```
module pumamod

!*******************************************!
! Portable University Model of the Atmosphere !
!*******************************************!
! Version: 17.0   16-Feb-2011                !
!*******************************************!
! Klaus Fraedrich                            !
! Frank Lunkeit  - Edilbert Kirk             !
! Frank Sielmann - Torben Kunz               !
! Hartmut Borth                              !
!*******************************************!
! Meteorologisches Institut                  !
! KlimaCampus - Universitaet Hamburg         !
!*******************************************!
! http://www.mi.uni-hamburg.de/puma          !
!*******************************************!

real :: wwt = 13713.4258

!************************************************************!
! The number of processes for processing on parallel machines  !
! NLAT/2 must be dividable by <npro>. npro can be set by the   !
! option -n <npro> when calling the puma executable            !
! This option is only available if the code is compiled with   !
! an mpi compiler.                                             !
!************************************************************!
integer :: npro = 1

!************************************************************!
! The horizontal resolution of PUMA is set by defining the     !
! number of latitudes <nlev> with the 1st. command line        !
! parameter and the number of levels with the 2nd. command     !
! parameter. A typical call for T42 is:                        !
! puma.x 64 10                                                 !
! which sets nlat=64 and nlev=10                               !
!************************************************************!
integer :: nlat = 32

!example values:  32,  48,  64, 128,  192,  256,  512,  1024
!truncation:     T21, T31, T42, T85, T127, T170, T341,  T682

integer :: nlev = 10

!*****************************************************!
! Grid related paramters, which are computed from the !
! command line arguments <nlat> and <nlev>            !
! Preset values are for T21 (nlat=32) and nlev=10     !
! ***************************************************!

integer :: nlem =    9 ! Levels - 1
integer :: nlep =   11 ! Levels + 1
integer :: nlsq =  100 ! Levels squared

integer :: nlon =   64 ! Longitudes = 2 * latitudes
integer :: nlah =   16 ! Half of latitudes
integer :: ntru =   21 ! (nlon-1) / 3
integer :: ntp1 =   22 ! ntru + 1
integer :: nzom =   44 ! Number of zonal modes
integer :: nrsp =  506 ! (ntru+1) * (ntru+2)
integer :: ncsp =  253 ! nrsp / 2
integer :: nspp =  506 ! nodes per process
integer :: nesp =  506 ! number of extended modes

integer :: nlpp =   32 ! Latitudes per process
integer :: nhpp =   16 ! Half latitudes per process
integer :: nhor = 2048 ! Horizontal part
integer :: nugp = 2048 ! Horizontal total
integer :: npgp = 1024 ! Horizontal total packed words
```



```
integer :: nud  =    6 ! I/O unit for diagnostic output

!***********!
! filenames !
!***********!
character (256) :: puma_namelist       = "puma_namelist"
character (256) :: puma_output         = "puma_output"
character (256) :: puma_diag           = "puma_diag"
character (256) :: puma_restart        = "puma_restart"
character (256) :: puma_status         = "puma_status"
character (256) :: efficiency_dat      = "efficiency.dat"
character (256) :: ppp_puma_txt        = "ppp-puma.txt"
character (256) :: puma_sp_init        = "puma_sp_init"

! ****************************************************************
! * For multiruns the instance number is appended to the filename *
! * e.g.: puma_namelist_1 puma_diag_1 etc. for instance # 1       *
! ****************************************************************

! ****************************************************************
! * Don't touch the following parameter definitions !            *
! ****************************************************************
integer, parameter :: PUMA   = 0        ! Model ID
integer, parameter :: PLASIM = 1        ! Model ID

parameter(MAXLEV = 100)                 ! Maximum level dimension
parameter(NROOT  =   0)                 ! Master node

parameter(PI     = 3.141592653589793D0) ! Pi
parameter(TWOPI  = PI + PI)             ! 2 Pi

parameter(AKAP_EARTH   = 0.286 )       ! Kappa Earth (Poisson constant R/Cp)
parameter(ALR_EARTH    = 0.0065)       ! Lapse rate Earth
parameter(GA_EARTH     = 9.80665)      ! Gravity Earth (mean on NN)
parameter(GASCON_EARTH = 287.0)        ! Gas constant for dry air on Earth
parameter(PSURF_EARTH  = 101100.0)     ! Mean Surface pressure [Pa] on Earth
                                       ! Trenberth 1981, J. Geoph. Res., Vol.86, 5238-5246
parameter(PLARAD_EARTH = 6371220.0)    ! Earth radius

parameter(PNU    = 0.02)               ! Time filter
parameter(PNU21  = 1.0 - 2.0*PNU)      ! Time filter 2

! ****************************************************************
! * EZ: Factor to multiply the spherical harmonic Y_(1,0) to get  *
! * the non-dimensional planetary vorticity 2 sin(phi). In PUMA   *
! * Y_(1,0) = sqrt(3/2)*sin(phi) (normalization factor 1/sqrt(2)).*
! * The time scale must be given by Tscale = 1/Omega             *
! ****************************************************************

parameter(EZ     = 1.632993161855452D0) ! ez = 1 / sqrt(3/8)


! ************************************************************
! * Planetary parameters & Scales                            *
! * -----------------------------                            *
! * The Puma model is formulated in non-dimensional form with  *
! * the planetary radius as length scale and the reciprocal of *
! * the planetary rotation rate as time scale. The temperature *
! * scale is given by the geopotential scale divided by the    *
! * gas constant.                                            *
! * For the time scale the length of the siderial day is used  *
! * as basic unit                                            *
! * The parameters are initialized for Earth settings. They    *
! * may be modified by the namelist file <puma_namelist>       *
! *                                                          *
! * The scales are derived internal quantities               *
! ************************************************************
```

```
real :: omega        = TWOPI / 86164.0 ! Default scaling
real :: sid_day      = 86164.0         ! Length of sideral day [sec] on Earth
real :: sol_day      = 86400.0         ! Length of solar   day [sec] on Earth
real :: plarad       = PLARAD_EARTH    ! Planetary radius [m] on Earth
real :: gascon       = GASCON_EARTH    ! Dry air gas consant [J/K kg] on Earth
real :: akap         = AKAP_EARTH      ! Kappa [] on Earth
real :: alr          = ALR_EARTH       ! average lapse rate [K/km] on Earth
real :: ga           = GA_EARTH        ! Gravity [m/sec*sec] on Earth
real :: psurf        = PSURF_EARTH     ! Mean surface pressure for EARTH [Pa]

real :: ww_time      = 0.0             ! time scale [sec] (day / 2 Pi)
real :: ww_scale     = 0.0             ! reciprocal of time scale [1/sec]
real :: cv           = 0.0             ! velocity scale [m/sec] on Earth
real :: ct           = 0.0             ! temperature scale [K] on Earth

! *************************
! * Global Integer Scalars *
! *************************

logical :: lrestart =  .false. ! Existing "puma_restart" sets to .true.
logical :: lselect  =  .false. ! true: disable some zonal waves
logical :: lspecsel =  .false. ! true: disable some spectral modes

integer :: model    = PUMA

integer :: kick     =  1 ! kick > 0 initializes eddy generation
integer :: nafter   =  0 ! write data interval 0: controlled by nwpd
integer :: nwpd     =  1 ! number of writes per day
integer :: ncoeff   =  0 ! number of modes to print
integer :: ndel     =  6 ! ndel
integer :: ndiag    = 12 ! write diagnostics interval
integer :: newsr    =  0 ! 1: recalculate sr1 and sr2 after restart
integer :: ngui     =  0 ! activate Graphical User Interface    !XW(Mar/25/2017)
 0=off
integer :: nkits    =  3 ! number of initial timesteps
integer :: nlevt    =  9 ! tropospheric levels (set_vertical_grid)
integer :: noutput  =  1 ! global switch for output on (1) or off (0)
integer :: nwspini  =  1 ! write sp_init after initialization
integer :: nrun     =  0 ! if (nstop == 0) nstop = nstep + nrun
integer :: nstep1   =  0 ! start step (for cpu statistics)
integer :: nstep    = -1 ! current timestep step 0: 01-Jan-0001  00:00
integer :: nstop    =  0 ! finishing timestep
integer :: ntspd    =  0 ! one day = ntspd timesteps
integer :: mpstep   =  0 ! minutes per step 0 = automatic
integer :: ncu      =  0 ! check unit (debug output)
integer :: nwrioro  =  1 ! controls output of orography
integer :: nextout  =  0 ! 1: extended output (entropy production)
integer :: nruido   =  0 ! 1: global constant, temporal noise
!                          2: spatio-temporal noise
!                          3: spatio-temporal equator symmetric
integer :: nseedlen =  0 ! length of random seed (set by lib call)
integer :: nmonths  =  0 ! Simulation time (1 month =  30 days)
integer :: nyears   =  1 ! simulation time (1 year  = 360 days)
integer :: nsponge  =  0 ! 1: Create sponge layer
integer :: nhelsua  =  0 ! 1: Set up Held & Suarez T_R field
!                            instead of original PUMA T_R field
!                         2: Set up Held & Suarez T_R field
!                            instead of original PUMA T_R field
!                            AND use latitudinally varying
!                            heating timescale in PUMA (H&Z(94)),
!                            irrelevant for PumaPreProcessor (ppp)
!                         3: Use latitudinally varying
!                            heating timescale in PUMA (H&Z(94)),
!                            irrelevant for PumaPreProcessor (ppp)
integer  :: ndiagp  = 0 ! 0/1 switch for grid point diabatic heating
integer  :: nconv   = 0 ! 0/1 switch for convecive heating
integer  :: nvg     = 0 ! type of vertical grid
                        ! 0 = linear
                        ! 1 = Scinocca & Haynes
```



```
                        ! 2 = Polvani & Kushner
integer  :: nenergy = 0 ! energy diagnostics (on/off 1/0)
integer  :: nentropy= 0 ! entropy diagnostics (on/off 1/0)
integer  :: ndheat  = 0 ! energy recycling (on/off 1/0)

integer  :: nradcv = 0 ! use two restoration fields



! ************************
! * Global Real Scalars *
! ************************

real :: alpha  =     1.0  ! Williams filter factor
real :: alrs   =     0.0  ! stratospheric lapse rate [K/m]
real :: delt              ! normalized timestep
real :: delt2             ! 2 * delt
real :: dtep   =    60.0  ! delta T equator <-> pole  [K]
real :: dtns   =   -70.0  ! delta T   north <-> south [K]
real :: dtrop  = 12000.0  ! Tropopause height [m]
real :: dttrp  =     2.0  ! Tropopause smoothing [K]
real :: dtzz   =    10.0  ! delta(Theta)/H additional lapserate in
                          ! Held & Suarez T_R field
real :: orofac =    1.0   ! factor to scale the orograpy
real :: plavor =    EZ    ! planetary vorticity
real :: psmean = PSURF_EARTH ! Mean of Ps on Earth
real :: rotspd =     1.0  ! rotation speed 1.0 = normal Earth rotation
real :: sigmax =  6.0e-7  ! sigma for top half level
real :: spstep =  0.0     ! seconds per step 0 = automatic
real :: diffts = 21600.0  ! diffusion time scale [sec]
real :: tac    =   360.0  ! length of annual cycle [days] (0 = no cycle)
real :: pac    =     0.0  ! phase of the annual cycle [days]
real :: tgr    =   288.0  ! Ground Temperature in mean profile [K]
real :: dvdiff =     0.0  ! vertical diffusion coefficient [m2/s]
!                         ! dvdiff =0. means no vertical diffusion
real :: disp   =     0.0  ! noise dispersion
real :: tauta  =    40.0  ! heating timescale far from surface
real :: tauts  =     4.0  ! heating timescale close to surface
real :: pspon  = 50.      ! apply sponge layer where p < pspon
!                         ! pressure [Pa]
real :: dcsponge = 0.5 / 86400.0 ! damping coefficient for sponge layer [1/sec]

! *************************
! * Global Spectral Arrays *
! *************************

real, allocatable :: sd(:,:)  ! Spectral Divergence
real, allocatable :: sdd(:,:) ! Difference between instances
real, allocatable :: st(:,:)  ! Spectral Temperature
real, allocatable :: std(:,:) ! Difference between instances
real, allocatable :: st1(:,:) ! Spectral Temperature at t-1 (for NEXTOUT == 1)
real, allocatable :: st2(:,:) ! Spectral Temperature at t-2 (for NEXTOUT == 1)
real, allocatable :: sz(:,:)  ! Spectral Vorticity
real, allocatable :: szd(:,:) ! Difference between instances
real, allocatable :: sp(:)    ! Spectral Pressure (ln Ps)
real, allocatable :: spd(:)   ! Difference between instances
real, allocatable :: sq(:,:)  ! For compatibility with PlaSim
real, allocatable :: sp1(:)   ! Spectral Pressure at t-1 (for NEXTOUT == 1)
real, allocatable :: sp2(:)   ! Spectral Pressure at t-2 (for NEXTOUT == 1)
real, allocatable :: so(:)    ! Spectral Orography
real, allocatable :: sr1(:,:) ! Spectral Restoration Temperature
real, allocatable :: sr2(:,:) ! Spectral Restoration Temperature

real, allocatable :: sdp(:,:) ! Spectral Divergence  Partial
real, allocatable :: stp(:,:) ! Spectral Temperature Partial
real, allocatable :: szp(:,:) ! Spectral Vorticity   Partial
real, allocatable :: spp(:)   ! Spectral Pressure    Partial
real, allocatable :: sop(:)   ! Spectral Orography   Partial
real, allocatable :: srp1(:,:)! Spectral Restoration Partial
```

```
real, allocatable :: srp2(:,:)! Spectral Restoration Partial

real, allocatable :: sdt(:,:) ! Spectral Divergence  Tendency
real, allocatable :: stt(:,:) ! Spectral Temperature Tendency
real, allocatable :: szt(:,:) ! Spectral Vorticity   Tendency
real, allocatable :: spt(:)   ! Spectral Pressure    Tendency

real, allocatable :: sdm(:,:) ! Spectral Divergence  Minus
real, allocatable :: stm(:,:) ! Spectral Temperature Minus
real, allocatable :: szm(:,:) ! Spectral Vorticity   Minus
real, allocatable :: spm(:)   ! Spectral Pressure    Minus

real, allocatable :: sak(:)   ! Hyper diffusion
real, allocatable :: srcn(:)  ! 1.0 / (n * (n+1))
real, allocatable :: span(:)  ! Pressure for diagnostics
real, allocatable :: spnorm(:)! Factors for output normalization

integer, allocatable :: nindex(:)  ! Holds wavenumber
integer, allocatable :: nscatsp(:) ! Used for reduce_scatter op
integer, allocatable :: nselzw(:)  ! Enable/disable selected zonal waves
integer, allocatable :: nselsp(:)  ! Enable/disable slected spectral modes

! ***************************
! * Global Gridpoint Arrays *
! ***************************

real, allocatable :: gd(:,:)     ! Divergence
real, allocatable :: gt(:,:)     ! Temperature
real, allocatable :: gz(:,:)     ! Vorticity
real, allocatable :: gu(:,:)     ! u * cos(phi)
real, allocatable :: gv(:,:)     ! v * cos(phi)
real, allocatable :: gp(:)       ! Ln(Ps)
real, allocatable :: gq(:,:)     ! For compatibilty with PlaSim
real, allocatable :: gfu(:,:)    ! Term Fu in Primitive Equations
real, allocatable :: gfv(:,:)    ! Term Fv in Primitive Equations
real, allocatable :: gut(:,:)    ! Term u * T
real, allocatable :: gvt(:,:)    ! Term v * T
real, allocatable :: gke(:,:)    ! Kinetic energy u * u + v * v
real, allocatable :: gpj(:)      ! d(Ln(Ps)) / d(mu)
real, allocatable :: rcsq(:)     ! 1 / cos2(phi)
real, allocatable :: ruido(:,:,:)! noise (nlon,nlat,nlev)
real, allocatable :: ruidop(:,:) ! noise partial (nhor,nlev)
real, allocatable :: gtdamp(:,:) ! 3D reciprocal damping times [1/sec]
                                 ! for relaxation in grid point space
                                 ! for radiative restoration temperature
                                 ! (e.g. for Held&Suarez)
real, allocatable :: gr1(:,:)    ! constant radiative restoration time scale
real, allocatable :: gr2(:,:)    ! variable radiative restoration time scale
real, allocatable :: gtdampc(:,:)! the same as gtdamp, but for convective
                                 ! restoration temperature
real, allocatable :: gr1c(:,:)   ! constant convective restoration time scale
real, allocatable :: gr2c(:,:)   ! variable convective restoration time scale

! **********************
! * Diagnostic Arrays *
! **********************

integer, allocatable :: ndil(:) ! Set diagnostics level

real, allocatable :: csu(:,:) ! Cross section u [m/s]
real, allocatable :: csv(:,:) ! Cross section v [m/s]
real, allocatable :: cst(:,:) ! Cross section T [Celsius]

real,allocatable :: denergy(:,:)     ! energy diagnostics
real,allocatable :: dentropy(:,:)    ! entropy diagnostics

! *******************
! * Latitude Arrays *
! *******************
```



```
character (3),allocatable :: chlat(:) ! label for latitudes
real (kind=8),allocatable :: sid(:)   ! sin(phi)
real (kind=8),allocatable :: gwd(:)   ! Gaussian weight (phi)
real,allocatable :: csq(:)            ! cos2(phi)
real,allocatable :: rcs(:)            ! 1/cos(phi)

! ****************
! * Level Arrays *
! ****************

real, allocatable :: t0(:)            ! reference temperature
real, allocatable :: t0d(:)           ! vertical t0 gradient
real              :: taur(MAXLEV)     ! tau R [sec]
real              :: tauf(MAXLEV)     ! tau F [sec]
real, allocatable :: damp(:)          ! 1.0 / (2 Pi * taur)
real, allocatable :: fric(:)          ! 1.0 / (2 Pi * tauf)

real, allocatable :: bm1(:,:,:)
real, allocatable :: dsigma(:)
real, allocatable :: rdsig(:)
real, allocatable :: sigma(:)         ! full level sigma
real, allocatable :: sigmh(:)         ! half level sigma
real, allocatable :: tkp(:)
real, allocatable :: c(:,:)
real, allocatable :: xlphi(:,:)       ! matrix Lphi (g)
real, allocatable :: xlt(:,:)         ! matrix LT (tau)

! ******************
! * Parallel Stuff *
! ******************

integer :: myworld = 0                  ! MPI variable
integer :: mpinfo  = 0                  ! MPI variable
integer :: mypid   = 0                  ! My Process Id
real    :: tmstart = 0.0                ! CPU time at start
real    :: tmstop  = 0.0                ! CPU time at stop
character(80), allocatable :: ympname(:) ! Processor name


! **********************
! * Multirun variables *
! **********************

integer :: mrworld =  0   ! MPI communication
integer :: mrinfo  =  0   ! MPI info
integer :: mrpid   = -1   ! MPI instance id
integer :: mrnum   =  0   ! MPI number of instances
integer :: mintru  =  0   ! Lowest resolution of all instances
integer :: mrdim   =  0   ! Exchange dimension  (min. NRSP)
integer :: nsync   =  0   ! Synchronization on or off
integer, allocatable :: mrtru(:) ! Truncations of members

real    :: syncstr  =  0.0 ! Coupling strength (0 .. 1)
real    :: syncsecs =  0.0 ! Coupling time [sec]

! ****************************************
! * GUI (Graphical User Interface for X11) *
! ****************************************

!XW(Mar/25/2017) to remove GUI:
!parameter (NPARCS = 10)          ! Number of GUI parameters
!integer :: nguidbg   =  0        ! Flag for GUI debug output    !XW(Mar/25/2017)
 0=off
!integer :: nshutdown =  0        ! Flag for shutdown request
!integer :: ndatim(6) = -1        ! Date & time array
!real(kind=4) :: parc(NPARCS)     ! Values of GUI parameters
!real(kind=4) :: crap(NPARCS)     ! Backup of parc(NPARCS)
!logical :: ldtep   = .FALSE.     ! DTEP changed by GUI
```

```
!logical :: ldtns   = .FALSE.    ! DTNS changed by GUI
!character(len=32) :: yplanet = "Earth"

! ***************
! * Random seed *
! ***************

integer              :: seed(8) = 0 ! settable in namelist
integer, allocatable :: meed(:)     ! machine dependent seed
real                 :: ganext = 0.0! y part of gaussian noise

end module pumamod

!***************!
! MODULE RADMOD !
!***************!

module radmod       ! Dummy declaration for compatibility
use pumamod         ! with PLASIM (needed in guimod)
end module radmod


! ***************** !
! * MODULE PPPMOD * !
! ***************** !

module prepmod
integer :: num_ppp     = 0
integer :: nlat_ppp(1) = 0
integer :: nlev_ppp(1) = 0

type ppp_type
   character (80)   :: name    ! name of variable or array
   logical          :: isint   ! .true. for integer
   integer          :: n       ! length of vector (1 for scalar)
   integer, pointer, dimension(:) :: pint    ! pointer to integer array
   real   , pointer, dimension(:) :: preal   ! pointer to real    array
end type ppp_type

type(ppp_type) :: ppp_tab(30)

contains
   subroutine ppp_def_int(pname,nvar,ndim)
   character (*)  :: pname
   integer,target :: nvar(ndim)

   num_ppp = num_ppp + 1
   ppp_tab(num_ppp)%name  =  '[' // trim(pname) // ']'
   ppp_tab(num_ppp)%isint =  .true.
   ppp_tab(num_ppp)%n     =  ndim
   ppp_tab(num_ppp)%pint  => nvar
   ppp_tab(num_ppp)%preal => null()
   return
   end subroutine ppp_def_int

   subroutine ppp_def_real(pname,rvar,ndim)
   character (*)  :: pname
   real   ,target :: rvar(ndim)

   num_ppp = num_ppp + 1
   ppp_tab(num_ppp)%name  =  '[' // trim(pname) // ']'
   ppp_tab(num_ppp)%isint =  .false.
   ppp_tab(num_ppp)%n     =  ndim
   ppp_tab(num_ppp)%pint  => null()
   ppp_tab(num_ppp)%preal => rvar
   return
   end subroutine ppp_def_real
end module prepmod
```



```
! *********************
! * PROGRAM PUMA_MAIN *
! *********************

program puma_main
use pumamod

! ***********
! * History *
! ***********

! 1972 - W. Bourke:
!        An efficient one-level primitive equation spectral model
!        Mon. Weath. Rev., 100, pp. 683-689

! 1975 - B.J. Hoskins and A.J. Simmons:
!        A multi-layer spectral model and the semi-implicit method
!        Qart. J. R. Met. Soc., 101, pp. 637-655

! 1993 - I.N. James and J.P. Dodd:
!        A Simplified Global Circulation Model
!        Users' Manual, Dept. of Meteorology, University of Reading

! 1998 - Klaus Fraedrich, Edilbert Kirk, Frank Lunkeit
!        Portable University Model of the Atmosphere
!        DKRZ Technical Report No. 16

! 2009 - PUMA Version 16.0
!        http://www.mi.uni-hamburg.de/puma

! ******************
! * Recent Changes *
! ******************

! 10-Jun-2002 - Puma Workshop - Documentation of subroutine SPECTRAL
! 04-Jul-2002 - Frank Lunkeit - Annual cycle
! 08-Jul-2002 - Edilbert Kirk - Factor for rotation speed
! 25-Sep-2002 - Puma Workshop - Documentation of subroutine CALCGP
! 11-Nov-2002 - Edilbert Kirk - Add Orography to output file
! 26-Feb-2003 - Edilbert Kirk - Read preprocessed initial file
! 07-Sep-2004 - Edilbert Kirk - Graphical User Interface
! 23-Aug-2006 - Torben Kunz   - Held & Suarez forcing
! 23-Aug-2006 - Torben Kunz   - new spacing schemes of sigma levels
! 23-Aug-2006 - Edilbert Kirk - individual selection of zonal waves
! 23-Aug-2006 - Edilbert Kirk - optimized Legendre trasnformation module
! 19-Feb-2007 - Edilbert Kirk - new flexible restart I/O
! 15-Sep-2009 - Edilbert Kirk - static arrays replaced by allocatable
! 15-Sep-2009 - Frank Lunkeit - diagnostics for entropy production
! 27-Sep-2010 - Edilbert Kirk - cleaned up ruido routines

call mpstart
call setfilenames
call opendiag
call read_resolution
call resolution
if (mrnum == 2) then
   call mrdimensions
endif
call allocate_arrays
call prolog
call master
call epilog
!XW(Mar/25/2017) to remove GUI: call guistop
call mpstop

print *, "STOP Normally!!!"
