**main.F**

c PCOM and BCOM in eta-coordinates

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'calendar.h'

include 'prog.h'

include 'grdvar.h'

integer mode\_c,mode\_t

namelist /contrl/ runlen,restrt

namelist /tsteps/ dtts,dtuv,dtsf

namelist /mixing/ am,ah,kappa\_m,kappa\_h,cdbot

namelist /filter/ afb1,afc1,aft1

namelist /io/ io\_tsuvp,io\_restr

c set scalar quantities

call setcon

c set resolution, t/u mask and the j-depended parameters

call grdvar

c initialization

call inirun

#ifdef gm90

call isopyi

#endif

10 year = 1 + (month-1)/12

mth = month - (month-1)/12\*12

c set euler forward/backward scheme at the beginning of every month

leapfrog\_t = .false.

leapfrog\_c = .false.

leapfrog\_b = .false.

euler\_back = .true.

c daily cycle

do 20 day=1,daypm(mth)

call rho\_ref

c interpolate the observed monthly mean data

call interp

do 30 mode\_t = 1,nss

call readyt

do 40 mode\_c = 1,ncc

call readyc

call barotr

call bclinc

40 continue

call tracer

call convect

30 continue

call diag

20 continue

end

**tracer.F**

real wka(imt,jmt,km),wkb(imt,jmt,km),pbar(imt,jmt),stf(imt,jmt)

common /works/ a(imt,jmt),b(imt,jmt),c(imt,jmt)

common /works/ wa(imt,jmt),wb(imt,jmt)

common /works/ flxa(imt,jmt),flxb(imt,jmt),dt(imt,jmt)

real fx,fy

common /wisop/ fx(imt,jmt),fy(imt,jmt)

#ifdef gm90

aidif = p5

#else

aidif = c1

#endif

c calculate time-averaged pbt and mass advections

do j=1,jmt

do i=1,imt

pmtp(i,j) = pmtp(i,j)\*onbc

enddo

enddo

do k=1,km

do j=1,jmt

do i=1,imt

ump(i,j,k) = ump(i,j,k)\*oncc (oncc=1/2)

vmp(i,j,k) = vmp(i,j,k)\*oncc

enddo

enddo

enddo

if(leapfrog\_t)then

do j=1,jmt

do i=1,imt

pbar(i,j) = (pmtp(i,j) + pmtm(i,j))\*p5

enddo

enddo

do k=1,km

do j=1,jmt

do i=1,imt

wka(i,j,k) = (ump(i,j,k)+umm(i,j,k))\*p5

wkb(i,j,k) = (vmp(i,j,k)+vmm(i,j,k))\*p5

enddo

enddo

enddo

else

do j=1,jmt

do i=1,imt

pbar(i,j) = pmtp(i,j)

enddo

enddo

do k=1,km

do j=1,jmt

do i=1,imt

wka(i,j,k) = ump(i,j,k)

wkb(i,j,k) = vmp(i,j,k)

enddo

enddo

enddo

endif

c calculate vertical mass advection

call upwelling(wka,wkb,w)

c calculate 1/2 mass advection across W & S face of T cells

c du & dv are used temporary as working arrays

do k=1,km

do j=2,jmt

do i=2,imt

du(i,j,k) = p25\*(wka(i-1,j,k)+wka(i-1,j-1,k))

dv(i,j,k) = p25\*(wkb(i,j-1,k)+wkb(i-1,j-1,k))

enddo

enddo

enddo

c calculate 2\*xbar(pbar) & 2\*ybar(pbar)

do j=1,jmt

do i=2,imt

fx(i,j) = (pbar(i,j)+pbar(i-1,j))\*p5

enddo

enddo

do j=2,jmt

do i=1,imt

fy(i,j) = (pbar(i,j)+pbar(i,j-1))\*p5

enddo

enddo

#ifdef gm90

c diffusion coefficients

call isopyc

do k=1,km

do j=1,jmt

do i=1,imt

wkb(i,j,k) = kappa\_h + ahisop\*k3(i,k,j,3)

enddo

enddo

enddo

#endif

c

if(leapfrog\_t) then

dtts2 = c2dtts

else

dtts2 = dtts

endif

c solve for one tracer at a time

c n = 1 => temperature

c n = 2 => salinity

do 1000 n=1,nt

c advections

do 100 k=1,km

do j=2,jmm

do i=2,imt

a(i,j) = du(i,j,k)\*(t(i,j,k,n,tau)-t(i-1,j,k,n,tau))

enddo

enddo

do j=2,jmt

do i=2,imm

b(i,j) = dv(i,j,k)\*(t(i,j,k,n,tau)-t(i,j-1,k,n,tau))

enddo

enddo

if(k.eq.1)then

do j=2,jmm

do i=2,imm

wa(i,j) = c0

enddo

enddo

else

do j=2,jmm

do i=2,imm

wa(i,j) = wb(i,j)

enddo

enddo

endif

c

do j=2,jmm

do i=2,imm

if(k.ge.itn(i,j))then

wb(i,j) = c0

else

wb(i,j) = w(i,j,k+1)\*p5\*(t(i,j,k,n,tau)-t(i,j,k+1,n,tau))

endif

enddo

enddo

c

do j=2,jmm

do i=2,imm

wka(i,j,k) = - rdxt(j)\*(a(i+1,j)+a(i,j))

& - rdyt(j)\*(b(i,j+1)+b(i,j))

& - rdz(k) \*(wa(i,j)+wb(i,j))

enddo

enddo

c

100 continue

c diffusions

#ifdef gm90

c compute the isopycnal/dipycnal mixing

c xz and yz isopycnal diffusive flux are solved explicitly;

c while zz component will be solved implicitly.

**call isoflux (wka,n)** (Advective isopycnal flux is added here)

#else

c horizontal diffusion

do 200 k=1,km

do j=2,jmm

do i=2,imt

a(i,j) = fx(i,j)\*(t(i,j,k,n,taum)-t(i-1,j,k,n,taum))\*

& tmask(i,j,k)\*tmask(i-1,j,k)

enddo

enddo

c

do j=2,jmt

do i=2,imm

b(i,j) = fy(i,j)\*(t(i,j,k,n,taum)-t(i,j-1,k,n,taum))\*

& tmask(i,j,k)\*tmask(i,j-1,k)

enddo

enddo

c

do j=2,jmm

do i=2,imm

wka(i,j,k) = wka(i,j,k) + ah\*( sdxt(j)\*(a(i+1,j)-a(i,j))+

& r1a(j)\*b(i,j+1)-r1b(j)\*b(i,j) )

(Standard diffusion added here!!)

enddo

enddo

200 continue

c

#endif

c vertical diffusion

do 300 k=1,km

if(k.eq.1)then

do j=2,jmm

do i=2,imm

flxa(i,j) = c0

enddo

enddo

else

do j=2,jmm

do i=2,imm

flxa(i,j) = flxb(i,j)

enddo

enddo

endif

c

do j=2,jmm

do i=2,imm

if(k.eq.itn(i,j))then

flxb(i,j) = c0

else if(k.lt.itn(i,j))then

flxb(i,j) = (t(i,j,k,n,taum)-t(i,j,k+1,n,taum))\*

#ifdef gm90

& gravr\*wkb(i,j,k)\*rdzw(k)

#else

& gravr\*kappa\_h \*rdzw(k)

#endif

else

flxb(i,j) = c0

endif

enddo

enddo

c

do j=2,jmm

do i=2,imm

wka(i,j,k) = wka(i,j,k) + rdz(k)\*(flxa(i,j)-flxb(i,j))\*aidif

enddo

enddo

300 continue

c + surface forcing

if(n.eq.1)then

c set sea surface heat flux b.c.

c

c dT/dt = D(T\*-T)/Cp/g/rho/dz

do j=2,jmm

do i=2,imm

ccc stf(i,j) = ddd(i,j)\*(bct(i,j)-t(i,j,1,n,tau))

stf(i,j) = gamma\_t \*(bct(i,j)-t(i,j,1,n,tau))

wka(i,j,1) = wka(i,j,1) + stf(i,j)\*rdz(1)\*aidif

enddo

enddo

c

else

c----------------------------------------------------

c set natural or restoring b.c. for salinity

c------------------------------------------------------

do j=2,jmm

do i=2,imm

#ifdef snbc

stf(i,j) = t(i,j,1,n,tau)\*emp(i,j)

#else

stf(i,j) = gamma\_s\*(bcs(i,j)-t(i,j,1,n,tau))\*pbar(i,j)\*dz(1)

#endif

wka(i,j,1) = wka(i,j,1) + stf(i,j)\*rdz(1)\*aidif

enddo

enddo

c

endif

c compute T&S at tau+1 time level. Doesn't include implicit diffusion

do k=1,km

do j=2,jmm

do i=2,imm

wka(i,j,k) = t(i,j,k,n,taum) +

& wka(i,j,k)/pbar(i,j)\*tmask(i,j,k)\*dtts2

enddo

enddo

enddo

c add the component due to implicit diffusion

c

#ifdef gm90

call invtri (wka,stf,wkb,aidif,dtts2,pbar)

#endif

c set T > -1.5 temporarily since no seaice model

if(n.eq.1)then

do k=1,km

do j=2,jmm

do i=2,imm

wka(i,j,k) = dmax1(tbice, wka(i,j,k))

enddo

enddo

enddo

endif

c T&S=wka & Filter & periodic b.c

if(leapfrog\_t) then

do k=1,km

do j=2,jmm

do i=2,imm

t(i,j,k,n,taum) = t(i,j,k,n,tau)

#ifdef asselin\_t

& \*aft2 + aft1\*(t(i,j,k,n,taum)+wka(i,j,k))

#endif

enddo

enddo

enddo

endif

c

do k=1,km

do j=2,jmm

do i=2,imm

t(i,j,k,n,tau) = wka(i,j,k)

enddo

enddo

c

do j=2,jmm

t(1 ,j,k,n,tau) = t(imm,j,k,n,tau)

t(imt,j,k,n,tau) = t(2 ,j,k,n,tau)

t(1 ,j,k,n,taum)= t(imm,j,k,n,taum)

t(imt,j,k,n,taum)= t(2 ,j,k,n,taum)

enddo

enddo

c

1000 continue

c

if(.not.leapfrog\_t) leapfrog\_t = .true.

c

return

end

**isopyc.F**

#ifdef gm90

c

c =================

**subroutine isopyi**

c =================

c Initialization for isopycnal mixing scheme

c

c Redi/Cox version + Gent\_McWilliams version

c

cifdef isopycxixspatialvar

c dciso1 = isopycnal tracer diffusivity coeffs modified based

c on the slopes of the isopycnal surfaces on the east face

c of "T" cells.

c dciso2 = isopycnal tracer diffusivity coeffs modified based

c on the slopes of the isopycnal surfaces on the north face

c of "T" cells.

c dslope = half length of the interval in which "ahisop" changes

c with a steep slope from about 0.9\*"ahisop" to about

c 0.1\*"ahisop"

c slopec = slope at which "ahisop" is equal to half of its

c original value

cendif

c

c dptlim = depth limits for the reference pressure levels(in cm).

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

c

real dptmid,t1,t2

c

c-----------------------------------------------

c USER INPUT

c initialize variables (all mixing units are cm\*\*2/sec.)

c------------------------------------------

c

slmxr = 100.0

ahisop = 1.e7

c

c define the isopycnal thickness diffusion coefficient

c

athkdf = 1.0e7

c

c reference pressure level intervals are defined (see isopyc.h").

c "dptlim" must have "nrpl+1" elements (see "param.h"). also,

c "dptlim" are in cm.

c

c REMARK: the first and the last elements of "dptlim" must be the

c depth at the top (0cm) and the maximum bottom depth,

c respectively. Also, the elements of "dptlim" must be in

c increasing order.

c

c dptlim(1) = 0.0e2

c dptlim(2) = 1000.0e2

c dptlim(3) = 2000.0e2

c dptlim(4) = 3000.0e2

c dptlim(5) = 4000.0e2

c dptlim(6) = 5000.0e2

c

c

c-----------------------------------------------------------

cdetermine the isopycnal reference pressure levels for the "t"

cgrid point levels,using the depths at the "t" grid points as the

c reference depth (pressure)

c---------------------------------------------------

c

c do k=1,km

c do m=2,nrpl+1

c if (z0(k).gt.dptlim(m-1) .and. z0(k).le.dptlim(m)) then

c kisrpl(k) = m-1

c go to 101

c endif

c enddo

101 continue

c

c if (kisrpl(k) .lt. 1 .or. kisrpl(k) .gt. nrpl) then

c write (6,9100) kisrpl(k), k

c stop 9999

c endif

c enddo

9100 format (/,' =>Error: kisrpl is ',i3,' at k ',1x,i3,' in isopyc.F')

c

c----------------------------------------------------

c the indices used in isopycnal mixing indicating the location of

c the reference pressure levels in the 20-level table of polynomial

c expansion variables are computed

c

c REMARK: because the polynomial expansion coefficients are

c functions of the reference potential temperature and

c salinity profiles, at the reference pressure level

c the corresponding potential temperature and salinity

c values will be used.

c-------------------------------------------------------

c

c do m=1,nrpl

c krplin(m) = 0

c enddo

c

c do m=2,nrpl+1

c dptmid = 0.5\*(dptlim(m-1)+dptlim(m))

c if (dptmid .le. z0(1)) then

c krplin(m-1) = 1

c elseif (dptmid .gt. z0(km)) then

c krplin(m-1) = km

c elseif (dptmid.gt.z0(1) .and. dptmid.le.z0(km)) then

c do k=2,km

c if (z0(k) .ge. dptmid) then

c t1 = z0(k)-dptmid

c t2 = dptmid-z0(k-1)

c if (t1 .gt. t2) then

c krplin(m-1) = k-1

c else

c krplin(m-1) = k

c endif

c go to 102

c endif

c enddo

102 continue

c endif

c if (krplin(m-1) .lt. 1 .or. krplin(m-1) .gt. km) then

c write (6,9110) krplin(m-1),m-1

c endif

c enddo

c

ccc write (6,96) (kisrpl(k),k=1,km)

ccc write (6,97) (krplin(m),m=1,nrpl)

c

c------------------------------------------------

c the isopycnal diffusion coefficient may be a function of depth. in

c the default configuration, the isopycnal diffusion coefficient is

c a constant: "fzisop", which multiplies "ahisop", is set to unity.

c if "ahisop" varies in the vertical, "fzisop" should contain this

c variation. the value of "ahisop" should be adjusted accordingly.

c-------------------------------------------------------

c

do k=1,km

fzisop(k) = c1

enddo

c

ccc write (6,98)

ccc write (6,99) (fzisop(k),k=1,km)

c

96 format (/,' isopycnal reference pressure levels (kisrpl) = ',

& 20(1x,i4))

97 format (/,' reference pressure level indices (krplin) = ',

& 20(1x,i4))

98 format (/,' vertical variation of "ahisop" (fzisop) = ')

99 format (5(1x,e12.6))

9110 format (/,' =>Error: krplin is ',i3,' at m ',1x,i3)

c

c

c------------------------------------------------------

c set reference depth for calculation of rhoi (m)

c-------------------------------------------------------

do k=1,km

kref(k) = z0(k)\*0.01

enddo

do k=1,km

rdz0(k) = c1/dz0(k)

enddo

c initialize arrays

c

do j=1,jmt

do k=1,km

do i=1,imt

K1(i,k,j,3) = c0

K3(i,k,j,1) = c0

K3(i,k,j,2) = c0

K3(i,k,j,3) = c0

K2(i,k,j,3) = c0

adv\_vetiso(i,k,j) = c0

adv\_vntiso(i,k,j) = c0

enddo

enddo

enddo

c

do j=1,jmt

do k=1,km

do i=1,imt

rhoi(i,k,j,xup) = c1

rhoi(i,k,j,xmd) = c1

rhoi(i,k,j,xlo) = c1

enddo

enddo

enddo

c

do m=1,3

do j=1,jmt

do k=1,km+1

do i=1,imt

e(i,k,j,m) = c0

enddo

enddo

enddo

enddo

c

do j=1,jmt

do k=0,km

do i=1,imt

adv\_vbtiso(i,k,j) = c0

enddo

enddo

enddo

return

end

c

**subroutine isopyc**

c

c Compute the isopycnal mixing tensor components and the

c isopycnal advection velocities which parameterize the effect

c of eddies on the isopycnals.

c Mixing tensor "K" is ...

c

c | 1.0 K1(,,,2) K1(,,,3) |

c | |

c K = | K2(,,,1) 1.0 K2(,,,3) |

c | |

c | K3(,,,1) K3(,,,2) K3(,,,3) |

c

c where K1(,,,2) and K2(,,,1) are set to 0.0 (neglected)

c output:

c rhoi = density at tau-1 referenced to pressure levels

c K1 = tensor components (1,2), (1,3) centered on east face

c of "T" cells

c K2 = tensor components (2,1), (2,3) centered on north face

c of "T" cells

c K3 = tensor components (3,1), (3,2), (3,3) centered on

c bottom face of "T" cells

cifdef gent\_mcwilliams

c adv\_vetiso = isopycnal advective vel on east face of "T" cell

c adv\_vntiso = isopycnal advective vel on north face of "T" cell

c (Note: this includes the cosine factor as in "adv\_vnt")

c adv\_vbtiso = isopycnal advective vel on bottom face of "T" cell

cendif

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'prog.h'

include 'isopyc.h'

real t0,s0,dens

integer mm1,mm2

c compute normalized densities for each isopycnal reference pressure

c level using a 3rd order polynomial fit to the equation of state.

c for each isopycnal reference pressure level, the same reference

c potential temperature, reference salinity and expansion coeff

c values are used at all of the vertical levels.

c

c Note: this density is used for the mixing tensor in both the

c Redi/Cox and Gent/McWilliams options

cxjin calculate densities using unesco(1981) state equation

do k=1,km

do j=2,jmm

do i=2,imm

if(tmask(i,j,k).gt.c0)then

mm1 = max(k-1,1)

mm2 = min(k+1,km)

t0 = t(i,j,k,1,tau)

s0 = t(i,j,k,2,tau)

rhoi(i,k,j,xup) = dens(t0,s0,kref(mm1))

rhoi(i,k,j,xmd) = dens(t0,s0,kref(k ))

rhoi(i,k,j,xlo) = dens(t0,s0,kref(mm2))

endif

enddo

enddo

enddo

do j=2,jmm

call setbcx (rhoi(1,1,j,xup), imt, km)

call setbcx (rhoi(1,1,j,xmd), imt, km)

call setbcx (rhoi(1,1,j,xlo), imt, km)

enddo

c evaluate K2(,,3) centered on the northern face of "T" cells

call k2\_3

c evaluate K1(,,3) centered on eastern face of "T" cells

call k1\_3

c evaluate K3(,,1..3) centered on bottom face of "T" cells

call k3\_123

c compute isopycnal advective velocities for tracers

call isoadv

return

end

c

c ===============

**subroutine k1\_3**

c ===============

c compute "K1(,,3)" at the center of the eastern face of "T" cells

c use "c1e10" to keep the exponents in range.

