm_bound,xyzp_bound,nproc)
if(limiter.eq.2) call uplim2(n,per,xyzm_bound,xyzp_bound,nproc)
if(limiter.eq.3) call uplim3(n,per,xyzm_bound,xyzp_bound,nproc)
if(limiter.eq.4) call uplim4(n,per,xyzm_bound,xyzp_bound,nproc)
if(limiter.eq.5) call uplim5(n,per,xyzm_bound,xyzp_bound,nproc)
c
cnn...

c ***** Entropy correction term *****

do m=1,nvar
do i=0,n
con=tvdsmu1(i)**2
gama=0.0d0
anu=alam(i,m)
aaa =al(i,m)
c
c need gt:n+1

cc computation of the first correction term
gama=aaa*(gt(i+1,m)-gt(i,m))/(aaa**2+eps)
if (abs(anu).ge.tvdsmu1(i)) then
qf=abs(anu)
else
qf=(anu**2 + con)/(2.d0*tvdsmu1(i))
endif

cmp(i,m)=0.5d0*qf
cc

cc computation of the second correction term

cc ww=anu+cmp(i,m)*gama ! term in the original upwind scheme

ww=cmp(i,m)*gama ! term in the modified upwind scheme

if (abs(ww).ge.tvdsmu1(i)) then
qf=abs(ww)
else
qf=(ww**2 + con)/(2.d0*tvdsmu1(i))
endif
cf(i,m)=qf
enddo
enddo

c ***** Entropy correction ends *****


c
c *** compute weights of 2nd-order TVD dissipation & 4th-order
c dissipation
c
call dis24(n,per,ep2,ep4,xyzm_bound,xyzp_bound,nproc)
c
do i=0,n
c
c need gt:n+1
c


cc ***** original Upwind Scheme *****

c e1 = (cmp(i,1)*(gt(i,1)+gt(i+1,1))-cf(i,1)*al(i,1))
c & *ep2(i,1) + alam(I,1)*ep4(i,1)
c e2 = (cmp(i,2)*(gt(i,2)+gt(i+1,2))-cf(i,2)*al(i,2))
c & *ep2(i,2) + alam(I,2)*ep4(i,2)
c e3 = (cmp(i,3)*(gt(i,3)+gt(i+1,3))-cf(i,3)*al(i,3))
```

```
c & *ep2(i,3) + alam(I,3)*ep4(i,3)
c e4 = (cmp(i,4)*(gt(i,4)+gt(i+1,4))-cf(i,4)*al(i,4))
c & *ep2(i,4) + alam(I,4)*ep4(i,4)
c e5 = (cmp(i,5)*(gt(i,5)+gt(i+1,5))-cf(i,5)*al(i,5))
c & *ep2(i,5) + alam(I,5)*ep4(i,5)
c e6 = (cmp(i,6)*(gt(i,6)+gt(i+1,6))-cf(i,6)*al(i,6))
c & *ep2(i,6) + alam(I,6)*ep4(i,6)

cc **********************************

if(tvds.eq.1)then

cc ***** Symmetric Scheme *****

e1 = -2.0d0*cmp(i,1)*(al(i,1)-gt(i,1))
& *ep2(i,1)
e2 = -2.0d0*cmp(i,2)*(al(i,2)-gt(i,2))
& *ep2(i,2)
e3 = -2.0d0*cmp(i,3)*(al(i,3)-gt(i,3))
& *ep2(i,3)
e4 = -2.0d0*cmp(i,4)*(al(i,4)-gt(i,4))
& *ep2(i,4)
e5 = -2.0d0*cmp(i,5)*(al(i,5)-gt(i,5))
& *ep2(i,5)
e6 = -2.0d0*cmp(i,6)*(al(i,6)-gt(i,6))
& *ep2(i,6)