stop
end program puma_main
```

```
! ***************************
! * SUBROUTINE SETFILENAMES *
! ***************************

subroutine setfilenames
use pumamod

character (3) :: mrext

if (mrpid <  0) return ! no multirun

write(mrext,'("_",i2.2)') mrpid

puma_namelist       = trim(puma_namelist      ) // mrext
puma_output         = trim(puma_output        ) // mrext
puma_diag           = trim(puma_diag          ) // mrext
puma_restart        = trim(puma_restart       ) // mrext
puma_status         = trim(puma_status        ) // mrext
efficiency_dat      = trim(efficiency_dat     ) // mrext
ppp_puma_txt        = trim(ppp_puma_txt       ) // mrext
puma_sp_init        = trim(puma_sp_init       ) // mrext

return
end


! ***********************
! * SUBROUTINE OPENDIAG *
! ***********************

subroutine opendiag
use pumamod

if (mypid == NROOT) then
   open(nud,file=puma_diag)
endif

return
end


! *****************************
! * SUBROUTINE ALLOCATE_ARRAYS *
! *****************************

subroutine allocate_arrays
use pumamod

allocate(sd(nesp,nlev))    ; sd(:,:)  = 0.0 ! Spectral Divergence
allocate(st(nesp,nlev))    ; st(:,:)  = 0.0 ! Spectral Temperature
allocate(sz(nesp,nlev))    ; sz(:,:)  = 0.0 ! Spectral Vorticity
allocate(sp(nesp))         ; sp(:)    = 0.0 ! Spectral Pressure (ln Ps)
allocate(so(nesp))         ; so(:)    = 0.0 ! Spectral Orography
allocate(sr1(nesp,nlev))   ; sr1(:,:) = 0.0 ! Spectral Restoration Temperature
allocate(sr2(nesp,nlev))   ; sr2(:,:) = 0.0 ! Spectral Restoration Temperature
allocate(sdp(nspp,nlev))   ; sdp(:,:) = 0.0 ! Spectral Divergence  Partial
allocate(stp(nspp,nlev))   ; stp(:,:) = 0.0 ! Spectral Temperature Partial
allocate(szp(nspp,nlev))   ; szp(:,:) = 0.0 ! Spectral Vorticity   Partial
allocate(spp(nspp))        ; spp(:)   = 0.0 ! Spectral Pressure    Partial
allocate(sop(nspp))        ; sop(:)   = 0.0 ! Spectral Orography   Partial
allocate(srp1(nspp,nlev)) ; srp1(:,:)= 0.0 ! Spectral Restoration Partial
allocate(srp2(nspp,nlev)) ; srp2(:,:)= 0.0 ! Spectral Restoration Partial
allocate(sdt(nspp,nlev))   ; sdt(:,:) = 0.0 ! Spectral Divergence  Tendency
allocate(stt(nspp,nlev))   ; stt(:,:) = 0.0 ! Spectral Temperature Tendency
allocate(szt(nspp,nlev))   ; szt(:,:) = 0.0 ! Spectral Vorticity   Tendency
allocate(spt(nspp))        ; spt(:)   = 0.0 ! Spectral Pressure    Tendency
allocate(sdm(nspp,nlev))   ; sdm(:,:) = 0.0 ! Spectral Divergence  Minus
```



```
allocate(stm(nspp,nlev))  ; stm(:,:) = 0.0 ! Spectral Temperature Minus
allocate(szm(nspp,nlev))  ; szm(:,:) = 0.0 ! Spectral Vorticity   Minus
allocate(spm(nspp))       ; spm(:)   = 0.0 ! Spectral Pressure    Minus
allocate(sak(nesp))       ; sak(:)   = 0.0 ! Hyper diffusion
allocate(srcn(nesp))      ; srcn(:)  = 0.0 ! 1.0 / (n * (n+1))
allocate(span(nesp))      ; span(:)  = 0.0 ! Pressure for diagnostics
allocate(spnorm(nesp))    ; spnorm(:)= 0.0 ! Factors for output normalization

allocate(nindex(nesp))    ; nindex(:)  = ntru ! Holds wavenumber
allocate(nscatsp(npro))   ; nscatsp(:) = nspp ! Used for reduce_scatter op
allocate(nselzw(0:ntru))  ; nselzw(:)  =    1 ! Enable selected zonal waves
allocate(nselsp(ncsp))    ; nselsp(:)  =    1 ! Enable slected spectral modes

allocate(gd(nhor,nlev))   ; gd(:,:)  = 0.0 ! Divergence
allocate(gt(nhor,nlev))   ; gt(:,:)  = 0.0 ! Temperature
allocate(gz(nhor,nlev))   ; gz(:,:)  = 0.0 ! Vorticity
allocate(gu(nhor,nlev))   ; gu(:,:)  = 0.0 ! u * cos(phi)
allocate(gv(nhor,nlev))   ; gv(:,:)  = 0.0 ! v * sin(phi)
allocate(gp(nhor))        ; gp(:)    = 0.0 ! Ln(Ps)
allocate(gfu(nhor,nlev))  ; gfu(:,:) = 0.0 ! Term Fu in Primitive Equations
allocate(gfv(nhor,nlev))  ; gfv(:,:) = 0.0 ! Term Fv in Primitive Equations
allocate(gut(nhor,nlev))  ; gut(:,:) = 0.0 ! Term u * T
allocate(gvt(nhor,nlev))  ; gvt(:,:) = 0.0 ! Term v * T
allocate(gke(nhor,nlev))  ; gke(:,:) = 0.0 ! Kinetic energy u * u + v * v
allocate(gpj(nhor))       ; gpj(:)   = 0.0 ! d(Ln(Ps)) / d(mu)


allocate(rcsq(nhor))      ; rcsq(:)  = 0.0 ! 1 / cos2(phi)

allocate(ndil(nlev))        ; ndil(:)  = 0
allocate(csu(nlat,nlev))    ; csu(:,:) = 0.0
allocate(csv(nlat,nlev))    ; csv(:,:) = 0.0
allocate(cst(nlat,nlev))    ; cst(:,:) = 0.0

allocate(chlat(nlat))       ; chlat(:) = ' '
allocate(sid(nlat))         ; sid(:)   = 0.0  ! sin(phi)
allocate(gwd(nlat))         ; gwd(:)   = 0.0  ! Gaussian weight (phi)
allocate(csq(nlat))         ; csq(:)   = 0.0  ! cos2(phi)
allocate(rcs(nlat))         ; rcs(:)   = 0.0  ! 1/cos(phi)

allocate(t0(nlev))     ; t0(:)     = 250.0  ! reference temperature
allocate(t0d(nlev))    ; t0d(:)    =   0.0  ! vertical t0 gradient
allocate(damp(nlev))   ; damp(:)   =   0.0  ! 1.0 / (2 Pi * taur)
allocate(fric(nlev))   ; fric(:)   =   0.0  ! 1.0 / (2 Pi * tauf)
allocate(dsigma(nlev)) ; dsigma(:) =   0.0
allocate(rdsig(nlev))  ; rdsig(:)  =   0.0
allocate(sigma(nlev))  ; sigma(:)  =   0.0
allocate(sigmh(nlev))  ; sigmh(:)  =   0.0
allocate(tkp(nlev))    ; tkp(:)    =   0.0
allocate(c(nlev,nlev)) ; c(:,:)    =   0.0
allocate(xlphi(nlev,nlev)) ; xlphi(:,:) = 0.0 ! matrix Lphi (g)
allocate(xlt(nlev,nlev))   ; xlt(:,:)   = 0.0 ! matrix LT (tau)
allocate(bm1(nlev,nlev,0:NTRU)) ; bm1(:,:,:)  = 0.0

if (mrnum == 2) then
   allocate(sdd(nesp,nlev))   ; sdd(:,:)  = 0.0
   allocate(std(nesp,nlev))   ; std(:,:)  = 0.0
   allocate(szd(nesp,nlev))   ; szd(:,:)  = 0.0
   allocate(spd(nesp     ))   ; spd(:  )  = 0.0
endif

return
end subroutine allocate_arrays


! =================
! SUBROUTINE PROLOG
! =================
```

```
subroutine prolog
use pumamod

character( 8) :: cpuma       = 'PUMA-II'
character(80) :: pumaversion = '16.0 (27-Sep-2010)'
real :: zsig(nlon*nlat)

if (mypid == NROOT) then
   call cpu_time(tmstart)
   write(nud,'(/," ************************************************")')
   write(nud,'(" * PUMA ",a43," *")') trim(pumaversion)
   write(nud,'(" ************************************************")')
   if (mrnum == 0) then
   write(nud,'(" * NTRU =",i4," NLEV =",i4," NLON = ",i4,"  NLAT =",i4," *")') &
      NTRU,NLEV,NLON,NLAT
   else
      do jpid = 1 , mrnum
   write(nud,'(" * PID =",i4," NTRU =",i4," NLEV = ",i4,"       *")') &
         jpid-1,mrtru(jpid),NLEV
      enddo
   endif
   write(nud,'(" ************************************************")')
   if (NPRO > 1) then
     write(nud,'(/," ************************************************")')
     do jpro = 1 , NPRO
        write(nud,'(" * CPU",i4,1x,a40," *")') jpro-1,ympname(jpro)
     enddo
     write(nud,'(" ************************************************")')
   endif
   call restart_ini(lrestart,puma_restart)
   call inigau(NLAT,sid,gwd)
   call inilat
   call legpri
   call readnl
   call ppp_interface
   call initpm
   call initsi
   call altlat(csq,NLAT) ! csq -> alternating grid
   !XW(Mar/25/2017) to remove GUI: if (ngui > 0) call guistart
   if (nrun == 0 .and. nstop  > 0) nrun = nstop-nstep
   if (nrun == 0) nrun = ntspd * (nyears * 360 + nmonths * 30)
   call initrandom      ! set random seed
endif ! (mypid == NROOT)

call mpbci(nruido)
call initruido       ! allocate ruido arrays


!XW(Mar/25/2017) to remove GUI: if (nshutdown > 0) return ! If something went wr
ong in the init routines

! **********************
! * broadcast & scatter *
! **********************

call mpscdn(sid,NHPP) ! real (kind=8)
call mpscdn(gwd,NHPP) ! real (kind=8)
call mpscrn(csq,NLPP)

do jlat = 1 , NLPP
   rcsq(1+(jlat-1)*NLON:jlat*NLON) = 1.0 / csq(jlat)
enddo

!     broadcast integer

call mpbci(kick    ) ! add noise for kick > 0
call mpbci(nafter  ) ! write data interval [steps]
call mpbci(nwpd    ) ! write data interval [writes per day]
call mpbci(ncoeff  ) ! number of modes to print
```



```
call mpbci(ndel    ) ! ndel
call mpbci(noutput ) ! global output switch
call mpbci(ndiag   ) ! write diagnostics interval
call mpbci(ngui    ) ! GUI on (1) or off (0)
call mpbci(nkits   ) ! number of initial timesteps
call mpbci(nlevt   ) ! tropospheric levels
call mpbci(nrun    ) ! if (nstop == 0) nstop = nstep + nrun
call mpbci(nstep   ) ! current timestep
call mpbci(nstop   ) ! finishing timestep
call mpbci(ntspd   ) ! number of timesteps per day
call mpbci(mpstep  ) ! minutes per step
call mpbci(nyears  ) ! simulation time
call mpbci(nmonths ) ! simulation time
call mpbci(nextout ) ! write extended output
call mpbci(nsponge)  ! Switch for sponge layer
call mpbci(nhelsua)  ! Held & Suarez forcing
call mpbci(ndiagp)   ! 0/1 switch for new grid point diabatic heating
call mpbci(nconv)    ! 0/1 switch for convective heating
call mpbci(nvg     ) ! Type of vertical grid
call mpbci(nenergy)  ! energy diagnostics
call mpbci(nentropy) ! entropy diagnostics
call mpbci(ndheat)   ! energy recycling
call mpbci(nradcv)   ! use two restoration fields

!     broadcast logical

call mpbcl(lrestart) ! true: read restart file, false: initial run
call mpbcl(lselect ) ! true: disable some zonal waves
call mpbcl(lspecsel) ! true: disable some spectral modes

!     broadcast real

call mpbcr(ww_time ) ! time scale [sec] = 1 day for Earth
call mpbcr(ww_scale) ! reciprocal of time scale [1/sec]
call mpbcr(v_scl   )
call mpbcr(ct      )
call mpbcr(cv      )
call mpbcr(sid_day )
call mpbcr(sol_day )
call mpbcr(plarad  )
call mpbcr(gascon  )
call mpbcr(akap    )
call mpbcr(alr     )
call mpbcr(ga      )
call mpbcr(psurf   )
call mpbcr(alpha   ) ! Williams factor for time filter
call mpbcr(dtep    ) ! equator-pole temperature difference
call mpbcr(dtns    )
call mpbcr(dtrop   )
call mpbcr(dttrp   )
call mpbcr(diffts  ) ! diffusion time scale [sec]
call mpbcr(tac     )
call mpbcr(pac     )
call mpbcr(plavor  )
call mpbcr(rotspd  )
call mpbcr(sigmax  ) ! sigma of top half level
call mpbcr(tgr     )
call mpbcr(dvdiff  )
call mpbcr(disp    )
call mpbcr(tauta   )
call mpbcr(tauts   )
call mpbcr(pspon   )
call mpbcr(dcsponge)
call mpbcr(spstep  ) ! seconds per step

!     broadcast integer arrays

call mpbcin(ndil  ,NLEV)
call mpbcin(nselzw,NTP1)
```

```
!     broadcast real arrays

call mpbcrn(damp  ,NLEV)
call mpbcrn(dsigma,NLEV)
call mpbcrn(fric  ,NLEV)
call mpbcrn(rdsig ,NLEV)
call mpbcrn(taur  ,NLEV)
call mpbcrn(sigma ,NLEV)
call mpbcrn(sigmh ,NLEV)
call mpbcrn(t0    ,NLEV)
call mpbcrn(t0d   ,NLEV)
call mpbcrn(tauf  ,NLEV)
call mpbcrn(tkp   ,NLEV)

call mpbcrn(c     ,NLSQ)
call mpbcrn(xlphi ,NLSQ)
call mpbcrn(xlt   ,NLSQ)

!     scatter integer arrays

call mpscin(nindex,NSPP)
call mpscrn(srcn  ,NSPP)
call mpscrn(sak   ,NSPP)

call legini(nlat,nlpp,nesp,nlev,plavor,sid,gwd)

if (lrestart) then
   call read_atmos_restart
   if (mypid == NROOT) then
      if (kick > 10) call noise(kick-10)
      if (newsr > 0) call setzt
   endif
else
   call initfd
endif

if (mypid == NROOT) then
   call printseed ! either namelist, clock initialized or from restart file
endif

!     broadcast spectral arrays

call mpbcrn(sp,NESP)
call mpbcrn(sd,NESP*NLEV)
call mpbcrn(st,NESP*NLEV)
call mpbcrn(sz,NESP*NLEV)

!     scatter spectral arrays

call mpscsp(sd,sdp,NLEV)
call mpscsp(st,stp,NLEV)
call mpscsp(sz,szp,NLEV)
call mpscsp(sr1,srp1,NLEV)
call mpscsp(sr2,srp2,NLEV)
call mpscsp(sp,spp,1)
call mpscsp(so,sop,1)

!     scatter gridpoint arrays

if (nruido > 0) call mpscgp(ruido,ruidop,NLEV)

!
!     initialize energy and entropy diagnostics
!
if(nenergy > 0) then
 allocate(denergy(NHOR,9))
 denergy(:,:)=0.
endif
```



```
if(nentropy > 0) then
 allocate(dentropy(NHOR,9))
 dentropy(:,:)=0.
endif
if(ndheat > 1 .and. mypid == NROOT) then
 open(9,file=efficiency_dat,form='formatted')
endif
!
!     write first service record containing sigma coordinates
!
if (mypid == NROOT) then
   if (noutput > 0) then
      istep = nstep
      if (istep > 0) istep = istep + nafter ! next write after restart
      open(40,file=puma_output,form='unformatted')
      call ntomin(istep,imin,ihour,iday,imonth,iyear)
      zsig(1:nlev) = sigmh(:)
      zsig(nlev+1:) = 0.0
      write(40) 333,0,iyear*10000+imonth*100+iday,0,nlon,nlat,nlev,ntru
      write(40) zsig
   endif ! (noutput > 0)
endif ! (mypid == NROOT)
return
end subroutine prolog


!====================!
! SUBROUTINE MASTER !
!==================== !

subroutine master
use pumamod

!XW(Mar/25/2017) to remove GUI: if (nshutdown > 0) return ! if something went wr
ong in prolog already

! ***************************
! * short initial timesteps *
! ***************************

ikits = nkits
do jkits = 1 , ikits
   delt  = spstep * ww_scale / (2**nkits)
   delt2 = delt + delt
   call gridpoint
   call makebm
   call spectral
   nkits = nkits - 1
enddo

delt  = spstep * ww_scale
delt2 = delt + delt
call makebm

nstep1 = nstep ! remember 1.st timestep

do jstep = 1 , nrun
   nstep = nstep + 1
   !XW(Mar/25/2017) to remove GUI: call ntomin(nstep,ndatim(5),ndatim(4),ndatim(
3),ndatim(2),ndatim(1))

!  ************************************************************
!  * calculation of non-linear quantities in grid point space *
!  ************************************************************

   call gridpoint

   if (mypid == NROOT) then
      if (mod(nstep,nafter) == 0 .and. noutput > 0) call outsp
```

```
      if (mod(nstep,ndiag ) == 0 .or. ngui > 0) call diag
      if (ncu > 0) call checkunit
   endif
   !XW(Mar/25/2017) to remove GUI: if (ngui > 0) call guistep_puma

!  ****************************
!  * adiabatic part of timestep *
!  ****************************

   call spectral
   if (mod(nstep,nafter) == 0 .and. noutput > 0) call outgp
   !XW(Mar/25/2017) to remove GUI: if (nshutdown > 0) return
enddo
return
end subroutine master


!     =================
!     SUBROUTINE EPILOG
!     =================

      subroutine epilog
      use pumamod
      real    (kind=8) :: zut,zst
      integer (kind=8) :: imem,ipr,ipf,isw,idr,idw

      if (mypid == NROOT) close(40) ! close output file

!     write restart file

      if (mypid == NROOT) then
         call restart_prepare(puma_status)
         sp(1) = psmean ! save psmean
         call put_restart_integer('nstep'    ,nstep    )
         call put_restart_integer('nlat'     ,NLAT     )
         call put_restart_integer('nlon'     ,NLON     )
         call put_restart_integer('nlev'     ,NLEV     )
         call put_restart_integer('nrsp'     ,NRSP     )

!        Save current random number generator seed

         call random_seed(get=meed)
         call put_restart_array('seed',meed,nseedlen,nseedlen,1)
         call put_restart_array('ganext',ganext,1,1,1)

         call put_restart_array('sz' ,sz ,NRSP,NESP,NLEV)
         call put_restart_array('sd' ,sd ,NRSP,NESP,NLEV)
         call put_restart_array('st' ,st ,NRSP,NESP,NLEV)
         call put_restart_array('sr1',sr1,NRSP,NESP,NLEV)
         call put_restart_array('sr2',sr2,NRSP,NESP,NLEV)
         call put_restart_array('sp' ,sp ,NRSP,NESP,   1)
         call put_restart_array('so' ,so ,NRSP,NESP,   1)
         if (nruido > 0) then
            call put_restart_array('ruido',ruido,nugp,nugp,nlev)
         endif
      endif

      call mpputsp('szm',szm,NSPP,NLEV)
      call mpputsp('sdm',sdm,NSPP,NLEV)
      call mpputsp('stm',stm,NSPP,NLEV)
      call mpputsp('spm',spm,NSPP,   1)

!     write gridpoint arrays

      if (allocated(gr1)) then
         call mpputgp('gr1',gr1,nhor,nlev)
      endif
      if (allocated(gr2)) then
         call mpputgp('gr2',gr2,nhor,nlev)
      endif
      if (allocated(gtdamp)) then
         call mpputgp('gtdamp',gtdamp,nhor,nlev)
      endif

      if (allocated(gr1c)) then
         call mpputgp('gr1c',gr1c,nhor,nlev)
      endif
      if (allocated(gr2c)) then
         call mpputgp('gr2c',gr2c,nhor,nlev)
      endif
      if (allocated(gtdampc)) then
         call mpputgp('gtdampc',gtdampc,nhor,nlev)
      endif

      if (mypid == NROOT) then
!        Get resource stats from function resources in file pumax.c

         ires = 1
         !XW/Mar-23-2017: this line need to call a function in pumax.c
         !ires = nresources(zut,zst,imem,ipr,ipf,isw,idr,idw)

         call cpu_time(tmstop)
         tmrun = tmstop - tmstart
         if (nstep > nstep1) then
            zspy = tmrun * 360.0 * real(ntspd) / (nstep - nstep1) ! sec / siy
            zypd = (24.0 * 3600.0 / zspy)                         ! siy / day
            write(nud,'(/,"****************************************")')
            if (zut > 0.0) &
            write(nud,  '("* User  time      : ", f10.3," sec * N/A")') zut
            if (zst > 0.0) &
            write(nud,  '("* System time     : ", f10.3," sec * N/A")') zst
            if (zut + zst > 0.0) tmrun = zut + zst
            write(nud,  '("* Total CPU time  : ", f10.3," sec *")') tmrun
            if (imem > 0) &
            write(nud,  '("* Memory usage    : ", f10.3," MB  * N/A")') imem * 0.000001
            if (ipr > 0 .and. ipr < 1000000) &
            write(nud,  '("* Page reclaims   : ", i6," pages  * N/A")') ipr
            if (ipf > 0 .and. ipf < 1000000) &
            write(nud,  '("* Page faults     : ", i6," pages  * N/A")') ipf
            if (isw > 0 .and. isw < 1000000) &
            write(nud,  '("* Page swaps      : ", i6," pages  * N/A")') isw
            if (idr > 0 .and. idr < 1000000) &
            write(nud,  '("* Disk read       : ", i6," blocks * N/A")') idr
            if (idw > 0 .and. idw < 1000000) &
            write(nud,  '("* Disk write      : ", i6," blocks * N/A")') idw
            write(nud,'("****************************************")')
            if (zspy < 600.0) then
               write(nud,'("* Seconds per sim year: ",i6,9x,"*")') nint(zspy)
            else if (zspy < 900000.0) then
               write(nud,'("* Minutes per sim year  ",i6,9x,"*")') nint(zspy/60.0)
            else
               write(nud,'("* Days per sim year:    ",i6,5x,"*")') nint(zspy/sol_day)
            endif
            write(nud,'("* Sim years per day  :",i7,9x,"* <-- Running Speed Index")') nint(zypd)
            write(nud,'("****************************************")')
         endif
      endif

      return
      end subroutine epilog

!     ============================
!     SUBROUTINE READ_ATMOS_RESTART
!     ============================

      subroutine read_atmos_restart
      use pumamod
```

```
      integer :: k = 0

!     read scalars and full spectral arrays

      if (mypid == NROOT) then
         call get_restart_integer('nstep',nstep)
         call get_restart_array('seed',meed,nseedlen,nseedlen,1)
         call get_restart_array('ganext',ganext,1,1,1)
         call get_restart_array('sz' ,sz ,NRSP,NESP,NLEV)
         call get_restart_array('sd' ,sd ,NRSP,NESP,NLEV)
         call get_restart_array('st' ,st ,NRSP,NESP,NLEV)
         call get_restart_array('sr1',sr1,NRSP,NESP,NLEV)
         call get_restart_array('sr2',sr2,NRSP,NESP,NLEV)
         call get_restart_array('sp' ,sp ,NRSP,NESP,   1)
         call get_restart_array('so' ,so ,NRSP,NESP,   1)
         if (nruido > 0) then
            call get_restart_array('ruido',ruido,nugp,nugp,nlev)
         endif
         psmean = sp(1)
         sp(1)  = 0.0
         call random_seed(put=meed)
      endif

      call mpbci(nstep)      ! broadcast current timestep
      call mpbcr(psmean)     ! broadcast mean surface pressure

!     read and scatter spectral arrays

      call mpgetsp('szm',szm,NSPP,NLEV)
      call mpgetsp('sdm',sdm,NSPP,NLEV)
      call mpgetsp('stm',stm,NSPP,NLEV)
      call mpgetsp('spm',spm,NSPP,   1)

!     allocate, read and scatter gridpoint arrays

      if (mypid == NROOT) call varseek('gr1',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gr1(nhor,nlev))
         call mpgetgp('gr1',gr1,nhor,nlev)
      endif
      if (mypid == NROOT) call varseek('gr2',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gr2(nhor,nlev))
         call mpgetgp('gr2',gr2,nhor,nlev)
      endif
      if (mypid == NROOT) call varseek('gtdamp',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gtdamp(nhor,nlev))
         call mpgetgp('gtdamp',gtdamp,nhor,nlev)
      endif
      if (mypid == NROOT) call varseek('gr1c',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gr1c(nhor,nlev))
         call mpgetgp('gr1c',gr1c,nhor,nlev)
      endif
      if (mypid == NROOT) call varseek('gr2c',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gr2c(nhor,nlev))
         call mpgetgp('gr2c',gr2c,nhor,nlev)
      endif
      if (mypid == NROOT) call varseek('gtdampc',ktmp)
      call mpbci(ktmp)
      if (ktmp > 0) then
         allocate(gtdampc(nhor,nlev))
```



```
         call mpgetgp('gtdampc',gtdampc,nhor,nlev)
      endif

      return
      end subroutine read_atmos_restart

!     =================
!     SUBROUTINE INITFD
!     =================

      subroutine initfd
      use pumamod

      if (nkits < 1) nkits = 1

!     Look for start data and read them if there

      call read_surf(129,so,     1,iread1)
      call read_surf(134,sp,     1,iread2)
      call read_surf(121,sr1,NLEV,iread3)
      call read_surf(122,sr2,NLEV,iread4)
      call read_vargp(123,NLEV,iread123)
      if (mypid == NROOT .and. iread123 == 0) then
         if (nhelsua > 1) then
            write(nud,*) "*** ERROR no *_surf_0123.sra file for Held&Suarez"
            stop
         endif
      endif

      if (ndiagp > 0) then
         call read_vargp(121,NLEV,iread121)
         call read_vargp(122,NLEV,iread122)
         if (.not. allocated(gtdamp)) then
            call read_vargp(123,NLEV,iread123)
         endif
         if (mypid == NROOT) then