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

c

real c1e10,eps,fxd,fxe,fxc,fxa,fxb,chkslp,olmask

c set local constants

c1e10 = 1.0e10

eps = 1.0e-25

c

c---------------------------------------------------

c d(rho\_barx\_barz)/dz centered on eastern face of "t" cells

c Note: values involving ocean surface and ocean bottom are

c estimated afterwards using a linear extrapolation

c-----------------------------------------------

c

do j=2,jmm

do k=2,km-1

fxd = c1e10\*p25\*rdz0(k)

do i=1,imm

e(i,k,j,3) = fxd\*(rhoi(i ,k-1,j,xlo) - rhoi(i ,k+1,j,xup)

& +rhoi(i+1,k-1,j,xlo) - rhoi(i+1,k+1,j,xup))

enddo

enddo

enddo

c

c

c---------------------------------------------------

c linearly extrapolate densities to ocean surface for calculation

c of d(rho\_barx\_barz)/dz involving level 1.

c

c REMARK: requires min(kmt(i,jrow)) = 2 cells in ocean.

c-------------------------------------------------

c

k = 1

fxd = c1e10\*rdz0(k)

fxe = dz0(k) + p5\*dz0(k+1)

do j=2,jmm

do i=1,imm

fxa = p5\*(rhoi(i,k+1,j,xup) + rhoi(i+1,k+1,j,xup))

fxb = p5\*(rhoi(i,k ,j,xmd) + rhoi(i+1,k ,j,xmd))

fxc = rdzw(k)\*(fxb\*fxe - fxa\*p5\*dz0(k))

e(i,k,j,3) = fxd\*(fxc - p5\*(fxa+fxb))

enddo

enddo

c

c---------------------------------------------------

c linearly extrapolate densities to ocean bottom for calculation

c of d(rho\_barx\_barz)/dz involving bottom level.

c------------------------------------------------

c

do j=2,jmm

do i=1,imm

e(i,km,j,3) = c0

enddo

enddo

c

do j=2,jmm

do i=1,imm

k = min(itn(i,j),itn(i+1,j))

if (k .ne. 0) then

fxe = dz0(k) + p5\*dz0(k-1)

fxa = p5\*(rhoi(i,k-1,j,xlo) + rhoi(i+1,k-1,j,xlo))

fxb = p5\*(rhoi(i,k ,j,xmd) + rhoi(i+1,k ,j,xmd))

fxc = rdzw(k-1)\*(fxb\*fxe - fxa\*p5\*dz0(k))

e(i,k,j,3) = rdz0(k)\*c1e10\*(p5\*(fxa+fxb) - fxc)

endif

enddo

enddo

c

c----------------------------------------------

c "e(,,,1)" = d(rho)/dx centered on east face of "T" cells

c "e(,,,2)" = d(rho\_barx\_bary)/dy centered on east face of "T" cells

c--------------------------------------------

c

do j=2,jmm

do k=1,km

do i=1,imm

e(i,k,j,1) = tmask(i,j,k)\*tmask(i+1,j,k)\*rdxt(j)\*c1e10\*

& (rhoi(i+1,k,j,xmd) - rhoi(i,k,j,xmd))

e(i,k,j,2) = p25\*rdy\*c1e10\*(

& rhoi(i ,k,j+1,xmd) - rhoi(i ,k,j-1,xmd)

& + rhoi(i+1,k,j+1,xmd) - rhoi(i+1,k,j-1,xmd))

enddo

enddo

enddo

c

c------------------------------------------------------

c if any one of the 4 neighboring corner grid points is a land point,

c set "e(i,k,j,2)" to zero. note that "e(i,k,j,2)" will be used

c only in the slope check.

c-----------------------------------------------------

c

do j=2,jmm

do k=1,km

do i=1,imm

olmask = tmask(i,j-1,k)\*tmask(i,j+1,k)\*tmask(i+1,j-1,k)

& \*tmask(i+1,j+1,k)

if (olmask .eq. c0) e(i,k,j,2) = c0

enddo

enddo

enddo

c

c-------------------------------------------------------

c impose zonal boundary conditions at "i"=1 and "imt"

c---------------------------------------------------

do j=2,jmm

call setbcx (e(1,1,j,1), imt, km)

call setbcx (e(1,1,j,2), imt, km)

call setbcx (e(1,1,j,3), imt, km)

enddo

c

c

c-----------------------------------------------------

c compute "K1", using "slmxr" to limit vertical slope of isopycnal

c to guard against numerical instabilities.

c--------------------------------------------------------

c

cjxz-----------------------------------------------------

c# ifdef isopycmixspatialvar

c do i=1,imt

c fxa = c0

c fxb = sign(1.,e(i,k,j,3))/(abs(e(i,k,j,3))+eps)

c slope = fxb\*sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)

c if (slope .le. c0 .and. slope .ge. (-c1/slmxr)) then

c fxa = p5\*(c1+tanh((slope+slopec)/dslope))

c endif

c dciso1(i,k,j) = fxa

c K1(i,k,j,3) = -fxb\*e(i,k,j,1)\*dciso1(i,k,j)

c enddo

c# else # endif

cjxz------------------------------------------------

do j=2,jmm

do k=1,km

do i=1,imt

cjxz chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr\*dtxsqr(k)

chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr

if (e(i,k,j,3) .gt. chkslp) e(i,k,j,3) = chkslp

enddo

do i=1,imt

K1(i,k,j,3) = (-e(i,k,j,1)\*e(i,k,j,3)\*fzisop(k))

& /(e(i,k,j,3)\*\*2+eps)

enddo

enddo

enddo

return

end

c

c ===============

**subroutine k2\_3**

c ===============

c=========================================

c compute "K2(,,3)" at the center of the northern face of "T" cells

c use "c1e10" to keep the exponents in range.

c===========================================

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

c

real c1e10,eps,fxd,fxe,fxc,fxa,fxb,chkslp,olmask

c

c----------------------------------------------------

c set local constants

c---------------------------------------------------

c1e10 = 1.0e10

eps = 1.0e-25

c

c------------------------------------------------

c d(rho\_bary\_barz)/dz centered on northern face of "T" cells

c Note: values involving ocean surface and ocean bottom are

c estimated afterwards using a linear extrapolation

c------------------------------------------------

c

do j=2,jmm

do k=2,km-1

fxd = c1e10\*p25\*rdz0(k)

do i=2,imm

e(i,k,j,3) = fxd\*(rhoi(i,k-1,j ,xlo) - rhoi(i,k+1,j ,xup)

& +rhoi(i,k-1,j+1,xlo) - rhoi(i,k+1,j+1,xup))

enddo

enddo

enddo

c

c-------------------------------------------------

c linearly extrapolate densities to ocean surface for calculation

c of d(rho\_bary\_barz)/dz involving level 1.

c------------------------------------------------

c

k = 1

fxd = c1e10\*rdz0(k)

fxe = dz0(k) + p5\*dz0(k+1)

do j=2,jmm

do i=2,imm

fxa = p5\*(rhoi(i,k+1,j,xup) + rhoi(i,k+1,j+1,xup))

fxb = p5\*(rhoi(i,k ,j,xmd) + rhoi(i,k ,j+1,xmd))

fxc = rdzw(k)\*(fxb\*fxe - fxa\*p5\*dz0(k))

e(i,k,j,3) = fxd\*(fxc - p5\*(fxa+fxb))

enddo

enddo

c

c--------------------------------------------------

c linearly extrapolate densities to ocean bottom for calculation

c of d(rho\_bary\_barz)/dz involving bottom level.

c--------------------------------------------------

c

do j=2,jmm

do i=2,imm

e(i,km,j,3) = c0

enddo

enddo

c

do j=2,jmm

do i=2,imm

k = min(itn(i,j),itn(i,j+1))

if (k .ne. 0) then

fxe = dz0(k) + p5\*dz0(k-1)

fxa = p5\*(rhoi(i,k-1,j,xlo) + rhoi(i,k-1,j+1,xlo))

fxb = p5\*(rhoi(i,k ,j,xmd) + rhoi(i,k ,j+1,xmd))

fxc = rdzw(k-1)\*(fxb\*fxe - fxa\*p5\*dz0(k))

e(i,k,j,3) = rdz0(k)\*c1e10\*(p5\*(fxa+fxb) - fxc)

endif

enddo

enddo

c

c--------------------------------------------

c "e(,,,1)" = d(rho\_barx\_bary)/dx centered on north face of "T" cells

c "e(,,,2)" = d(rho)/dy on north face of "T" cells

c-----------------------------------------------

c

do j=2,jmm

do k=1,km

do i=2,imm

e(i,k,j,1) = rdxt(j)\*p25\*c1e10\*(

& rhoi(i+1,k,j+1,xmd) - rhoi(i-1,k,j+1,xmd)

& + rhoi(i+1,k,j ,xmd) - rhoi(i-1,k,j ,xmd))

e(i,k,j,2) = tmask(i,j,k)\*tmask(i,j+1,k)\*rdy\*c1e10

& \*(rhoi(i,k,j+1,xmd) - rhoi(i,k,j,xmd))

enddo

enddo

enddo

c

c-----------------------------------------

c if any one of the 4 neighboring corner grid points is a land point,

c set "e(i,k,j,1)" to zero. note that "e(i,k,j,1)" will be used

c only in the slope check.

c----------------------------------------------

c

do j=2,jmm

do k=1,km

do i=2,imm

olmask = tmask(i-1,j+1,k)\*tmask(i+1,j+1,k)\*tmask(i-1,j,k)

& \*tmask(i+1,j,k)

if (olmask .eq. c0) e(i,k,j,1) = c0

enddo

enddo

enddo

c

c----------------------------------------------------

c impose zonal boundary conditions at "i"=1 and "imt"

c---------------------------------------------

c

do j=2,jmm

call setbcx (e(1,1,j,1), imt, km)

call setbcx (e(1,1,j,2), imt, km)

call setbcx (e(1,1,j,3), imt, km)

enddo

c

c

c------------------------------------------------

c compute "K2", using "slmxr" to limit vertical slope of isopycnal

c to guard against numerical instabilities.

c-------------------------------------------

c

c# ifdef isopycmixspatialvar

c do i=1,imt

c fxa = c0

c fxb = sign(1.,e(i,k,j,3))/(abs(e(i,k,j,3))+eps)

c slope = fxb\*sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)

c if (slope .le. c0 .and. slope .ge. (-c1/slmxr)) then

c fxa = p5\*(c1+tanh((slope+slopec)/dslope))

c endif

c dciso2(i,k,j) = fxa

c K2(i,k,j,3) = -fxb\*e(i,k,j,2)\*dciso2(i,k,j)

c enddo

c# else # endif

cjxz--------------------------------------------

do j=2,jmm

do k=1,km

do i=1,imt

cjxz chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr\*dtxsqr(k)

chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr

if (e(i,k,j,3) .gt. chkslp) e(i,k,j,3) = chkslp

enddo

do i=1,imt

K2(i,k,j,3) = (-e(i,k,j,2)\*e(i,k,j,3)\*fzisop(k))

& /(e(i,k,j,3)\*\*2+eps)

enddo

enddo

enddo

return

end

c

c =================

**subroutine k3\_123**

c =================

c

c==================================================

c compute K2(,,,1:3) at the center of the bottom face of "T" cells

c use "c1e10" to keep the exponents in range.

c==================================================

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

c

real c1e10,eps,chkslp,ahfctr,fxa

c

c--------------------------------------------------

c set local constants

c--------------------------------------------------

c1e10 = 1.0e10

eps = 1.0e-25

c

c

do j=2,jmm

do k=2,km

do i=2,imm

e(i,k-1,j,1) = rdxt(j)\*p25\*c1e10

& \*(tmask(i-1,j,k-1)\*tmask(i,j,k-1)\*(rhoi(i,k-1,j,xlo)

& -rhoi(i-1,k-1,j,xlo))

& +tmask(i,j,k-1)\*tmask(i+1,j,k-1)\*(rhoi(i+1,k-1,j,xlo)

& -rhoi(i,k-1,j,xlo))

& +tmask(i-1,j,k)\*tmask(i,j,k)\*(rhoi(i,k,j,xmd)

& -rhoi(i-1,k,j,xmd))

& +tmask(i,j,k)\*tmask(i+1,j,k)\*(rhoi(i+1,k,j,xmd)

& -rhoi(i,k,j,xmd)))

c

e(i,k-1,j,2) = rdy\*p25\*c1e10

& \*(tmask(i,j-1,k-1)\*tmask(i,j,k-1)\*(rhoi(i,k-1,j,xlo)

& -rhoi(i,k-1,j-1,xlo))

& +tmask(i,j,k-1)\*tmask(i,j+1,k-1)\*(rhoi(i,k-1,j+1,xlo)

& -rhoi(i,k-1,j,xlo))

& +tmask(i,j-1,k)\*tmask(i,j,k)\*(rhoi(i,k,j,xmd)

& -rhoi(i,k,j-1,xmd))

& +tmask(i,j,k)\*tmask(i,j+1,k)\*(rhoi(i,k,j+1,xmd)

& -rhoi(i,k,j,xmd)))

c

e(i,k-1,j,3) = rdzw(k-1)\*tmask(i,j,k-1)\*tmask(i,j,k)\*c1e10

& \*(rhoi(i,k-1,j,xlo) - rhoi(i,k,j,xmd))

c

enddo

enddo

k = km

do i=2,imm

e(i,k,j,1) = c0

e(i,k,j,2) = c0

e(i,k,j,3) = c0

enddo

enddo

c

c--------------------------------------------------

c compute "K3", using "slmxr" to limit vertical slope of isopycnal

c to guard against numerical instabilities.

c--------------------------------------------

c

c# ifdef isopycmixspatialvar

c do i=2,imt-1

c fxa = c0

c fxb = sign(1.,e(i,k,j,3))/(abs(e(i,k,j,3))+eps)

c slope = fxb\*sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)

c if (slope .le. c0 .and. slope .ge. (-c1/slmxr)) then

c fxa = p5\*(c1+tanh((slope+slopec)/dslope))

c endif

c fxc = fxb\*fxa

c K3(i,k,j,1) = -fxc\*e(i,k,j,1)

c K3(i,k,j,2) = -fxc\*e(i,k,j,2)

c K3(i,k,j,3) = fxb\*fxb\*(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)

c & \*fxa

c enddo

c# else # endif

do j=2,jmm

do k=1,km

do i=2,imt-1

cjxz chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr\*dtxsqr(k)

chkslp = -sqrt(e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*slmxr

if (e(i,k,j,3) .gt. chkslp) e(i,k,j,3) = chkslp

enddo

fxa = p5\*(fzisop(min(km,k+1))+fzisop(k))

do i=2,imt-1

ahfctr = fxa/(e(i,k,j,3)\*\*2+eps)

K3(i,k,j,1) = -e(i,k,j,3)\*e(i,k,j,1)\*ahfctr

K3(i,k,j,2) = -e(i,k,j,3)\*e(i,k,j,2)\*ahfctr

K3(i,k,j,3) = (e(i,k,j,1)\*\*2+e(i,k,j,2)\*\*2)\*ahfctr

enddo

enddo

enddo

c

c----------------------------------------------

c impose zonal boundary conditions at "i"=1 and "imt"

c--------------------------------------------

c

do j=2,jmm

call setbcx (K3(1,1,j,1), imt, km)

call setbcx (K3(1,1,j,2), imt, km)

call setbcx (K3(1,1,j,3), imt, km)

enddo

return

end

c

c

c

c ===================================

subroutine setbcx (a, imt, jmtorkm)

c ===================================

implicit none

integer imt,jmtorkm,k

real a(imt,jmtorkm)

do k=1,jmtorkm

a(1,k) = a(imt-1,k)

a(imt,k) = a(2,k)

enddo

return

end

c

c =================

**subroutine isoadv**

c =================

c

c===============================================

c compute isopycnal transport velocities.

c===========================================

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

c

integer jstrt

real fxa

c

real fx,fy

common /wisop/ fx(imt,jmt),fy(imt,jmt)

c

c--------------------------------------------

c compute the meridional component of the isopycnal mixing velocity

c at the center of the northern face of the "t" cells.

c----------------------------------------

c

do j=2,jmm

do k=2,km-1

fxa = -p5\*rdz0(k)\*athkdf\*cosu(j)

do i=1,imt

adv\_vntiso(i,k,j) = fxa\*tmask(i,j,k)\*tmask(i,j+1,k)\*(

& K2(i,k-1,j,3) - K2(i,k+1,j,3))

enddo

enddo

enddo

c

c consider the top and bottom levels. "K2" is assumed to be zero

c at the ocean top and bottom.