cc ****************************

elseif(tvds.eq.2)then

cc ***** modified Upwind Scheme *****

e1 = (cmp(i,1)*(gt(i,1)+gt(i+1,1))-(cf(i,1)+cmp(i,1))*al(i,1))
& *ep2(i,1) + alam(I,1)*ep4(i,1)
e2 = (cmp(i,2)*(gt(i,2)+gt(i+1,2))-(cf(i,2)+cmp(i,2))*al(i,2))
& *ep2(i,2) + alam(I,2)*ep4(i,2)
e3 = (cmp(i,3)*(gt(i,3)+gt(i+1,3))-(cf(i,3)+cmp(i,3))*al(i,3))
& *ep2(i,3) + alam(I,3)*ep4(i,3)
e4 = (cmp(i,4)*(gt(i,4)+gt(i+1,4))-(cf(i,4)+cmp(i,4))*al(i,4))
& *ep2(i,4) + alam(I,4)*ep4(i,4)
e5 = (cmp(i,5)*(gt(i,5)+gt(i+1,5))-(cf(i,5)+cmp(i,5))*al(i,5))
& *ep2(i,5) + alam(I,5)*ep4(i,5)
e6 = (cmp(i,6)*(gt(i,6)+gt(i+1,6))-(cf(i,6)+cmp(i,6))*al(i,6))
& *ep2(i,6) + alam(I,6)*ep4(i,6)

cc **********************************

endif
eff(i,1)=e1*r(i,1,1)+e2*r(i,1,2)+e3*r(i,1,3)+e4*r(i,1,4)+
& e5*r(i,1,5)+e6*r(i,1,6)
eff(i,2)=e1*r(i,2,1)+e2*r(i,2,2)+e3*r(i,2,3)+e4*r(i,2,4)+
& e5*r(i,2,5)+e6*r(i,2,6)
eff(i,3)=e1*r(i,3,1)+e2*r(i,3,2)+e3*r(i,3,3)+e4*r(i,3,4)+
& e5*r(i,3,5)+e6*r(i,3,6)
eff(i,4)=e1*r(i,4,1)+e2*r(i,4,2)+e3*r(i,4,3)+e4*r(i,4,4)+
& e5*r(i,4,5)+e6*r(i,4,6)
eff(i,5)=e1*r(i,5,1)+e2*r(i,5,2)+e3*r(i,5,3)+e4*r(i,5,4)+
& e5*r(i,5,5)+e6*r(i,5,6)
eff(i,6)=e1*r(i,6,1)+e2*r(i,6,2)+e3*r(i,6,3)+e4*r(i,6,4)+
& e5*r(i,6,5)+e6*r(i,6,6)
enddo
c
return
end
c***5***10********20********30********40********50********60********70********80
subroutine uplim1(n,per,xyzm_bound,xyzp_bound,nproc)
use tvdmod
```

```
use incl3d

implicit none
c
c Subroutine arguments.
c
integer :: n
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc
c
integer :: i,k,m
real(kind=rk) :: aal,aalm,amn,eps,ss,aalp,amn1,amn2,minmod1
& ,minmod2
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
c
c Computes minmod(jm,j)
c

if(tvds.eq.1)then

cc ***** Limiter 1 for symmetric scheme *****
eps=0.0000001d0
do m=1,nvar
do i=0,n+1
aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs((aal-aalm)/(aal+aalm+eps))
ss =sign(1.0d0,al(i,m))
amn1 = min(aal,ss*al(i-1,m))
amn2 = min(aal,ss*al(i+1,m))
minmod1 = ss*max(amn1,0.0d0)
minmod2 = ss*max(amn2,0.0d0)
gt(i,m)=minmod1+minmod2-al(i,m)

enddo
enddo
cc ****************************************

elseif(tvds.eq.2)then

cc ***** Limiter 1 for upwind scheme *****
eps=0.0000001d0
do m=1,nvar
do i=0,n+1
aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs(aal-aalm)/(aal+aalm+eps)
ss =sign(1.0d0,al(i,m))
amn =min(aal,ss*al(i-1,m))
gt(i,m)=ss*max(amn,0.0d0)
enddo
enddo

cc ***************************************

endif

c repeat the TH cal. later in DIS24 routine!!!
c
if(xyzm_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(m,k)=gt(n+m,k)
th(m,k)=th(n+m,k)
end do
endif
```