            if (iread121==0 .or. iread122==0 .or. iread123==0) then
               write(nud,*) "*** ERROR not all fields (121,122,123) for grid point heating found"
               stop
            endif
         endif
      endif

      if (nconv > 0) then
         call read_vargp(124,NLEV,iread124)
         call read_vargp(125,NLEV,iread125)
         call read_vargp(126,NLEV,iread126)
         if (mypid == NROOT) then
            if (iread124==0 .or. iread125==0 .or. iread126==0) then
               write(nud,*) "*** ERROR not all fields (124,125,126) for convective heating found"
               stop
            endif
         endif
      endif

      if (mypid == NROOT) then
         if (iread1==0 .or. iread2==0 .or. iread3==0 .or. iread4==0) then
            call setzt ! setup for aqua-planet
         else
            psmean = psurf * exp(spnorm(1) * sp(1))
            sp(1)  = 0.0
            so(:) = so(:) / (cv * cv) ! descale from [m2/s2]
            sr1(:,:) = sr1(:,:) / ct  ! descale from [K]
            sr2(:,:) = sr2(:,:) / ct  ! descale from [K]
            sr1(1,:) = sr1(1,:) - t0(:) * sqrt(2.0) ! subtract profile
            write(nud,'(a,f8.2,a)') ' Mean of Ps = ',0.01*psmean, '[hPa]'
         endif
      endif
```

```
!     Add initial noise if wanted

      if (mypid == NROOT) then
         call printprofile
         if (kick > 10) then
            call noise(kick-10)
         else
            call noise(kick)
         endif
      endif ! (mypid == NROOT)

      call mpscsp(sp,spm,1)
      if (mypid == NROOT) then
          st(1,:) = sr1(1,:)
         stm(1,:) = sr1(1,:)
          sz(3,:) = plavor
         szm(3,:) = plavor
      endif
      return
      end


!     ==========================
!     SUBROUTINE READ_RESOLUTION
!     ==========================

      subroutine read_resolution
      use pumamod

      character (80) :: ylat
      character (80) :: ylev

      if (mypid == NROOT) then
         call get_command_argument(1,ylat)
         call get_command_argument(2,ylev)
         read(ylat,*) nlat
         read(ylev,*) nlev
      endif

      call mpbci(nlat)
      call mpbci(nlev)
      return
      end


!     ====================
!     SUBROUTINE RESOLUTION
!     ====================

      subroutine resolution
      use pumamod

      nlem = nlev - 1
      nlep = nlev + 1
      nlsq = nlev * nlev

      nlon = nlat + nlat ! Longitudes
      nlah = nlat / 2
      nlpp = nlat / npro
      nhpp = nlah / npro
      nhor = nlon * nlpp
      nugp = nlon * nlat
      npgp = nugp / 2

      ntru = (nlon - 1) / 3
      ntp1 = ntru + 1
      nzom = ntp1 + ntp1
      nrsp = (ntru + 1) * (ntru + 2)
      ncsp = nrsp / 2
```



```
      nspp = (nrsp + npro - 1) / npro
      nesp = nspp * npro
      nesp = nesp + 3 - mod(nesp-1,4)

      return
      end


!     =================
!     SUBROUTINE READNL
!     =================

      subroutine readnl
      use pumamod

!     This workaround is necessaray, because allocatable arrays are
!     not allowed in namelists for FORTRAN versions < F2003

      integer, parameter :: MAXSELZW =  42
      integer, parameter :: MAXSELSP = ((MAXSELZW+1) * (MAXSELZW+2)) / 2
      integer :: nselect(0:MAXSELZW) = 1      ! NSELECT can be used up tp T42
      integer :: nspecsel(MAXSELSP)  = 1      ! Default setting: all modes activ
e
      integer :: ndl(MAXLEV)         = 0      ! Diagnostics off
      real    :: sigmah(MAXLEV)      = 0.0    ! Half level sigma
      real    :: t0k(MAXLEV)         = 250.0  ! Reference temperature

      namelist /puma_nl/ &
        akap    , alpha   , alr     , alrs    , diffts  , disp    &
      , dtep    , dtns    , dtrop   , dttrp   , dtzz    , dvdiff  &
      , ga      , gascon  &
      , kick    , mpstep  , nafter  , ncoeff  , nconv   , ncu     &
      , ndel    , ndheat  , ndiag   , ndiagp  , ndl     , nenergy &
      , nentropy, newsr   , nextout , ngui    , nguidbg , nhelsua &
      , nkits   &
      , nlevt   , nmonths , noutput , nradcv  , nruido  , nrun    &
      , nselect , nspecsel, nsponge , nstep   , nstop   , nsync   &
      , ntspd   , nvg     , nwpd    , nwspini , nyears  &
      , orofac  , pac     , plarad  , pspon   , psurf   &
      , rotspd  , seed    , sid_day , sigmah  , sigmax  , dcsponge&
      , spstep  , syncsecs, syncstr , t0k     , tauf    , taur    &
      , tac     , tauta   , tauts   , tgr     &
      , ww_time

      open(13,file=puma_namelist,iostat=ios)
      if (ios == 0) then
         read (13,puma_nl)
         close(13)
      endif

!--- modify basic scales according to namelist

      if (ww_time < 1.0) ww_time = sid_day / TWOPI ! time scale
      ww_scale = 1.0 / ww_time      ! reciprocal of time scale 1/Omega
      cv    = plarad*ww_scale       ! velocity scale (velocity at the equator)
      ct    = cv*cv/gascon    ! temperature scale from hydrostatic equation
      if (mpstep == 0 .and. spstep == 0.0) then ! automatic timestep
         mpstep = (60 * 32) / nlat              ! 60 min for T21
         spstep = mpstep * 60.0
      endif
      if (spstep == 0.0) spstep = 60.0 * mpstep
      if (ntspd == 0) ntspd = sol_day / spstep

      nafter = ntspd                            ! daily output
      if (nwpd > 0 .and. nwpd <= ntspd) then
         nafter = ntspd / nwpd
      endif
      if (ndiag  < 1) ndiag  = ntspd * 10       ! every 10th. day
```

```
      if (syncsecs > 0.0) syncstr = spstep / syncsecs
      if (syncstr  > 1.0) syncstr = 1.0

      write(nud,puma_nl)

      itru = ntru
      if (itru > MAXSELZW) itru = MAXSELZW
      icsp = ncsp
      if (icsp > MAXSELSP) icsp = MAXSELSP
      ilev = nlev
      if (ilev > MAXLEV)   ilev = MAXLEV

      nselzw(0:itru) = nselect(0:itru)  ! Copy values to allocated array
      nselsp(1:icsp) = nspecsel(1:icsp)
      ndil(1:ilev)   = ndl(1:ilev)
      sigmh(1:ilev)  = sigmah(1:ilev)
      t0(1:ilev)     = t0k(1:ilev)

      return
      end


      subroutine ppp_read_i(a,ndim,nread)
      integer :: a(ndim)
      integer :: n

      nread = 0
      read (15,*) n
      if (n < 1 .or. n > ndim) return
      read (15,*) a(1:n)
      nread = n
      return
      end


      subroutine ppp_read_r(a,ndim,nread)
      real    :: a(ndim)
      integer :: n

      nread = 0
      read (15,*) n
      if (n < 1 .or. n > ndim) return
      read (15,*) a(1:n)
      nread = n
      return
      end


!     ========================
!     SUBROUTINE PPP_INTERFACE
!     ========================

      subroutine ppp_interface
      use pumamod
      use prepmod
      logical :: lexist
      integer :: iostat
      integer :: n
      integer :: ivar
      character (80) :: yname
      character (80) :: yfo1 = '("* ",A," = ",G10.4," *")'

      inquire(file=ppp_puma_txt,exist=lexist)
      if (.not. lexist) return

      call ppp_def_int('NLAT',nlat_ppp,1)
      call ppp_def_int('NLEV',nlev_ppp,1)

      call ppp_def_real('SIGMH',sigmh,nlev)
```



```
      write(nud,*) "******************************"
      write(nud,*) "* Reading file <",trim(ppp_puma_txt),"> *"
      write(nud,*) "******************************"
      open (15,file=ppp_puma_txt)
      read (15,'(A)',iostat=iostat) yname
      do while (trim(yname) /= '[END]' .and. iostat == 0)
         do j = 1 , num_ppp
            if (trim(yname) == ppp_tab(j)%name) then
               if (ppp_tab(j)%isint) then
                  call ppp_read_i(ppp_tab(j)%pint,ppp_tab(j)%n,iread)
                  if (iread == 0) then
                     write(nud,*) "*** ERROR reading ",trim(yname)," from ",trim(ppp_
puma_txt)
                     stop
                  else if (iread == 1) then
                     write(nud,'("* ",A," = ",I10," *")') yname(1:15),ppp_tab(j)%pint
                  else
                     write(nud,'("* ",A," :",I5," items *")') yname(1:15),iread
                  endif
               else
                  call ppp_read_r(ppp_tab(j)%preal,ppp_tab(j)%n,iread)
                  if (iread == 0) then
                     write(nud,*) "*** ERROR reading ",trim(yname)," from ",trim(ppp_
puma_txt)
                     stop
                  else if (iread == 1) then
                     write(nud,yfo1) yname(1:15),ppp_tab(j)%preal
                  else
                     write(nud,'("* ",A," :",I5," items *")') yname(1:15),iread
                  endif
               endif
               exit
            endif
         enddo
         read (15,'(A)',iostat=iostat) yname
      enddo
      if (nlat_ppp(1) /= 0 .and. nlat_ppp(1) /= nlat) then
         write(nud,*) "*** ERROR *** ERROR *** ERROR *** ERROR ***"
         write(nud,*) "# of latitudes mismatch in preprocessor PPP and PUMA"
         write(nud,*) "NLAT in PPP : ",nlat_ppp," <",trim(ppp_puma_txt),">"
         write(nud,*) "NLAT in PUMA : ",nlat
         write(nud,*) "Aborting ..."
         stop
      endif
      if (nlev_ppp(1) /= 0 .and. nlev_ppp(1) /= nlev) then
         write(nud,*) "*** ERROR *** ERROR *** ERROR *** ERROR ***"
         write(nud,*) "# of levels mismatch in preprocessor PPP and PUMA"
         write(nud,*) "NLEV in PPP : ",nlev_ppp," <",trim(ppp_puma_txt),">"
         write(nud,*) "NLEV in PUMA : ",nlev
         write(nud,*) "Aborting ..."
         stop
      endif
      write(nud,*) "******************************"

      return
      end subroutine ppp_interface


!     ============================
!     SUBROUTINE SELECT_ZONAL_WAVES
!     ============================

      subroutine select_zonal_waves
      use pumamod

      if (sum(nselzw(:)) /= NTP1) then ! some wavenumbers disabled
         lselect = .true.
      endif
```

```
      return
      end

!     ================================
!     SUBROUTINE SELECT_SPECTRAL_MODES
!     ================================

      subroutine select_spectral_modes
      use pumamod

      if (sum(nselsp(:)) /= NCSP) then ! some modes disabled
         lspecsel = .true.
      endif
      return
      end

!     =====================
!     * SET VERTICAL GRID *
!     =====================

      subroutine set_vertical_grid

      use pumamod

      if (sigmh(NLEV) /= 0.0) return ! Already read in from namelist puma

      if (nvg == 1) then               ! Scinocca & Haynes sigma levels

         if (nlevt >= NLEV) then       ! Security check for 'nlevt'
            write(nud,*) '*** ERROR *** nlevt >= NLEV'
            write(nud,*) 'Number of levels (NLEV): ',NLEV
            write(nud,*) 'Number of tropospheric levels (nlevt): ',nlevt
         endif

!     troposphere: linear spacing in sigma
!     stratosphere: linear spacing in log(sigma)
!     after (see their Appendix):
!     Scinocca, J. F. and P. H. Haynes (1998): Dynamical forcing of
!        stratospheric planetary waves by tropospheric baroclinic eddies.
!        J. Atmos. Sci., 55 (14), 2361-2392

!     Here, zsigtran is set to sigma at dtrop (tropopause height for
!     construction of restoration temperature field). If tgr=288.15K,
!     ALR=0.0065K/km and dtrop=11.km, then zsigtran=0.223 (=0.1 in
!     Scinocca and Haynes (1998)).
!     A smoothing of the transition between linear and logarithmic
!     spacing, as noted in Scinocca and Haynes (1998), is not yet
!     implemented.

         zsigtran = (1. - alr * dtrop / tgr)**(ga/(gascon*alr))
         zsigmin = 1. - (1. - zsigtran) / real(nlevt)

         do jlev=1,NLEV
            if (jlev == 1) then
               sigmh(jlev) = SIGMAX
            elseif (jlev > 1 .and. jlev < NLEV - nlevt) then
               sigmh(jlev) = exp((log(SIGMAX) - log(zsigtran))          &
     &              / real(NLEV - nlevt - 1) * real(NLEV - nlevt - jlev) &
     &              + log(zsigtran))
            elseif (jlev >= NLEV - nlevt .and. jlev < NLEV - 1) then
               sigmh(jlev) = (zsigtran - zsigmin) / real(nlevt - 1)     &
     &                       * real(NLEV - 1 - jlev) + zsigmin
            elseif (jlev == NLEV - 1) then
               sigmh(jlev) = zsigmin
            elseif (jlev == NLEV) then
               sigmh(jlev) = 1.
            endif
         enddo
         return  ! case nvg == 1 finished
```



```
      else if (nvg == 2) then   ! Polvani & Kushner sigma levels
         inl = int(real(NLEV)/(1.0 - sigmax**(1.0/5.0)))
         do jlev=1,NLEV
            sigmh(jlev) = (real(jlev + inl - NLEV) / real(inl))**5
         enddo
         return

!     Default (nvg == 0) : equidistant sigma levels

      else
         do jlev = 1 , NLEV
            sigmh(jlev) = real(jlev) / real(NLEV)
         enddo
      endif

      return
      end


!     =================
!     SUBROUTINE INITPM
!     =================

      subroutine initpm
      use pumamod

      real (kind=8) :: radea,zakk,zzakk
      real :: zsigb           ! sigma_b for Held & Suarez frictional
!                               and heating timescales

      radea  = plarad             ! planet radius in high precision
      plavor = EZ * rotspd * omega * ww_time ! planetary vorticity

!     ************************************************************
!     * carries out all initialisation of model prior to running. *
!     * major sections identified with comments.                  *
!     * this s/r sets the model parameters and all resolution     *
!     * dependent quantities.                                     *
!     ************************************************************

      if (lrestart) nkits=0

!     ************************************************
!     * Check for enabling / disabling zonal wavenumbers *
!     ************************************************

      call select_zonal_waves
      if (npro == 1) call select_spectral_modes

!     *********************
!     * set vertical grid *
!     *********************

      call set_vertical_grid

      dsigma(1      ) = sigmh(1)
      dsigma(2:NLEV) = sigmh(2:NLEV) - sigmh(1:NLEM)

      rdsig(:) = 0.5 / dsigma(:)

      sigma(1      ) = 0.5 * sigmh(1)
      sigma(2:NLEV) = 0.5 * (sigmh(1:NLEM) + sigmh(2:NLEV))

!     Initialize profile of tau R if not set in namelist

      if (taur(NLEV) == 0.0) then
         do jlev = 1 , NLEV
            taur(jlev) = sid_day * 50.0 * atan(1.0 - sigma(jlev))
            if (taur(jlev) > 30.0 * sid_day) taur(jlev) = 30.0 * sid_day
```

```
         enddo
      endif

!     Initialize profile of tau F if not set in namelist

      if (tauf(NLEV) == 0.0) then
         do jlev = 1 , NLEV
            if (sigma(jlev) > 0.8) then
               tauf(jlev) = exp(10.0 * (1.0 - sigma(jlev))) / 2.718 * sid_day
            endif
         enddo
      endif

!     Compute 1.0 / (2 Pi * tau) for efficient use in calculations
!     A day is 2 Pi in non dimensional units using omega as scaling

      where (taur(1:NLEV) > 0.0)
!        damp(1:NLEV) = ww_time / taur(1:NLEV)
         damp(1:NLEV) = wwt     / taur(1:NLEV)
      endwhere

      where (tauf(1:NLEV) > 0.0)
!         fric(1:NLEV) = ww_time / tauf(1:NLEV)
         fric(1:NLEV) = wwt     / tauf(1:NLEV)
      endwhere

      if (nsponge == 1) call sponge


!     annual cycle period and phase in timesteps

      if (tac > 0.0) tac = TWOPI / (ntspd * tac)
      pac = pac * ntspd

!     compute internal diffusion parameter

      jdelh = ndel/2
      if (diffts > 0.0) then
!        zakk = ww_scale*(radea**ndel)/(TWOPI*diffts/sol_day &
         zakk = 1.0/wwt*(radea**ndel)/(TWOPI*diffts/sol_day &
              * ((NTRU*(NTRU+1.))**jdelh))
      else
         zakk = 0.0
      endif
      zzakk = zakk / (1.0/wwt*(radea**ndel))

!     set coefficients which depend on wavenumber

      zrsq2 = 1.0 / sqrt(2.0)

      jr =-1
      jw = 0
      do jm=0,NTRU
         do jn=jm,NTRU
            jr=jr+2
            ji=jr+1
            jw=jw+1
            nindex(jr)=jn
            nindex(ji)=jn
            spnorm(jr)=zrsq2
            spnorm(ji)=zrsq2
            zsq = jn * (jn+1)
            if (jn > 0) then
               srcn(jr) = 1.0 / zsq
               srcn(ji) = srcn(jr)
            endif
            sak(jr) = -zzakk * zsq**jdelh
            sak(ji) = sak(jr)
         enddo
```



```
            zrsq2=-zrsq2
         enddo

! finally make temperatures dimensionless

         dtns  = dtns    / ct
         dtep  = dtep    / ct
!        dttrp = dttrp   / ct
         t0(:) = t0(:) / ct

!        print out

         write(nud,8120)
         write(nud,8000)
         write(nud,8010) NLEV
         write(nud,8020) NTRU
         write(nud,8030) NLAT
         write(nud,8040) NLON
         if (zakk == 0.0) then
            write(nud,8060)
         else
            write(nud,8070) ndel
            write(nud,8080)
            write(nud,8090) zakk,ndel
            write(nud,8100) diffts
         endif
         write(nud,8110) PNU
         write(nud,8000)
         write(nud,8120)
         return

 8000 format('*************************************************')
 8010 format('* NLEV = ',i6,'  Number of levels              *')
 8020 format('* NTRU = ',i6,'  Triangular truncation         *')
 8030 format('* NLAT = ',i6,'  Number of latitudes           *')
 8040 format('* NLON = ',i6,'  Number of longitues           *')
 8060 format('*           No lateral dissipation          *')
 8070 format('* ndel = ',i6,'  Lateral dissipation          *')
 8080 format('* on vorticity, divergence and temperature    *')
 8090 format('* with diffusion coefficient = ',e13.4,' m**',i1,'/s *')
 8100 format('* e-folding time for smallest scale is ',f7.0,' sec *')
 8110 format('* Robert time filter with parameter PNU =',f8.3,'  *')
 8120 format(/)
      end


!     =================
!     SUBROUTINE MAKEBM
!     =================

      subroutine makebm
      use pumamod

      zdeltsq = delt * delt

      do jlev1 = 1 , NLEV
         do jlev2 = 1 , NLEV
            zaq = zdeltsq * (t0(jlev1) * dsigma(jlev2)&
     &          + dot_product(xlphi(:,jlev1),xlt(jlev2,:)))
            bm1(jlev2,jlev1,1:NTRU) = zaq
         enddo
      enddo

      do jn=1,NTRU
         do jlev = 1 , NLEV
            bm1(jlev,jlev,jn) = bm1(jlev,jlev,jn) + 1.0 / (jn*(jn+1))
         enddo
         call minvers(bm1(1,1,jn),NLEV)
      enddo
```

```
      return
      end

!     =================
!     SUBROUTINE INITSI
!     =================

      subroutine initsi
      use pumamod

!     ******************************************
!     * Initialisation of the Semi Implicit scheme *
!     ******************************************

      dimension zalp(NLEV),zh(NLEV)
      dimension ztautk(NLEV,NLEV)
      dimension ztaudt(NLEV,NLEV)

      tkp(:) = akap * t0(:)
      t0d(1:NLEM) = t0(2:NLEV) - t0(1:NLEM)

      zalp(2:NLEV) = log(sigmh(2:NLEV)) - log(sigmh(1:NLEM))

      xlphi(:,:) = 0.0
      xlphi(1,1) = 1.0
      do jlev = 2 , NLEV
         xlphi(jlev,jlev) = 1.0 - zalp(jlev)*sigmh(jlev-1)/dsigma(jlev)
         xlphi(jlev,1:jlev-1) = zalp(jlev)
      enddo

      do jlev = 1 , NLEV
         c(jlev,:) = xlphi(:,jlev) * (dsigma(jlev) / dsigma(:))
      enddo

!     **********************   tkp(i) = t0(i) * AKAP
!     * matrix xlt - part 1 *
!     **********************

      do jlev = 1 , NLEV
         ztautk(:,jlev) = tkp(jlev) * c(:,jlev)
      enddo

!     *********************   dsigma(i) = sigmh(i) - sigmh(i-1)
!     * matrix xlt part 2 *   rdsig (i) = 0.5 / dsigma(i)
!     *********************

      ztaudt(1,1)      = 0.5 * t0d(1) * (sigmh(1) - 1.0)
      ztaudt(2:NLEV,1) = 0.5 * t0d(1) *  dsigma(2:NLEV)

      do j= 2 , NLEV
         do i = 1 , j-1
            ztaudt(i,j) =  dsigma(i) * rdsig(j) &
            * (t0d(j-1) * (sigmh(j-1)-1.0) + t0d(j) * (sigmh(j)-1.0))
         enddo
            ztaudt(j,j) =  0.5                  &
            * (t0d(j-1) *  sigmh(j-1)      + t0d(j) * (sigmh(j)-1.0))
         do i = j+1 , NLEV
            ztaudt(i,j) =  dsigma(i) * rdsig(j) &
            * (t0d(j-1) *  sigmh(j-1)      + t0d(j) *  sigmh(j)     )
         enddo
      enddo

      xlt(:,:) = ztautk(:,:) + ztaudt(:,:)

!     xlt finished

      zfctr=0.001*cv*cv/ga
      do jlev=1,NLEV
         zh(jlev) = dot_product(xlphi(:,jlev),t0(:)) * zfctr
```



```
      enddo

!     *********************************
!     * write out vertical information *
!     *********************************

      ilev = min(NLEV,5)
      write(nud,9001)
      write(nud,9002)
      write(nud,9003)
      write(nud,9002)
      do jlev=1,NLEV
        write(nud,9004) jlev,sigma(jlev),t0(jlev)*ct,zh(jlev)
      enddo
      write(nud,9002)
      write(nud,9001)

!     matrix c

      write(nud,9012)
      write(nud,9013) 'c',(jlev,jlev=1,ilev)
      write(nud,9012)
      do jlev=1,NLEV
        write(nud,9014) jlev,(c(i,jlev),i=1,ilev)
      enddo
      write(nud,9012)
      write(nud,9001)

!     matrix xlphi

      write(nud,9012)
      write(nud,9013) 'xlphi',(jlev,jlev=1,ilev)
      write(nud,9012)
      do jlev=1,NLEV
        write(nud,9014) jlev,(xlphi(i,jlev),i=1,ilev)
      enddo
      write(nud,9012)
      write(nud,9001)
      return
 9001 format(/)
 9002 format(33('*'))
 9003 format('* Lv *  Sigma Basic-T  Height *')
 9004 format('*',i3,' *',3f8.3,' *')
 9012 format(69('*'))
 9013 format('* Lv *',a5,i7,4i12,' *')
 9014 format('*',i3,' *',5f12.8,' *')
      end

!     =====================
!     SUBROUTINE INITRANDOM
!     =====================

      subroutine initrandom
      use pumamod
      integer :: i, clock

!     Set random number generator seed

      call random_seed(size=nseedlen)
      allocate(meed(nseedlen))

!     Take seed from namelist parameter 'SEED' ?