c

k = 1

fxa = -p5\*rdz0(k)\*athkdf

do j=2,jmm

do i=1,imt

adv\_vntiso(i,k,j) = -fxa\*tmask(i,j,k)\*tmask(i,j+1,k)\*cosu(j)

& \*(K2(i,k,j,3) + K2(i,k+1,j,3))

enddo

enddo

c

do j=2,jmm

do i=1,imt

adv\_vntiso(i,km,j) = c0

enddo

enddo

c

do j=2,jmm

do i=1,imt

k = min(itn(i,j),itn(i,j+1))

if (k .ne. 0) then

adv\_vntiso(i,k,j) = -p5\*rdz0(k)\*athkdf\*cosu(j)\*tmask(i,j,k)

& \*tmask(i,j+1,k)\*(K2(i,k,j,3) + K2(i,k-1,j,3))

endif

enddo

enddo

c

cxjin------------------------------

do j=2,jmm

do k=1,km

do i=1,imt

adv\_vntiso(i,k,j) = adv\_vntiso(i,k,j) \* fy(i,j+1)

enddo

enddo

enddo

cxjin------------------------------

c

c------------------------------------

c compute the zonal component of the isopycnal mixing velocity

c at the center of the eastern face of the "t" grid box.

c------------------------------------------------

c

cjxz jstrt = max(js,jsmw)

jstrt = 2

c

do j=jstrt,jmm

do k=2,km-1

fxa = -p5\*rdz0(k)\*athkdf

do i=1,imm

adv\_vetiso(i,k,j) = fxa\*tmask(i,j,k)\*tmask(i+1,j,k)

& \*(K1(i,k-1,j,3) - K1(i,k+1,j,3))

enddo

enddo

enddo

c

c consider the top and bottom levels. "K1" is assumed to be zero

c at the ocean top and bottom.

c

k = 1

fxa = -p5\*rdz0(k)\*athkdf

do j=jstrt,jmm

do i=1,imm

adv\_vetiso(i,k,j) = -fxa\*tmask(i,j,k)\*tmask(i+1,j,k)

& \*(K1(i,k,j,3)+K1(i,k+1,j,3))

enddo

enddo

c

do j=jstrt,jmm

do i=1,imm

adv\_vetiso(i,km,j) = c0

enddo

enddo

c

do j=jstrt,jmm

do i=1,imm

k = min(itn(i,j),itn(i+1,j))

if (k .ne. 0) then

adv\_vetiso(i,k,j) = -p5\*rdz0(k)\*athkdf\*tmask(i,j,k)

& \*tmask(i+1,j,k)\*(K1(i,k,j,3)+K1(i,k-1,j,3))

endif

enddo

enddo

c

cxjin------------------------------

do j=2,jmm

do k=1,km

do i=1,imm

adv\_vetiso(i,k,j) = adv\_vetiso(i,k,j) \* fx(i+1,j)

enddo

enddo

enddo

cxjin------------------------------

c

c----------------------------------------------

c set the boundary conditions

c--------------------------------------------

do j=jstrt,jmm

call setbcx (adv\_vetiso(1,1,j), imt, km)

enddo

c

c--------------------------------------------

c compute the vertical component of the isopycnal mixing velocity

c at the center of the bottom face of the "t" cells, using the

c continuity equation for the isopycnal mixing velocities

c-----------------------------------------------

c

c do j=jstrt,jmm

c do i=1,imt

c adv\_vbtiso(i,0,j) = c0

c enddo

c enddo

c

c do j=jstrt,jmm

c do k=1,km-1

c do i=2,imt

c adv\_vbtiso(i,k,j) = dz0(k)\*(

c & (adv\_vetiso(i,k,j) - adv\_vetiso(i-1,k,j))\*rdxt(j) +

c & (adv\_vntiso(i,k,j) - adv\_vntiso(i,k,j-1))\*rdyt(j))

c enddo

c enddo

c enddo

c

c do j=jstrt,jmm

c do k=1,km-1

c do i=2,imt

c adv\_vbtiso(i,k,j) = adv\_vbtiso(i,k,j) + adv\_vbtiso(i,k-1,j)

c enddo

c enddo

c enddo

cc

c do j=jstrt,jmm

c do i=2,imt

c adv\_vbtiso(i,itn(i,j),j) = c0

c enddo

c enddo

c

c---------------------------------------------------

c set the boundary conditions

c---------------------------------------------------

c do j=jstrt,jmm

c call setbcx (adv\_vbtiso(1,0,j), imt, km+1)

c enddo

c

c

cxjin-------------------------------------------------

call isow (adv\_vetiso,adv\_vntiso,adv\_vbtiso)

cxjin-------------------------------------------------

c

c

return

end

c =============================

subroutine **isoflux (tf,mtrace)**

c =============================

c isopycnal diffusive tracer fluxes are computed.

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'isopyc.h'

include 'prog.h'

include 'scalar.h'

c

real fx(imt,jmt),fy(imt,jmt)

c

real temp(imt,jmt,km),tf(imt,jmt,km)

real wki(imt),wkj(jmt),wkk(kmp1)

real fk31(0:km),fk32(0:km)

integer mtrace

real fxa,fxb,fxc,fxe

c

wki(imt) = c0

wkj(1) = c0

wkj(jmt) = c0

wkk(1) = c0

wkk(kmp1) = c0

c

do i=1,imt

do j=1,jmt

do k=1,km

temp(i,j,k)=c0

enddo

enddo

enddo

c

m = mtrace

c

c------------------------------------------------

c first compute the vertical tracer flux "temp" at the northern

c face of "t" cells.

c-----------------------------------------------

c

do k=2,km-1

do j=1,jmm

do i=1,imt

temp(i,j,k)=p25\*rdz0(k)\*

& (t(i,j+1,k-1,m,taum)-t(i,j+1,k+1,m,taum)

& +t(i,j ,k-1,m,taum)-t(i,j, k+1,m,taum))

enddo

enddo

enddo

c

c----------------------------------------------

c now consider the top level, assuming that the surface tracer

c values are the same as the ones at "k"=1

c---------------------------------------------

c

k = 1

do j=1,jmm

do i=1,imt

temp(i,j,k) = p25\*rdz0(k)\*

& (t(i,j+1,k,m,taum)-t(i,j+1,k+1,m,taum)

& +t(i,j ,k,m,taum)-t(i,j, k+1,m,taum))

enddo

enddo

c

c--------------------------------------------------

c finally, consider the bottom level. the extrapolative estimator

c is used to compute the tracer values at the ocean bottom.

c-------------------------------------------------

c

do j=1,jmt

do i=1,imt

temp(i,j,km) = 0.0

enddo

enddo

c

do j=1,jmm

do i=1,imt

k = min(itn(i,j),itn(i,j+1))

if (k .ne. 0) then

fxe = dz0(k) + p5\*dz0(k-1)

fxa = p5\*(t(i,j+1,k-1,m,taum) + t(i,j,k-1,m,taum))

fxb = p5\*(t(i,j+1,k ,m,taum) + t(i,j,k ,m,taum))

fxc = rdzw(k-1)\*(fxb\*fxe-fxa\*p5\*dz0(k))

temp(i,j,k) = rdz0(k)\*(p5\*(fxa+fxb) - fxc)

endif

enddo

enddo

c

c------------------------------------------------------

c compute of meridional tracer flux

c add in the effects of the along isopycnal diffusion computed

c using "K2" component of the tensor and apply land/sea masks

c------------------------------------------------------

c

c fzisop(k)=1.0

c

do 100 k=1,km

do 100 i=2,imm

do j=1,jmm

wkj(j) = ( rdy\*(t(i,j+1,k,m,taum)-t(i,j,k,m,taum))

& + K2(i,k,j,3)\*temp(i,j,k) )\*

& cosu(j)\*tmask(i,j,k)\*tmask(i,j+1,k)\*fy(i,j+1)\*p5

enddo

do j=2,jmm

tf(i,j,k) = tf(i,j,k) + ahisop\*rdyt(j)\*(wkj(j)-wkj(j-1))

enddo

100 continue

c

c---------------------------------------------------

c compute the vertical tracer flux "temp" at the eastern

c face of "t" cells.

c------------------------------------------------

c

do k=2,km-1

do j=2,jmm

do i=1,imm

temp(i,j,k)=p25\*rdz0(k)\*

& (t(i+1,j,k-1,m,taum)-t(i+1,j,k+1,m,taum)

& +t(i ,j,k-1,m,taum)-t(i ,j,k+1,m,taum))

enddo

enddo

enddo

c

c------------------------------------------------------

c now consider the top level, assuming that the surface tracer

c values are the same as the ones at "k"=1

c--------------------------------------------

c

k = 1

do j=2,jmm

do i=1,imm

temp(i,j,k)=p25\*rdz0(k)\*

& (t(i+1,j,k,m,taum)-t(i+1,j,k+1,m,taum)

& +t(i ,j,k,m,taum)-t(i ,j,k+1,m,taum))

enddo

enddo

c

c---------------------------------------------------

c finally, consider the bottom level. the extrapolative estimator

c is used to compute the tracer values at the ocean bottom.

c--------------------------------------------------

c

do j=2,jmm

do i=1,imm

temp(i,j,km) = 0.0

enddo

enddo

c

do j=2,jmm

do i=1,imm

k = min(itn(i,j),itn(i+1,j))

if (k .ne. 0) then

fxe = dz0(k) + p5\*dz0(k-1)

fxa = p5\*(t(i,j,k-1,m,taum)+t(i+1,j,k-1,m,taum))

fxb = p5\*(t(i,j,k,m,taum)+t(i+1,j,k,m,taum))

fxc = rdzw(k-1)\*(fxb\*fxe - fxa\*p5\*dz0(k))

temp(i,j,k) = rdz0(k)\*(p5\*(fxa+fxb)-fxc)

endif

enddo

enddo

c

c-------------------------------------------------

c compute of zonal tracer flux

c add in the effects of the along isopycnal diffusion computed

c using "K1" component of the tensor and apply land/sea masks

c---------------------------------------------------

c

c fzisop(k)=1.0

c dxtr=const

c

do 120 k=1,km

do 120 j=2,jmm

do i=1,imm

wki(i) = ( rdxt(j)\*(t(i+1,j,k,m,taum)-t(i,j,k,m,taum))

& + K1(i,k,j,3)\*temp(i,j,k) )\*

& tmask(i+1,j,k)\*tmask(i,j,k)\*fx(i+1,j)\*p5

enddo

do i=2,imm

tf(i,j,k) = tf(i,j,k) + ahisop\*rdxt(j)\*(wki(i)-wki(i-1))

enddo

120 continue

c

c

c------------------------------------------

c compute the vertical tracer flux "diff\_fbiso" containing the K31

c and K32 components which are to be solved explicitly. The K33

c component will be treated semi-implicitly

c--------------------------------------

c

c----------------------------------

c at ocean surface the flux is set to zero to reflect the no tracer

c flux condition. Same condition is also imposed at ocean bottom.

c---------------------------------------

do k=0,km

fk31(k)=0.0

fk32(k)=0.0

enddo

c

do 140 j=2,jmm

do 140 i=2,imm

c

do k=2, min(itn(i,j)+1,km)

fk31(k-1) = ahisop\*p25\*rdxt(j)\*K3(i,k-1,j,1)\*tmask(i,j,k)\*(

& tmask(i-1,j,k )\*(t(i,j,k ,m,taum)-t(i-1,j,k ,m,taum))+

& tmask(i-1,j,k-1)\*(t(i,j,k-1,m,taum)-t(i-1,j,k-1,m,taum))+

& tmask(i+1,j,k )\*(t(i+1,j,k,m,taum)-t(i,j ,k ,m,taum))+

& tmask(i+1,j,k-1)\*(t(i+1,j,k-1,m,taum)-t(i,j,k-1,m,taum)) )

enddo

c

do k=2, min(itn(i,j)+1,km)

fk32(k-1) = ahisop\*p25\*rdy\*K3(i,k-1,j,2)\*tmask(i,j,k)\*(

& tmask(i,j-1,k )\*(t(i,j,k ,m,taum)-t(i,j-1,k ,m,taum))+

& tmask(i,j-1,k-1)\*(t(i,j,k-1,m,taum)-t(i,j-1,k-1,m,taum))+

& tmask(i,j+1,k )\*(t(i,j+1,k,m,taum)-t(i,j ,k ,m,taum))+

& tmask(i,j+1,k-1)\*(t(i,j+1,k-1,m,taum)-t(i,j,k-1,m,taum)) )

enddo

c

do k=1,km

tf(i,j,k) = tf(i,j,k) + gravr\*rdz(k)\*(

& (fk31(k-1)-fk31(k))+(fk32(k-1)-fk32(k)) )

enddo

140 continue

c

c

cifdef gent\_mcwilliams

c

c--------------------------------------------

c compute the meridional component of the isopycnal velocity mixing

c-------------------------------------------

c

c

do k=1,km

do i=2,imm

do j=2,jmm

wkj(j) = adv\_vntiso(i,k,j)\*

& (t(i,j+1,k,m,taum)-t(i,j,k,m,taum))

enddo

do j=2,jmm

tf(i,j,k) = tf(i,j,k) - rdyt(j)\*(wkj(j)+wkj(j-1))\*p5

enddo

enddo

enddo

c

c

c-----------------------------------------------

c compute the zonal component of the isopycnal velocity mixing

c-----------------------------------------------

c

do k=1,km

do j=2,jmm

do i=1,imm

wki(i) = adv\_vetiso(i,k,j)\*

& (t(i+1,j,k,m,taum)-t(i,j,k,m,taum))

enddo

do i=2,imm

tf(i,j,k) = tf(i,j,k) - rdxt(j)\*(wki(i)+wki(i-1))\*p5

enddo

enddo

enddo

c

c-------------------------------------------------

c compute the vertical component of the isopycnal velocity mixing

c-------------------------------------------------

c

do j=2,jmm

do i=2,imm

do k=2, min(itn(i,j)+1,km)

wkk(k) = adv\_vbtiso(i,k-1,j)\*

& (t(i,j,k-1,m,taum)-t(i,j,k,m,taum))

enddo

do k=1,itn(i,j)

tf(i,j,k) = tf(i,j,k) - (wkk(k)+wkk(k+1))\*rdz(k)\*p5

enddo

enddo

enddo

c

return

end

#endif

c

c

c ======================

**subroutine isow(u,v,w)**

c ======================

c calculate vertical mass advection: Ps\*dz/dt

c upward is positive

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'scalar.h'

c

real u(imt,km,jmt),v(imt,km,jmt),w(imt,0:km,jmt)

real dp(imt,jmt)

c

do j=1,jmt

do i=1,imt

w(i,0,j) = c0

dp(i,j) = c0

enddo

enddo

c

do j=2,jmm

do k=1,km

do i=2,imm

w(i,k,j) = dz(k)\*((u(i,k,j)-u(i-1,k,j))\*rdxt(j) +

& (v(i,k,j)-v(i,k,j-1))\*rdyt(j))

dp(i,j) = dp(i,j) - w(i,k,j)

enddo

enddo

enddo

c

do j=2,jmm

do k=1,km-1

do i=2,imm

w(i,k,j)=(w(i,k-1,j)+dz(k)\*dp(i,j)/pn(i,j)+w(i,k,j))

& \*tmask(i,j,k+1)

enddo

enddo

enddo

c

do j=2,jmm

do i=2,imt

w(i,itn(i,j),j) = c0

enddo

enddo

c

c

do k=1,km

do j=2,jmm

w(1 ,k,j) = w(imm,k,j)

w(imt,k,j) = w(2 ,k,j)

enddo

enddo

c

return

end

**barotr.F**

subroutine barotr

c compute pbt, upb & vpb at "tau+1" time level

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

include 'cvbc.h'

integer mode\_b

real a,b,dp,adu,adv,etax,etay,du2,dv2,t1,t2,t3

common /works/ a(imt,jmt),b(imt,jmt),dp(imt,jmt)

common /works/ adu(imt,jmt),adv(imt,jmt)

common /works/ etax(imt,jmt),etay(imt,jmt)

common /works/ du2(imt,jmt),dv2(imt,jmt)

do 1000 mode\_b = 1,nbb

c compute the "artificial" horizontal viscosity

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0)then

adu(i,j) = am\*( r1c(j)\*(upb(i,j+1,taum)-upb(i,j,taum)) -

& r1d(j)\*(upb(i,j,taum)-upb(i,j-1,taum)) +

& sdxu(j)\*(upb(i+1,j,taum)-c2\*upb(i,j,taum)+

& upb(i-1,j,taum)) )\*c2

adv(i,j) = am\*( r1c(j)\*(vpb(i,j+1,taum)-vpb(i,j,taum)) -

& r1d(j)\*(vpb(i,j,taum)-vpb(i,j-1,taum)) +

& sdxu(j)\*(vpb(i+1,j,taum)-c2\*vpb(i,j,taum)+

& vpb(i-1,j,taum)) )\*c2

endif

enddo

enddo

if(mode\_b.eq.1) then

do j=2,jmm

do i=2,imm

dub(i,j) = dub(i,j) - adu(i,j)

dvb(i,j) = dvb(i,j) - adv(i,j)

enddo

enddo

end if

c

c

1001 continue

c

c-----------------------------------------------

c square root of pbt at the U cell

c-----------------------------------------------

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0)then

spbt(i,j) = p5\*sqrt(pbt(i,j ,tau) + pbt(i+1,j ,tau) +

& pbt(i,j+1,tau) + pbt(i+1,j+1,tau))

endif

spbt(1 ,j) = spbt(imm,j)

spbt(imt,j) = spbt(2 ,j)

enddo

enddo

c

c

c----------------------------------------------------

c calculate the pressure gradients due to change of free surface

c---------------------------------------------

#ifdef boussinesq

do j=2,jmm

do i=1,imt

a(i,j) = phib(i,j)\*(c1-pbt(i,j,tau))

enddo

enddo

#endif

c

c

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0)then

c

etax(i,j) = spbt(i,j)\*p5\*(

#ifdef boussinesq

& (a(i+1,j )-a(i,j ))\*pdxn(i,j)