```
else
gt(-1,k) = gt(1,k)
gt(0,k) = gt(1,k)
th(-1,k) = th(1,k)
th(0,k) = th(1,k)
endif
enddo
end if
c
if(xyzp_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(n+1-m,k)=gt(-m+1,k)
th(n+1-m,k)=th(-m+1,k)
end do
endif
else
gt(n+1,k) = gt(n,k)
gt(n+2,k) = gt(n,k)
th(n+1,k) = th(n,k)
th(n+2,k) = th(n,k)
endif
enddo
end if
c
return
end
c***5***10********20********30********40********50********60********70********80
subroutine uplim2(n,per,xyzm_bound,xyzp_bound,nproc)
c
c Van Leer limiter
c
use tvdmod
use incl3d
c
implicit none
integer :: n
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc
c
integer :: i,k,m
real(kind=rk) :: aal,aalm,all,ar,den,eps,ss,amn
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
c

if(tvds.eq.1)then

cc ***** Limiter 2 for symmetric scheme *****
eps=0.0000001d0
do m=1,nvar
do i=0,n+1

aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=(abs((aal-aalm)/(aal+aalm+eps)))**3.d0
ss =sign(1.0d0,al(i,m))
amn =min(aal,ss*al(i-1,m),ss*al(i+1,m))
gt(i,m)=ss*max(amn,0.0d0)

enddo
enddo
cc ****************************************

elseif(tvds.eq.2)then
```

```
cc ***** Limiter 2 (Van Leer limiter) for upwind scheme *****

eps=0.0000001d0
do m=1,nvar
do i=0,n+1
aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs(aal-aalm)/(aal+aalm+eps)
all = al(i-1,m)
ar = al(i,m)
ss = all*ar
den = all + ar
gt(i,m)= 0.0d0
if(den.ne.0.0d0) gt(i,m)= ( ss + abs(ss) ) / den
enddo
enddo

cc *******************************************************

endif


if(xyzm_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(m,k)=gt(n+m,k)
th(m,k)=th(n+m,k)
end do
endif
else
gt(-1,k) = gt(1,k)
gt(0,k) = gt(1,k)
th(-1,k) = th(1,k)
th(0,k) = th(1,k)
endif
enddo
end if
c
if(xyzp_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(n+1-m,k)=gt(-m+1,k)
th(n+1-m,k)=th(-m+1,k)
end do
endif
else
gt(n+1,k) = gt(n,k)
gt(n+2,k) = gt(n,k)
th(n+1,k) = th(n,k)
th(n+2,k) = th(n,k)
endif
enddo
end if
c
return
end
c***5***10********20********30********40********50********60********70********80
subroutine uplim3(n,per,xyzm_bound,xyzp_bound,nproc)
use tvdmod
use incl3d

implicit none
integer :: n
```

```
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc
c
integer :: i,k,m,nupmax
real(kind=rk) :: aal,aalm,all,ar,del2,eps,gtt,ss,amn
cnn...
real(kind=rk),dimension(-1:ijkm,nvar) :: wm

if(tvds.eq.1)then

cc ***** Limiter 3 for symmetric scheme *****
eps=0.0000001d0
do m=1,nvar
do i=0,n+1

aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs((aal-aalm)/(aal+aalm+eps))
ss =sign(1.0d0,al(i,m))
amn =min(2.d0*aal,ss*2.d0*al(i-1,m),ss*2.d0*al(i+1,m)
& ,ss*0.5d0*(al(i-1,m)+al(i+1,m)))
gt(i,m)=ss*max(amn,0.0d0)

enddo
enddo
cc ****************************************

elseif(tvds.eq.2)then

cc ***** Limiter 3 (p41 HCY) for upwind scheme *****
c
del2=1.d-7
eps=0.0000001d0
c

do m=1,nvar
do i=0,n+1
aalm = abs(al(i-1,m))
aal = abs(al(i,m))
th(i,m)= (abs(aal-aalm)/(aal+aalm+eps))
all = al(i-1,m)
ar = al(i,m)
gtt = all*(ar*ar+del2)+ar*(all*all+del2)
gt(i,m)= gtt/(ar*ar+all*all+2.0d0*del2)
enddo
enddo

cc ************************************************

endif

nupmax = 0
if(xyzm_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,nupmax
gt(m,k)=gt(n+m,k)
th(m,k)=th(n+m,k)
end do
endif
else
gt(-1,k) = gt(1,k)
gt(0,k) = gt(1,k)
th(-1,k) = th(1,k)
th(0,k) = th(1,k)
endif
enddo
```