      if (seed(1) /= 0) then
         meed(:) = 0
         i = nseedlen
         if (i > 8) i = 8
         meed(1:i) = seed(1:i)
      else
```

```
         call system_clock(count=clock)
         meed(:) = clock + 37 * (/(i,i=1,nseedlen)/)
      endif
      call random_seed(put=meed)
      return
      end

!     ====================
!     SUBROUTINE PRINTSEED
!     ====================

      subroutine printseed
      use pumamod
      integer :: i

      write (nud,9020)
      write (nud,9010)
      do i = 1 , nseedlen
         write (nud,9000) i,meed(i)
      enddo
      write (nud,9010)
      write (nud,9020)
      return
 9000 format('* seed(',i1,') = ',i10,' *')
 9010 format('************************')
 9020 format(/)
      end

!     ====================
!     SUBROUTINE INITRUIDO
!     ====================

      subroutine initruido
      use pumamod
      if (nruido > 0) then
         allocate(ruido(nlon,nlat,nlev))
         allocate(ruidop(nhor,nlev))
         ruido = 77
         ruidop = 88
      endif
      return
      end

!     ====================
!     SUBROUTINE STEPRUIDO
!     ====================

      subroutine stepruido
      use pumamod
      real :: zr

      if (mypid == NROOT) then
         if (nruido == 1) then
            zr = disp*gasdev()
            ruido(:,:,:) = zr
         elseif (nruido == 2) then
            do jlev=1,NLEV
               do jlat=1,NLAT
                  do jlon=1,NLON
                     ruido(jlon,jlat,jlev) = disp*gasdev()
                  enddo
               enddo
            enddo
         elseif (nruido == 3) then
            do jlev=1,NLEV
               do jlat=1,NLAT,2
                  do jlon=1,NLON
                     ruido(jlon,jlat  ,jlev) = disp*gasdev()
                     ruido(jlon,jlat+1,jlev) = ruido(jlon,jlat,jlev)
```



```
                  enddo
               enddo
            enddo
         endif
      endif ! (mypid == NROOT)

      call mpscgp(ruido,ruidop,NLEV)
      return
      end

!     ==================
!     SUBROUTINE MINVERS
!     ==================

      subroutine minvers(a,n)
      dimension a(n,n),b(n,n),indx(n)

      b = 0.0
      do j = 1 , n
         b(j,j) = 1.0
      enddo
      call ludcmp(a,n,indx)
      do j = 1 , n
         call lubksb(a,n,indx,b(1,j))
      enddo
      a = b
      return
      end

!     =================
!     SUBROUTINE LUBKSB
!     =================

      subroutine lubksb(a,n,indx,b)
      dimension a(n,n),b(n),indx(n)
      k = 0
      do i = 1 , n
         l    = indx(i)
         sum  = b(l)
         b(l) = b(i)
         if (k > 0) then
            do j = k , i-1
               sum = sum - a(i,j) * b(j)
            enddo
         else if (sum /= 0.0) then
            k = i
         endif
         b(i) = sum
      enddo

      do i = n , 1 , -1
         sum = b(i)
         do j = i+1 , n
            sum = sum - a(i,j) * b(j)
         enddo
         b(i) = sum / a(i,i)
      enddo
      return
      end

!     =================
!     SUBROUTINE LUDCMP
!     =================

      subroutine ludcmp(a,n,indx)
      dimension a(n,n),indx(n),vv(n)

      d = 1.0
      vv = 1.0 / maxval(abs(a),2)
```

```
      do 19 j = 1 , n
         do i = 2 , j-1
            a(i,j) = a(i,j) - dot_product(a(i,1:i-1),a(1:i-1,j))
         enddo
         aamax = 0.0
         do i = j , n
            if (j > 1) &
     &      a(i,j) = a(i,j) - dot_product(a(i,1:j-1),a(1:j-1,j))
            dum = vv(i) * abs(a(i,j))
            if (dum .ge. aamax) then
               imax = i
               aamax = dum
            endif
         enddo
         if (j .ne. imax) then
            do 17 k = 1 , n
               dum = a(imax,k)
               a(imax,k) = a(j,k)
               a(j,k) = dum
   17       continue
            d = -d
            vv(imax) = vv(j)
         endif
         indx(j) = imax
         if (a(j,j) == 0.0) a(j,j) = tiny(a(j,j))
         if (j < n) a(j+1:n,j) = a(j+1:n,j) / a(j,j)
   19 continue
      return
      end

!     ============================
!     SUBROUTINE FILTER_ZONAL_WAVES
!     ============================

      subroutine filter_zonal_waves(pfc)
      use pumamod
      dimension pfc(2,NLON/2,NLPP)

      do jlat = 1 , NLPP
         pfc(1,1:NTP1,jlat) = pfc(1,1:NTP1,jlat) * nselzw(:)
         pfc(2,1:NTP1,jlat) = pfc(2,1:NTP1,jlat) * nselzw(:)
      enddo

      return
      end


!     ================================
!     SUBROUTINE FILTER_SPECTRAL_MODES
!     ================================

      subroutine filter_spectral_modes
      use pumamod

      j =  0
      k = -1
      do m = 0 , NTRU
         do n = m , NTRU
            k = k + 2
            j = j + 1
            if (nselsp(j) == 0) then
               spp(k:k+1  ) = 0.0
               sdp(k:k+1,:) = 0.0
               stp(k:k+1,:) = 0.0
               spt(k:k+1  ) = 0.0
               sdt(k:k+1,:) = 0.0
               stt(k:k+1,:) = 0.0
               spm(k:k+1  ) = 0.0
```



```
               sdm(k:k+1,:) = 0.0
               stm(k:k+1,:) = 0.0
               srp1(k:k+1,:) = 0.0
               srp2(k:k+1,:) = 0.0
               if (n < NTRU) then
                  szp(k+2:k+3,:) = 0.0
                  szt(k+2:k+3,:) = 0.0
                  szm(k+2:k+3,:) = 0.0
               endif
            endif
         enddo
      enddo

      return
      end


!     ================
!     SUBROUTINE NOISE
!     ================

      subroutine noise(kickval)
      use pumamod

!     kickval = -1 : read ln(ps) from puma_sp_init
!     kickval =  0 : model runs zonally symmetric with no eddies
!     kickval =  1 : add white noise to ln(Ps) asymmetric hemispheres
!     kickval =  2 : add white noise to ln(Ps) symmetric to the equator
!     kickval =  3 : force mode(1,2) of ln(Ps) allowing reproducable runs
!     kickval =  4 : add white noise to symmetric zonal wavenumbers 7 of ln(Ps)

      integer :: kickval
      integer :: jsp, jsp1, jn, jm
      integer :: jr, ji, ins
      real    :: zr, zi, zscale, zrand

      zscale = 0.000001          ! amplitude of noise
      zr     = 0.001             ! kickval=3 value for mode(1,2) real
      zi     = 0.0005            ! kickval=3 value for mode(1,2) imag

      select case (kickval)
      case (-1)
         open(71, file=puma_sp_init,form='unformatted',iostat=iostat)
         if (iostat /= 0) then
            write(nud,*) ' *** kick=-1: needs file <',trim(puma_sp_init),'> ***'
            stop
         endif
         read(71,iostat=iostat) sp(:)
         if (iostat /= 0) then
            write(nud,*) ' *** error reading file <',trim(puma_sp_init),'> ***'
            stop
         endif
         close(71)
         write(nud,*) 'initial ln(ps) field read from <',trim(puma_sp_init),'>'
         return
      case (0)                   ! do nothing
      case (1)
         jsp1=2*NTP1+1
         do jsp=jsp1,NRSP
            call random_number(zrand)
            if (mrpid > 0) zrand = zrand + mrpid * 0.01
            sp(jsp)=sp(jsp)+zscale*(zrand-0.5)
         enddo
         write(nud,*) 'white noise added'
      case (2)
         jr=2*NTP1-1
         do jm=1,NTRU
            do jn=jm,NTRU
               jr=jr+2
```

```
               ji=jr+1
               if (mod(jn+jm,2) == 0) then
                  call random_number(zrand)
                  if (mrpid > 0) zrand = zrand + mrpid * 0.01
                  sp(jr)=sp(jr)+zscale*(zrand-0.5)
                  sp(ji)=sp(ji)+zscale*(zrand-0.5)
               endif
            enddo
         enddo
         write(nud,*) 'symmetric white noise added'
      case (3)
         sp(2*NTP1+3) = sp(2*NTP1+3) + zr
         sp(2*NTP1+4) = sp(2*NTP1+4) + zi
         write(nud,*) 'mode(1,2) of ln(Ps) set to (',sp(2*NTP1+3),',',sp(2*NTP1+4),')'
      case (4)
         jr=2*NTP1-1
         do jm=1,NTRU
            do jn=jm,NTRU
               jr=jr+2
               ji=jr+1
               if (mod(jn+jm,2) == 0 .and. jm == 7) then
                  call random_number(zrand)
                  sp(jr)=sp(jr)+zscale*(zrand-0.5)
                  sp(ji)=sp(ji)+zscale*(zrand-0.5)
               endif
            enddo
         enddo
         write(nud,*) 'symmetric zonal wavenumbers 7 of ln(Ps) perturbed',     &
     &         'with white noise.'
      case default
         write(nud,*) 'Value ',kickval  ,' for kickval not implemented.'
         stop
      end select

      if (nwspini == 1) then
         open(71, file=puma_sp_init, form='unformatted')
         write(71) sp(:)
         close(71)
      endif

      return
      end

!     ================
!     SUBROUTINE SETZT
!     ================
      subroutine setzt
      use pumamod

!     ************************************************************
!     * Set up the restoration temperature fields sr1 and sr2    *
!     * for aqua planet conditions.                              *
!     * The temperature at sigma = 1 is <tgr>, entered in kelvin. *
!     * The lapse rate of ALR K/m is assumed under the tropopause *
!     * and zero above. The tropopause is defined by <dtrop>.    *
!     * The smoothing ot the tropopause depends on <dttrp>.      *
!     ************************************************************

      dimension ztrs(NLEV)  ! Mean profile
      dimension zfac(NLEV)

      sr1(:,:) = 0.0 ! NESP,NLEV
      sr2(:,:) = 0.0 ! NESP,NLEV

!     Temperatures in [K]

      zsigprev = 1.0  ! sigma value
      ztprev   = tgr  ! Temperature [K]
      zzprev   = 0.0  ! Height      [m]
```



```
      do jlev = NLEV , 1 , -1   ! from bottom to top of atmosphere
        zzp=zzprev+(gascon*ztprev/ga)*log(zsigprev/sigma(jlev))
        ztp=tgr-dtrop*alr ! temperature at tropopause
        ztp=ztp+sqrt((.5*alr*(zzp-dtrop))**2+dttrp**2)
        ztp=ztp-.5*alr*(zzp-dtrop)
        ztpm=.5*(ztprev+ztp)
        zzpp=zzprev+(gascon*ztpm/ga)*log(zsigprev/sigma(jlev))
        ztpp=tgr-dtrop*alr
        ztpp=ztpp+sqrt((.5*alr*(zzpp-dtrop))**2+dttrp**2)
        ztpp=ztpp-.5*alr*(zzpp-dtrop)
        ztrs(jlev)=ztpp
        zzprev=zzprev+(.5*(ztpp+ztprev)*gascon/ga)*log(zsigprev/sigma(jlev))
        ztprev=ztpp
        zsigprev=sigma(jlev)
      enddo

      do jlev=1,NLEV
         ztrs(jlev)=ztrs(jlev)/ct
      enddo

!*****************************************************************
! loop to set array zfac - this controls temperature gradients as a
! function of sigma in tres. it is a sine wave from one at
! sigma = 1 to zero at stps (sigma at the tropopause) .
!*****************************************************************
! first find sigma at dtrop
!
      zttrop=tgr-dtrop*alr
      ztps=(zttrop/tgr)**(ga/(alr*gascon))
!
! now the latitudinal variation in tres is set up ( this being in terms
! of a deviation from t0 which is usually constant with height)
!
      zsqrt2  = sqrt(2.0)
      zsqrt04 = sqrt(0.4)
      zsqrt6  = sqrt(6.0)
      do 2100 jlev=1,NLEV
        zfac(jlev)=sin(0.5*PI*(sigma(jlev)-ztps)/(1.-ztps))
        if (zfac(jlev).lt.0.0) zfac(jlev)=0.0
        sr1(1,jlev)=zsqrt2*(ztrs(jlev)-t0(jlev))
        sr2(3,jlev)=(1./zsqrt6)*dtns*zfac(jlev)
        sr1(5,jlev)=-2./3.*zsqrt04*dtep*zfac(jlev)
 2100 continue
      write(nud,*) '*************************************************'
      write(nud,*) '* Restoration Temperature set up for aqua planet *'
      write(nud,*) '*************************************************'
      return
      end

!     ======================
!     SUBROUTINE PRINTPROFILE
!     ======================

      subroutine printprofile
      use pumamod

!     *********************************
!     * write out vertical information *
!     *********************************

      write(nud,9001)
      write(nud,9002)
      write(nud,9003)
      write(nud,9002)

      do jlev=1,NLEV
         zt = (sr1(1,jlev)/sqrt(2.0) + t0(jlev)) * ct
         if (tauf(jlev) > 0.1) then
```

```
            write(nud,9004) jlev,sigma(jlev),zt,taur(jlev),tauf(jlev)
         else
            write(nud,9005) jlev,sigma(jlev),zt,taur(jlev)
         endif
      enddo

      write(nud,9002)
      write(nud,9001)
      return
 9001 format(/)
 9002 format(46('*'))
 9003 format('* Lv *   Sigma Restor-T tauR [s]  tauF [s]  *')
 9004 format('*',i3,' *',f8.3,f9.3,2e10.3,' *')
 9005 format('*',i3,' *',f8.3,f9.3,e10.3,'    -*')
      end


!     ====================
!     SUBROUTINE READ_SURF
!     ====================

      subroutine read_surf(kcode,psp,klev,kread)
      use pumamod

      logical :: lexist
      integer :: kread
      integer :: ihead(8)
      character(len=256) :: yfilename
      real :: psp(NESP,klev)
      real :: zgp(NUGP,klev)
      real :: zpp(NHOR,klev)

      kread = 0
      if (mypid == NROOT) then
         if (NLAT < 1000) then
         write(yfilename,'("N",I3.3,"_surf_",I4.4,".sra")') NLAT,kcode
         else
         write(yfilename,'("N",I4.4,"_surf_",I4.4,".sra")') NLAT,kcode
         endif
         inquire(file=yfilename,exist=lexist)
      endif
      call mpbcl(lexist)
      if (.not. lexist) return

      if (mypid == NROOT) then
         open(65,file=yfilename,form='formatted')
         write(nud,*) 'Reading file <',trim(yfilename),'>'
         do jlev = 1 , klev
            read (65,*) ihead(:)
            read (65,*) zgp(:,jlev)
         enddo
         close(65)
         if (kcode == 134) then
            write(nud,*) "Converting Ps to LnPs"
            zscale   = log(100.0) - log(psurf) ! Input [hPa] / PSURF [Pa]
            zgp(:,:) = log(zgp(:,:)) + zscale
         endif
         call reg2alt(zgp,klev)
      endif ! (mypid == NROOT)
      call mpscgp(zgp,zpp,klev)
      call gp2fc(zpp,NLON,NLPP*klev)
      do jlev = 1 , klev
         call fc2sp(zpp(1,jlev),psp(1,jlev))
      enddo
      call mpsum(psp,klev)
      kread = 1
      return
      end subroutine read_surf
```



```
!     ====================
!     SUBROUTINE READ_VARGP
!     ====================

      subroutine read_vargp(kcode,klev,kread)
      use pumamod

      logical :: lexist
      integer :: ihead(8)
      character(len=256) :: yfilename
      real :: zgp(NUGP,klev)

      kread = 0
      if (mypid == NROOT) then
         if (NLAT < 1000) then
         write(yfilename,'("N",I3.3,"_surf_",I4.4,".sra")') NLAT,kcode
         else
         write(yfilename,'("N",I4.4,"_surf_",I4.4,".sra")') NLAT,kcode
         endif
         inquire(file=yfilename,exist=lexist)
      endif
      call mpbcl(lexist)
      if (.not. lexist) then
         if (mypid == NROOT) then
            write(nud,*) 'File <',trim(yfilename),'> not found'
         endif
         return
      endif

      if (mypid == NROOT) then
         open(65,file=yfilename,form='formatted')
         write(nud,*) 'Reading file <',trim(yfilename),'>'
         do jlev = 1 , klev
            read (65,*) ihead(:)
            read (65,*) zgp(:,jlev)
         enddo
         close(65)
         call reg2alt(zgp,klev)
      endif ! (mypid == NROOT)

      select case(kcode)
         case(121)
            !--- non-dimensionalize and shift const radiative rest. temp.
            if (mypid == NROOT) then
               zgp(:,:) = zgp(:,:)/ct
               do jhor = 1,nugp
                  zgp(jhor,:) = zgp(jhor,:) - t0(:)
               enddo
            endif
            allocate(gr1(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gr1 allocated'
            endif
            call mpscgp(zgp,gr1,klev)
         case(122)
            !--- non-dimensionalize variable. radiative rest. temp.
            if (mypid == NROOT) then
               zgp(:,:) = zgp(:,:)/ct
            endif
            allocate(gr2(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gr2 allocated'
            endif
            call mpscgp(zgp,gr2,klev)
         case(123)
            !--- non-dimensionalize radiative relaxation time scale
            if (mypid == NROOT) then
```

```
               zgp(:,:) = zgp(:,:)/ww_scale
            endif
            allocate(gtdamp(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gtdamp allocated'
            endif
            call mpscgp(zgp,gtdamp,klev)
         case(124)
            !--- non-dimensionalize and shift const. convective rest. temp.
            if (mypid == NROOT) then
               zgp(:,:) = zgp(:,:)/ct
               do jhor = 1,nugp
                  zgp(jhor,:) = zgp(jhor,:) - t0(:)
               enddo
            endif
            allocate(gr1c(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gr1c allocated'
            endif
            call mpscgp(zgp,gr1c,klev)
         case(125)
            !--- non-dimensionalize variable. convective rest. temp.
            if (mypid == NROOT) then
               zgp(:,:) = zgp(:,:)/ct
            endif
            allocate(gr2c(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gr2c allocated'
            endif
            call mpscgp(zgp,gr2c,klev)
         case(126)
            !--- non-dimensionalize convective relaxation time scale
            if (mypid == NROOT) then
               zgp(:,:) = zgp(:,:)/ww_scale
            endif
            allocate(gtdampc(nhor,klev))
            if (mypid == NROOT) then
               write(nud,*) 'Field gtdampc allocated'
            endif
            call mpscgp(zgp,gtdampc,klev)
      end select
      kread = 1
      return
      end subroutine read_vargp

!     ===============
!     SUBROUTINE DIAG
!     ===============

      subroutine diag
      use pumamod
      if (noutput > 0 .and. mod(nstep,ndiag) == 0) then
         if (ncoeff > 0) call prisp
         call xsect
      endif
      call energy
      return
      end

!     ================
!     SUBROUTINE PRISP
!     ================

      subroutine prisp
      use pumamod

      character(30) :: title

      scale = 100.0
```



```
      title = 'Vorticity [10-2]'
      do 100 jlev=1,NLEV
         if (ndil(jlev).ne.0) call wrspam(sz(1,jlev),jlev,title,scale)
  100 continue

      title = 'Divergence [10-2]'
      do 200 jlev=1,NLEV
         if (ndil(jlev).ne.0) call wrspam(sd(1,jlev),jlev,title,scale)
  200 continue

      scale = 1000.0
      title = 'Temperature [10-3]'
      do 300 jlev=1,NLEV
         if (ndil(jlev).ne.0) call wrspam(st(1,jlev),jlev,title,scale)
  300 continue

      title = 'Pressure [10-3]'
      call wrspam(sp,0,title,scale)

      return
      end

!     ====================
!     SUBROUTINE POWERSPEC
!     ====================

      subroutine powerspec(pf,pspec)
      use pumamod
      real :: pf(2,NCSP)
      real :: pspec(NTP1)

      do j = 1 , NTP1
         pspec(j) = 0.5 * (pf(1,j) * pf(1,j) + pf(2,j) * pf(2,j))
      enddo

      j = NTP1 + 1
      do m = 2 , NTP1
         do l = m , NTP1
            pspec(l) = pspec(l) + pf(1,j) * pf(1,j) + pf(2,j) * pf(2,j)
            j = j + 1
         enddo
      enddo
      return
      end

!     ======================
!     SUBROUTINE POWERPRINT
!     ======================

      subroutine powerprint(text,pspec)
      use pumamod
      character(3) :: text
      real :: pspec(NTP1)

      zmax = maxval(pspec(:))
      if (zmax <= 1.0e-20) return
      zsca = 10 ** (4 - int(log10(zmax)))
      write(nud,1000) text,(int(pspec(j)*zsca),j=2,13)
      return
 1000 format('* Power(',a3,')',i8,11i5,' *')
      end



!     ==============
!     FUNCTION RMSSP
!     ==============
```

```
      function rmssp(pf)
      use pumamod
      real pf(NESP,NLEV)

      zsum = 0.0
      do jlev = 1 , NLEV
         zsum = zsum + dsigma(jlev)&
     &        * (dot_product(pf(1:NZOM,jlev),pf(1:NZOM,jlev)) * 0.5&
     &        +  dot_product(pf(NZOM+1:NRSP,jlev),pf(NZOM+1:NRSP,jlev)))
      enddo
      rmssp = zsum
      return
      end

!     =================
!     SUBROUTINE ENERGY
!     =================

      subroutine energy
      use pumamod

      parameter (idim=5) ! Number of scalars for GUI timeseries

!     calculates various global diagnostic quantities
!     remove planetary vorticity so sz contains relative vorticity

      real :: spec(NTP1)
      real (kind=4) ziso(idim)

      sz(3,:) = sz(3,:) - plavor

!     ********************************************
!     calculate means - zpsitot rms vorticity
!                       zchitot rms divergence
!                       ztmptot rms temperature
!                       ztotp  ie+pe potential energy
!                       zamsp mean surface pressure
!     ********************************************

      zsqrt2 = sqrt(2.0)
      zamsp  = 1.0 + span(1) / zsqrt2
      zst    = dot_product(dsigma(:),st(1,:)) / zsqrt2
      ztout1 = dot_product(dsigma(:),t0(:))

      ztout2 = 0.0
      zst2b  = 0.0
      ztoti  = 0.0
      do jlev = 1 , NLEV
         ztout2 = ztout2 + dsigma(jlev) * t0(jlev) * t0(jlev)
         zst2b  = zst2b  + dsigma(jlev) * t0(jlev) * st(1,jlev)
         ztoti  = ztoti + dsigma(jlev)&
     &          * (dot_product(span(1:NZOM),st(1:NZOM,jlev)) * 0.5&
     &          +  dot_product(span(NZOM+1:NRSP),st(NZOM+1:NRSP,jlev)))
      enddo

      ztotp = dot_product(span(1:NZOM),so(1:NZOM)) * 0.5&
     &      + dot_product(span(NZOM+1:NRSP),so(NZOM+1:NRSP))&
     &      + so(1)/zsqrt2 + (zamsp*ztout1+ztoti+zst) / akap

      zpsitot = sqrt(rmssp(sz))
      zchitot = sqrt(rmssp(sd))
      ztmptot = sqrt(rmssp(st)+ztout2+zst2b*zsqrt2)

      ziso(1) = ct * (spnorm(1) * st(1,NLEV) + t0(NLEV)) - 273.16 ! T(NLEV) [C]
      ziso(2) = ww_scale * zchitot * 1.0e6
      ziso(3) = ztmptot
      ziso(4) = ztotp
      ziso(5) = sz(3,2)
      !XW(Mar/25/2017) to remove GUI: call guiput("SCALAR" // char(0) ,ziso,idim
```



```
,1,1)

!     restore sz to absolute vorticity

      sz(3,:) = sz(3,:) + plavor

      if (mod(nstep,ndiag) /= 0) return ! was called for GUI only
      write(nud,9001)
      write(nud,9002) nstep,zpsitot,zchitot,ztmptot,ztotp,zamsp
      write(nud,9002)
      write(nud,9011) (j,j=1,12)
      write(nud,9012)
      call powerspec(span,spec)
      call powerprint('Pre',spec)
      call powerspec(sz(1,NLEV),spec)
      call powerprint('Vor',spec)
      call powerspec(sd(1,NLEV),spec)
      call powerprint('Div',spec)
      call powerspec(st(1,NLEV),spec)
      call powerprint('Tem',spec)
      return
 9001 format(/,'  nstep    rms z      rms d      rms t      &
     & pe+ie      msp')
 9002 format(i10,4x,4g12.5,g15.8)
!9009 format('*',75(' '),' *')
!9010 format('* Power(',a,') ',7e9.2,' *')
 9011 format('* Wavenumber ',i8,11i5,' *')
 9012 format('',78('*'))
      end

!     =================
!     SUBROUTINE NTOMIN
!     =================

      subroutine ntomin(kstep,imin,ihou,iday,imon,iyea)
      use pumamod
      istep = kstep                           ! day [0-29] month [0-11]
      if (istep .lt. 0) istep = 0             ! min [0-59] hour  [0-23]
      imin = mod(istep,ntspd) * 1440 / ntspd  ! minutes of current day
      ihou = imin / 60                        ! hours   of current day
      imin = imin - ihou * 60                 ! minutes of current hour
      iday = istep / ntspd                    ! days    in this run
      imon = iday / 30                        ! months  in this run
      iday = iday - imon * 30                 ! days    of current month
      iyea = imon / 12                        ! years   in this run
      imon = imon - iyea * 12                 ! month   of current year
      iday = iday + 1
      imon = imon + 1
      iyea = iyea + 1
      return
      end

!     =================
!     SUBROUTINE NTODAT
!     =================

      subroutine ntodat(istep,datch)
      character(18) :: datch
      character(3) :: mona(12)
      data mona /'Jan','Feb','Mar','Apr','May','Jun',&
     &          'Jul','Aug','Sep','Oct','Nov','Dec'/
      call ntomin(istep,imin,ihou,iday,imon,iyea)
      write(datch,20030) iday,mona(imon),iyea,ihou,imin
20030 format(i2,'-',a3,'-',i4.4,2x,i2,':',i2.2)
      end


!     =================
!     SUBROUTINE WRSPAM
```

```
!     =================

      subroutine wrspam(ps,klev,title,scale)
      use pumamod
!