& +(a(i+1,j+1)-a(i,j+1))\*pdxs(i,j+1) +

#else

& + phibx(i,j) +

#endif

& (pbt(i+1,j ,tau)-pbt(i,j ,tau))\*pbxn(i,j)

& +(pbt(i+1,j+1,tau)-pbt(i,j+1,tau))\*pbxs(i,j+1) +

& (pbt(i+1,j ,tau)+pbt(i,j ,tau))\*pcxn(i,j)

& +(pbt(i+1,j+1,tau)+pbt(i,j+1,tau))\*pcxs(i,j+1) )

c

c

etay(i,j) = spbt(i,j)\*p5\*(

#ifdef boussinesq

& (a(i ,j+1)-a(i ,j))\*pdye(i,j)

& +(a(i+1,j+1)-a(i+1,j))\*pdyw(i+1,j) +

#else

& + phiby(i,j) +

#endif

& (pbt(i ,j+1,tau)-pbt(i, j,tau))\*pbye(i ,j)

& +(pbt(i+1,j+1,tau)-pbt(i+1,j,tau))\*pbyw(i+1,j) +

& (pbt(i ,j+1,tau)+pbt(i, j,tau))\*pcye(i ,j)

& +(pbt(i+1,j+1,tau)+pbt(i+1,j,tau))\*pcyw(i+1,j) )

c

endif

enddo

enddo

c

c----------------------------------------------

c advection + viscosiy + pressure gradient + coriolis

c-------------------------------------------------

c

do j=1,jmt

do i=1,imt

if(ivn(i,j).gt.0)then

a(i,j) = dub(i,j) + adu(i,j) + ff(j)\*vpb(i,j,tau) - etax(i,j)

b(i,j) = dvb(i,j) + adv(i,j) - ff(j)\*upb(i,j,tau) - etay(i,j)

else

a(i,j) = c0

b(i,j) = c0

endif

enddo

enddo

c

c

c------------------------------------

c coriolis adjustment

c----------------------------------------

c

if(leapfrog\_b)then

do j=1,jmt

do i=1,imt

du2(i,j) = ebla(j)\*a(i,j) + eblb(j)\*b(i,j)

dv2(i,j) = ebla(j)\*b(i,j) - eblb(j)\*a(i,j)

enddo

enddo

else

do j=1,jmt

do i=1,imt

du2(i,j) = ebea(j)\*a(i,j) + ebeb(j)\*b(i,j)

dv2(i,j) = ebea(j)\*b(i,j) - ebeb(j)\*a(i,j)

enddo

enddo

endif

c

c

c-----------------------------------------------

c calculate the change of Pbt

c------------------------------------------

c

do j=1,jmt

do i=1,imt

a(i,j) = zu(i,j)\*spbt(i,j)\*upb(i,j,tau)

b(i,j) = zu(i,j)\*spbt(i,j)\*vpb(i,j,tau)\*cosu(j)

enddo

enddo

c

c

do j=2,jmm

do i=2,imm

if(itn(i,j).gt.0)then

dp(i,j) = ( - p5\*(rdxt(j)\*(a(i,j)+a(i,j-1)-a(i-1,j)-a(i-1,j-1))

& +rdyt(j)\*(b(i,j)+b(i-1,j)-b(i,j-1)-b(i-1,j-1)))

#ifdef snbc

& - emp(i,j)

#endif

& )/pn(i,j)

else

dp(i,j) = c0

endif

enddo

enddo

c

c

#ifdef smth

print\*, ' check I&J boundary for smooth!!!!!!'

stop 111222

call smths(du2,umask, 2, 5) !! from 68S-74S

call smths(dv2,umask, 2, 5)

call smths(du2,umask,73,76) !! from 68N-74N

call smths(dv2,umask,73,76)

#endif

c

c

c

c-----------------------------------------------------

c compute pbt, upb & vpb at tau+1 time level

c-------------------------------------------------

c

if(leapfrog\_b) go to 120

c

do j=2,jmm

do i=2,imm

upb(i,j,tau) = upb(i,j,taum) + du2(i,j)\*dtsf

vpb(i,j,tau) = vpb(i,j,taum) + dv2(i,j)\*dtsf

pbt(i,j,tau) = pbt(i,j,taum) + dp(i,j) \*dtsf

enddo

enddo

c

if(euler\_back) then

euler\_back = .false.

go to 1001

endif

c

leapfrog\_b = .true.

c

go to 150

c

c

120 continue

c

c

do j=2,jmm

do i=2,imm

t1 = pbt(i,j,taum) + dp (i,j)\*c2dtsf

t2 = upb(i,j,taum) + du2(i,j)\*c2dtsf

t3 = vpb(i,j,taum) + dv2(i,j)\*c2dtsf

#ifdef asselin\_b

pbt(i,j,taum) = afb2\*pbt(i,j,tau)+afb1\*(pbt(i,j,taum)+t1)

upb(i,j,taum) = afb2\*upb(i,j,tau)+afb1\*(upb(i,j,taum)+t2)

vpb(i,j,taum) = afb2\*vpb(i,j,tau)+afb1\*(vpb(i,j,taum)+t3)

#else

pbt(i,j,taum) = pbt(i,j,tau)

upb(i,j,taum) = upb(i,j,tau)

vpb(i,j,taum) = vpb(i,j,tau)

#endif

pbt(i,j,tau) = t1

upb(i,j,tau) = t2

vpb(i,j,tau) = t3

enddo

enddo

c

150 continue

c

c

do j=2,jmm

upb(1 ,j,tau) = upb(imm,j,tau)

upb(imt,j,tau) = upb(2 ,j,tau)

vpb(1 ,j,tau) = vpb(imm,j,tau)

vpb(imt,j,tau) = vpb(2 ,j,tau)

pbt(1 ,j,tau) = pbt(imm,j,tau)

pbt(imt,j,tau) = pbt(2 ,j,tau)

upb(1 ,j,taum) = upb(imm,j,taum)

upb(imt,j,taum) = upb(2 ,j,taum)

vpb(1 ,j,taum) = vpb(imm,j,taum)

vpb(imt,j,taum) = vpb(2 ,j,taum)

pbt(1 ,j,taum) = pbt(imm,j,taum)

pbt(imt,j,taum) = pbt(2 ,j,taum)

enddo

c

c

c------------------------------------------------------

c constract time-averaged pbt for solving baroclinc and tracer Eqs.

c-------------------------------------------------------

do j=2,jmm

do i=1,imt

pmup(i,j) = pmup(i,j) + pbt(i,j,tau)

pmtp(i,j) = pmtp(i,j) + pbt(i,j,tau)

enddo

enddo

c

c

1000 continue

c

return

end

**bclinc.F**

c =================

subroutine bclinc

c =================

c

c compute up & vp at tau+1 time level

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

c

real pbar,etax,etay,px,py,a,b,c,t1,t2

common /works/ a(imt,jmt),b(imt,jmt),c(imt,jmt)

common /works/ px(imt,jmt),py(imt,jmt)

common /works/ etax(imt,jmt),etay(imt,jmt)

common /works/ pbar(imt,jmt)

c

c

c---------------------------------------------------

c calculate time-averaged pbt & sqrt(pbt)

c--------------------------------------------------

do j=1,jmt

do i=1,imt

pmup(i,j) = pmup(i,j)\*onbb

enddo

enddo

c

if(leapfrog\_c)then

do j=1,jmt

do i=1,imt

pbar(i,j) = (pmup(i,j)+pmum(i,j))\*p5

enddo

enddo

else

do j=1,jmt

do i=1,imt

pbar(i,j) = pmup(i,j)

enddo

enddo

endif

c

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0)then

spbt(i,j) = p5\*sqrt(pbar(i,j ) + pbar(i+1,j ) +

& pbar(i,j+1) + pbar(i+1,j+1))

endif

spbt(1 ,j) = spbt(imm,j)

spbt(imt,j) = spbt(2 ,j)

enddo

enddo

c

c

do 100 k=1,km

c

c------------------------------------------------------

c calculate pressure gradients \* spbt

c------------------------------------------------------

#ifdef boussinesq

c

c a = geopotential

c b = pressure

c px = (rho)xbar\*(a)x + (b)x

c py = (rho)ybar\*(a)y + (b)y

c

do j=2,jmm

do i=1,imt

a(i,j) = phib(i,j)-pbar(i,j)\*(z(k)+phib(i,j))

b(i,j) = pbar(i,j)\*rhodp(i,j,k)

enddo

enddo

do j=2,jmm

do i=2,imm

px(i,j) = ( (rho(i,j,k)+rho(i+1,j,k))\*p5\*(a(i+1,j)-a(i,j)) +

& (b(i+1,j)-b(i,j)) )\*rdxt(j)

py(i,j) = ( (rho(i,j,k)+rho(i,j+1,k))\*p5\*(a(i,j+1)-a(i,j)) +

& (b(i,j+1)-b(i,j)) )\*rdy

enddo

enddo

#else

c

c a = geopotential

c b = pressure

c px = (a)x + (rho)xbar\*(b)x

c py = (a)y + (rho)ybar\*(b)y

c

do j=2,jmm

do i=1,imt

a(i,j) = phib(i,j) + pbar(i,j)\*rhodp(i,j,k)

b(i,j) = pbar(i,j)\*z(k)

enddo

enddo

c

do j=2,jmm

do i=2,imm

px(i,j) = ( (rho(i,j,k)+rho(i+1,j,k))\*p5\*(b(i+1,j)-b(i,j)) +

& (a(i+1,j)-a(i,j)) )\*rdxt(j)

py(i,j) = ( (rho(i,j,k)+rho(i,j+1,k))\*p5\*(b(i,j+1)-b(i,j)) +

& (a(i,j+1)-a(i,j)) )\*rdy

enddo

enddo

#endif

c

do j=2,jmm

do i=2,imm

etax(i,j) = (px(i,j)+px(i,j+1))\*p5\*spbt(i,j)

etay(i,j) = (py(i,j)+py(i+1,j))\*p5\*spbt(i,j)

enddo

enddo

c

c

c------------------------------------------------

c advection + diffusion + pressure gradients + coriolis

c------------------------------------------------

do j=2,jmm

do i=2,imm

a(i,j) = du(i,j,k) + ff(j)\*vp(i,j,k,tau) - etax(i,j)

b(i,j) = dv(i,j,k) - ff(j)\*up(i,j,k,tau) - etay(i,j)

enddo

enddo

c

c

c----------------------------------------

c coriolis adjustment

c-------------------------------------------------

if(leapfrog\_c)then

do j=2,jmm

do i=2,imm

du(i,j,k) = (epla(j)\*a(i,j) + eplb(j)\*b(i,j))\*umask(i,j,k)

dv(i,j,k) = (epla(j)\*b(i,j) - eplb(j)\*a(i,j))\*umask(i,j,k)

enddo

enddo

else

do j=2,jmm

do i=2,imm

du(i,j,k) = (epea(j)\*a(i,j) + epeb(j)\*b(i,j))\*umask(i,j,k)

dv(i,j,k) = (epea(j)\*b(i,j) - epeb(j)\*a(i,j))\*umask(i,j,k)

enddo

enddo

endif

c

100 continue

c

c

c------------------------------------------------------

c compute up & vp at tau+1 time level

c------------------------------------------------------

c

if(leapfrog\_c) go to 120

c

do k=1,km

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

up(i,j,k,tau) = up(i,j,k,taum) + du(i,j,k)\*dtuv

vp(i,j,k,tau) = vp(i,j,k,taum) + dv(i,j,k)\*dtuv

endif

enddo

enddo

enddo

c

c@@@ interaction between barotropic and baroclinic modes

c

call vinteg(up(1,1,1,tau),a)

call vinteg(vp(1,1,1,tau),b)

c

do k=1,km

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

up(i,j,k,tau) = up(i,j,k,tau) - a(i,j) + upb(i,j,tau)

vp(i,j,k,tau) = vp(i,j,k,tau) - b(i,j) + vpb(i,j,tau)

endif

enddo

enddo

enddo

c

leapfrog\_c = .true.

c

go to 150

c

120 continue

c

do k=1,km

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

du(i,j,k) = up(i,j,k,taum) + du(i,j,k)\*c2dtuv

dv(i,j,k) = vp(i,j,k,taum) + dv(i,j,k)\*c2dtuv

endif

enddo

enddo

enddo

c

c@@@ interaction between barotropic and baroclinic modes

c

call vinteg(du,a)

call vinteg(dv,b)

c

do k=1,km

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

t1 = du(i,j,k) - a(i,j) + upb(i,j,tau)

t2 = dv(i,j,k) - b(i,j) + vpb(i,j,tau)

#ifdef asselin\_c

up(i,j,k,taum) = afc2\*up(i,j,k,tau)+afc1\*(up(i,j,k,taum)+t1)

vp(i,j,k,taum) = afc2\*vp(i,j,k,tau)+afc1\*(vp(i,j,k,taum)+t2)

#else

up(i,j,k,taum) = up(i,j,k,tau)

vp(i,j,k,taum) = vp(i,j,k,tau)

#endif

up(i,j,k,tau) = t1

vp(i,j,k,tau) = t2

endif

enddo

enddo

enddo

c

150 continue

c

c

do k=1,km

do j=2,jmm

up(1 ,j,k,tau) = up(imm,j,k,tau)

up(imt,j,k,tau) = up(2 ,j,k,tau)

vp(1 ,j,k,tau) = vp(imm,j,k,tau)

vp(imt,j,k,tau) = vp(2 ,j,k,tau)

up(1 ,j,k,taum) = up(imm,j,k,taum)

up(imt,j,k,taum) = up(2 ,j,k,taum)

vp(1 ,j,k,taum) = vp(imm,j,k,taum)

vp(imt,j,k,taum) = vp(2 ,j,k,taum)

enddo

enddo

c

c

c--------------------------------------------------

c constract time-averaged mass advections

c-----------------------------------------------

do j=2,jmm

do i=2,imm

spbt(i,j) = p5\*sqrt(pbt(i,j ,tau) + pbt(i+1,j ,tau) +

& pbt(i,j+1,tau) + pbt(i+1,j+1,tau))

enddo

spbt(1 ,j) = spbt(imm,j)

spbt(imt,j) = spbt(2 ,j)

enddo

c

do k=1,km

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

ump(i,j,k) = ump(i,j,k) + up(i,j,k,tau)\*spbt(i,j)

vmp(i,j,k) = vmp(i,j,k) + vp(i,j,k,tau)\*spbt(i,j)\*cosu(j)

endif

enddo

ump(1 ,j,k) = ump(imm,j,k)

ump(imt,j,k) = ump(2 ,j,k)

vmp(1 ,j,k) = vmp(imm,j,k)

vmp(imt,j,k) = vmp(2 ,j,k)

enddo

enddo

c

c

return

end

**convect.F**

c ==================

subroutine convect

c ==================

c

c a full convective adjustment scheme, based on GFDL's version

c

c ---------------------------------------------------------------

c kcon = maximum number of levels at this location

c lcon = counts levels down

c lcona = upper layer of a convective part of water column

c lconb = lower layer of a convective part of water column

c rhoup = density anomaly referenced to same level

c rholo = density anomaly referenced to level below

c dztsum = sum of layer thicknesses

c trasum = sum of layer tracer values

c tramix = mixed tracer value after convection

c lcven = number of levels ventilated (convection to surface)

c ---------------------------------------------------------------

c

implicit none

include 'param.h'

include 'grdvar.h'

include 'prog.h'

c

real rhoup(km),rholo(km),trasum(2)

real tup,sup,tlo,slo,dztsum,tramix

integer kcon,lcven,l1,l,lcon,lcona,lconb,lmix

c

c

do j=2,jmm

do i=2,imm

c

kcon = itn(i,j)

if (kcon .eq. 0) goto 1310

c

lcven = 1

lcon = 0

c

do l=1,kcon-1

l1 = l+1

tup = t(i,j,l1,1,tau)

sup = t(i,j,l1,2,tau)

tlo = t(i,j, l,1,tau)

slo = t(i,j, l,2,tau)

rhoup(l1) = pt(i,j,l1)\*tup + ps(i,j,l1)\*sup

rholo(l) = pt(i,j,l1)\*tlo + ps(i,j,l1)\*slo

enddo

c

c

c 1. initial search for uppermost unstable pair; if none is

c found, move on to next column

c

do k=kcon-1,1,-1

if (rholo(k) .gt. rhoup(k+1)) lcon = k

enddo

c

if (lcon .eq. 0) goto 1310

c

1319 lcona = lcon

lconb = lcon + 1

c

c 2. mix the first two unstable layers

c

dztsum = dz(lcona) + dz(lconb)

do n=1,2

trasum(n) = t(i,j,lcona,n,tau)\*dz(lcona) +

& t(i,j,lconb,n,tau)\*dz(lconb)

tramix = trasum(n) / dztsum

t(i,j,lcona,n,tau) = tramix

t(i,j,lconb,n,tau) = tramix

enddo

c

c 3. test layer below lconb

c

1306 continue

if (lconb .eq. kcon) goto 1308

c

l1 = lconb + 1

rholo(lconb) = pt(i,j,l1)\*t(i,j,lconb,1,tau) +

& ps(i,j,l1)\*t(i,j,lconb,2,tau)

c

if (rholo(lconb) .gt. rhoup(l1)) then

lconb = lconb+1

dztsum = dztsum + dz(lconb)

do n=1,2

trasum(n) = trasum(n) + t(i,j,lconb,n,tau)\*dz(lconb)

tramix = trasum(n) / dztsum

do lmix=lcona,lconb

t(i,j,lmix,n,tau) = tramix

enddo

enddo

goto 1306

end if

c

c 4. test layer above lcona

c

1308 continue

if (lcona .gt. 1) then

l1 = lcona-1

rholo(l1) = pt(i,j,lcona)\*t(i,j,l1,1,tau) +

& ps(i,j,lcona)\*t(i,j,l1,2,tau)

rhoup(lcona) = pt(i,j,lcona)\*t(i,j,lcona,1,tau) +

& ps(i,j,lcona)\*t(i,j,lcona,2,tau)

if (rholo(lcona-1) .gt. rhoup(lcona)) then

lcona = lcona-1

dztsum = dztsum + dz(lcona)

do n=1,2

trasum(n) = trasum(n) + t(i,j,lcona,n,tau)\*dz(lcona)

tramix = trasum(n) / dztsum

do lmix=lcona,lconb

t(i,j,lmix,n,tau) = tramix

enddo

enddo

goto 1306

end if

end if

c

c 5. remember the total number of levels mixed by convection

c in this water column, as well as the ventilated column

c

if (lcona .eq. 1) lcven = lconb - lcona + 1

c

c 6. resume search if step 3. and 4. have been passed and this

c unstable part of the water column has thus been removed,

c i.e. find further unstable areas further down the column

c

if (lconb .eq. kcon) goto 1310

lcon = lconb

c

1302 continue

lcon = lcon + 1

c

if (lcon .eq. kcon) goto 1310

c

if (rholo(lcon) .le. rhoup(lcon+1)) goto 1302

c

goto 1319

1310 continue

c

enddo

enddo

c

do n=1,2

do k=1,km

do j=2,jmm

t(1 ,j,k,n,tau) = t(imm,j,k,n,tau)

t(imt,j,k,n,tau) = t(2 ,j,k,n,tau)

enddo

enddo

enddo

c

return

end

density.F

c==================

subroutine density

c ==================

c calculate density/rho\_0(BCOM) or reciprocal of density(PCOM)