```
end if

if(xyzp_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,nupmax
gt(n+1-m,k)=gt(-m+1,k)
th(n+1-m,k)=th(-m+1,k)
end do
endif
else
gt(n+1,k) = gt(n,k)
gt(n+2,k) = gt(n,k)
th(n+1,k) = th(n,k)
th(n+2,k) = th(n,k)
endif
enddo
end if
c
return
end
c***5***10********20********30********40********50********60********70********80
subroutine uplim4(n,per,xyzm_bound,xyzp_bound,nproc)
c
c Superbee
c
use tvdmod
use incl3d

implicit none
integer :: n
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc
c
integer :: i,k,m,nupmax
real(kind=rk) :: aal,aalm,amn1,amn2,eps,ss
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
c
c Limiter qhat = minmod(jm,j,jp)
c
eps=0.0000001d0
do m=1,nvar
do i=0,n+1
aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs(aal-aalm)/(aal+aalm+eps)
ss =sign(1.0d0,al(i,m))
amn1 =min(2.0d0*aal,ss*al(i-1,m))
amn2 =min(aal,2.0d0*ss*al(i-1,m))
gt(i,m)=ss*max(amn1,amn2,0.0d0)
enddo
enddo
c
nupmax = 0
if(xyzm_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,nupmax
gt(m,k)=gt(n-2*nupmax+m,k)
th(m,k)=th(n-2*nupmax+m,k)
end do
endif
else
gt(-1,k) = gt(1,k)
gt(0,k) = gt(1,k)
```

```
th(-1,k) = th(1,k)
th(0,k) = th(1,k)
endif
enddo
end if
c
if(xyzp_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,nupmax
gt(n+1-m,k)=gt(2*nupmax-m+1,k)
th(n+1-m,k)=th(2*nupmax-m+1,k)
end do
endif
else
gt(n+1,k) = gt(n,k)
gt(n+2,k) = gt(n,k)
th(n+1,k) = th(n,k)
th(n+2,k) = th(n,k)
endif
enddo
end if
c
return
c
end
c***5***10********20********30********40********50********60********70********80
subroutine uplim5(n,per,xyzm_bound,xyzp_bound,nproc)
use tvdmod
use incl3d

implicit none
integer :: n
logical :: per,xyzm_bound,xyzp_bound
integer :: nproc
c
integer :: i,k,m
real(kind=rk) :: aal,aalm,amn,eps,ss
real(kind=rk),dimension(-1:ijkm,nvar) :: wm
c
c Colella-Woodward Limiter
c Computes minmod(2*jm,2*j,0.5d0*(j+jm))
c
eps=0.0000001d0
do m=1,nvar
do i=0,n+1
aalm =abs(al(i-1,m))
aal =abs(al(i,m))
th(i,m)=abs(aal-aalm)/(aal+aalm+eps)
ss =sign(1.0d0,al(i,m))
amn =min(2.0d0*aal,ss*2.0d0*al(i-1,m)
& ,0.5d0*ss*(al(i-1,m)+al(i,m)))
gt(i,m)=ss*max(amn,0.0d0)
enddo
enddo
c
if(xyzm_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(m,k)=gt(n+m,k)
th(m,k)=th(n+m,k)
end do
endif
```

```
else
gt(-1,k) = gt(1,k)
gt(0,k) = gt(1,k)
th(-1,k) = th(1,k)
th(0,k) = th(1,k)
endif
enddo
end if
c
if(xyzp_bound) then
do k=1,nvar
if(per) then
if (nproc.eq.1) then
do m=-1,0
gt(n+1-m,k)=gt(-m+1,k)
th(n+1-m,k)=th(-m+1,k)
end do
endif
else
gt(n+1,k) = gt(n,k)
gt(n+2,k) = gt(n,k)
th(n+1,k) = th(n,k)
th(n+2,k) = th(n,k)
endif
enddo
end if
c
return
end
c***5***10********20********30********40********50********60********70********80
```