      dimension ps(NRSP)
      character(30) :: title
      character(18) :: datch

!     cab(i)=real(scale*sqrt(ps(i+i-1)*ps(i+i-1)+ps(i+i)*ps(i+i)))

      call ntodat(nstep,datch)
      write(nud,'(1x)')
      write(nud,20000)
      write(nud,20030) datch,title,klev
      write(nud,20000)
      write(nud,20020) (i,i=0,9)
      write(nud,20000)
      write(nud,20100) (cab(i),i=1,10)
      write(nud,20200) (cab(i),i=NTRU+2,NTRU+10)
      write(nud,20300) (cab(i),i=2*NTRU+2,2*NTRU+9)
      write(nud,20400) (cab(i),i=3*NTRU+1,3*NTRU+7)
      write(nud,20000)
      write(nud,'(1x)')

20000 format(78('*'))
20020 format('* n *',10i7,' *')
20030 format('*  *',a18,2x,a30,'  Level ',i2,11x,'*')
20100 format('* 0 *',f8.2,9f7.2,' *')
20200 format('* 1 *',8x,9f7.2,' *')
20300 format('* 2 *',15x,8f7.2,' *')
20400 format('* 3 *',22x,7f7.2,' *')
      contains
      function cab(i)
         cab = scale * sqrt(ps(i+i-1)*ps(i+i-1)+ps(i+i)*ps(i+i))
      end function cab
      end

!     ===============
!     SUBROUTINE WRZS
!     ===============

      subroutine wrzs(zs,title,scale)
      use pumamod
!
      dimension zs(NLAT,NLEV)
      character(30) :: title
      character(18) :: datch

      ip = NLAT / 16
      ia = ip/2
      ib = ia + 7 * ip
      id = NLAT + 1 - ia
      ic = id - 7 * ip

      call ntodat(nstep,datch)
      write(nud,'(1x)')
      write(nud,20000)
      write(nud,20030) datch,title
      write(nud,20000)
      write(nud,20020) (chlat(i),i=ia,ib,ip),(chlat(j),j=ic,id,ip)
      write(nud,20000)
      do 200 jlev = 1 , NLEV
         write(nud,20100) jlev,((int(zs(i,jlev)*scale)),i=ia,ib,ip),&
     &                         ((int(zs(j,jlev)*scale)),j=ic,id,ip),jlev
  200 continue
      write(nud,20000)
      write(nud,'(1x)')
```



```
20000 format(78('*'))
20020 format('* Lv *',16(1x,a3),' * Lv *')
20030 format('*  *',a18,2x,a30,20x,'*')
20100 format('*',i2,' *',16i4,' *',i2,' *')
      end

!     ================
!     SUBROUTINE XSECT
!     ================

      subroutine xsect
      use pumamod
      character(30) :: title

      scale = 10.0
      title = 'Zonal Wind [0.1 m/s]'
      call wrzs(csu,title,scale)
      title = 'Meridional Wind [0.1 m/s]'
      call wrzs(csv,title,scale)
      scale = 1.0
      title = 'Temperature [C]'
      call wrzs(cst,title,scale)
      return
      end

!     ==================
!     SUBROUTINE WRITESP
!     ==================

      subroutine writesp(kunit,pf,kcode,klev,pscale,poff)
      use pumamod
      real    :: pf(NRSP)
      real    :: zf(NRSP)
      integer :: ihead(8)

      call ntomin(nstep,nmin,nhour,nday,nmonth,nyear)

      ihead(1) = kcode
      ihead(2) = klev
      ihead(3) = nday + 100 * nmonth + 10000 * nyear
      ihead(4) = nmin + 100 * nhour
      ihead(5) = NRSP
      ihead(6) = 1
      ihead(7) = 1
      ihead(8) = 0

!     normalize ECHAM compatible and scale to physical dimensions

      zf(:) = pf(:) * spnorm(1:NRSP) * pscale
      zf(1) = zf(1) + poff ! Add offset if necessary
      write(kunit) ihead
      write(kunit) zf

      return
      end

!     ==================
!     SUBROUTINE WRITEGP
!     ==================

      subroutine writegp(kunit,pf,kcode,klev)
      use pumamod
      real :: pf(NHOR)
      real :: zf(NUGP)
      integer :: ihead(8)

      call mpgagp(zf,pf,1)

      if (mypid == NROOT) then
```

```
         call alt2reg(zf,1)
         call ntomin(nstep,nmin,nhour,nday,nmonth,nyear)

         ihead(1) = kcode
         ihead(2) = klev
         ihead(3) = nday + 100 * nmonth + 10000 * nyear
         ihead(4) = nmin + 100 * nhour
         ihead(5) = NLON
         ihead(6) = NLAT
         ihead(7) = 1
         ihead(8) = 0

         write(kunit) ihead
         write(kunit) zf
      endif

      return
      end


!     ================
!     SUBROUTINE OUTSP
!     ================

      subroutine outsp
      use pumamod
      real zsr(NESP)

      if (nwrioro == 1) then
         call writesp(40,so,129,0,cv*cv,0.0)
         nwrioro = 0
      endif

      if (nextout == 1) then
         call writesp(40,sp2,40,0,1.0,log(psmean))
         call writesp(40,sp1,41,0,1.0,log(psmean))
         do jlev = 1,NLEV
            call writesp(40,st2(1,jlev),42,jlev,ct,t0(jlev)*ct)
         enddo
         do jlev = 1,NLEV
            call writesp(40,st1(1,jlev),43,jlev,ct,t0(jlev)*ct)
         enddo
      endif

!     ************
!     * pressure *
!     ************

      call writesp(40,sp,152,0,1.0,log(psmean))

!     ***************
!     * temperature *
!     ***************

      do jlev = 1 , NLEV
         call writesp(40,st(1,jlev),130,jlev,ct,t0(jlev)*ct)
      enddo

!     ********************
!     * res. temperature *
!     ********************

      zampl = cos((real(nstep)-pac)*tac)
      do jlev = 1 , NLEV
         zsr(:)=sr1(:,jlev)+sr2(:,jlev)*zampl
         call writesp(40,zsr,154,jlev,ct,t0(jlev)*ct)
      enddo

!     **************
```



```
!     * divergence *
!     **************

      do jlev = 1 , NLEV
         call writesp(40,sd(1,jlev),155,jlev,ww_scale,0.0)
      enddo

!     *************
!     * vorticity *
!     *************

      do jlev = 1 , NLEV
         zsave = sz(3,jlev)
         sz(3,jlev) = sz(3,jlev) - plavor
         call writesp(40,sz(1,jlev),138,jlev,ww_scale,0.0)
         sz(3,jlev) = zsave
      enddo

      return
      end

!     ================
!     SUBROUTINE OUTGP
!     ================

      subroutine outgp
      use pumamod
      real zhelp(NHOR)
!
!     energy diagnostics
!
      if(nenergy > 0) then
       do je=1,9
        jcode=300+je
        zhelp(:)=denergy(:,je)
        call writegp(40,zhelp,jcode,0)
       enddo
      endif
      if(nentropy > 0) then
       do je=1,9
        jcode=310+je
        zhelp(:)=dentropy(:,je)
        call writegp(40,zhelp,jcode,0)
       enddo
      endif
!
      return
      end


!     ====================
!     SUBROUTINE CHECKUNIT
!     ====================

      subroutine checkunit
      use pumamod

      write(ncu,1000) nstep,'sp( 1 )',sp(1),sp(1)*spnorm(1)+log(psmean)
      write(ncu,1000) nstep,'st( 1,1)',st(1,1),st(1,1)*spnorm(1)*ct+t0(1)*ct
      write(ncu,1000) nstep,'sd( 1,1)',sd(1,1),sd(1,1)*spnorm(1)*ww_scale
      write(ncu,1000) nstep,'sz( 1,1)',sz(1,1),sz(1,1)*spnorm(1)*ww_scale

      write(ncu,1000) nstep,'st( 1,NLEV)',st(1,NLEV),st(1,NLEV)*spnorm(1)*ct+t0(5)
*ct
      write(ncu,1000) nstep,'sd( 1,NLEV)',sd(1,NLEV),sd(1,NLEV)*spnorm(1)*ww_scal
e
      write(ncu,1000) nstep,'sz( 1,NLEV)',sz(1,NLEV),sz(1,NLEV)*spnorm(1)*ww_scal
e
```

```
      if (100 < NRSP) then
      write(ncu,1000) nstep,'sp(100 )',sp(100),sp(100)*spnorm(100)
      write(ncu,1000) nstep,'st(100,NLEV)',st(100,NLEV),st(100,NLEV)*spnorm(100)*c
t
      write(ncu,1000) nstep,'sd(100,NLEV)',sd(100,NLEV),sd(100,NLEV)*spnorm(100)*w
w_scale
      write(ncu,1000) nstep,'sz(100,NLEV)',sz(100,NLEV),sz(100,NLEV)*spnorm(100)*w
w_scale
      endif

      return
 1000 format(i5,1x,a,1x,2f14.7)
      end


!     ====================
!     * SUBROUTINE LEGPRI *
!     ====================

      subroutine legpri
      use pumamod

      write(nud,231)
      write(nud,232)
      write(nud,233)
      write(nud,232)
      do 14 jlat = 1 , NLAT
         zalat = asin(sid(jlat))*180.0/PI
         write(nud,234) jlat,zalat,csq(jlat),gwd(jlat)
   14 continue
      write(nud,232)
      write(nud,231)
      return
  231 format(/)
  232 format(37('*'))
  233 format('* No *  Lat *     csq   weight *')
  234 format('*',i4,' *',f6.1,' *',2f10.4,' *')
      end


!     =================
!     SUBROUTINE INILAT
!     =================

      subroutine inilat
      use pumamod
      real (kind=8) :: zcsq

      do jlat = 1 , NLAT
         zcsq       = 1.0 - sid(jlat) * sid(jlat)
         csq(jlat)  = zcsq
         rcs(jlat)  = 1.0 / sqrt(zcsq)
      enddo
      do jlat = 1 , NLAT/2
         ideg = nint(180.0/PI * asin(sid(jlat)))
         write(chlat(jlat),'(i2,a1)') ideg,'N'
         write(chlat(NLAT+1-jlat),'(i2,a1)') ideg,'S'
      enddo
      return
      end


!     ====================
!     SUBROUTINE GRIDPOINT
!     ====================

      subroutine gridpoint
      use pumamod
```



```
      real gtn(NLON,NLPP,NLEV)
      real gvpp(NHOR)
      real gpmt(NLON,NLPP)
      real sdf(NESP,NLEV)
      real stf(NESP,NLEV)
      real szf(NESP,NLEV)
      real spf(NESP)
      real zgp(NLON,NLAT)
      real zgpp(NHOR)
      real (kind=4) :: zcs(NLAT,NLEV)
      real (kind=4) :: zsp(NRSP)

      do jlev = 1 , NLEV
         call sp2fc(sd(1,jlev),gd(1,jlev))
         call sp2fc(st(1,jlev),gt(1,jlev))
         call sp2fc(sz(1,jlev),gz(1,jlev))
      enddo

      call sp2fc(sp,gp)              ! LnPs
      call sp2fcdmu(sp,gpj)          ! d(lnps) / d(mu)
!     divergence, vorticity -> u*cos(phi), v*cos(phi)
      do jlev = 1 , NLEV
         call dv2uv(sd(1,jlev),sz(1,jlev),gu(1,jlev),gv(1,jlev))
      enddo
      if (lselect) then
         call filter_zonal_waves(gp)
         call filter_zonal_waves(gpj)
         do jlev = 1 , NLEV
            call filter_zonal_waves(gu(1,jlev))
            call filter_zonal_waves(gv(1,jlev))
            call filter_zonal_waves(gd(1,jlev))
            call filter_zonal_waves(gt(1,jlev))
            call filter_zonal_waves(gz(1,jlev))
         enddo
      endif

      if (ngui > 0 .or. mod(nstep,ndiag) == 0) then
        do jlev = 1 , NLEV
          do jlat = 1 , NLPP
            sec = cv / sqrt(csq(jlat))
            csu(jlat,jlev) = gu(1+(jlat-1)*NLON,jlev) * sec
            csv(jlat,jlev) = gv(1+(jlat-1)*NLON,jlev) * sec
            cst(jlat,jlev) =(gt(1+(jlat-1)*NLON,jlev) + t0(jlev))*ct-273.16
          enddo
        enddo
      endif

      do jlat = 1 , NLPP
         do jlon = 1 , NLON-1 , 2
           gpmt(jlon  ,jlat) = -gp(jlon+1+(jlat-1)*NLON) * ((jlon-1)/2)
           gpmt(jlon+1,jlat) =  gp(jlon  +(jlat-1)*NLON) * ((jlon-1)/2)
         end do
      end do

      call fc2gp(gu ,NLON,NLPP*NLEV)
      call fc2gp(gv ,NLON,NLPP*NLEV)
      call fc2gp(gt ,NLON,NLPP*NLEV)
      call fc2gp(gd ,NLON,NLPP*NLEV)
      call fc2gp(gz ,NLON,NLPP*NLEV)
      call fc2gp(gpj,NLON,NLPP)
      call fc2gp(gpmt,NLON,NLPP)

      call calcgp(gtn,gpmt,gvpp)

      gut(:,:) = gu(:,:) * gt(:,:)
      gvt(:,:) = gv(:,:) * gt(:,:)
      gke(:,:) = gu(:,:) * gu(:,:) + gv(:,:) * gv(:,:)

      call gp2fc(gtn ,NLON,NLPP*NLEV)
```

```
      call gp2fc(gut ,NLON,NLPP*NLEV)
      call gp2fc(gvt ,NLON,NLPP*NLEV)
      call gp2fc(gfv ,NLON,NLPP*NLEV)
      call gp2fc(gfu ,NLON,NLPP*NLEV)
      call gp2fc(gke ,NLON,NLPP*NLEV)
      call gp2fc(gvpp,NLON,NLPP     )

      call fc2sp(gvpp,spf)

      if (lselect) then
         call filter_zonal_waves(gvpp)
         do jlev = 1 , NLEV
            call filter_zonal_waves(gtn(1,1,jlev))
            call filter_zonal_waves(gut(1,jlev))
            call filter_zonal_waves(gvt(1,jlev))
            call filter_zonal_waves(gfv(1,jlev))
            call filter_zonal_waves(gfu(1,jlev))
            call filter_zonal_waves(gke(1,jlev))
         enddo
      endif

      do jlev = 1 , NLEV
         call mktend(sdf(1,jlev),stf(1,jlev),szf(1,jlev),gtn(1,1,jlev),&
         gfu(1,jlev),gfv(1,jlev),gke(1,jlev),gut(1,jlev),gvt(1,jlev))
      enddo

      if (nruido > 0) call stepruido
      call mpsumsc(spf,spt,1)
      call mpsumsc(stf,stt,NLEV)
      call mpsumsc(sdf,sdt,NLEV)
      call mpsumsc(szf,szt,NLEV)

      if (ngui > 0 .or. mod(nstep,ndiag) == 0) then
         call fc2gp(gp,NLON,NLPP)
         zgpp(:) = exp(gp)                  ! LnPs -> Ps
         call mpgagp(zgp,zgpp,1)            ! zgp = Ps (full grid)
         !XW(Mar/25/2017) to remove GUI:
         !if (ngui > 0) then
         !   call guips(zgp,psmean)
         !   call guigv("GU" // char(0),gu)
         !   call guigv("GV" // char(0),gv)
         !   call guigt(gt)
         !endif
         zgpp(:) =  zgpp(:) - 1.0           ! Mean(LnPs) = 0  <-> Mean(Ps) = 1
         call gp2fc(zgpp,NLON,NLPP)
         call fc2sp(zgpp,span)

         call mpsum(span,1)                 ! span = Ps spectral
         call mpgacs(csu)
         call mpgacs(csv)
         call mpgacs(cst)
         if (mypid == NROOT) then
            call altcs(csu)
            call altcs(csv)
            call altcs(cst)
            !XW(Mar/25/2017) to remove GUI:
            !if (ngui > 0) then
            !   zcs(:,:) = csu(:,:)
            !   call guiput("CSU"  // char(0) ,zcs ,NLAT,NLEV,1)
            !   zcs(:,:) = csv(:,:)
            !   call guiput("CSV"  // char(0) ,zcs ,NLAT,NLEV,1)
            !   zcs(:,:) = cst(:,:)
            !   call guiput("CST"  // char(0) ,zcs ,NLAT,NLEV,1)
            !   zsp(:) = span(1:NRSP)
            !   call guiput("SPAN" // char(0) ,zsp ,NCSP,-NTP1,1)
            !endif
         endif
      endif
      return
```



```
      end

!     =================
!     SUBROUTINE CALCGP
!     =================
      subroutine calcgp(gtn,gpm,gvp)

      use pumamod

!     Comments by Torben Kunz and Guido Schroeder

!     Compute nonlinear tendencies in grid point space.
!     Hoskins and Simmons 1975 (Q.J.R.Meteorol.Soc.,101,637-655) (HS75)

!     For terms calculated in this routine, see HS75, eqs. (8)-(10) and
!     appendix I:
!     - script Fu, Fv as contributions to script D: gl. arrays gfu, gfv
!     - script T: returned as gtn
!     - script P: returned as gvp


!     parameters (in)
!     ---------------

!     gpm --              d(ln(ps)) / d(lambda)

!     parameters (out)
!     ---------------

!     gtn -- temperature tendency
!     gvp -- vertical integral of (u,v) * grad(ln(ps))

!     global arrays variable in time
!     ------------------------------

!     gfu, gfv -- terms Fu, Fv in primitive equations,
!                 see HS75 (eqs. (1), (2))
!     gu, gv   -- components u, v of horizontal velocity vector
!     gd       -- divergence D
!     gz       -- absolute vorticity
!     gt       -- temperature deviation T'

!     global arrays constant in time
!     ------------------------------

!     t0d   -- reference temperature difference between two adjacent
!              full levels
!     tkp   -- reference temperature times kappa (global parameter AKAP)
!     rdsig -- 1 / (2 * dsigma)
!     rcsq  -- 1 / (1 - mu^2)

!     notations used in subsequent comments
!     -------------------------------------

!     aINTb(A)dsigma :<=> the integral of A over the interval [a,b]
!                         with respect to sigma

      real gtn(NHOR,NLEV)
      real gpm(NHOR)     , gvp(NHOR)
      real zsdotp(NHOR,NLEM),zsumd(NHOR),zsumvp(NHOR),zsumvpm(NHOR)
      real ztpta(NHOR),ztptb(NHOR)
      real zvgpg(NHOR,NLEV)
      real gtd(NHOR,NLEM)
      real gud(NHOR,NLEM)
      real gvd(NHOR,NLEM)

!     1.
!     1.1 zvgpg: (u,v) * grad(ln(ps))
```

```
      do jlev = 1 , NLEV
         zvgpg(:,jlev) = rcsq  * (gu(:,jlev)*gpm(:)+gv(:,jlev)*gpj(:))
      enddo

!     1.2 Calculate vertical integral of A = D + (u,v) * grad(ln(ps)),
!         separated into divergence and ln(ps) advection.
!         zsumd  : 0INT1(D)dsigma
!         gvp    : 0INT1[(u,v) * grad ln(ps)]dsigma
!         zsdotp : 0INTsigma(A)dsigma

      zsumd = dsigma(1) * gd(:,1)
      gvp   = dsigma(1) * zvgpg(:,1)
      zsdotp(:,1) = zsumd + gvp

      do jlev = 2 , NLEM
         zsumd = zsumd + dsigma(jlev) * gd(:,jlev)
         gvp   = gvp   + dsigma(jlev) * zvgpg(:,jlev)
         zsdotp(:,jlev) = zsumd + gvp
      enddo

      zsumd = zsumd + dsigma(NLEV) * gd(:,NLEV)
      gvp   = gvp   + dsigma(NLEV) * zvgpg(:,NLEV)

!     2. Calculate vertical velocity and vertical advection terms
!        on half levels.

      do jlev = 1 , NLEM
         zsdotp(:,jlev) = (sigmh(jlev) * (zsumd+gvp) - zsdotp(:,jlev))
      enddo

      gtd(:,:) = zsdotp(:,:) * (gt(:,2:NLEV) - gt(:,1:NLEM))
      gud(:,:) = zsdotp(:,:) * (gu(:,2:NLEV) - gu(:,1:NLEM))
      gvd(:,:) = zsdotp(:,:) * (gv(:,2:NLEV) - gv(:,1:NLEM))

!     3. Calculate nonlinear contributions to temperature tendency and
!        nonlinear terms Fu, Fv as used in vorticity and
!        divergence equation.

!     3.1 top level:

!     3.1.1 zsumvp: 0INTsigma[(u,v) * grad(ln(ps))]dsigma

      zsumvp = zvgpg(:,1) * dsigma(1)

!     3.1.2 Calculation of gtn, gfv and gfu as for inner levels (3.2),
!           but somewhat simplified:
!           a) For the top level the following equation holds in the
!              discretized form: (1/sigma)*0INTsigma(A)dsigma == A
!              (HS75, second equation following eq. (7)). Therefore,
!              (3.2.3) simplifies to -kappa*T' * D and (3.2.4) vanishes.
!           b) Vertical advection terms (gtd, gud, gvd (see section 2)
!              and vertical T0 advection (3.2.6)) vanish at upper
!              boundary (sigma == 0).

      gtn(:,1) = (1.0-akap) * gt(:,1) * gd(:,1) - rdsig(1) * (gtd(:,1) &
               + t0d(1) * (sigmh(1)*gvp-zsumvp))

      gfv(:,1) = - gu(:,1)*gz(:,1) - gpj(:)*gt(:,1) - rdsig(1)*gvd(:,1)
      gfu(:,1) =   gv(:,1)*gz(:,1) - gpm(:)*gt(:,1) - rdsig(1)*gud(:,1)

!     3.2 inner levels:

      do jlev = 2 , NLEM

!        3.2.1 ztpta: (1/sigma)*0INTsigma(A-D)dsigma
!              ztptb: (1/sigma)*0INTsigma(A)dsigma
!              Matrix c contains factors for discretized integration, see
!              HS75 (second equation following eq. (7)).
```



```
         ztpta = c(1,jlev) *  zvgpg(:,1)
         ztptb = c(1,jlev) * (zvgpg(:,1) + gd(:,1))

         do jlej = 2 , jlev
            ztpta = ztpta + c(jlej,jlev) *  zvgpg(:,jlej)
            ztptb = ztptb + c(jlej,jlev) * (zvgpg(:,jlej) + gd(:,jlej))
         enddo

         zsumvpm = zsumvp
         zsumvp  = zsumvp + zvgpg(:,jlev) * dsigma(jlev)

!        3.2.2 D * T'

         gtn(:,jlev) = gt(:,jlev) * gd(:,jlev)

!        3.2.3 kappa*T' *
!                    [(u,v)*grad(ln(ps)) - (1/sigma)*0INTsigma(A)dsigma]

         gtn(:,jlev) = gtn(:,jlev)                                        &
     &       + akap * gt(:,jlev) * (zvgpg(:,jlev) - ztptb)

!        3.2.4 kappa*T0 *
!                  [(u,v)*grad(ln(ps)) - (1/sigma)*0INTsigma(A-D)dsigma]

         gtn(:,jlev) = gtn(:,jlev)                                        &
     &       + tkp(jlev) * (zvgpg(:,jlev) - ztpta)

!        3.2.5 Calculate vertical T' advection on full levels by
!              averaging two half level advection terms (gtd, calculated
!              in section 2).

!        and

!        3.2.6 Calculate vertical T0 advection on full levels by
!              averaging two half level advection terms.

         gtn(:,jlev) = gtn(:,jlev)                                        &
     &       - rdsig(jlev) * (gtd(:,jlev) + gtd(:,jlev-1))                &
     &         +(sigmh(jlev)   * gvp - zsumvp)  * t0d(jlev)              &
     &         +(sigmh(jlev-1) * gvp - zsumvpm) * t0d(jlev-1))

!        3.2.7 terms Fv, Fu, see HS75 (equations following eq. (5));
!              vertical advection terms interpolated to full levels by
!              averaging two half level advection terms.