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

include 'cvbc.h'

c

real t0,s0,p0,dens,rdens

c

c calculate density/rho\_0 or reciprocal of density

c

do k=1,km

do j=2,jmm

do i=2,imm

if(tmask(i,j,k).gt.c0)then

t0 = t(i,j,k,1,tau)

s0 = t(i,j,k,2,tau)

#ifdef boussinesq

p0 = fixp(i,j,k)

rho(i,j,k) = dens(t0,s0,p0)\*rrho\_0

#else

p0 = (pbt(i,j,tau)\*z(k) + bcp(i,j))\*decibar

rho(i,j,k) = rdens(t0,s0,p0)

#endif

endif

enddo

rho(1 ,j,k) = rho(imm,j,k)

rho(imt,j,k) = rho(2 ,j,k)

enddo

enddo

c

return

end

c

c

c ==================

subroutine rho\_ref

c ==================

c

c pt = {partial rho} over {partial temperature}

c ps = {partial rho} over {partial salinity }

c

c potential density = pt\*(t-t0) + ps\*(s-s0) + den0

c

c it will compare pt\*t+ps\*s rather than potential density

c itself in subroutine convect

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

c

real t1,s1,p1,dens

real rhop,rhoq

c

do k=1,km

do j=2,jmm

do i=2,imm

if(tmask(i,j,k).gt.c0)then

t1 = t(i,j,k,1,tau)

s1 = t(i,j,k,2,tau)

#ifdef boussinesq

p1 = fixp(i,j,k)

#else

p1 = pbt(i,j,tau)\*z(k)\*decibar

#endif

rhop = dens(t1+deltat,s1,p1)

rhoq = dens(t1-deltat,s1,p1)

pt(i,j,k) = (rhop-rhoq)\*rdeltat

rhop = dens(t1,s1+deltas,p1)

rhoq = dens(t1,s1-deltas,p1)

ps(i,j,k) = (rhop-rhoq)\*rdeltas

endif

enddo

pt(1 ,j,k) = pt(imm,j,k)

pt(imt,j,k) = pt(2 ,j,k)

ps(1 ,j,k) = ps(imm,j,k)

ps(imt,j,k) = ps(2 ,j,k)

enddo

enddo

c

return

end

c

grdvar.F

c =================

subroutine grdvar

c =================

c

c set resolution, t/u mask and the j,k-depended parameters

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

c

real rdx,t1

c

c ---------------------------------------------------------------

c horizontal resolution (-1S to 61N) 2x2

c ---------------------------------------------------------------

real dlam,dphi,phis

parameter (dlam=2.0)

parameter (dphi=2.0)

parameter (phis=0.0)

c

c ---------------------------------------------------------------

c vertical resolution (depth at the center of T grids, in cm)

c ---------------------------------------------------------------

data z0/

& 1500.00, 4574.87, 7873.52, 11542.41, 15723.96, 20554.94,

& 26165.04, 32675.40, 40197.28, 48830.88, 58664.21, 69772.14,

& 82215.58, 96040.78,111278.89,127945.55,146040.78,165548.91,

& 186438.81,208664.22,232164.22,256863.95,282675.41,309498.38,

& 337221.59,365723.94,394875.72,424540.16,454574.84,484833.32/

c

c ---------------------------------------------------------------

c model's topography

c ---------------------------------------------------------------

open(72,file='topog.data',form='unformatted',status='old')

read(72) itn

close(72)

c

c ---------------------------------------------------------------

c set up T/U mask

c ---------------------------------------------------------------

do k=1,km

do j=1,jmt

do i=1,imt

if(k.le.itn(i,j))then

tmask(i,j,k)=c1

else

tmask(i,j,k)=c0

endif

enddo

enddo

enddo

c

do k=1,km

do j=1,jmt

do i=1,imt

umask(i,j,k)=c0

enddo

enddo

do j=2,jmm

do i=2,imm

umask(i,j,k)=tmask(i,j,k)\*tmask(i+1,j,k)

& \*tmask(i,j+1,k)\*tmask(i+1,j+1,k)

enddo

ccc umask(1 ,j,k) = umask(imm,j,k)

ccc umask(imt,j,k) = umask(2 ,j,k)

enddo

enddo

c

c

do j=1,jmt

do i=1,imt

ivn(i,j)=0

do k=1,km

if(umask(i,j,k).gt.c0) ivn(i,j) = ivn(i,j) + 1

enddo

enddo

enddo

c

c ---------------------------------------------------------------

c geopotential at bottom

c ---------------------------------------------------------------

do j=1,jmt

do i=1,imt

t1 = c0

do k=1,itn(i,j)

t1 = -t1-z0(k)\*c2

enddo

phib(i,j) = t1 \* grav

enddo

enddo

c

c ---------------------------------------------------------------

c calculate pn & z & dz

c ---------------------------------------------------------------

c pn = constant bottom pressure for PCOM; -1\*phib for BCOM

c z = eta at the center of T grids

c dz = thinkness of T grids

c

call setpn

c

c

c ---------------------------------------------------------------

c rdz = 1/dz

c zu = pn at "u" cell

c rzu = 1/zu

c rdzw = 1/(layer thickness at "w" cell (cm))

c ---------------------------------------------------------------

do k=1,km

rdz(k)=c1/dz(k)

enddo

c

do j=1,jmt

do i=1,imt

zu (i,j) = c0

rzu(i,j) = c0

do k=1,ivn(i,j)

zu(i,j) = zu(i,j) + dz(k)

enddo

if(zu(i,j).gt.c0) rzu(i,j) = c1/zu(i,j)

enddo

enddo

c

do k=1,kmm1

rdzw(k) = c1/(z0(k+1) - z0(k))

enddo

rdzw(km) = rdzw(kmm1)

c

c ---------------------------------------------------------------

c calculate latitude for starting filter

c ---------------------------------------------------------------

do j=1,jmt

jstn = j

if((phis+dphi\*(j-1)).ge.65) go to 10

enddo

10 jedn = jmm

c

do j=1,jmt

jeds = j

if((phis+dphi\*(j-1)).ge.-65) then

jeds=j

go to 20

endif

enddo

20 jsts = 2

c

c

c ---------------------------------------------------------------

c calculate parameters which are the function of grid

c ---------------------------------------------------------------

c

do j=1,jmt

lat(j) = phis + dphi\*(j-1)

enddo

c

do j=1,jmt

cost(j) = cos((lat(j)-dphi\*p5)\*torad)

cosu(j) = cos(lat(j)\*torad)

ff (j) = sin(lat(j)\*torad)\*c2\*omega

enddo

c

rdx = c1/(radius\*dlam\*torad)

rdy = c1/(radius\*dphi\*torad)

c

do j=1,jmt

rdxt(j) = rdx / cost(j)

rdxu(j) = rdx / cosu(j)

rdyt(j) = rdy / cost(j)

rdyu(j) = rdy / cosu(j)

enddo

c

do j=1,jmt

sdxt(j) = rdxt(j)\*rdxt(j)

sdxu(j) = rdxu(j)\*rdxu(j)\*p5

enddo

c

do j=2,jmt

r1a(j) = cosu(j) /cost(j)\*(rdy\*\*2)

r1b(j) = cosu(j-1)/cost(j)\*(rdy\*\*2)

enddo

r1a(1) = r1a(2)

r1b(1) = r1b(2)

c

do j=1,jmm

r1c(j) = p5\*cost(j+1)/cosu(j)\*(rdy\*\*2)

r1d(j) = p5\*cost(j) /cosu(j)\*(rdy\*\*2)

enddo

r1c(jmt) = r1c(jmm)

r1d(jmt) = r1d(jmm)

c

do j=1,jmt

t1 = sqrt(c1-cosu(j)\*\*2)/cosu(j)

cv1(j) = (c1-t1\*t1)/(radius\*\*2)

cv2(j) = t1\*rdxu(j)/radius

enddo

c

c

do j=1,jmt

dxdyt(j) = c1/rdy/rdxt(j)

dxdyu(j) = c1/rdy/rdxu(j)

enddo

c

area = c0

do j=1,jmt

do i=2,imm

area = area + dxdyt(j)\*tmask(i,j,1)

enddo

enddo

c

return

end

inirun.F

C =================

subroutine inirun

c =================

c initialization

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

include 'cvbc.h'

include 'calendar.h'

include 'diag.h'

c

real a,b,c,d,t0

common /works/ a(imt,jmt),b(imt,jmt),c(imt,jmt)

common /works/ d(imt,jmt,km)

real fx,fy

common /wisop/ fx(imt,jmt),fy(imt,jmt)

c

c-------------------------------------------------

c asselin temporal filter parameter

c------------------------------------------------

afb2 = c1-c2\*afb1

afc2 = c1-c2\*afc1

aft2 = c1-c2\*aft1

c

c

c------------------------------------------------

c set contral parameter of integration

c--------------------------------------------------

nss = int(24.0/dtts)

ncc = int(dtts/dtuv)

nbb = int(dtuv\*c60/dtsf)

c

#ifdef accelerate

ncc = 1

#endif

c

onbb = c1/float(nbb+1)

oncc = c1/float(ncc+1)

onbc = c1/float(nbb\*ncc+1)

c

write(6,801)nss,ncc,nbb

801 format(/1x,'nss=',i3,2x,'ncc=',i3,2x,'nbb=',i3/)

c

c

c------------------------------------------------------

c set time step to the unit of second

c-------------------------------------------------

dtsf = dtsf \* c60

dtuv = dtuv \* c3600

dtts = dtts \* c3600

c

c2dtsf = dtsf \* c2

c2dtuv = dtuv \* c2

c2dtts = dtts \* c2

c

c

c----------------------------------------------------

c parameters used for semi-implicitly handle of Coriolis term

c--------------------------------------------------

do j=1,jmt

t0 = p5\*ff(j)\*dtuv

epea(j) = c1/(c1+t0\*t0)

epeb(j) = t0/(c1+t0\*t0)

enddo

c

do j=1,jmt

t0 = p5\*ff(j)\*c2dtuv

epla(j) = c1/(c1+t0\*t0)

eplb(j) = t0/(c1+t0\*t0)

enddo

c

do j=1,jmt

t0 = p5\*ff(j)\*dtsf

ebea(j) = c1/(c1+t0\*t0)

ebeb(j) = t0/(c1+t0\*t0)

enddo

c

do j=1,jmt

t0 = p5\*ff(j)\*c2dtsf

ebla(j) = c1/(c1+t0\*t0)

eblb(j) = t0/(c1+t0\*t0)

enddo

c

c------------------------------------------------------

c surface forcing fields

c------------------------------------------------------

c bcf(i,j,12,1) : sea surface zonal windstres (dynes/cm\*\*2)

c bcf(i,j,12,2) : sea surface meridional windstres (dynes/cm\*\*2)

c bcf(i,j,12,3) : sea surface air temperature (celsius)

c bcf(i,j,12,4) : sea surface air presure (dynes/cm\*\*2)

c bcf(i,j,12,5) : sea surface salinity (psu)

c bcf(i,j,12,6) : rate of evaporation minus precipitation (cm/s)

c bcf(i,j,12,7) : coefficient for calculation of HF (w/m2/c)

c

open(80,file='sbcf.data',form='unformatted',status='old')

read(80) bcf

close(80)

c

c

c------------------------------------------------------

c read Levitus annual mean temperature and salinity

c------------------------------------------------------

open(81,file='tsobs.data',status='old',form='unformatted')

read(81)pt,ps

close(81)

c

do k=1,km

do j=1,jmt

pt(1 ,j,k) = pt(imm,j,k)

pt(imt,j,k) = pt(2 ,j,k)

ps(1 ,j,k) = ps(imm,j,k)

ps(imt,j,k) = ps(2 ,j,k)

enddo

enddo

c

do k=1,km

do j=1,jmt

do i=1,imt

t(i,j,k,1,tau ) = pt(i,j,k)

t(i,j,k,2,tau ) = ps(i,j,k)

t(i,j,k,1,taum) = t(i,j,k,1,tau)

t(i,j,k,2,taum) = t(i,j,k,2,tau)

enddo

enddo

enddo

c

c

c------------------------------------------------------

c initialize pbt & rho & fixp (for BCOM)

c----------------------------------------------------

call setpbt

c

do j=1,jmt

do i=1,imt

pbt(i,j,taum) = pbt(i,j,tau)

enddo

enddo

c

c

c--------------------------------------------------------

c initialize velocities

c--------------------------------------------------------

do n=1,2

do k=1,km

do j=1,jmt

do i=1,imt

up(i,j,k,n) = c0

vp(i,j,k,n) = c0

enddo

enddo

enddo

do j=1,jmt

do i=1,imt

upb(i,j,n) = c0

vpb(i,j,n) = c0

enddo

enddo

enddo

c

c-------------------------------------------------

c initialize diagnostic variables in prog.h

c-----------------------------------------------

do j=1,jmt

do i=1,imt

spbt(i,j) = c1

enddo

enddo

c

do k=1,kmp1

do j=1,jmt

do i=1,imt

w(i,j,k) = c0

enddo

enddo

enddo

c

c---------------------------------------------------

c initialize wroking arrays in prog.h

c---------------------------------------------------

do j=1,jmt

do i=1,imt

dub(i,j) = c0

dvb(i,j) = c0

enddo

enddo

c

do k=1,km

do j=1,jmt

do i=1,imt

du(i,j,k) = c0

dv(i,j,k) = c0

diffu(i,j,k) = c0

diffv(i,j,k) = c0

pt(i,j,k) = c0

ps(i,j,k) = c0

enddo

enddo

enddo

c

do j=1,jmt

do i=1,imt

pmup(i,j) = pbt(i,j,tau)

pmtp(i,j) = pbt(i,j,tau)

pmum(i,j) = pbt(i,j,tau)

pmtm(i,j) = pbt(i,j,tau)

enddo

enddo

c

do k=1,km

do j=1,jmt

do i=1,imt

ump(i,j,k) = c0

vmp(i,j,k) = c0

umm(i,j,k) = c0

vmm(i,j,k) = c0

enddo

enddo

enddo

c

c--------------------------------------------------

c initialize common /works/

c--------------------------------------------------

do k=1,km

do j=1,jmt

do i=1,imt

a(i,j) = c0

b(i,j) = c0

c(i,j) = c0

d(i,j,k) = c0

enddo

enddo

enddo

do j=1,jmt

do i=1,imt

fx(i,j) = c1

fy(i,j) = c1

enddo

enddo

c

c---------------------------------------------------

c initialize pressure terms

c-----------------------------------------------

do j=1,jmt

do i=1,imt

pax (i,j) = c0

pay (i,j) = c0

pbxn(i,j) = c0

pcxn(i,j) = c0

pdxn(i,j) = c0

pbxs(i,j) = c0

pcxs(i,j) = c0

pdxs(i,j) = c0

pbye(i,j) = c0

pcye(i,j) = c0

pdye(i,j) = c0

pbyw(i,j) = c0

pcyw(i,j) = c0

pdyw(i,j) = c0

do k=1,km

rhodp(i,j,k) = c0

enddo

enddo

enddo

c

do j=1,jmt

do i=1,imt

phibx(i,j) = c0

phiby(i,j) = c0

enddo

enddo

do j=2,jmm

do i=2,imm

phibx(i,j) = (phib(i+1,j )-phib(i,j ))\*rdxt(j ) +

& (phib(i+1,j+1)-phib(i,j+1))\*rdxt(j+1)