         gfv(:,jlev) = - gu(:,jlev)*gz(:,jlev) - gpj(:)*gt(:,jlev)       &
     &                 - rdsig(jlev)*(gvd(:,jlev) + gvd(:,jlev-1))

         gfu(:,jlev) =   gv(:,jlev)*gz(:,jlev) - gpm(:)*gt(:,jlev)       &
     &                 - rdsig(jlev)*(gud(:,jlev) + gud(:,jlev-1))
      enddo

!     3.3 bottom level

!     3.3.1 ztpta, ztptb: see 3.2.1

      ztpta = c(1,NLEV) *  zvgpg(:,1)
      ztptb = c(1,NLEV) * (zvgpg(:,1) + gd(:,1))

      do jlej = 2 , NLEV
         ztpta = ztpta + c(jlej,NLEV) *  zvgpg(:,jlej)
         ztptb = ztptb + c(jlej,NLEV) * (zvgpg(:,jlej) + gd(:,jlej))
      enddo

!     3.3.2 Calculation of gtn, gfv and gfu as for inner levels (3.2),
!           but somewhat simplified:
!           Vertical advection terms (gtd, gud, gvd (see section 2) and
!           vertical T0 advection (3.2.6)) vanish at
!           lower boundary (sigma == 1).
```

```
      gtn(:,NLEV) = gt(:,NLEV) * gd(:,NLEV)                              &
     &            + akap*gt(:,NLEV)*(zvgpg(:,NLEV)-ztptb)                &
     &            + tkp(NLEV)*(zvgpg(:,NLEV)-ztpta)                      &
     &            - rdsig(NLEV) * (gtd(:,NLEM)                           &
     &            + t0d(NLEM)*(sigmh(NLEM)*gvp-zsumvp))

      gfv(:,NLEV) = -gu(:,NLEV) * gz(:,NLEV) - gpj(:) * gt(:,NLEV)       &
     &              - rdsig(NLEV) * gvd(:,NLEM)
      gfu(:,NLEV) =  gv(:,NLEV) * gz(:,NLEV) - gpm(:) * gt(:,NLEV)       &
     &              - rdsig(NLEV) * gud(:,NLEM)

!     3.3.3 Add gaussian noise to T (controlled by nruido)

      if (nruido > 0) gtn(:,:) = gtn(:,:) + ruidop(:,:)

      return
      end

!     ====================
!     SUBROUTINE SPECTRAL
!     ====================

      subroutine spectral
      use pumamod

!*    Add adiabatic and diabatic tendencies - perform leapfrog

!     The adiabatic tendencies are added using the semi implicit scheme
!     Hoskins and Simmons 1975 (Q.J.R.Meteorol.Soc.,101,637-655) (HS75)
!     To compare the code directly with HS75 the following notes might
!     be helpful (in addition to the comments below):

!     Name rule for global arrays <abc>:
!     a : representation (s=spectral, g=grid, z=local)
!     b : variable (p=ln(ps), d=divergence, z=vorticity, t=temperature)
!     c : modifier (m=previous timestep, p=present timestep, t=tendency)

!     global arrays variable in time
!     ------------------------------

!     spt - pressure    tendency HS75 (10)
!     sdt - divergence  tendency HS75 ( 8)
!     szt - vorticity   tendency
!     stt - temperature tendency HS75 ( 9)

!     spm - pressure    at previous timestep
!     sdm - divergence  at previous timestep
!     szm - vorticity   at previous timestep
!     stm - temperature at previous timestep

!     spp - pressure    at present timestep
!     sdp - divergence  at present timestep
!     szp - vorticity   at present timestep
!     stp - temperature at present timestep

!     global arrays constant in time
!     ------------------------------

!     sak(NSPP)      -     = hyper diffusion
!     sop(NSPP)      - g*  = orography as geopotential
!     srp1(NSPP,NLEV) - Tr = radiative equilibrium temperature (annual mean)
!     srp2(NSPP,NLEV) - Tr = radiative equilibrium temperature (annual cycle)
!     nindex(NSPP)   - n   = total wavenumber n for spectral modes
!     srcn(NSPP)   - 1/Cn  = 1.0 / (n * (n+1))
!     damp(NLEV)  1/tau R  = time constant for newtonian cooling
!     fric(NLEV)  1/tau F  = time constant for Rayleigh friction

      real zpm(NSPP)      ! new spm
```



```
      real zdm(NSPP,NLEV) ! new sdm
      real zzm(NSPP,NLEV) ! new szm
      real ztm(NSPP,NLEV) ! new stm
      real zwp(NSPP)      ! timefilter delta pm
      real zwd(NSPP,NLEV) ! timefilter delta sd
      real zwz(NSPP,NLEV) ! timefilter delta sz
      real zwt(NSPP,NLEV) ! timefilter delta st
      real zsrp(NSPP)     ! restoring temperature (mean + annual cycle)

      real zgt(NSPP,NLEV) ! work array

      real,allocatable :: zstte(:,:,:) ! temp. tendencies for energy diag.
      real,allocatable :: zszte(:,:,:) ! vort. tendencies for energy recycling
      real,allocatable :: zsdte(:,:,:) ! div. tendencies for energy recycling
      real,allocatable :: zdps(:)      ! surf pressure for energy diag
      real,allocatable :: zsp(:)       ! surf pressure spectral
      real,allocatable :: zspf(:)      ! surf pressure spectral
      real,allocatable :: zspt(:)      ! surf pressure tendency

      real,allocatable :: zst(:,:)     ! temperature for entropy diagnostics
      real,allocatable :: zstt(:,:)    ! tem. tendencies for entropy diag.
      real,allocatable :: ztgp(:,:)    !
      real,allocatable :: zdtgp(:,:)   !
      real,allocatable :: zsum1(:)
      real,allocatable :: zgw(:)

!     0. Special code for experiments with mode filtering

      if (lspecsel) call filter_spectral_modes

!     1. Initialize local arrays

      zpm(:)   = spp(:)
      zdm(:,:) = sdp(:,:)
      zzm(:,:) = szp(:,:)
      ztm(:,:) = stp(:,:)
!
!     allocate diagnostic arrays if needed
!
      if(nenergy > 0 .or. nentropy > 0 .or. ndheat > 0) then
       allocate(zstte(NSPP,NLEV,3))
      endif
      if(ndheat > 0) then
       allocate(zszte(NSPP,NLEV,2))
       allocate(zsdte(NSPP,NLEV,2))
      endif
!
!     allocate and compute surface pressure if needed
!
      if(nenergy > 0 .or. nentropy > 0 .or. ndheat > 0) then
       allocate(zspt(NSPP))
       allocate(zsp(NSPP))
      endif

!     2. Calculate divergence on timelevel t (sdt) HS75 (17)
!        which will replace the divergence tendency sdt
!        (semi implicit scheme)

!        The vertical scheme has being changed to the ECMWF scheme
!        (see e.g. Simmons and Burridge 1981, Mon.Wea.Rev.,109,758-766).
!        in this scheme, matrix xlphi (g) differs from that in HS75.

!         z0 : reference temperature To
!         zq : 1.0 / Cn
!         zt : xlphi * script T - To * script P
!         zm : xlphi * T        + To * ln(Ps)(t-dt)

!         (note that phi is needed in HS75 (17) and, therefore,
!         the surface geopotential phi* [sop] is added
```

```
      do jlev=1,NLEV
         z0 = t0(jlev)
         do jsp=1,NSPP
            zq = srcn(jsp)                    ! 1.0 / (n * (n + 1))
            zt = dot_product(xlphi(:,jlev),stt(jsp,:)) - z0 * spt(jsp)
            zm = dot_product(xlphi(:,jlev),stm(jsp,:)) + z0 * spm(jsp)
            za = sdt(jsp,jlev) * zq
            zb = sdm(jsp,jlev) * zq
            zgt(jsp,jlev) = zb + delt * (za + zm + sop(jsp) + zt * delt)
         enddo
      enddo

!     bm1 is the invers of matrix (1/cn I+B dt**2) (lhs HS75 (17))

      do jlev = 1 , NLEV
         do jsp = 1 , NSPP
            jn = nindex(jsp)              ! total wavenumber n
            sdt(jsp,jlev) = dot_product(zgt(jsp,:),bm1(:,jlev,jn))
         enddo
      enddo

!     3. Calculate surface pressure tendency -ln(ps) HS75 (15)

      do jlev = 1 , NLEV
         spt(:) = spt(:) + dsigma(jlev) * sdt(:,jlev)
      enddo

!     4. Calculate temperature tendency   HS75 (14)

      do jlev = 1 , NLEV
       do jsp = 1 , NSPP
        stt(jsp,jlev)=stt(jsp,jlev)-dot_product(xlt(:,jlev),sdt(jsp,:))
       enddo
      enddo

!     5. Add tendencies

      spp(:)   = spm(:) - delt2 * spt(:)      ! spt = -ln(ps) tendency
      sdp(:,:) =   2.0 * sdt(:,:) - sdm(:,:) ! sdt = sdm + delt * tend.
      szp(:,:) = delt2 * szt(:,:) + szm(:,:) ! vorticity
      stp(:,:) = delt2 * stt(:,:) + stm(:,:) ! temperature

      if(nenergy > 0) then
       zspt(:)=-spt(:)
       call mkenerdiag(stm,stt,spm,zspt,denergy(:,1))
      endif
      if(nentropy > 0) then
       call mkentrodiag(stm,stt,spm,dentropy(:,1))
      endif

!     6. Calculate newtonian cooling, friction and biharmonic diffusion
!        (srp - stp) * damp = (Tr' -T') / tau R = newtonian cooling
!        srp1 = annual mean  component
!        srp2 = annual cycle component
!        sak  = diffusion
!        fric = friction
!        zampl = annual cycle

      zampl = cos((real(nstep)-pac)*tac)

      if (nhelsua == 0 .or. nhelsua == 1) then
         do jlev=1,NLEV
            zsrp(:)=srp1(:,jlev)+srp2(:,jlev)*zampl
            sdt(:,jlev) =  sdp(:,jlev) * (sak(1:NSPP) - fric(jlev))
            szt(:,jlev) =  szp(:,jlev) * (sak(1:NSPP) - fric(jlev))
            stt(:,jlev) = (zsrp(:) - stp(:,jlev)) * damp(jlev)          &
     &                                            + stp(:,jlev)  * sak(1:NSPP)
            if(nenergy > 0) then
```



```
             zstte(:,jlev,2)=(zsrp(:)-stp(:,jlev))*damp(jlev)
             zstte(:,jlev,3)=stp(:,jlev)*sak(1:NSPP)
            endif
            if(ndheat > 0) then
             zsdte(:,jlev,1) =  -sdp(:,jlev) * fric(jlev)
             zszte(:,jlev,1) =  -szp(:,jlev) * fric(jlev)
             zsdte(:,jlev,2) =  sdp(:,jlev) * sak(1:NSPP)
             zszte(:,jlev,2) =  szp(:,jlev) * sak(1:NSPP)
            endif
         enddo
      elseif (nhelsua == 2 .or. nhelsua == 3 .or. ndiagp > 0) then
         if (ndiagp == 0) then
            call heatgp(zampl)  ! stt(:,:) = Newtonian cooling
         else
            call diagp(zampl)  ! stt(:,:) = Newtonian cooling
         endif
         if(nenergy > 0) then
          zstte(:,:,2)=stt(:,:)
         endif
         do jlev=1,NLEV
            sdt(:,jlev) = sdp(:,jlev) * (sak(1:NSPP) - fric(jlev))
            szt(:,jlev) = szp(:,jlev) * (sak(1:NSPP) - fric(jlev))
            stt(:,jlev) = stt(:,jlev) + stp(:,jlev) * sak(1:NSPP)
            if(nenergy > 0) then
             zstte(:,jlev,3)=stp(:,jlev)*sak(1:NSPP)
            endif
            if(ndheat > 0) then
             zsdte(:,jlev,1) =  -sdp(:,jlev) * fric(jlev)
             zszte(:,jlev,1) =  -szp(:,jlev) * fric(jlev)
             zsdte(:,jlev,2) =  sdp(:,jlev) * sak(1:NSPP)
             zszte(:,jlev,2) =  szp(:,jlev) * sak(1:NSPP)
            endif
         enddo
      endif

!        Conserve ln(ps) by forcing mode(0,0) to zero
!        Correct vorticity by canceling the friction and diffusion
!        applied to planetary vorticity
!        Only root node processes the first NSPP modes

      if(nenergy > 0) then
       zspt(:)=0.
       call mkenerdiag(stp,zstte(:,:,2),spp,zspt,denergy(:,2))
       call mkenerdiag(stp,zstte(:,:,3),spp,zspt,denergy(:,3))
      endif
      if(nentropy > 0) then
       call mkentrodiag(stp,zstte(:,:,2),spp,dentropy(:,2))
       call mkentrodiag(stp,zstte(:,:,3),spp,dentropy(:,3))
      endif
      if(nenergy > 0 .or. nentropy > 0 .or. ndheat > 0) then
       zsp(:)=spp(:)
       zstte(:,:,1)=stt(:,:)
      endif

      if (mypid == NROOT) then
         spp(1) = 0.0
         spp(2) = 0.0
         szt(3,:) = szt(3,:) + plavor * (fric(:) - sak(3))
         if(ndheat > 0) then
          zszte(3,:,1) = zszte(3,:,1) + plavor * fric(:)
          zszte(3,:,2) = zszte(3,:,2) - plavor * sak(3)
         endif
      endif
!
!     6b) call for vertical diffusion
!

      if(dvdiff > 0.) call vdiff(stp,szp,sdp,stt,szt,sdt)
```

```
!
!     recycle kin energy dissipation
!

      if(ndheat > 0) then
       call mkdheat(zszte(:,:,1),zszte(:,:,2)                          &
     &             ,zsdte(:,:,1),zsdte(:,:,2),zsp)
      endif


      if(nenergy > 0 .or. nentropy > 0) then
       zstte(:,:,1)=stt(:,:)-zstte(:,:,1)
      endif
      if(nenergy > 0) then
       call mkenerdiag(stp,zstte(:,:,1),zsp,zspt,denergy(:,4))
      endif
      if(nentropy > 0) then
       zstte(:,:,1)=stt(:,:)-zstte(:,:,1)
       call mkentrodiag(stp,zstte(:,:,1),zsp,dentropy(:,4))
      endif
      if(nenergy > 0 .or. nentropy > 0) then
       zstte(:,:,1)=0.
       zspt(:)=(spp(:)-zsp(:))/delt2
      endif
      if(nenergy > 0) then
       call mkenerdiag(stp,zstte(:,:,1),zsp,zspt,denergy(:,8))
      endif
      if(nentropy > 0) then
       call mkentrodiag(stp,zstte(:,:,1),zsp,dentropy(:,8))
      endif

!
!     diagnostics of efficiency
!

      if(ndheat > 1) then
       zcp=gascon/akap
       allocate(zst(NESP,NLEV))
       allocate(zstt(NESP,NLEV))
       allocate(zspf(NESP))
       allocate(ztgp(NHOR,NLEV))
       allocate(zdtgp(NHOR,NLEV))
       allocate(zdps(NHOR))
       allocate(zsum1(4))
       allocate(zgw(NHOR))
       jhor=0
       do jlat=1,NHPP
        do jlon=1,NLON*2
         jhor=jhor+1
         zgw(jhor)=gwd(jlat)
        enddo
       enddo
       call mpgallsp(zst,stp,NLEV)
       call mpgallsp(zstt,stt,NLEV)
       call mpgallsp(zspf,zsp,1)
       do jlev = 1 , NLEV
          call sp2fc(zst(1,jlev),ztgp(1,jlev))
          call sp2fc(zstt(1,jlev),zdtgp(1,jlev))
       enddo
       call sp2fc(zspf,zdps)
       call fc2gp(ztgp,NLON,NLPP*NLEV)
       call fc2gp(zdtgp,NLON,NLPP*NLEV)
       call fc2gp(zdps,NLON,NLPP)
       zdps(:)=psurf*exp(zdps(:))
       zsum1(:)=0.
       do jlev=1,NLEV
        ztgp(:,jlev)=ct*(ztgp(:,jlev)+t0(jlev))
        zdtgp(:,jlev)=ct*ww_scale*zdtgp(:,jlev)
        zsum1(1)=zsum1(1)+SUM(zdtgp(:,jlev)*zgw(:)                     &
```



```
     &                        *zcp*zdps(:)/ga*dsigma(jlev)             &
     &                        ,mask=(zdtgp(:,jlev) >= 0.))
        zsum1(2)=zsum1(2)+SUM(zdtgp(:,jlev)*zgw(:)                     &
     &                        *zcp*zdps(:)/ga*dsigma(jlev)             &
     &                        ,mask=(zdtgp(:,jlev) < 0.))
        zsum1(3)=zsum1(3)+SUM(zdtgp(:,jlev)/ztgp(:,jlev)*zgw(:)        &
     &                        *zcp*zdps(:)/ga*dsigma(jlev)             &
     &                        ,mask=(zdtgp(:,jlev) >= 0.))
        zsum1(4)=zsum1(4)+SUM(zdtgp(:,jlev)/ztgp(:,jlev)*zgw(:)        &
     &                        *zcp*zdps(:)/ga*dsigma(jlev)             &
     &                        ,mask=(zdtgp(:,jlev) < 0.))
       enddo
       zsum3=SUM(zgw(:))
       call mpsumbcr(zsum1,4)
       call mpsumbcr(zsum3,1)
       zsum1(:)=zsum1(:)/zsum3
       if(mypid == NROOT) then
        ztp=zsum1(1)/zsum1(3)
        zztm=zsum1(2)/zsum1(4)
        write(9,*) zsum1(:),zsum1(1)/zsum1(3),zsum1(2)/zsum1(4)        &
     &            ,(ztp-zztm)/ztp
       endif
       deallocate(zst)
       deallocate(zstt)
       deallocate(zspf)
       deallocate(ztgp)
       deallocate(zdps)
       deallocate(zdtgp)
       deallocate(zsum1)
       deallocate(zgw)
      endif

!     7. Add newtonian cooling, friction and diffusion tendencies

      sdp(:,:) = sdp(:,:) + delt2 * sdt(:,:)
      szp(:,:) = szp(:,:) + delt2 * szt(:,:)
      stp(:,:) = stp(:,:) + delt2 * stt(:,:)

!     11. Coupling for synchronization runs

      if (mrnum == 2 .and. nsync > 0) then
         call mrdiff(stp,std,NESP,NLEV)
         call mrdiff(sdp,sdd,NESP,NLEV)
         call mrdiff(szp,szd,NESP,NLEV)
         call mrdiff(spp,spd,NESP,   1)
         stp(:,:) = stp(:,:) + syncstr * std(:,:)
         sdp(:,:) = sdp(:,:) + syncstr * sdd(:,:)
         szp(:,:) = szp(:,:) + syncstr * szd(:,:)
         spp(:  ) = spp(:  ) + syncstr * spd(:  )

      endif

!     8. Apply Robert Asselin time filter (not for short initial timesteps)
!        d(t) = pnu * f(t-1) + pnu * f(t+1) - 2 * pnu * f(t)

      if (nkits == 0) then
         zwp(:)   = pnu * (spm(:)   + spp(:)   - 2.0 * zpm(:)  )
         zwd(:,:) = pnu * (sdm(:,:) + sdp(:,:) - 2.0 * zdm(:,:))
         zwz(:,:) = pnu * (szm(:,:) + szp(:,:) - 2.0 * zzm(:,:))
         zwt(:,:) = pnu * (stm(:,:) + stp(:,:) - 2.0 * ztm(:,:))

!        Add Robert-Asselin-Williams filter value to f(t)

         spm(:)   = zpm(:)   + alpha * zwp(:)
         sdm(:,:) = zdm(:,:) + alpha * zwd(:,:)
         szm(:,:) = zzm(:,:) + alpha * zwz(:,:)
         stm(:,:) = ztm(:,:) + alpha * zwt(:,:)

!        Add filter value to f(t+1)
```

```
         spp(:)   = spp(:)   - (1.0 - alpha) * zwp(:)
         sdp(:,:) = sdp(:,:) - (1.0 - alpha) * zwd(:,:)
         szp(:,:) = szp(:,:) - (1.0 - alpha) * zwz(:,:)
         stp(:,:) = stp(:,:) - (1.0 - alpha) * zwt(:,:)
      endif

      if (nenergy > 0 .or. nentropy > 0) then
       zstte(:,:,1)=(stm(:,:)-ztm(:,:))/delt2
       zspt(:)=(spm(:)-zpm(:))/delt2
      endif
      if(nenergy > 0) then
       call mkenerdiag(ztm,zstte(:,:,1),zpm,zspt,denergy(:,9))
      endif
      if (nentropy > 0) then
       call mkentrodiag(ztm,zstte(:,:,1),zpm,dentropy(:,9))
      endif

!     9. Save spectral arrays for extended output

      if (nextout == 1 .and. mypid == NROOT) then
         if (mod(nstep,nafter) == nafter - 2) then
            if (.not. allocated(st2)) allocate(st2(nesp,nlev))
            st2(:,:) = st(:,:)
            if (.not. allocated(sp2)) allocate(sp2(nesp))
            sp2(:) = sp(:)
         endif
         if (mod(nstep,nafter) == nafter - 1) then
            if (.not. allocated(st1)) allocate(st1(nesp,nlev))
            st1(:,:) = st(:,:)
            if (.not. allocated(sp1)) allocate(sp1(nesp))
            sp1(:) = sp(:)
         endif
      endif

!     10. Gather spectral modes from all processes

      call mpgallsp(sp,spp,   1)
      call mpgallsp(sd,sdp,NLEV)
      call mpgallsp(sz,szp,NLEV)
      call mpgallsp(st,stp,NLEV)

      if(nenergy > 0 .or. nentropy > 0) then
       deallocate(zstte)
      endif
      if(ndheat > 0) then
       deallocate(zszte)
       deallocate(zsdte)
      endif
      if(nenergy > 0 .or. nentropy > 0 .or. ndheat > 0) then
       deallocate(zsp)
       deallocate(zspt)
      endif

      return
      end

      subroutine mrcheck(f)
      use pumamod
      real :: f(*)
      write (nud,'(/,i3,8f8.4)')  0,f(1:16:2)
      write (nud,'( i3,8f8.4)')   8,f(17:32:2)
      write (nud,'( i3,8f8.4)')  16,f(33:48:2)
      write (nud,'( i3,8f8.4)')  24,f(49:64:2)
      write (nud,'( i3,8f8.4)')  32,f(65:80:2)
      return
      end
```



```
!     ================
!     SUBROUTINE DIAGP
!     ================

      subroutine diagp(zampl)
      use pumamod

      real :: zstf(NESP,NLEV)
      real :: zgr12(NHOR,NLEV)
      real :: zgtt(NHOR,NLEV)
      real :: gr12(NHOR,NLEV)
      real :: gr12c(NHOR,NLEV)


      real :: gdtmp(NHOR)

      real :: zampl

      !--- transform temperature and divergence to grid point space
      call mpgallsp(st,stp,NLEV)
      if (nconv > 0) then
         call mpgallsp(sd,sdp,NLEV)
      endif
      do jlev=1,NLEV
         call sp2fc(st(1,jlev)   ,gt(1,jlev)   )
         if (nconv > 0) then
            call sp2fc(sd(1,jlev)   ,gd(1,jlev)   )
         endif
      enddo
      call fc2gp(gt   ,NLON,NLPP*NLEV)
      if (nconv > 0) then
         call fc2gp(gd   ,NLON,NLPP*NLEV)
      endif


      !--- radiative temperature tendencies
      gr12(:,:) = gr1(:,:) + gr2(:,:)*zampl
      zgtt(:,:) = (gr12(:,:) - gt(:,:))*gtdamp(:,:)

      !--- add convective temperature tendencies
      if (nconv > 0) then
         gdtmp(:) = gd(:,nlev)
         do jlev = 1,nlev
            where (gdtmp < 0.0)
               gr12c(:,jlev) = gr1c(:,jlev) + gr2c(:,jlev)*zampl
               zgtt(:,jlev)  = zgtt(:,jlev) + (gr12c(:,jlev) - gt(:,jlev))*gtdam
pc(:,jlev)
            endwhere
         enddo
      endif

      !--- transform temperature tendencies to spectral space
      call gp2fc(zgtt ,NLON,NLPP*NLEV)
      do jlev=1,NLEV
         call fc2sp(zgtt(1,jlev),zstf(1,jlev))
      enddo
      call mpsumsc(zstf,stt,NLEV)

      return
      end subroutine diagp

!     =================
!     SUBROUTINE HEATGP
!     =================

      subroutine heatgp(zampl)
      use pumamod

      real :: zsr12(NESP,NLEV)
```

```
      real :: zsrp12(NSPP,NLEV)
      real :: zstf(NESP,NLEV)
      real :: zgr12(NHOR,NLEV)
      real :: zgtt(NHOR,NLEV)

      real :: zampl

      zsrp12(:,:)=srp1(:,:)+srp2(:,:)*zampl
      call mpgallsp(zsr12,zsrp12,NLEV)
      call mpgallsp(st,stp,NLEV)
      do jlev=1,NLEV
         call sp2fc(zsr12(1,jlev),zgr12(1,jlev))
         call sp2fc(st(1,jlev)   ,gt(1,jlev)   )
      enddo
      call fc2gp(zgr12,NLON,NLPP*NLEV)
      call fc2gp(gt   ,NLON,NLPP*NLEV)

!     Newtonian cooling

      zgtt(:,:) = (zgr12(:,:) - gt(:,:)) * gtdamp(:,:)

      call gp2fc(zgtt ,NLON,NLPP*NLEV)
      do jlev=1,NLEV
         call fc2sp(zgtt(1,jlev),zstf(1,jlev))
      enddo
      call mpsumsc(zstf,stt,NLEV)

      return
      end

!     ================
!     SUBROUTINE VDIFF
!     ================

      subroutine vdiff(pt,pz,pd,ptt,pzt,pdt)
      use pumamod
!
      parameter(ztref=250.)
      real pt(NSPP,NLEV),pz(NSPP,NLEV),pd(NSPP,NLEV)
      real ptt(NSPP,NLEV),pzt(NSPP,NLEV),pdt(NSPP,NLEV)
      real ztn(NSPP,NLEV),zzn(NSPP,NLEV),zdn(NSPP,NLEV)
      real zebs(NLEM)
      real zskap(NLEV),zskaph(NLEV)
      real zkdiff(NLEM)
!
      zdelt=delt2/ww_scale
      zkonst1=ga*zdelt/gascon
      zkonst2=zkonst1*ga/gascon
!
      zskap(:)=sigma(:)**akap
      zskaph(:)=sigmh(:)**akap
!