phiby(i,j) = (phib(i,j+1)-phib(i,j)+phib(i+1,j+1)-phib(i+1,j))\*rdy

enddo

enddo

c

do j=2,jmm

phibx(1 ,j) = phibx(imm,j)

phibx(imt,j) = phibx(2 ,j)

phiby(1 ,j) = phiby(imm,j)

phiby(imt,j) = phiby(2 ,j)

enddo

c

c

c------------------------------------------------

c initialize the cumulative monthes of the integration

c----------------------------------------------

month = 1

c

c------------------------------------------------

c read in data for restarting integration

c-------------------------------------------------

if(restrt)then

open(22,file='restr.data',form='unformatted',status='old')

read(22)month,t,pbt,up,vp,upb,vpb,pmtp,ump,vmp

close(22)

cxin month = month + 1

month = 1 !this will print out, strating from month 1

endif

c

return

end

**invtri.F**

#ifdef gm90

c =================

subroutine invtri (wk,topbc,dcb,aidif,dtts2,pbar)

c =================

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'scalar.h'

c

real a8(km),b8(km),c8(km),d8(km),e8(0:km),f8(0:km),g8

real wk(imt,jmt,km),topbc(imt,jmt),dcb(imt,jmt,km),pbar(imt,jmt)

real aidif,dtts2,grp,topbct

c

do 100 j=2,jmm

do 100 i=2,imm

if(itn(i,j).eq.0)goto 100

c

grp = gravr/pbar(i,j)

topbct = topbc(i,j)/gravr

c

do k=2,itn(i,j)

a8(k) = dcb(i,j,k-1)\*rdzw(k-1)\*grp\*rdz(k)\*dtts2\*aidif

c8(k) = dcb(i,j,k )\*rdzw(k )\*grp\*rdz(k)\*dtts2\*aidif

b8(k) = c1 + a8(k) + c8(k)

d8(k) = wk(i,j,k)

e8(k-1) = c0

f8(k-1) = c0

enddo

c

c b. c. at top

k = 1

a8(k) = grp\*rdz(k)\*dtts2\*aidif

c8(k) = dcb(i,j,k)\*rdzw(k)\*grp\*rdz(k)\*dtts2\*aidif

b8(k) = c1 + c8(k)

d8(k) = wk(i,j,k)

e8(k-1) = c0

f8(k-1) = c0

c

c b. c. at bottom

m = itn(i,j)

b8(m) = c1 + a8(m)

c8(m) = grp\*rdz(m)\*dtts2\*aidif

e8(m) = c0

f8(m) = c0

c

c now invert

do k=itn(i,j),1,-1

g8 = c1/(b8(k)-c8(k)\*e8(k))

e8(k-1) = a8(k)\*g8

f8(k-1) = (d8(k)+c8(k)\*f8(k))\*g8

enddo

c

c b.c. at surface

wk(i,j,1) = (e8(0)\*topbct + f8(0))\*tmask(i,j,1)

do k=2,itn(i,j)

wk(i,j,k)=(e8(k-1)\*wk(i,j,k-1)+f8(k-1))\*tmask(i,j,k)

enddo

c

100 continue

return

end

#endif

prsgrd.F

c =================

subroutine prsgrd

c =================

c

c pressure gradient terms

c

c rhodp = int(rho), from

c PCOM: z - pn at T grids, rho is reciprocal of density

c BCOM: 0 - z at T grids, rho is density

c

c PCOM x-component

c 1) pax = atmospheric pressure gradient

c 2) pbxn = int(rhodp\_xbar+z\*rho\_xbar) at ivn(i,j)

c 3) pbxs = int(rhodp\_xbar+z\*rho\_xbar) at ivn(i,j-1)

c 4) pcxn = int((rhodp)x) at ivn(i,j)

c 5) pcxs = int((rhodp)x) at ivn(i,j-1)

c

c PCOM y-component

c 1) pay = atmospheric pressure gradient

c 2) pbye = int(rhodp\_ybar+z\*rho\_ybar) at ivn(i,j)

c 3) pbyw = int(rhodp\_ybar+z\*rho\_ybar) at ivn(i-1,j)

c 4) pcye = int((rhodp)y) at ivn(i,j)

c 5) pcyw = int((rhodp)y) at ivn(i-1,j)

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

include 'cvbc.h'

c

real wka,wkb,wkc,wk3

c

common /works/ wka(imt,jmt),wkb(imt,jmt),wkc(imt,jmt)

common /works/ wk3(imt,jmt,km)

real wkz(kmp1)

c atmospheric pressure gradients

c NOTE: set density=const for both PCOM and BCOM

do j=2,jmt

do i=2,imm

wka(i,j) = rrho\_0\*(bcp(i+1,j)-bcp(i,j))\*rdxt(j)

enddo

enddo

do j=2,jmm

do i=2,imt

wkb(i,j) = rrho\_0\*(bcp(i,j+1)-bcp(i,j))\*rdy

enddo

enddo

do j=2,jmm

do i=2,imm

pax(i,j) = (wka(i,j)+wka(i,j+1))\*p5

pay(i,j) = (wkb(i,j)+wkb(i+1,j))\*p5

enddo

enddo

c PCOM: rhodp(i,j,k) = int(1/density) from z to pn at T grids

c BCOM: rhodp(i,j,k) = int(density) from 0 to z at T grids

do j=2,jmm

do i=2,imm

wkz(1) = c0

do k=1,itn(i,j)

wkz(k+1) = wkz(k) + rho(i,j,k)\*dz(k)

enddo

do k=1,itn(i,j)

#ifdef boussinesq

rhodp(i,j,k) = (wkz(k+1)+wkz(k))\*p5

#else

rhodp(i,j,k) = wkz(itn(i,j)+1) - (wkz(k+1)+wkz(k))\*p5

#endif

enddo

enddo

enddo

c

do k=1,km

do j=2,jmm

rhodp(1 ,j,k) = rhodp(imm,j,k)

rhodp(imt,j,k) = rhodp(2 ,j,k)

enddo

enddo

c (rhodp)xbar & (rho)xbar\*z at (i+1/2,j,k)

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rhodp(i+1,j,k)+rhodp(i,j,k))\*p5

#ifdef boussinesq

& - (rho(i,j,k)+rho(i+1,j,k))\*p5\*z(k)

#else

& + (rho(i,j,k)+rho(i+1,j,k))\*p5\*z(k)

#endif

enddo

enddo

enddo

call vinteg\_ns(wk3,pbxn,pbxs)

c (rhodp)x at (i+1/2,j,k)

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rhodp(i+1,j,k)-rhodp(i,j,k))\*rdxt(j)

enddo

enddo

enddo

call vinteg\_ns(wk3,pcxn,pcxs)

c (rhodp)ybar & (rho)ybar at (i,j+1/2,k)

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rhodp(i,j+1,k)+rhodp(i,j,k))\*p5

#ifdef boussinesq

& - (rho(i,j,k)+rho(i,j+1,k))\*p5\*z(k)

#else

& + (rho(i,j,k)+rho(i,j+1,k))\*p5\*z(k)

#endif

enddo

enddo

enddo

call vinteg\_ew(wk3,pbye,pbyw)

c

c---------------------------------------------------

c (rhodp)y at (i,j+1/2,k)

c----------------------------------------------------

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rhodp(i,j+1,k)-rhodp(i,j,k))\*rdy

enddo

enddo

enddo

call vinteg\_ew(wk3,pcye,pcyw)

c

c

#ifdef boussinesq

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rho(i,j,k)+rho(i+1,j,k))\*p5

enddo

enddo

enddo

call vinteg\_ns(wk3,pdxn,pdxs)

c

do k=1,km

do j=2,jmm

do i=2,imm

wk3(i,j,k) = (rho(i,j,k)+rho(i,j+1,k))\*p5

enddo

enddo

enddo

call vinteg\_ew(wk3,pdye,pdyw)

#endif

c

do j=2,jmm

do i=2,imm

pbxn(i,j) = pbxn(i,j)\*rdxt(j)

pbxs(i,j) = pbxs(i,j)\*rdxt(j)

pbye(i,j) = pbye(i,j)\*rdy

pbyw(i,j) = pbyw(i,j)\*rdy

c

pdxn(i,j) = pdxn(i,j)\*rdxt(j)

pdxs(i,j) = pdxs(i,j)\*rdxt(j)

pdye(i,j) = pdye(i,j)\*rdy

pdyw(i,j) = pdyw(i,j)\*rdy

c

pcxn(i,j) = pcxn(i,j)\*p5

pcxs(i,j) = pcxs(i,j)\*p5

pcye(i,j) = pcye(i,j)\*p5

pcyw(i,j) = pcyw(i,j)\*p5

enddo

enddo

c

c

do j=1,jmt

c

pax(1 ,j) = pax(imm,j)

pax(imt,j) = pax(2 ,j)

pay(1 ,j) = pay(imm,j)

pay(imt,j) = pay(2 ,j)

c

pbxn(1 ,j) = pbxn(imm,j)

pbxn(imt,j) = pbxn(2 ,j)

pbxs(1 ,j) = pbxs(imm,j)

pbxs(imt,j) = pbxs(2 ,j)

pcxn(1 ,j) = pcxn(imm,j)

pcxn(imt,j) = pcxn(2 ,j)

pcxs(1 ,j) = pcxs(imm,j)

pcxs(imt,j) = pcxs(2 ,j)

pdxn(1 ,j) = pdxn(imm,j)

pdxn(imt,j) = pdxn(2 ,j)

pdxs(1 ,j) = pdxs(imm,j)

pdxs(imt,j) = pdxs(2 ,j)

c

pbye(1 ,j) = pbye(imm,j)

pbye(imt,j) = pbye(2 ,j)

pbyw(1 ,j) = pbyw(imm,j)

pbyw(imt,j) = pbyw(2 ,j)

pcye(1 ,j) = pcye(imm,j)

pcye(imt,j) = pcye(2 ,j)

pcyw(1 ,j) = pcyw(imm,j)

pcyw(imt,j) = pcyw(2 ,j)

pdye(1 ,j) = pdye(imm,j)

pdye(imt,j) = pdye(2 ,j)

pdyw(1 ,j) = pdyw(imm,j)

pdyw(imt,j) = pdyw(2 ,j)

c

enddo

c

return

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_readyc.F\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0100664\_0000764\_0000764\_00000020707\_07733607066\_011461\_ 0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ustar \_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_c

c =================

subroutine readyc

c =================

c momentum advections, viscosities & atmpospheric pressure terms

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

include 'cvbc.h'

c

real ubar,vbar,abc,uvmag,t1,t2,t3

real a,b,c,u,v,ux,vx,wua,wub,wva,wvb

c

common /works/ a(imt,jmt),b(imt,jmt),c(imt,jmt)

common /works/ u(imt,jmt),v(imt,jmt)

common /works/ ux(imt,jmt),vx(imt,jmt)

common /works/ wua(imt,jmt),wub(imt,jmt)

common /works/ wva(imt,jmt),wvb(imt,jmt)

c

c

c--------------------------------------------------

c initialize time-averaged pbt (used in baroclinic Eq)

c------------------------------------------------

c

do j=1,jmt

do i=1,imt

pmum(i,j) = pmup(i,j)

pmup(i,j) = pbt (i,j,tau)

enddo

enddo

c

c

c------------------------------------------

c calculate vertical mass advection (dz/dt \* pbt)

c-----------------------------------------------

c

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0)then

spbt(i,j) = p5\*sqrt(pbt(i,j ,tau) + pbt(i+1,j ,tau) +

& pbt(i,j+1,tau) + pbt(i+1,j+1,tau))

endif

enddo

spbt(1 ,j) = spbt(imm,j)

spbt(imt,j) = spbt(2 ,j)

enddo

c

c du,dv are used temporarily for mass advection

c

do k=1,km

do j=1,jmt

do i=1,imt

if(umask(i,j,k).gt.c0)then

du(i,j,k) = spbt(i,j)\*up(i,j,k,tau)

dv(i,j,k) = spbt(i,j)\*vp(i,j,k,tau)\*cosu(j)

else

du(i,j,k) = c0

dv(i,j,k) = c0

endif

enddo

enddo

enddo

c

call upwelling(du,dv,w)

c

c calculate vertical velocity on the surface of U cell

c

do k=1,km

do j=2,jmt

do i=2,imt

a(i,j) = w(i,j,k)/pbt(i,j,tau)

enddo

enddo

do j=2,jmm

do i=2,imm

w(i,j,k) = p25\*(a(i,j)+a(i+1,j)+a(i,j+1)+a(i+1,j+1))

enddo

enddo

enddo

c

c

c------------------------------------------

c calculate advections & horizontal viscosities

c-------------------------------------------

c

do 100 k=1,km

c

c calculate horizontal advection velocities

c

do j=1,jmt

do i=1,imt

if(umask(i,j,k).gt.c0)then

u(i,j) = up(i,j,k,tau)/spbt(i,j)

v(i,j) = vp(i,j,k,tau)/spbt(i,j)

else

u(i,j) = c0

v(i,j) = c0

endif

enddo

enddo

c

c U-advection

c

do j=2,jmm

do i=2,imt

ubar = p5\*(u(i,j)+u(i-1,j))

a(i,j) = p5\*(up(i,j,k,tau)+up(i-1,j,k,tau))\*ubar

b(i,j) = p5\*(vp(i,j,k,tau)+vp(i-1,j,k,tau))\*ubar

c(i,j) = ubar

enddo

enddo

c

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

abc = c(i+1,j)-c(i,j)

ux(i,j) = rdxu(j)\*((a(i+1,j)-a(i,j)) - p5\*up(i,j,k,tau)\*abc)

vx(i,j) = rdxu(j)\*((b(i+1,j)-b(i,j)) - p5\*vp(i,j,k,tau)\*abc)

endif

enddo

enddo

c

c

c V-advection

c

do j=2,jmt

do i=2,imm

vbar = p5\*(v(i,j)\*cosu(j)+v(i,j-1)\*cosu(j-1))

a(i,j) = p5\*(up(i,j,k,tau)+up(i,j-1,k,tau))\*vbar

b(i,j) = p5\*(vp(i,j,k,tau)+vp(i,j-1,k,tau))\*vbar

c(i,j) = vbar

enddo

enddo

c

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

abc = c(i,j+1)-c(i,j)

ux(i,j) = ux(i,j)+rdyu(j)\*((a(i,j+1)-a(i,j))-p5\*up(i,j,k,tau)\*abc)

vx(i,j) = vx(i,j)+rdyu(j)\*((b(i,j+1)-b(i,j))-p5\*vp(i,j,k,tau)\*abc)

endif

enddo

enddo

c

c

c W-advection

c

if(k.eq.1)then

do j=2,jmm

do i=2,imm

wua(i,j) = c0

wva(i,j) = c0

enddo

enddo

else

do j=2,jmm

do i=2,imm

wua(i,j) = wub(i,j)

wva(i,j) = wvb(i,j)

enddo

enddo

endif

c

do j=2,jmm

do i=2,imm

if(k.ge.ivn(i,j))then

wub(i,j) = c0

wvb(i,j) = c0

else

wub(i,j) = w(i,j,k+1)\*(up(i,j,k,tau)+up(i,j,k+1,tau))

wvb(i,j) = w(i,j,k+1)\*(vp(i,j,k,tau)+vp(i,j,k+1,tau))

endif

enddo

enddo

c

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

abc = w(i,j,k) - w(i,j,k+1)

ux(i,j) = ux(i,j)+p5\*rdz(k)\*(wua(i,j)-wub(i,j)-up(i,j,k,tau)\*abc)

vx(i,j) = vx(i,j)+p5\*rdz(k)\*(wva(i,j)-wvb(i,j)-vp(i,j,k,tau)\*abc)

endif

enddo

enddo

c

c

c---------------------------------------------------

c + advection + Pa + viscosities(t=taum)

c---------------------------------------------------

c

if(leapfrog\_c)then

do j=2,jmm

do i=2,imm

du(i,j,k) = - ux(i,j) - pax(i,j)\*spbt(i,j) + diffu(i,j,k)

dv(i,j,k) = - vx(i,j) - pay(i,j)\*spbt(i,j) + diffv(i,j,k)

enddo

enddo

else

do j=2,jmm

do i=2,imm

du(i,j,k) = - ux(i,j) - pax(i,j)\*spbt(i,j)

dv(i,j,k) = - vx(i,j) - pay(i,j)\*spbt(i,j)

enddo

enddo

endif

c

c

c-------------------------------------------------

c calculate horizontal viscosities in tau time level

c-------------------------------------------------

c

do j=2,jmm

do i=2,imt

abc = pbt(i,j,tau)+pbt(i,j+1,tau)

ux(i,j) = abc\*(u(i,j)-u(i-1,j))

vx(i,j) = abc\*(v(i,j)-v(i-1,j))

enddo

enddo

do j=2,jmt

do i=2,imm

abc = pbt(i,j,tau)+pbt(i+1,j,tau)

a(i,j) = abc\*(u(i,j)-u(i,j-1))

b(i,j) = abc\*(v(i,j)-v(i,j-1))

enddo

enddo

c

do j=2,jmm

do i=2,imm

if(umask(i,j,k).gt.c0)then

diffu(i,j,k) = am\*( (sdxu(j)\*(ux(i+1,j)-ux(i,j))+r1c(j)\*a(i,j+1)

& -r1d(j)\*a(i,j))/spbt(i,j)

& + cv1(j)\*up(i,j,k,tau)

& - cv2(j)\*(v(i+1,j)-v(i-1,j))\*spbt(i,j) )

c

diffv(i,j,k) = am\*( (sdxu(j)\*(vx(i+1,j)-vx(i,j))+r1c(j)\*b(i,j+1)

& -r1d(j)\*b(i,j))/spbt(i,j)

& + cv1(j)\*vp(i,j,k,tau)