!     1) modified diffusion coefficents
!
      do jlev=1,NLEM
       jlp=jlev+1
       zkdiff(jlev)=zkonst2*sigmh(jlev)*sigmh(jlev)/(ztref*ztref)     &
     &              *dvdiff/(sigma(jlp)-sigma(jlev))
      enddo
!
!     2. semi implicit scheme
!
!     2a momentum
!
!     top layer elimination
!
      zebs(1)=zkdiff(1)/(dsigma(1)+zkdiff(1))
      zdn(:,1)=dsigma(1)*pd(:,1)/(dsigma(1)+zkdiff(1))
      zzn(:,1)=dsigma(1)*pz(:,1)/(dsigma(1)+zkdiff(1))
```



```
!
!     middle layer elimination
!
      do jlev=2,NLEM
       jlem=jlev-1
       zebs(jlev)=zkdiff(jlev)/(dsigma(jlev)+zkdiff(jlev)               &
     &           +zkdiff(jlem)*(1.-zebs(jlem)))
       zdn(:,jlev)=(pd(:,jlev)*dsigma(jlev)+zkdiff(jlem)*zdn(:,jlem))   &
     &            /(dsigma(jlev)+zkdiff(jlev)                           &
     &            +zkdiff(jlem)*(1.-zebs(jlem)))
       zzn(:,jlev)=(pz(:,jlev)*dsigma(jlev)+zkdiff(jlem)*zzn(:,jlem))   &
     &            /(dsigma(jlev)+zkdiff(jlev)                           &
     &            +zkdiff(jlem)*(1.-zebs(jlem)))
      enddo
!
!     bottom layer elimination
!
      zdn(:,NLEV)=(pd(:,NLEV)*dsigma(NLEV)+zkdiff(NLEM)*zdn(:,NLEM))    &
     &           /(dsigma(NLEV)+zkdiff(NLEM)*(1.-zebs(NLEM)))
      zzn(:,NLEV)=(pz(:,NLEV)*dsigma(NLEV)+zkdiff(NLEM)*zzn(:,NLEM))    &
     &           /(dsigma(NLEV)+zkdiff(NLEM)*(1.-zebs(NLEM)))
!
!     back-substitution
!
      do jlev=NLEM,1,-1
       jlep=jlev+1
       zdn(:,jlev)=zdn(:,jlev)+zebs(jlev)*zdn(:,jlep)
       zzn(:,jlev)=zzn(:,jlev)+zebs(jlev)*zzn(:,jlep)
      enddo
!
!     tendencies
!
      pdt(:,1:NLEV)=pdt(:,1:NLEV)+(zdn(:,1:NLEV)-pd(:,1:NLEV))/delt2
      pzt(:,1:NLEV)=pzt(:,1:NLEV)+(zzn(:,1:NLEV)-pz(:,1:NLEV))/delt2
!
!     2c potential temperature
!
      do jlev=1,NLEM
       zkdiff(jlev)=zkdiff(jlev)*zskaph(jlev)
      enddo
!
!     semi implicit scheme
!
!     top layer elimination
!
      zebs(1)=zkdiff(1)/(dsigma(1)+zkdiff(1)/zskap(1))
      ztn(:,1)=dsigma(1)*pt(:,1)/(dsigma(1)+zkdiff(1)/zskap(1))
!
!     middle layer elimination
!
      do jlev=2,NLEM
       jlem=jlev-1
       zebs(jlev)=zkdiff(jlev)/(dsigma(jlev)+(zkdiff(jlev)              &
     &           +zkdiff(jlem)*(1.-zebs(jlem)/zskap(jlem)))/zskap(jlev))
       ztn(:,jlev)=(pt(:,jlev)*dsigma(jlev)                             &
     &             +zkdiff(jlem)/zskap(jlem)*ztn(:,jlem))               &
     &            /(dsigma(jlev)+(zkdiff(jlev)                          &
     &             +zkdiff(jlem)*(1.-zebs(jlem)/zskap(jlem)))           &
     &             /zskap(jlev))
      enddo
!
!     bottom layer elimination
!
      ztn(:,NLEV)=(pt(:,NLEV)*dsigma(NLEV)                              &
     &            +zkdiff(NLEM)*ztn(:,NLEM)/zskap(NLEM))                &
     &           /(dsigma(NLEV)+zkdiff(NLEM)/zskap(NLEV)                &
     &                         *(1.-zebs(NLEM)/zskap(NLEM)))
!
!     back-substitution
```

```
!
      do jlev=NLEM,1,-1
       jlep=jlev+1
       ztn(:,jlev)=ztn(:,jlev)+zebs(jlev)*ztn(:,jlep)/zskap(jlep)
      enddo
!
!     tendencies
!
      ptt(:,1:NLEV)=ptt(:,1:NLEV)+(ztn(:,1:NLEV)-pt(:,1:NLEV))/delt2
!
      return
      end subroutine vdiff

!     =================
!     SUBROUTINE GASDEV
!     =================

!     Gaussian noise generator with zero mean and unit variance.

      real function gasdev()
      use pumamod
      implicit none
      real :: fr, vx, vy, ra

      if (ganext == 0.0) then
         ra = 2.0
         do while (ra >= 1.0 .or. ra < 1.0e-20)
            call random_number(vx)
            call random_number(vy)
            vx = 2.0 * vx - 1.0
            vy = 2.0 * vy - 1.0
            ra = vx * vx + vy * vy
         enddo
         fr = sqrt(-2.0 * log(ra) / ra)
         gasdev = vx * fr
         ganext = vy * fr
      else
         gasdev = ganext
         ganext = 0.0
      endif

      return
      end

!     =================
!     SUBROUTINE SPONGE
!     =================

      subroutine sponge
      use pumamod

      real :: zp

!     This introduces a simple sponge layer to the highest model levels
!     by applying Rayleigh friction there, according to
!     Polvani & Kushner (2002, GRL), see their appendix.

      write(nud,*)
      write(nud,9991)
      write(nud,9997)
      write(nud,9991)
      write(nud,9996)
      write(nud,9991)
      do jlev=1,NLEV
         zp = sigma(jlev)*psurf
         if (zp < pspon) then
            fric(jlev) = (dcsponge * sol_day &
                 * ((pspon - zp) / pspon)**2) / TWOPI
         endif
```



```
!        some output
         if (zp > pspon) then
            if (fric(jlev) == 0) then
               write(nud,9992) jlev
            else
               write(nud,9993) jlev, fric(jlev)*TWOPI
            endif
         else
            if (fric(jlev) == 0) then
               write(nud,9994) jlev
            else
               write(nud,9995) jlev, fric(jlev)*TWOPI
            endif
         endif
      enddo
      write(nud,9991)
      write(nud,*)
      return
 9991 format(33('*'))
 9992 format('*',i4,' *',7('-'),' *       *')
 9993 format('*',i4,' *',f7.4,' *       *')
 9994 format('*',i4,' *',7('-'),' *',' SPONGE    *')
 9995 format('*',i4,' *',f7.4,' *',' SPONGE    *')
 9996 format('* Lv *[1/day] *       *')
 9997 format('* Rayleigh damping coefficients *')
      end


!     =====================
!     SUBROUTINE MKENERDIAG
!     =====================

      subroutine mkenerdiag(pst,pstt,psp,pspt,penergy)
      use pumamod
!
      real :: pst(NSPP,NLEV),pstt(NSPP,NLEV)
      real :: psp(NSPP),pspt(NSPP)
      real :: penergy(NHOR)
!
      real :: zsttf(NESP,NLEV),zstf(NESP,NLEV)
      real :: zsptf(NESP),zspf(NESP)
      real :: zgtt(NHOR,NLEV),zgt(NHOR,NLEV)
      real :: zgps(NHOR),zgpst(NHOR)
      real :: ztm(NHOR)
!
      zcp=gascon/akap
      zdelt=delt2/ww_scale
!
      call mpgallsp(zsttf,pstt,NLEV)
      call mpgallsp(zstf,pst,NLEV)
      call mpgallsp(zsptf,pspt,1)
      call mpgallsp(zspf,psp,1)

      do jlev=1,NLEV
       call sp2fc(zsttf(:,jlev),zgtt(:,jlev))
       call sp2fc(zstf(:,jlev),zgt(:,jlev))
      enddo
      call sp2fc(zsptf,zgpst)
      call sp2fc(zspf,zgps)
      call fc2gp(zgtt,NLON,NLPP*NLEV)
      call fc2gp(zgt,NLON,NLPP*NLEV)
      call fc2gp(zgps,NLON,NLPP)
      call fc2gp(zgpst,NLON,NLPP)
      zgpst(:)=psurf*(exp(zgps(:)+delt2*zgpst(:))-exp(zgps(:)))/zdelt
      zgps(:)=psurf*exp(zgps(:))+zdelt*zgpst(:)
      zgtt(:,:)=ct*ww_scale*zgtt(:,:)
      do jlev=1,NLEV
       zgt(:,jlev)=ct*(zgt(:,jlev)+t0(jlev))
```

```
      enddo
!
      ztm(:)=0.
      penergy(:)=0.
      do jlev=1,NLEV
       ztm(:)=ztm(:)+zgt(:,jlev)*dsigma(jlev)
       penergy(:)=penergy(:)+zgtt(:,jlev)*dsigma(jlev)
      enddo
      penergy(:)=ztm(:)*zcp*zgpst(:)/ga                                  &
     &          +penergy(:)*zcp*zgps(:)/ga
!
      return
      end

!     ======================
!     SUBROUTINE MKENTRODIAG
!     ======================

      subroutine mkentrodiag(pst,pstt,psp,pentropy)
      use pumamod
!
      real :: pst(NSPP,NLEV),pstt(NSPP,NLEV)
      real :: psp(NSPP)
      real :: pentropy(NHOR)
!
      real :: zsttf(NESP,NLEV),zstf(NESP,NLEV)
      real :: zspf(NESP)
      real :: zgtt(NHOR,NLEV),zgt(NHOR,NLEV)
      real :: zgps(NHOR)
!
      zcp=gascon/akap
!
      call mpgallsp(zsttf,pstt,NLEV)
      call mpgallsp(zstf,pst,NLEV)
      call mpgallsp(zspf,psp,1)

      do jlev=1,NLEV
       call sp2fc(zsttf(:,jlev),zgtt(:,jlev))
       call sp2fc(zstf(:,jlev),zgt(:,jlev))
      enddo
      call sp2fc(zspf,zgps)
      call fc2gp(zgtt,NLON,NLPP*NLEV)
      call fc2gp(zgt,NLON,NLPP*NLEV)
      call fc2gp(zgps,NLON,NLPP)
      zgps(:)=psurf*exp(zgps(:))
      zgtt(:,:)=ct*ww_scale*zgtt(:,:)
      do jlev=1,NLEV
       zgt(:,jlev)=ct*(zgt(:,jlev)+t0(jlev))
      enddo
!
      pentropy(:)=0.
      do jlev=1,NLEV
       pentropy(:)=pentropy(:)+zgtt(:,jlev)*dsigma(jlev)/zgt(:,jlev)
      enddo
      pentropy(:)=pentropy(:)*zcp*zgps(:)/ga
!
      return
      end

!     ==================
!     SUBROUTINE MKDHEAT
!     ==================

      subroutine mkdheat(zszt1,zszt2,zsdt1,zsdt2,zsp)
      use pumamod
!
!     'recycle' kin. energy loss by heating the environment
!
!     zszt1/zsdt1 : vorticity/divergence tendency due to friction
```



```
!     zszt2/zsdt2 : vorticity/divergence tendency fue to diffusion
!     zp          : surface pressure
!
      real zszt1(NSPP,NLEV),zszt2(NSPP,NLEV)
      real zsdt1(NSPP,NLEV),zsdt2(NSPP,NLEV)
      real zsp(NSPP)
      real zp(NHOR)
!
      real zsd(NESP,NLEV),zsz(NESP,NLEV)
      real zspf(NESP),zspt(NSPP)
      real zsdp(NSPP,NLEV),zszp(NSPP,NLEV)
      real zu(NHOR,NLEV),zun(NHOR,NLEV),zdu1(NHOR,NLEV),zdu2(NHOR,NLEV)
      real zv(NHOR,NLEV),zvn(NHOR,NLEV),zdv1(NHOR,NLEV),zdv2(NHOR,NLEV)
      real zdtdt1(NHOR,NLEV),zdtdt2(NHOR,NLEV),zdtdt3(NHOR,NLEV)
!
      real zdtdt(NHOR,NLEV),zdekin(NHOR,NLEV)
!
      real zsde(NSPP,NLEV),zsdef(NESP,NLEV)
      real zstt(NSPP,NLEV),zstf(NESP,NLEV)
      real zstt1(NSPP,NLEV),zstf1(NESP,NLEV),zstt3(NSPP,NLEV)
      real zstt2(NSPP,NLEV),zstf2(NESP,NLEV),zstf3(NESP,NLEV)
!
!     some constants
!
      zdelt=delt2/ww_scale     ! timestep in s
      zcp=gascon/akap    ! heat capacity
!
!     'recycle' friction
!
!     a) gather the 'partial' field of z and d, and make u and v
!        at old time level
!
      zsdp(:,:)=sdp(:,:)
      zszp(:,:)=szp(:,:)
      call mpgallsp(zsd,zsdp,NLEV)
      call mpgallsp(zsz,zszp,NLEV)
      do jlev = 1 , NLEV
         call dv2uv(zsd(1,jlev),zsz(1,jlev),zu(1,jlev),zv(1,jlev))
      enddo
      call fc2gp(zu,NLON,NLPP*NLEV)
      call fc2gp(zv,NLON,NLPP*NLEV)
!
!     b) add fricton tendencies and create new u and v
!
      zsdp(:,:)=sdp(:,:)+zsdt1(:,:)*delt2
      zszp(:,:)=szp(:,:)+zszt1(:,:)*delt2
      call mpgallsp(zsd,zsdp,NLEV)
      call mpgallsp(zsz,zszp,NLEV)
      do jlev = 1 , NLEV
         call dv2uv(zsd(1,jlev),zsz(1,jlev),zun(1,jlev),zvn(1,jlev))
      enddo
      call fc2gp(zun,NLON,NLPP*NLEV)
      call fc2gp(zvn,NLON,NLPP*NLEV)
!
!     c) compute temperature tendency
!
      do jlev=1,NLEV
       zu(:,jlev)=cv*zu(:,jlev)*SQRT(rcsq(:))
       zv(:,jlev)=cv*zv(:,jlev)*SQRT(rcsq(:))
       zun(:,jlev)=cv*zun(:,jlev)*SQRT(rcsq(:))
       zvn(:,jlev)=cv*zvn(:,jlev)*SQRT(rcsq(:))
       zdu1(:,jlev)=zun(:,jlev)-zu(:,jlev)
       zdv1(:,jlev)=zvn(:,jlev)-zv(:,jlev)
       zdtdt1(:,jlev)=-(zun(:,jlev)*zun(:,jlev)                          &
     &                 -zu(:,jlev)*zu(:,jlev)                            &
     &                 +zvn(:,jlev)*zvn(:,jlev)                          &
     &                 -zv(:,jlev)*zv(:,jlev))*0.5/zdelt/zcp
      enddo
```

```
!
!     'recycle' momentum diffusion
!
!     a) add tendencies and create new u and v and get surface pressure
!
!
      zsdp(:,:)=sdp(:,:)+zsdt2(:,:)*delt2
      zszp(:,:)=szp(:,:)+zszt2(:,:)*delt2
      call mpgallsp(zsd,zsdp,NLEV)
      call mpgallsp(zsz,zszp,NLEV)
      call mpgallsp(zspf,zsp,1)
      do jlev = 1 , NLEV
         call dv2uv(zsd(1,jlev),zsz(1,jlev),zun(1,jlev),zvn(1,jlev))
      enddo
      call fc2gp(zun,NLON,NLPP*NLEV)
      call fc2gp(zvn,NLON,NLPP*NLEV)
      call sp2fc(zspf,zp)
      call fc2gp(zp,NLON,NLPP)
      zp(:)=psurf*exp(zp(:))
!
!     b) compute loss of kinetic energy
!        (note: only the global average change of kin. e. is 'lost'
!         the other changes are just diffusion)
!
      do jlev = 1 , NLEV
       zun(:,jlev)=cv*zun(:,jlev)*SQRT(rcsq(:))
       zvn(:,jlev)=cv*zvn(:,jlev)*SQRT(rcsq(:))
       zdu2(:,jlev)=zun(:,jlev)-zu(:,jlev)
       zdv2(:,jlev)=zvn(:,jlev)-zv(:,jlev)
       zdekin(:,jlev)=(zun(:,jlev)*zun(:,jlev)                          &
     &                -zu(:,jlev)*zu(:,jlev)                            &
     &                +zvn(:,jlev)*zvn(:,jlev)                          &
     &                -zv(:,jlev)*zv(:,jlev))*0.5/zdelt                 &
     &               *zp(:)/ga*dsigma(jlev)
      enddo
!
!     c) get the global average and transform it back
!
      call gp2fc(zdekin,NLON,NLPP*NLEV)
      do jlev=1,NLEV
       call fc2sp(zdekin(:,jlev),zsdef(:,jlev))
      enddo
      call mpsumsc(zsdef,zsde,NLEV)
      call mpgallsp(zsdef,zsde,NLEV)
      zsdef(2:NESP,:)=0.
      do jlev = 1 , NLEV
         call sp2fc(zsdef(1,jlev),zdekin(1,jlev))
      enddo
      call fc2gp(zdekin,NLON,NLPP*NLEV)
!
!     d) compute temperature tendency
!
      do jlev=1,NLEV
       zdtdt2(:,jlev)=-zdekin(:,jlev)*ga/zp(:)/dsigma(jlev)/zcp
       zdtdt3(:,jlev)=-(zdu1(:,jlev)*zdu2(:,jlev)                       &
     &                 +zdv1(:,jlev)*zdv2(:,jlev))/zdelt/zcp
      enddo
!
      zdtdt1(:,:)=zdtdt1(:,:)/ct/ww_scale
      zdtdt2(:,:)=zdtdt2(:,:)/ct/ww_scale
      zdtdt3(:,:)=zdtdt3(:,:)/ct/ww_scale
!
      call gp2fc(zdtdt1,NLON,NLPP*NLEV)
      call gp2fc(zdtdt2,NLON,NLPP*NLEV)
      call gp2fc(zdtdt3,NLON,NLPP*NLEV)
      do jlev=1,NLEV
       call fc2sp(zdtdt1(:,jlev),zstf1(:,jlev))
       call fc2sp(zdtdt2(:,jlev),zstf2(:,jlev))
       call fc2sp(zdtdt3(:,jlev),zstf3(:,jlev))
```



```
      enddo
      call mpsumsc(zstf1,zstt1,NLEV)
      call mpsumsc(zstf2,zstt2,NLEV)
      call mpsumsc(zstf3,zstt3,NLEV)
!
!     add the temprature tendencies
!
      stt(:,:)=stt(:,:)+zstt1(:,:)+zstt2(:,:)+zstt3(:,:)
!
!     energy diagnostics
!
      if(nenergy > 0) then
       zspt(:)=0.
       call mkenerdiag(stp,zstt1,zsp,zspt,denergy(:,5))
       call mkenerdiag(stp,zstt2,zsp,zspt,denergy(:,6))
       call mkenerdiag(stp,zstt3,zsp,zspt,denergy(:,7))
      endif
      if(nentropy > 0) then
       call mkentrodiag(stp,zstt1,zsp,dentropy(:,5))
       call mkentrodiag(stp,zstt2,zsp,dentropy(:,6))
       call mkentrodiag(stp,zstt3,zsp,dentropy(:,7))
      endif

!
      return
      end subroutine mkdheat

!     =================
!     SUBROUTINE MKEKIN
!     =================

      subroutine mkekin(zszp,zsdp,zp,zekin)
      use pumamod
!
      real zszp(NSPP,NLEV),zsdp(NSPP,NLEV)
      real zp(NHOR),zekin(NHOR)
!
      real zsd(NESP,NLEV),zsz(NESP,NLEV)
      real zu(NHOR,NLEV),zv(NHOR,NLEV)
!
!     some constants
!
      zdelt=delt2/ww_scale      ! timestep in s
      zcp=gascon/akap     ! heat capacity
!
      call mpgallsp(zsd,zsdp,NLEV)
      call mpgallsp(zsz,zszp,NLEV)
      do jlev = 1 , NLEV
         call dv2uv(zsd(1,jlev),zsz(1,jlev),zu(1,jlev),zv(1,jlev))
      enddo
      call fc2gp(zu,NLON,NLPP*NLEV)
      call fc2gp(zv,NLON,NLPP*NLEV)
!
      zekin(:)=0.
      do jlev = 1 , NLEV
       zu(:,jlev)=cv*zu(:,jlev)*SQRT(rcsq(:))
       zv(:,jlev)=cv*zv(:,jlev)*SQRT(rcsq(:))
       zekin(:)=(zu(:,jlev)*zu(:,jlev)+zv(:,jlev)*zv(:,jlev))*0.5       &
     &          *zp(:)/ga*dsigma(jlev)+zekin(:)
      enddo
!
      return
      end
      subroutine mkekin2(zszp,zsdp,zspp,zekin)
      use pumamod
!
      real zszp(NSPP,NLEV),zsdp(NSPP,NLEV),zspp(NSPP)
      real zp(NHOR),zekin(NHOR)
!
```

```
      real zsd(NESP,NLEV),zsz(NESP,NLEV),zsp(NESP)
      real zu(NHOR,NLEV),zv(NHOR,NLEV)
!
!     some constants
!
      zdelt=delt2/ww_scale     ! timestep in s
      zcp=gascon/akap    ! heat capacity
!
      call mpgallsp(zsd,zsdp,NLEV)
      call mpgallsp(zsz,zszp,NLEV)
      call mpgallsp(zsp,zspp,NLEV)
      do jlev = 1 , NLEV
         call dv2uv(zsd(1,jlev),zsz(1,jlev),zu(1,jlev),zv(1,jlev))
      enddo
      call sp2fc(zsp,zp)
      call fc2gp(zu,NLON,NLPP*NLEV)
      call fc2gp(zv,NLON,NLPP*NLEV)
      call fc2gp(zp,NLON,NLPP)
!
      zp(:)=psurf*exp(zp(:))
      zekin(:)=0.
      do jlev = 1 , NLEV
       zu(:,jlev)=cv*zu(:,jlev)*SQRT(rcsq(:))
       zv(:,jlev)=cv*zv(:,jlev)*SQRT(rcsq(:))
       zekin(:)=(zu(:,jlev)*zu(:,jlev)+zv(:,jlev)*zv(:,jlev))*0.5       &
     &          *zp(:)/ga*dsigma(jlev)+zekin(:)
      enddo
!
      return
      end


!     =================
!     SUBROUTINE MKEPOT
!     =================

      subroutine mkepot(zstp,zp,zepot)
      use pumamod
!
      real zstp(NSPP,NLEV)
      real zp(NHOR),zepot(NHOR)
!
      real zst(NESP,NLEV)
      real zt(NHOR,NLEV)
!
!     some constants
!
      zdelt=delt2/ww_scale     ! timestep in s
      zcp=gascon/akap    ! heat capacity
!
      call mpgallsp(zst,zstp,NLEV)
      do jlev = 1 , NLEV
       call sp2fc(zst(1,jlev),zt(1,jlev))
      enddo
      call fc2gp(zt,NLON,NLPP*NLEV)
!
      zepot(:)=0.
      do jlev = 1 , NLEV
       zt(:,jlev)=ct*(zt(:,jlev)+t0(jlev))
       zepot(:)=zt(:,jlev)*zcp   &
     &          *zp(:)/ga*dsigma(jlev)+zepot(:)
      enddo
!
      return
      end
      subroutine mkepot2(zstp,zspp,zepot)
      use pumamod
!
      real zstp(NSPP,NLEV),zspp(NSPP)
```



```
      real zp(NHOR),zepot(NHOR)
!
      real zst(NESP,NLEV),zsp(NESP)
      real zt(NHOR,NLEV)
!