& + cv2(j)\*(u(i+1,j)-u(i-1,j))\*spbt(i,j) )

endif

enddo

enddo

c

100 continue

c

c

c------------------------------------------------------

c NOTE: u(i,j) & v(i,j) = u&v at bottom (k=ivn(i,j))

c-------------------------------------------------

c

c

c----------------------------------------------

c calculate vertical viscosities in tau time level

c----------------------------------------------

c

c 2-D working array

do j=2,jmm

do i=2,imm

if(ivn(i,j).gt.0) a(i,j)=kappa\_m\*gravr/spbt(i,j)\*\*2

enddo

enddo

c

c

do 200 k=1,km

c

if(k.eq.1)then

c windstress bcu&bcv

do j=2,jmm

do i=2,imm

wua(i,j) = bcu(i,j)\*gravr/spbt(i,j)\*rrho\_0

wva(i,j) = bcv(i,j)\*gravr/spbt(i,j)\*rrho\_0

enddo

enddo

else

do j=2,jmm

do i=2,imm

wua(i,j) = wub(i,j)

wva(i,j) = wvb(i,j)

enddo

enddo

endif

c

do j=2,jmm

do i=2,imm

if(k.eq.ivn(i,j))then

c bottom drag (assume rho=c1 for both PCOM and BCOM)

uvmag = sqrt(u(i,j)\*\*2+v(i,j)\*\*2)

abc = grav/spbt(i,j)\*cdbot\*uvmag

wub(i,j) = u(i,j)\*abc

wvb(i,j) = v(i,j)\*abc

else if(k.lt.ivn(i,j))then

wub(i,j) = a(i,j)\*(up(i,j,k,tau)-up(i,j,k+1,tau))\*rdzw(k)

wvb(i,j) = a(i,j)\*(vp(i,j,k,tau)-vp(i,j,k+1,tau))\*rdzw(k)

else

wub(i,j) = c0

wvb(i,j) = c0

endif

enddo

enddo

c

c

do j=2,jmm

do i=2,imm

diffu(i,j,k) = diffu(i,j,k) + rdz(k)\*(wua(i,j)-wub(i,j))

diffv(i,j,k) = diffv(i,j,k) + rdz(k)\*(wva(i,j)-wvb(i,j))

enddo

enddo

c

c

c--------------------------------------------------

c + diffusion (at tau time level) on euler forward time step

c---------------------------------------------------

c

if(.not.leapfrog\_c)then

do j=2,jmm

do i=2,imm

du(i,j,k) = du(i,j,k) + diffu(i,j,k)

dv(i,j,k) = dv(i,j,k) + diffv(i,j,k)

enddo

enddo

endif

c

200 continue

c

c

c---------------------------------------------------

c vertical integration of du&dv

c---------------------------------------------------

c

call vinteg(du,dub)

call vinteg(dv,dvb)

c

c

return

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_readyt.F\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0100664\_0000764\_0000764\_00000003131\_07733607066\_011472\_ 0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ustar \_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_c

c =================

subroutine readyt

c =================

c calculate the variables depended on stratification

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

c

real a,t1,t2,t3,rdens

c

common /works/ a(imt,jmt)

c

c

c---------------------------------------------------

c calculate density/rho\_0 (BCOM) or 1/density (PCOM)

c---------------------------------------------------

call density

c

c

c---------------------------------------------------

c initialize the time-averaged pbt & up vp

c---------------------------------------------------

do j=1,jmt

do i=1,imt

pmtm(i,j) = pmtp(i,j)

pmtp(i,j) = pbt (i,j,tau)

enddo

enddo

c

do j=2,jmm

do i=2,imm

spbt(i,j) = p5\*sqrt(pbt(i,j ,tau) + pbt(i+1,j ,tau) +

& pbt(i,j+1,tau) + pbt(i+1,j+1,tau))

enddo

spbt(1 ,j) = spbt(imm,j)

spbt(imt,j) = spbt(2 ,j)

enddo

c

do k=1,km

do j=1,jmt

do i=1,imt

umm(i,j,k) = ump(i,j,k)

vmm(i,j,k) = vmp(i,j,k)

ump(i,j,k) = up(i,j,k,tau)\*spbt(i,j)

vmp(i,j,k) = vp(i,j,k,tau)\*spbt(i,j)\*cosu(j)

enddo

enddo

enddo

c

c

c-------------------------------------------------

c calculate pressure gradients related to stratification

c-------------------------------------------------

call prsgrd

c

c

return

end

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_setcon.F\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_0100664\_0000764\_0000764\_00000006561\_07733607066\_011507\_ 0\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ustar \_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_huang\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_c =================

subroutine setcon

c =================

c set scalar quantities

c

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'calendar.h'

c

c----------------------------

c calendar

c------------------------------

data monname /'jan','feb','mar','apr','may','jun',

& 'jul','aug','sep','oct','nov','dec'/

c data daymd /16,14,16,15,16,15,16,16,15,16,15,16/

c data daypm /31,28,31,30,31,30,31,31,30,31,30,31/

data daymd /15,15,15,15,15,15,15,15,15,15,15,15/

data daypm /30,30,30,30,30,30,30,30,30,30,30,30/

c

c

c-------------------------------------------

c physical parameters

c-----------------------------------

c am = constant lateral viscosity coeff for momentum

c ah = constant lateral diffusion coeff for tracers

c kappa\_m = constant vertical viscosity coefficient (cm\*\*2/sec)

c kappa\_h = constant vertical diffusion coefficient (cm\*\*2/sec)

c cdbot = parameter for bottom drag

c

am = 3.0e8

ah = 1.0e7

kappa\_m = 1.0

kappa\_h = 0.3

cdbot = 2.6e-3

c

#ifdef boussinesq

gravr = grav

#else

gravr = grav\*rho\_0

#endif

c

c change the unit of pressure from dynes/cm\*\*2 to decibars

c 1 dynes/cm\*\*2 = 0.1 N/m\*\*2 = 1.0e-3 mbar = 1.0e-5 decibars

c

decibar = 1.0e-5

c

c deltax is used to compute {partial rho}/{partial x}, x=p,t,s

deltap = 1.0e-2

deltat = 1.0e-2

deltas = 1.0e-3

rdeltap = p5/deltap

rdeltat = p5/deltat

rdeltas = p5/deltas

c

c

c-----------------------------------------------

c newtonia restoring coefficient for heat flux

c-----------------------------------------------

c gamma\_t = grav \* ddd/cp

c ddd = 40W/m\*\*2/K

c cp = cpecific heat capacity of sea water, 3901 J/kg/K

c 1.0e-1 = unit change

c

gamma\_t = grav\*40.0/3901.0\*1.0e-1

c

c

c-------------------------------------------

c newtonia restoring time scale for salinity

c-------------------------------------------

gamma\_s = r120\*secday

c

c

c-------------------------------------------------

c asselin temporal filter parameter

c--------------------------------------------------

afb1 = 0.25

afc1 = 0.25

aft1 = 0.25

c

c

c-----------------------------------------------------

c time-step of integration

c--------------------------------------------------

c dtsf: time step for adv\_1 (input unit: min)

c dtuv: time step for adv\_2 (input unit: hour)

c dtts: time step for tracer (input unit: hour)

c

dtsf = 2.0

dtuv = 1.0

dtts = 24.0

c

c

c------------------------------------------------------

c parameters for model's output

c-----------------------------------------------------

c io\_tsuvp = interval for output time average variables (unit year)

c io\_restr = interval for output restr.data (unit year)

c

io\_tsuvp = 1

io\_restr = 1

c

c

return

end

setpbt.F

subroutine setpbt

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

include 'prog.h'

real t0,s0,p0,dens,prea,preb,pre(kmp1)

do k=1,km

do j=1,jmt

do i=1,imt

rho(i,j,k) = c1

pbt(i,j,tau) = c1

enddo

enddo

enddo

c

#ifdef boussinesq

do k=1,km

do j=1,jmt

do i=1,imt

fixp(i,j,k) = c0

enddo

enddo

enddo

#endif

c

c---------------------------------------------------------

c initialize pbt & rho

c---------------------------------------------------------

do 100 j=1,jmt

do 100 i=1,imt

if(itn(i,j).eq.0) goto 100

c

pre(1) = c0

do k=1,itn(i,j)

pre(k+1) = pre(k) + dz(k)

enddo

c

do k=1,itn(i,j)

t0 = pt(i,j,k)

s0 = ps(i,j,k)

p0 = (pre(k)+pre(k+1))\*p5\*decibar

rho(i,j,k) = dens(t0,s0,p0)

enddo

c

pbt(i,j,tau) = c1

c

c

#ifdef boussinesq

do k=1,itn(i,j)

fixp(i,j,k) = (pre(k)+pre(k+1))\*p5\*decibar

enddo

#endif

c

100 continue

#ifdef boussinesq

do j=1,jmt

do i=1,imt

pbt(i,j,tau) = c1

enddo

enddo

#endif

c

ctmp-------------------------------------------------------------

c print\*, 'pbt(90,40,1) =', pbt(90,40,tau)

ctmp-------------------------------------------------------------

return

end

setpn.F

subroutine setpn

c calculate pn, z & dz

c PCOM pn = constant bottom pressure (dy/cm2)

c BCOM pn = grav \* thickness (cm\*cm/s2)

c z = eta; dz=1/z

implicit none

include 'param.h'

include 'pconst.h'

include 'scalar.h'

include 'grdvar.h'

c

real p0,dens,pres

real pre(kmp1),denz(km)

real t30(km),s30(km)

c

c---------------------------------------------------

c input global averaged TS stratification

c-------------------------------------------------

open(82,file='ts30.data',status='old',form='formatted')

read(82,\*) t30,s30

close(82)

c calculate depth of each layer (in cm)

dz0(1) = z0(1)\*c2

do k=2,km

dz0(k) = (z0(k)-z0(k-1))\*c2 - dz0(k-1)

enddo

c calculate pressure at the bottom of each layer

c set an initial density

do k=1,km

denz(k) = c1

enddo

c

pres = c0

c

25 pre(1) = c0

do k=1,km

pre(k+1) = pre(k) + denz(k)\*grav\*dz0(k)

enddo

c

do k=1,km

p0 = (pre(k)+pre(k+1))\*p5\*decibar

denz(k) = dens(t30(k),s30(k),p0)

enddo

if(abs(pre(kmp1)-pres).gt.c1em4)then

pres = pre(kmp1)

goto 25

endif

c calculate z & dz

#ifdef boussinesq

do k=1,km

z(k) = z0(k) \* grav

dz(k) = dz0(k) \* grav

enddo

#else

do k=1,km

z(k) = (pre(k+1)+pre(k))\*p5

dz(k) = pre(k+1)-pre(k)

enddo

#endif

c calculate pn

#ifdef boussinesq

do j=1,jmt

do i=1,imt

if(itn(i,j).eq.0) then

pn(i,j) = c1

else

pn(i,j) = abs(phib(i,j))

endif

enddo

enddo

#else

do j=1,jmt

do i=1,imt

if(itn(i,j).eq.0) then

pn(i,j) = c1

else

pn(i,j) = pre(itn(i,j)+1)

endif

enddo

enddo

#endif

return

end

upwelling.F

c ===========================

subroutine upwelling(u,v,w)

c ===========================

c

c calculate vertical mass advection: pbt\*dz/dt

c upward is positive

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

include 'cvbc.h'

include 'scalar.h'

c

real a,b,dp,t1

real u(imt,jmt,km),v(imt,jmt,km),w(imt,jmt,kmp1)

real c(imt,jmt,km)

c

common /works/ a(imt,jmt),b(imt,jmt),dp(imt,jmt)

c

c

do j=2,jmm

do i=2,imm

dp(i,j) = c0

enddo

enddo

c

do k=1,km

c

do j=2,jmm

do i=1,imm

a(i,j) = u(i,j,k)+u(i,j-1,k)

enddo

enddo

do j=1,jmm

do i=2,imm

b(i,j) = v(i,j,k)+v(i-1,j,k)

enddo

enddo

c

do j=2,jmm

do i=2,imm

t1 = p5\*(rdxt(j)\*(a(i,j)-a(i-1,j))+rdyt(j)\*(b(i,j)-b(i,j-1)))

dp(i,j) = dp(i,j) - t1\*dz(k)

c(i,j,k) = t1

enddo

enddo

c

enddo

c calculate pbt\*dz/dt on the surface of T cell

#ifdef snbc

do j=2,jmm

do i=2,imm

w(i,j,1) = emp(i,j) ! note that w is positive upward

dp(i,j) = dp(i,j) - emp(i,j)

enddo

enddo

#endif

c

do k=2,km

do j=2,jmm

do i=2,imm

w(i,j,k)=(w(i,j,k-1)+dz(k-1)\*(dp(i,j)/pn(i,j)+c(i,j,k-1)))

& \*tmask(i,j,k)

enddo

enddo

enddo

c

do k=1,km

do j=2,jmm

w(1 ,j,k) = w(imm,j,k)

w(imt,j,k) = w(2 ,j,k)

enddo

enddo

c

return

end

vinteg.F

c ==========================

subroutine vinteg(wk3,wk2)

c ==========================

c calculate vertical integration at "u" grid

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

real wk3(imt,jmt,km),wk2(imt,jmt)

c

do j=1,jmt

do i=1,imt

wk2(i,j)=c0

do k=1,ivn(i,j)

wk2(i,j)=wk2(i,j)+dz(k)\*wk3(i,j,k)

enddo

wk2(i,j)=wk2(i,j)\*rzu(i,j)

enddo

enddo

c

return

end

c ==============================

subroutine vinteg\_ns(wk,an,as)

c ==============================

c calculate vertical integration at "u" grid

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

c

real wk(imt,jmt,km),an(imt,jmt),as(imt,jmt)

real tmax,tmin

c

do j=2,jmm

do i=2,imm

m = min(ivn(i,j),ivn(i,j-1))

n = max(ivn(i,j),ivn(i,j-1))

tmin = c0

do k=1,m

tmin = tmin + wk(i,j,k)\*dz(k)

enddo

tmax = tmin

do k=m+1,n

tmax = tmax + wk(i,j,k)\*dz(k)

enddo

if(ivn(i,j).eq.m) then

an(i,j) = tmin\*rzu(i,j)

as(i,j) = tmax\*rzu(i,j-1)

else

an(i,j) = tmax\*rzu(i,j)

as(i,j) = tmin\*rzu(i,j-1)

endif

enddo

enddo

return

end

subroutine vinteg\_ew(wk,ae,aw)

c calculate vertical integration at "u" grid

c

implicit none

include 'param.h'

include 'pconst.h'

include 'grdvar.h'

c

real wk(imt,jmt,km),ae(imt,jmt),aw(imt,jmt)

real tmax,tmin

c

do j=2,jmm

do i=2,imm

m = min(ivn(i,j),ivn(i-1,j))

n = max(ivn(i,j),ivn(i-1,j))

tmin = c0

do k=1,m

tmin = tmin + wk(i,j,k)\*dz(k)

enddo

tmax = tmin

do k=m+1,n

tmax = tmax + wk(i,j,k)\*dz(k)

enddo

if(ivn(i,j).eq.m) then

ae(i,j) = tmin\*rzu(i,j)

aw(i,j) = tmax\*rzu(i-1,j)

else

ae(i,j) = tmax\*rzu(i,j)

aw(i,j) = tmin\*rzu(i-1,j)

endif

enddo

enddo

return

end

calendar.h

c======================= include file "calendar.h"======================

c

c calendar specification arrays

c

c monname = character names of months

c daypm = array of month lengths in days

c daymd = array of the middle date in monthes

c month = cumulative monthes of the integration

c year = the year of model

c mth = calendar month (from 1 to 12)

c day = calendar day (from 1 to daypm(mth))

integer daypm,daymd

integer month,year,mth,day

character monname(12)\*3

c

common /calend/ monname

common /calend/ daypm(12), daymd(12)

common /calend/ month,year,mth,day

cvbc.h

c===== include file "cvbc.h" ==========================

c

c vertical boundary condition variables:

c

c bcf = monthly mean surface forcing fileds. use linear

c interpolation to produce:

c

c bcu = bcf(1) : sea surface zonal windstres (dynes/cm\*\*2)

c bcv = bcf(2) : sea surface meridional windstres(dynes/cm\*\*2)

c bct = bcf(3) : sea surface air temperature (celsius)

c bcp = bcf(4) : sea surface air presure (dynes/cm\*\*2)

c bcs = bcf(5) : sea surface salinity (model unit)

c emp = bcf(6) : rate of evaporation minus precipitation (cm/s)

c ddd = bcf(7) : coefficient for calculation of HF (w/m2/c)

c

real\*4 bcf,bcu,bcv,bct,bcp,bcs,emp,ddd

c

common /cvbc/ bcf(imt,jmt,12,7)

common /cvbc/ bcu(imt,jmt),bcv(imt,jmt)

common /cvbc/ bct(imt,jmt),bcp(imt,jmt),bcs(imt,jmt)

common /cvbc/ emp(imt,jmt),ddd(imt,jmt)

diag.h

c======== include file "diag.h" ==========================

c

c tmn = monthly/annual averaged potential temperature (c)

c smn = monthly/annual averaged salinity (model unit)

c pmn = monthly/annual averaged pbt (dynes/cm\*\*2)

c umn = monthly/annual averaged up

c vmn = monthly/annual averaged vp

c

real tmn,smn,pmn,umn,vmn

c

common /cdiag/ tmn(imt,jmt,km),smn(imt,jmt,km)

common /cdiag/ umn(imt,jmt,km),vmn(imt,jmt,km)

common /cdiag/ pmn(imt,jmt)

grdvar.h

c== include file "grdvar.h" ========================

c

c variables which are functions of the grid

c

c z0 = depth at the center of "t" and "u" grid cells

c dz0 = layer-thickness in z-coordinate

c z = i.e. "eat", at the center of "t" and "u" grid cells

c dz = thickness of "t" and "u" grid cells

c rdz = reciprocal of dz

c rdzw = reciprocal of thickness of "w" cell (in cm)

c phib = potential high at bottom

c pn = constant bottom pressure (PCOM); (-1)\*phib (BCOM)

c zu = int(dz) of "u" grid cell

c rzu = reciprocal of zu

c

real z0,dz0,z,dz,rdz,rdzw,phib,pn,rzu,zu

common /grdv/ z0(km),dz0(km),z(km),dz(km),rdz(km),rdzw(km)

common /grdv/ phib(imt,jmt),pn(imt,jmt),rzu(imt,jmt),zu(imt,jmt)

c

c

c tmask = tracer cell land/sea mask = (0.0, 1.0) on (land, sea)

c umask = velocity cell land/sea mask = (0.0, 1.0) on (land, sea)

real tmask,umask

integer itn,ivn

common /grdv/ tmask(imt,jmt,km),umask(imt,jmt,km)

common /grdv/ itn(imt,jmt),ivn(imt,jmt)

c

c

c cost = cosine of "t" grid point latitude

c cosu = cosine of "u,v" grid point latitude

c ff = 2\*omega\*sine(j)

crdxt = reciprocal of longitudinal width of "t" grid box (in cm)

crdxu = reciprocal of longitudinal width of "u" grid box (in cm)

c rdy = reciprocal of latitudinal height

c rdyt = reciprocal of (latitudinal height \* cost)

c rdyu = reciprocal of (latitudinal height \* cosu)

c dxdyt = dxt \* dy

c dxdyu = dxu \* dy

c area = global ocean area

c cv1 = square of sinu/cosu/a

c cv2 = sinu/cosu \* rdxu/radius

c sdxt = 1/2 square of rdxt

c sdxu = 1/2 square of rdxu

c r1# = coefficients for calculation of diffusion

c ep# = coefficients for semi-implicitly handle of Coriolis term

c

real lat,cost,cosu,ff

real rdxt,rdxu,rdyt,rdyu,rdy

real dxdyt,dxdyu,area

real cv1,cv2,sdxt,sdxu,r1a,r1b,r1c,r1d

real ebea,ebeb,ebla,eblb,epea,epeb,epla,eplb

c

common /grdv/ lat(jmt),cost(jmt),cosu(jmt),ff(jmt)

common /grdv/ rdxt(jmt),rdxu(jmt),rdyt(jmt),rdyu(jmt),rdy

common /grdv/ dxdyt(jmt),dxdyu(jmt),area

common /grdv/ cv1(jmt),cv2(jmt),sdxt(jmt),sdxu(jmt)

common /grdv/ r1a(jmt),r1b(jmt),r1c(jmt),r1d(jmt)

common /grdv/ ebea(jmt),ebeb(jmt),ebla(jmt),eblb(jmt)

common /grdv/ epea(jmt),epeb(jmt),epla(jmt),eplb(jmt)

c

isopyc.h

c isopycnal diffusion variables:

c

c ahisop = isopycnal tracer diffusivity (cm\*\*2/sec)

c athkdf = isopycnal thickness diffusivity (cm\*\*2/sec)

c dptlim = depth limits for the reference pressure levels (in cm).

c the mid point of the two neighboring "dptlim" elements is

c used as the reference pressure (depth) for that interval,

c i.e.,

c

c reference pressure level reference depth

c --------------- -----------------------------

c 1 0.5\*(dptlim(1)+dptlim(2))

c 2 0.5\*(dptlim(2)+dptlim(3))

c . .

c nrpl 0.5\*(dptlim(nrpl)+dptlim(nrpl+1))

c

c REMARK: the first and the last elements of "dptlim" must

c be the depth at the top (0m) and the maximum bottom

c depth, respectively. also, the elements of "dptlim"

c must be in increasing order.

c e = scratch array

c K1 = (1,3) component of the isopycnal mixing tensor computed

c at the center of the eastern face of the "t" grid cell

c K2 = (2,3) component of the isopycnal mixing tensor computed

c at the center of the northern face of the "t" grid cell

c K3 = (3,.) components of the isopycnal mixing tensor

c computed at the top face of the "t" grid cell

c (,,1) --> (3,1) component

c (,,2) --> (3,2) component

c (,,3) --> (3,3) component

c REMARK: (1,1) and (2,2) components of the tensor are

c assumed to be unity. also, (1,2) and (2,1)

c components are assumed to be zero.

c fzisop = function containing the vertical variation of the isopycnal

c diffusion coefficient. "fzisop" multiplies "ahisop".

c kisrpl = isopycnal reference pressure levels for the "t" grid

c point levels computed based on the depth (pressure) at the

c "t" grid points

c krplin = indices indicating the location of the reference pressure

c depth in the 20-level table of polynomial expansion

c variables

c slmxr = reciprocal of maximum slope of isopycnals allowed in mixing

c scheme to prevent excessively large vertical mixing that

c could create numerical instabilities. furthermore, the

c form of the isopycnal diffusion tensor incorporates the

c assumption that horizontal density gradients are much

c smaller than vertical gradients. "slmxr" is also used to

c satisfy this assumption. a value of 100 for "slmxr"

c translates to a slope of 1:100.

c

c rhoi = potential density at "t" cell centers

c

c# ifdef isopycmixspatialvar

c dciso1 = isopycnal tracer diffusivity coefficients modified based

c on the slopes of the isopycnal surfaces on the east face

c of "T" cells.

cdciso2 =isopycnal tracer diffusivity coefficients modified based

c on the slopes of the isopycnal surfaces on the north face

c of "T" cells.

c dslope = half length of the interval in which "ahisop" changes

c with a steep slope from about 0.9\*"ahisop" to about

c 0.1\*"ahisop"

cslopec =slope at which "ahisop" is equal to half of its original

c value

c

c REMARK: 0 <= "slopec", "dslope" <= 1/"slmxr".

c REMARK: because the vertical gradient of density must be

c less than zero for isopycnal mixing, 1/"slmxr" is

c actually a negative maximum slope. this fact is

c taken into account in "isopyc.F". consequently,

c "slmxr" must be a positive number in "blkdta.F".

c# endif

c# ifdef gent\_mcwilliams

c adv\_vetiso = zonal isopycnal mixing velocity computed at the

c center of the eastern face of the "t" cells

c adv\_vntiso = meridional isopycnal mixing velocity computed at

c the center of the northern face of the "t" cells

c (Note: this includes the cosine as in "adv\_vnt")

c adv\_vbtiso = vertical isopycnal mixing velocity computed at the

c center of the top face of the "t" cells

c adv\_fbiso = "adv\_vbtiso" \* (tracer) evaluated at the center of

c the bottom face of the "t" cells

c# endif

c

cmove nrpl = number of reference presure levels used in isopycnal

c integer nrpl

c parameter (nrpl=5)

integer xup,xmd,xlo

parameter(xup=1,xmd=2,xlo=3)

c

real rhoi,e

real K1,K2,K3

real ahisop, athkdf, fzisop, slmxr

real adv\_vetiso

real adv\_vntiso

real adv\_vbtiso

real dciso1, dciso2

real dslope, slopec

cxjin

real kref,rdz0

c

common /cisop/ rhoi(imt,km,jmt,xup:xlo)

common /cisop/ e(imt,kmp1,jmt,3)

common /cisop/ K1(imt,km,jmt,3:3), K2(imt,km,jmt,3:3)

common /cisop/ K3(imt,km,jmt,1:3)

common /cisop/ ahisop, athkdf, fzisop(km), slmxr

cmove common /cisopi/ kisrpl(km), krplin(nrpl),dptlim(nrpl+1)

common /cisop/ adv\_vetiso(imt,km,jmt)

common /cisop/ adv\_vntiso(imt,km,jmt)

common /cisop/ adv\_vbtiso(imt,0:km,jmt)

common /cisop/ dciso1(imt,km,jmt), dciso2(imt,km,jmt)

common /cisop/ dslope, slopec

common /cisop/ kref(km),rdz0(km)

param.h

c=========== include file "param.h" =========================

c

c main parameter file which sets ocean characteristics:

c

c

c imt = number of grid points in longitudinal direction

c jmt = number of grid points in latitudinal direction

c km = number of the sigma levels

c nt = number of tracers (temperature, salinity, ...)

c

integer imt,jmt,km,nt,imm,jmm,kmp1,kmm1,i,j,k,n,m

c

parameter (imt=32,jmt=32,km=30)

parameter (nt=2)

parameter (imm=imt-1,jmm=jmt-1,kmp1=km+1,kmm1=km-1)

pconst.h

c======== include file "pconst.h" ========================

c rules for parameter constants

c use prefix of "c" for whole real numbers (eg: c57 for 57.0)

c use "m" after prefix to designate negative values (minus sign)

c (eg: cm7 for -7.0)

c use prefix of "p" for non repeating fractions (eg: p5 for 0.5)

c use prefix of "r" for reciprocals (eg: r3 for 1/3.0)

ccombine use of prefix above and "e" for scientific notation with

c (eg: c5e4 for 5.0e4, c1em10 for 1.0e-10)

real c0,c1,c2,c3,c4,c5,c6,c25,c35

parameter (c0=0.0,c1=1.0,c2=2.0,c3=3.0,c4=4.0)

parameter (c5=5.0,c6=6.0,c25=25.0,c35=35.0)

c

real p125,p25,p5

parameter (p125=0.125,p25=0.25, p5=0.5)

c

real c60,c1440,c3600

parameter (c60=60.0, c1440=1440.0, c3600=3600.0)

c

real c1e3,c1em4,c1em12

parameter (c1e3=1.0e3,c1em4=1.0e-4,c1em12=1.0e-12)

c

real r120,r150,r180,r365

parameter (r120=c1/120.0,r150=c1/150.0)

parameter (r180=c1/180.0,r365=c1/365.0)

c

real secday

parameter (secday=c1/(c60\*c1440))

c

real rho\_0,rrho\_0

parameter (rho\_0=1.029,rrho\_0=c1/rho\_0)

c

real tbice

parameter (tbice=-1.5)

c

c

real pi,torad,radius,omega,grav

parameter (pi = 3.141592653589793)

parameter (torad = pi\*r180 )

parameter (radius = 6370.0d5 )

parameter (omega = pi/43082.0 )

parameter (grav = 980.0)

c

c grav = earth's gravitational acceleration (cm/sec\*\*2)

c

prog.h

include file "prog.h" ==========================

c

c variables used for prognostic and diagnostic equations

c

c tau = tau time level

c taum = tau-1 time level

c

integer tau,taum

parameter (tau=1,taum=2)

c

c----------------------------------

c arrays for prognostic equations:

c--------------------------------------------

c

c t = tracer (nt=1 for temperature, 2 for salinity)

c up = u \* sqrt (pbt) (cm/s \* spbt)

c vp = v \* sqrt (pbt) (cm/s \* spbt)

c pbt=pressure difference between bottom and surface dynes/cm\*\*2)

c upb = barotropic component of up (cm/s \* spbt)

c vpb = barotropic component of vp (cm/s \* spbt)

c

cnote:temperature is potential temperature in degrees Celsius and

csalinity is in "model units",the deviation from 0.035g of salt/g

c of water, assuming a water density of 1 g/cm\*\*3

c

real t,up,vp,pbt,upb,vpb

c

common /vprog/ t(imt,jmt,km,nt,2)

common /vprog/ up(imt,jmt,km,2),vp(imt,jmt,km,2)

common /vprog/ pbt(imt,jmt,2)

common /vprog/ upb(imt,jmt,2),vpb(imt,jmt,2)

c arrays for diagnostic equations:

c

c spbt = sqrt(pbt)

c rho = density

c w = pbt \* dz/dt at "t" cell or dz/dt at "u" cell

c

real spbt,rho,w

c

common /vdiag/ spbt(imt,jmt)

common /vdiag/ rho(imt,jmt,km)

common /vdiag/ w(imt,jmt,kmp1)

c-------------------------------------------

c woring arrays

c-------------------------------------------

c du = advection + diffusion + pressure gradients for up

c dv = advection + diffusion + pressure dradients for vp

c dub = vertical integration of du

c dvb = vertical integration of dv

c

real du,dv,dub,dvb

common /vwork/ du(imt,jmt,km),dv(imt,jmt,km)

common /vwork/ dub(imt,jmt),dvb(imt,jmt)

c diffu = viscosity term for up eq.

c diffv = viscosity term for vp eq.

c pt = {partial rho} over {partial reference temperature}

c ps = {partial rho} over {partial reference salinity}

real diffu,diffv,pt,ps

common /vwork/ diffu(imt,jmt,km),diffv(imt,jmt,km)

common /vwork/ pt(imt,jmt,km),ps(imt,jmt,km)

c pmup = pbt at tau+1/2 time level (baroclinic eq)

c pmum = pbt at tau-1/2 time level ''

c pmtp = pbt at tau+1/2 time level (tracer eq)

c pmtm = pbt at tau-1/2 time level ''

c ump = u at tau+1/2 time level ''

c umm = u at tau-1/2 time level ''

c vmp = v at tau+1/2 time level ''

c vmm = v at tau-1/2 time level ''

c

real pmup,pmum,pmtp,pmtm,ump,umm,vmp,vmm

common /vwork/ pmup(imt,jmt),pmum(imt,jmt)

common /vwork/ pmtp(imt,jmt),pmtm(imt,jmt)

common /vwork/ ump(imt,jmt,km),umm(imt,jmt,km)

common /vwork/ vmp(imt,jmt,km),vmm(imt,jmt,km)

c arrays rlated to pressure gradents

real rhodp,pax,pay

real pbxn,pbxs,pcxn,pcxs,pdxn,pdxs

real pbye,pbyw,pcye,pcyw,pdye,pdyw

real phibx,phiby

c

common /vprsg/ rhodp(imt,jmt,km)

common /vprsg/ pax(imt,jmt),pay(imt,jmt)

common /vprsg/ pbxn(imt,jmt),pbxs(imt,jmt)

common /vprsg/ pcxn(imt,jmt),pcxs(imt,jmt)

common /vprsg/ pdxn(imt,jmt),pdxs(imt,jmt)

common /vprsg/ pbye(imt,jmt),pbyw(imt,jmt)

common /vprsg/ pcye(imt,jmt),pcyw(imt,jmt)

common /vprsg/ pdye(imt,jmt),pdyw(imt,jmt)

common /vprsg/ phibx(imt,jmt),phiby(imt,jmt)

c

c#ifdef boussinesq

real fixp

common /vprsg/ fixp(imt,jmt,km)

c#endif

scalar.h

c=== include file "scalar.h" ========================

c scalar quantities:

c

c dtts = time step for tracers (in seconds)

c dtuv = time step for solving baroclinic eq (in seconds)

c dtsf = time step for solving barotropic eq (in seconds)

c c2dtts = 2\*dtts

c c2dtuv = 2\*dtuv

c c2dtsf = 2\*dtsf

c nss = number of time steps for tracer eq

c ncc = number of time steps for baroclinic eq

c nbb = number of time steps for barotropic eq

c

c af# = asselin temporal filter parameters

c jst/jed = starting/ending latitude for filter

c decibar = unit factor for density calculation

c delta# = delta #, for calculation of {partial rho}/{partial #}

c

c am = constant lateral viscosity coeff for momentum

c ah = constant lateral diffusion coeff for tracers

c kappa\_m = constant vertical viscosity coefficient (cm\*\*2/sec)

c kappa\_h = constant vertical diffusion coefficient (cm\*\*2/sec)

c gamma\_t = parameter for calculation of surface heat flux

c gamma\_s = parameter for surface salinity b.c.

c cdbot = parameter for bottom drag

c

c ==================

c control variables:

c ==================

c

c runlen = integration length (in months)

c

c restrt = (true,false) indicates that this run is a

c (restart, start from initial conditions)

c

c euler\_back = (false,true) on a (eular forward, backward) time step

c in "adv-1"

c

c leapfrog\_b = (false,true) on a (eular backward, normal leapfrog)

c time step in "adv-1"

c

c leapfrog\_c = (false,true) on a (eular forward, normal leapfrog)

c time step in "adv-2"

c

c leapfrog\_t = (false,true) on a (eular forward, normal leapfrog)

c time step in "tracer"

c

c io\_tsuvp = interval for annual/monthly mean output (in year)

c io\_restr = interval for saving data for restarting (in year)

c

integer nss,ncc,nbb,jstn,jedn,jsts,jeds

real onbb,oncc,onbc

real dtts,dtuv,dtsf,c2dtts,c2dtuv,c2dtsf

real afb1,afc1,aft1,afb2,afc2,aft2

real decibar,deltap,rdeltap,deltat,deltas,rdeltat,rdeltas

real am,ah,kappa\_m,kappa\_h,gamma\_t,gamma\_s,cdbot,gravr

c

common /scalar/ nss,ncc,nbb,onbb,oncc,onbc

common /scalar/ dtts,dtuv,dtsf,c2dtts,c2dtuv,c2dtsf

common /scalar/ fb1,afc1,aft1,afb2,afc2,aft2,jstn,jedn,jsts,jeds

common /scalar/ decibar

common /scalar/ deltap,rdeltap,deltat,deltas,rdeltat,rdeltas

common /scalar/ am,ah,kappa\_m,kappa\_h,gamma\_t,gamma\_s,cdbot

common /scalar/ gravr

c

real runlen

integer io\_tsuvp,io\_restr

logical euler\_back,leapfrog\_b,leapfrog\_c,leapfrog\_t

logical restrt

c

common /switcr/ runlen,restrt

common /switcr/ euler\_back,leapfrog\_b,leapfrog\_c,leapfrog\_t

common /switcr/ io\_tsuvp,io\_restr