!     some constants
!
      zdelt=delt2/ww_scale     ! timestep in s
      zcp=gascon/akap    ! heat capacity
!
      call mpgallsp(zst,zstp,NLEV)
      call mpgallsp(zsp,zspp,1)
      do jlev = 1 , NLEV
       call sp2fc(zst(1,jlev),zt(1,jlev))
      enddo
      call sp2fc(zsp,zp)
      call fc2gp(zt,NLON,NLPP*NLEV)
      call fc2gp(zp,NLON,NLPP)
!
      zp(:)=psurf*exp(zp(:))
      zepot(:)=0.
      do jlev = 1 , NLEV
       zt(:,jlev)=ct*(zt(:,jlev)+t0(jlev))
       zepot(:)=zt(:,jlev)*zcp   &
     &          *zp(:)/ga*dsigma(jlev)+zepot(:)
      enddo
!
      return
      end
```

```
! ************************************************************************
! * module legsym - direct and indirect Legendre transformation routines *
! * using symmetric and antisymmetric fourier coefficients               *
! * E. Kirk 08-Sep-2010 tested for T21 - T682 resolutions 32 & 64 bit    *
! ************************************************************************

module legsym
! ************************
! * Legendre Polynomials *
! ************************

integer :: ntru
integer :: ntp1
integer :: ncsp
integer :: nesp
integer :: nlon
integer :: nlpp
integer :: nhpp
integer :: nlat
integer :: nlev
real    :: plavor

real   , allocatable :: qi(:,:) ! P(m,n) = Associated Legendre Polynomials
real   , allocatable :: qj(:,:) ! Q(m,n) = Used for d/d(mu)
real   , allocatable :: qc(:,:) ! P(m,n) * gwd              used in fc2sp
real   , allocatable :: qe(:,:) ! Q(mn,) * gwd / cos2       used in mktend
real   , allocatable :: qq(:,:) ! P(m,n) * gwd / cos2 * n * (n+1) / 2  "
real   , allocatable :: qu(:,:) ! P(m,n) / (n*(n+1)) * m    used in dv2uv
real   , allocatable :: qv(:,:) ! Q(m,n) / (n*(n+1))        used in dv2uv
complex, allocatable :: qx(:,:) ! P(m,n) * gwd / cos2 * m   used in mktend
end module legsym

! =================
! SUBROUTINE LEGINI
! =================

subroutine legini(klat,klpp,kesp,klev,vorpla,sid,gwd)
use legsym
implicit none

integer :: klat
integer :: klpp
integer :: kesp
integer :: klev

real          :: vorpla   ! planetary vorticity
real (kind=8) :: sid(*)   ! sin(phi)
real (kind=8) :: gwd(*)   ! Gaussian weight (phi)

integer :: jlat ! Latitude
integer :: lm
integer :: m
integer :: n

real (kind=8) :: amsq
real (kind=8) :: z1
real (kind=8) :: z2
real (kind=8) :: z3
real (kind=8) :: f1m
real (kind=8) :: f2m
real (kind=8) :: znn1
real (kind=8) :: zsin    ! sin
real (kind=8) :: zcsq    ! cos2
real (kind=8) :: zcos    ! cos
real (kind=8) :: zgwd    ! gw
real (kind=8) :: zgwdcsq ! gw / cos2
real (kind=8) :: zpli(kesp)
real (kind=8) :: zpld(kesp)
```



```
nlat = klat
nlpp = klpp
nhpp = klpp / 2
nlon = klat + klat
ntru = (nlon - 1) /3
ntp1 = ntru + 1
ncsp = ((ntru + 1) * (ntru + 2)) / 2
nesp = kesp
nlev = klev

plavor = vorpla

allocate(qi(ncsp,nhpp))
allocate(qj(ncsp,nhpp))
allocate(qc(ncsp,nhpp))
allocate(qe(ncsp,nhpp))
allocate(qx(ncsp,nhpp))
allocate(qq(ncsp,nhpp))
allocate(qu(ncsp,nhpp))
allocate(qv(ncsp,nhpp))

do jlat = 1 , nhpp

! set p(0,0) and p(0,1)

   zgwd    = gwd(jlat)             ! gaussian weight - from inigau
   zsin    = sid(jlat)             ! sin(phi) - from inigau
   zcsq    = 1.0_8 - zsin * zsin   ! cos(phi) squared
   zgwdcsq = zgwd / zcsq           ! weight / cos squared
   zcos    = sqrt(zcsq)            ! cos(phi)
   f1m     = sqrt(1.5_8)
   zpli(1) = sqrt(0.5_8)
   zpli(2) = f1m * zsin
   zpld(1) = 0.0
   lm      = 2

! loop over wavenumbers

   do m = 0 , ntru
      if (m > 0) then
         lm  = lm + 1
         f2m = -f1m * sqrt(zcsq / (m+m))
         f1m =  f2m * sqrt(m+m + 3.0_8)
         zpli(lm) = f2m
         if (lm < ncsp) then
            lm = lm + 1
            zpli(lm  ) =       f1m * zsin
            zpld(lm-1) =  -m * f2m * zsin
         endif ! (lm < ncsp)
      endif ! (m > 0)

      amsq = m * m

      do n = m+2 , ntru
         lm = lm + 1
         z1 = sqrt(((n-1)*(n-1) - amsq) / (4*(n-1)*(n-1)-1))
         z2 = zsin * zpli(lm-1) - z1 * zpli(lm-2)
         zpli(lm  ) = z2 * sqrt((4*n*n-1) / (n*n-amsq))
         zpld(lm-1) = (1-n) * z2 + n * z1 * zpli(lm-2)
      enddo ! n

      if (lm < ncsp) then ! mode (m,ntru)
         z3 = sqrt((ntru*ntru-amsq) / (4*ntru*ntru-1))
         zpld(lm)=-ntru*zsin*zpli(lm) + (ntru+ntru+1)*zpli(lm-1)*z3
      else                ! mode (ntru,ntru)
         zpld(lm)=-ntru*zsin*zpli(lm)
      endif
   enddo ! m
```

```
   lm = 0
   do m = 0 , ntru
      do n = m , ntru
           lm = lm + 1
           znn1 = 0.0
           if (n > 0) znn1 = 1.0_8 / (n*(n+1))
           qi(lm,jlat) = zpli(lm)
           qj(lm,jlat) = zpld(lm)
           qc(lm,jlat) = zpli(lm) * zgwd
           qu(lm,jlat) = zpli(lm) * znn1 * m
           qv(lm,jlat) = zpld(lm) * znn1
           qe(lm,jlat) = zpld(lm) * zgwdcsq
           qq(lm,jlat) = zpli(lm) * zgwdcsq * n * (n+1) * 0.5_8
           qx(lm,jlat) = zpli(lm) * zgwdcsq * m * (0.0,1.0)
      enddo ! n
   enddo ! m
enddo ! jlat
return
end


! ================
! SUBROUTINE FC2SP
! ================

subroutine fc2sp(fc,sp)
use legsym
implicit none
complex, intent(in ) :: fc(nlon,nhpp)
complex, intent(out) :: sp(nesp/2)

integer :: l ! Index for latitude
integer :: m ! Index for zonal wavenumber
integer :: w ! Index for spherical harmonic
integer :: e ! Index for last wavenumber

sp(:) = (0.0,0.0)

do l = 1 , nhpp
   w = 1
   do m = 1 , ntp1
      e = w + ntp1 - m
      sp(w  :e:2) = sp(w  :e:2) + qc(w  :e:2,l) * (fc(m,l) + fc(m+nlat,l))
      sp(w+1:e:2) = sp(w+1:e:2) + qc(w+1:e:2,l) * (fc(m,l) - fc(m+nlat,l))
      w = e + 1
   enddo ! m
enddo ! l
return
end


! ================
! SUBROUTINE SP2FC
! ================

subroutine sp2fc(sp,fc) ! Spectral to Fourier
use legsym
implicit none

complex :: sp(ncsp)      ! Coefficients of spherical harmonics
complex :: fc(nlon,nhpp) ! Fourier coefficients

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Index for spectral mode
integer :: e ! Index for last wavenumber

complex :: fs,fa
```



```
fc(:,:) = (0.0,0.0)

do l = 1 , nhpp
  w = 1
  do m = 1 ,ntp1
    e = w + ntp1 - m
    fs = dot_product(qi(w  :e:2,l),sp(w  :e:2))
    fa = dot_product(qi(w+1:e:2,l),sp(w+1:e:2))
    fc(m     ,l) = fs + fa
    fc(m+nlat,l) = fs - fa
    w = e + 1
  enddo ! m
enddo ! l
return
end


! ====================
! SUBROUTINE SP2FCDMU
! ====================

subroutine sp2fcdmu(sp,fc) ! Spectral to Fourier d/dmu
use legsym
implicit none

complex :: sp(ncsp)      ! Coefficients of spherical harmonics
complex :: fc(nlon,nhpp) ! Fourier coefficients

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Index for spectral mode
integer :: e ! Index for last wavenumber

complex :: fs,fa

fc(:,:) = (0.0,0.0)

do l = 1 , nhpp
  w = 1
  do m = 1 , ntp1
    e = w + ntp1 - m
    fs = dot_product(qj(w  :e:2,l),sp(w  :e:2))
    fa = dot_product(qj(w+1:e:2,l),sp(w+1:e:2))
    fc(m     ,l) = fa + fs
    fc(m+nlat,l) = fa - fs
    w = e + 1
  enddo ! m
enddo ! l
return
end


! ================
! SUBROUTINE DV2UV
! ================

! This is an alternative subroutine for computing U and V from Div and Vor.
! It looks much prettier than the regular one, but unfortunately it is slower
! if compiling with "gfortran".
! I leave it (unused) in this module for educational purposes.

subroutine dv2uv_alt(pd,pz,pu,pv)
use legsym
implicit none
complex, parameter :: i = (0.0,1.0)
complex :: pd(nesp/2)    ! Spherical harmonics  of divergence
complex :: pz(nesp/2)    ! Spherical harmonics  of vorticity
complex :: pu(nlon,nhpp) ! Fourier coefficients of u
complex :: pv(nlon,nhpp) ! Fourier coefficients of v
```

```
complex :: zsave,uds,vds,uzs,vzs,uda,vda,uza,vza

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Loop index for spectral mode
integer :: e ! End index

pu(:,:) = (0.0,0.0)
pv(:,:) = (0.0,0.0)

zsave = pz(2)                       ! Save mode(0,1) of vorticity
pz(2) = zsave - cmplx(plavor,0.0)  ! Convert pz from absolute to relative vortic
ity

do l = 1 , nhpp
  w = 1
  do m = 1 ,ntp1
    e = w + ntp1 - m
    uds = i * dot_product(qu(w  :e:2,l),pd(w:  e:2))
    vds =     dot_product(qv(w  :e:2,l),pd(w:  e:2))
    uzs = i * dot_product(qu(w  :e:2,l),pz(w:  e:2))
    vzs =     dot_product(qv(w  :e:2,l),pz(w:  e:2))
    uda = i * dot_product(qu(w+1:e:2,l),pd(w+1:e:2))
    vda =     dot_product(qv(w+1:e:2,l),pd(w+1:e:2))
    uza = i * dot_product(qu(w+1:e:2,l),pz(w+1:e:2))
    vza =     dot_product(qv(w+1:e:2,l),pz(w+1:e:2))
    pu(m     ,l) =  vzs - uds + vza - uda
    pu(m+nlat,l) = -vzs - uds + vza + uda
    pv(m     ,l) = -vds - uzs - vda - uza
    pv(m+nlat,l) =  vds - uzs - vda + uza
    w = e + 1
  enddo ! m
enddo ! l
pz(2) = zsave
return
end


! ================
! SUBROUTINE DV2UV
! ================

subroutine dv2uv(pd,pz,pu,pv)
use legsym
implicit none

real :: pd(2,nesp/2)    ! Spherical harmonics  of divergence
real :: pz(2,nesp/2)    ! Spherical harmonics  of vorticity
real :: pu(2,nlon,nhpp) ! Fourier coefficients of u
real :: pv(2,nlon,nhpp) ! Fourier coefficients of v
real :: zsave
real :: unr,uni,usr,usi,vnr,vni,vsr,vsi
real :: zdr,zdi,zzr,zzi

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: n ! Loop index for total wavenumber n
integer :: w ! Loop index for spectral mode

pu(:,:,:) = 0.0
pv(:,:,:) = 0.0

zsave = pz(1,2)           ! Save mode(0,1) of vorticity
pz(1,2) = zsave - plavor  ! Convert pz from absolute to relative vorticity

do l = 1 , nhpp
   w = 1
   do m = 1 , ntp1
      unr = 0.0 ! u - north - real
```



```
      uni = 0.0 ! u - north - imag
      usr = 0.0 ! u - south - real
      usi = 0.0 ! u - south - imag
      vnr = 0.0 ! v - north - real
      vni = 0.0 ! v - north - imag
      vsr = 0.0 ! v - south - real
      vsi = 0.0 ! v - south - imag

!     process two modes per iteration, one symmetric (m+n = even) and one anti
!     we start the loop with (n=m), so the starting mode is always symmetric

      do n = m , ntru , 2
         zdr = qu(w,l) * pd(1,w)  ! symmetric mode
         zdi = qu(w,l) * pd(2,w)
         zzr = qv(w,l) * pz(1,w)
         zzi = qv(w,l) * pz(2,w)
         unr = unr + zzr + zdi
         uni = uni + zzi - zdr
         usr = usr - zzr + zdi
         usi = usi - zzi - zdr
         zzr = qu(w,l) * pz(1,w)
         zzi = qu(w,l) * pz(2,w)
         zdr = qv(w,l) * pd(1,w)
         zdi = qv(w,l) * pd(2,w)
         vnr = vnr + zzi - zdr
         vni = vni - zzr - zdi
         vsr = vsr + zzi + zdr
         vsi = vsi - zzr + zdi
         w = w + 1
         zdr = qu(w,l) * pd(1,w)  ! antisymmetric mode
         zdi = qu(w,l) * pd(2,w)
         zzr = qv(w,l) * pz(1,w)
         zzi = qv(w,l) * pz(2,w)
         unr = unr + zzr + zdi
         uni = uni + zzi - zdr
         usr = usr + zzr - zdi
         usi = usi + zzi + zdr
         zzr = qu(w,l) * pz(1,w)
         zzi = qu(w,l) * pz(2,w)
         zdr = qv(w,l) * pd(1,w)
         zdi = qv(w,l) * pd(2,w)
         vnr = vnr + zzi - zdr
         vni = vni - zzr - zdi
         vsr = vsr - zzi - zdr
         vsi = vsi + zzr - zdi
         w = w + 1
      enddo
      if (n == ntp1) then          ! additional symmetric mode
         zdr = qu(w,l) * pd(1,w)  ! if (ntp1-m) is even
         zdi = qu(w,l) * pd(2,w)
         zzr = qv(w,l) * pz(1,w)
         zzi = qv(w,l) * pz(2,w)
         unr = unr + zzr + zdi
         uni = uni + zzi - zdr
         usr = usr - zzr + zdi
         usi = usi - zzi - zdr
         zzr = qu(w,l) * pz(1,w)
         zzi = qu(w,l) * pz(2,w)
         zdr = qv(w,l) * pd(1,w)
         zdi = qv(w,l) * pd(2,w)
         vnr = vnr + zzi - zdr
         vni = vni - zzr - zdi
         vsr = vsr + zzi + zdr
         vsi = vsi - zzr + zdi
         w = w + 1
      endif
      pu(1,m     ,l) = unr
      pu(2,m     ,l) = uni
      pu(1,m+nlat,l) = usr
```

```
      pu(2,m+nlat,l) = usi
      pv(1,m     ,l) = vnr
      pv(2,m     ,l) = vni
      pv(1,m+nlat,l) = vsr
      pv(2,m+nlat,l) = vsi
   enddo ! m
enddo ! l
pz(1,2) = zsave ! Restore pz to absolute vorticity
return
end


! =================
! SUBROUTINE MKTEND
! =================

subroutine mktend(d,t,z,tn,fu,fv,ke,ut,vt)
use legsym
implicit none

complex, intent(in) :: tn(nlon,nhpp)
complex, intent(in) :: fu(nlon,nhpp)
complex, intent(in) :: fv(nlon,nhpp)
complex, intent(in) :: ke(nlon,nhpp)
complex, intent(in) :: ut(nlon,nhpp)
complex, intent(in) :: vt(nlon,nhpp)

complex, intent(out) :: d(nesp/2)
complex, intent(out) :: t(nesp/2)
complex, intent(out) :: z(nesp/2)

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Loop index for spectral mode
integer :: e ! End index for w

complex :: fus,fua,fvs,fva,kes,kea,tns,tna,uts,uta,vts,vta

d(:) = (0.0,0.0) ! divergence
t(:) = (0.0,0.0) ! temperature
z(:) = (0.0,0.0) ! vorticity

do l = 1 , nhpp  ! process pairs of Nort-South latitudes
   w = 1
   do m = 1 , ntp1
      kes = ke(m,l) + ke(m+nlat,l) ; kea = ke(m,l) - ke(m+nlat,l)
      fvs = fv(m,l) + fv(m+nlat,l) ; fva = fv(m,l) - fv(m+nlat,l)
      fus = fu(m,l) + fu(m+nlat,l) ; fua = fu(m,l) - fu(m+nlat,l)
      uts = ut(m,l) + ut(m+nlat,l) ; uta = ut(m,l) - ut(m+nlat,l)
      vts = vt(m,l) + vt(m+nlat,l) ; vta = vt(m,l) - vt(m+nlat,l)
      tns = tn(m,l) + tn(m+nlat,l) ; tna = tn(m,l) - tn(m+nlat,l)
      e = w + ntp1 - m    ! vector of symmetric modes
      d(w:e:2) = d(w:e:2) + qq(w:e:2,l) * kes - qe(w:e:2,l) * fva + qx(w:e:2,l)
* fus
      t(w:e:2) = t(w:e:2) + qe(w:e:2,l) * vta + qc(w:e:2,l) * tns - qx(w:e:2,l)
* uts
      z(w:e:2) = z(w:e:2) + qe(w:e:2,l) * fua + qx(w:e:2,l) * fvs
      w = w + 1           ! vector of antisymmetric modes
      d(w:e:2) = d(w:e:2) + qq(w:e:2,l) * kea - qe(w:e:2,l) * fvs + qx(w:e:2,l)
* fua
      t(w:e:2) = t(w:e:2) + qe(w:e:2,l) * vts + qc(w:e:2,l) * tna - qx(w:e:2,l)
* uta
      z(w:e:2) = z(w:e:2) + qe(w:e:2,l) * fus + qx(w:e:2,l) * fva
      w = e + 1
   enddo ! m
enddo ! l
return
end
```



```
! ================
! SUBROUTINE QTEND
! ================

subroutine qtend_old(q,qn,uq,vq)
use legsym
implicit none

complex, intent(in) :: qn(nlon,nhpp)
complex, intent(in) :: uq(nlon,nhpp)
complex, intent(in) :: vq(nlon,nhpp)

complex, intent(out) :: q(nesp/2)

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Loop index for spectral mode
integer :: e ! End index for w

complex :: qns,qna,uqs,uqa,vqs,vqa

q(:) = (0.0,0.0) ! humidity

do l = 1 , nhpp  ! process pairs of Nort-Souqh latitudes
   w = 1
   do m = 1 , ntp1
      uqs = uq(m,l) + uq(m+nlat,l) ; uqa = uq(m,l) - uq(m+nlat,l)
      vqs = vq(m,l) + vq(m+nlat,l) ; vqa = vq(m,l) - vq(m+nlat,l)
      qns = qn(m,l) + qn(m+nlat,l) ; qna = qn(m,l) - qn(m+nlat,l)
      e = w + ntp1 - m    ! vector of symmetric modes
      q(w:e:2) = q(w:e:2) + qe(w:e:2,l) * vqa + qc(w:e:2,l) * qns - qx(w:e:2,l)
* uqs
      w = w + 1           ! vector of antisymmetric modes
      q(w:e:2) = q(w:e:2) + qe(w:e:2,l) * vqs + qc(w:e:2,l) * qna - qx(w:e:2,l)
* uqa
      w = e + 1
   enddo ! m
enddo ! l
return
end


! ================
! SUBROUTINE UV2DV
! ================

subroutine uv2dv(pu,pv,pd,pz)
use legsym
implicit none

complex, intent(in) :: pu(nlon,nhpp)
complex, intent(in) :: pv(nlon,nhpp)

complex, intent(out) :: pd(nesp/2)
complex, intent(out) :: pz(nesp/2)

integer :: l ! Loop index for latitude
integer :: m ! Loop index for zonal wavenumber m
integer :: w ! Loop index for spectral mode
integer :: e ! End index for w

complex :: zus,zua,zvs,zva

pd(:) = (0.0,0.0) ! divergence
pz(:) = (0.0,0.0) ! vorticity

do l = 1 , nhpp   ! process pairs of Nort-Souqh latitudes
   w = 1
```

```
   do m = 1 , NTP1
      zus = pu(m,l) + pu(m+nlat,l) ; zua = pu(m,l) - pu(m+nlat,l)
      zvs = pv(m,l) + pv(m+nlat,l) ; zva = pv(m,l) - pv(m+nlat,l)
      e = w + ntp1 - m    ! vector of symmetric modes
      pz(w:e:2) = pz(w:e:2) + qe(w:e:2,l) * zua + qx(w:e:2,l) * zvs
      pd(w:e:2) = pd(w:e:2) - qe(w:e:2,l) * zva + qx(w:e:2,l) * zus
      w = w + 1           ! vector of antisymmetric modes
      pz(w:e:2) = pz(w:e:2) + qe(w:e:2,l) * zus + qx(w:e:2,l) * zva
      pd(w:e:2) = pd(w:e:2) - qe(w:e:2,l) * zvs + qx(w:e:2,l) * zua
      w = e + 1
   enddo ! m
enddo ! l

return
end


! ==================
! SUBROUTINE REG2ALT
! ==================

subroutine reg2alt(pr,klev)
use legsym
implicit none

real :: pr(nlon,nlat,klev)
real :: pa(nlon,nlat,klev)

integer :: jlat
integer :: klev

do jlat = 1 , nlat / 2
  pa(:,2*jlat-1,:) = pr(:,jlat       ,:)
  pa(:,2*jlat  ,:) = pr(:,nlat-jlat+1,:)
enddo

pr = pa

return
end


! ==================
! SUBROUTINE ALT2REG
! ==================

subroutine alt2reg(pa,klev)
use legsym
implicit none

real :: pa(nlon,nlat,klev)
real :: pr(nlon,nlat,klev)

integer :: jlat
integer :: klev

do jlat = 1 , nlat / 2
  pr(:,jlat       ,:) = pa(:,2*jlat-1,:)
  pr(:,nlat-jlat+1,:) = pa(:,2*jlat  ,:)
enddo

pa = pr

return
end


! ================
! SUBROUTINE ALTCS
```



```
! ================

subroutine altcs(pcs)
use legsym
implicit none
real :: pcs(nlat,nlev)
real :: pal(nlat,nlev)
integer :: jlat

do jlat = 1 , nlat / 2
  pal(jlat       ,:) = pcs(2*jlat-1,:)
  pal(nlat-jlat+1,:) = pcs(2*jlat  ,:)
enddo

pcs = pal

return
end


! =================
! SUBROUTINE ALTLAT
! =================

subroutine altlat(pr,klat)
implicit none
integer :: jlat
integer :: klat
real    :: pr(klat)  ! regular     grid
real    :: pa(klat)  ! alternating grid

do jlat = 1 , klat / 2
  pa(2*jlat-1) = pr(jlat       )
  pa(2*jlat  ) = pr(klat-jlat+1)
enddo

pr(:) = pa(:)

return
end
```

```
! =================
! SUBROUTINE INIGAU
! =================

subroutine inigau(klat,pz0,pzw)        ! pz0 & pzw are (kind=8) reals !!!
implicit none
integer                   :: klat       ! Number of Gaussian latitudes
real (kind=8)             :: pz0(klat)  ! Gaussian abscissas
real (kind=8)             :: pzw(klat)  ! Gaussian weights
integer                   :: jlat       ! Latitudinal loop index
integer                   :: jiter      ! Iteration loop index
integer      , parameter :: NITER = 50 ! Maximum # of iterations
real (kind=8), parameter :: PI    =  3.14159265358979_8
real (kind=8), parameter :: ZEPS  =  1.0e-16 ! Convergence criterion
real (kind=8) :: z0,z1,z2,z3,z4,z5
real (kind=8) :: ql,qld

! Compute Gaussian abscissas & weights

z0 = PI / (2*klat+1)
z1 = 1.0_8 / (klat*klat*8)
z4 = 2.0_8 / (klat*klat)

do jlat = 1 , klat/2
   z2 = z0 * (2*jlat - 0.5_8)
   z2 = cos(z2 + z1 / tan(z2))
   do jiter = 1 , NITER
      z3 = ql(klat,z2) * qld(klat,z2)
      z2 = z2 - z3
      if (abs(z3) < ZEPS) exit ! converged
   enddo ! jiter
   z5 = ql(klat-1,z2) / sqrt(klat - 0.5_8)
   pz0(jlat) = z2
   pzw(jlat) = z4 * (1.0_8 - z2 * z2) / (z5 * z5)
   pz0(klat-jlat+1) = -z2
   pzw(klat-jlat+1) = pzw(jlat)
enddo ! jlat

return
end subroutine inigau

! ===========
! FUNCTION QL
! ===========

real (kind=8) function ql(k,p)
implicit none
integer      , intent(IN) :: k
real (kind=8), intent(IN) :: p
real (kind=8) :: z0,z1,z2,z3,z4
integer :: j
z0 = acos(p)
z1 = 1.0
z2 = 0.0
do j = k , 0 , -2
   z3 = z1 * cos(z0 * j)
   z2 = z2 + z3
   z4 = (k-j+1) * (k+j) * 0.5_8
   z1 = z1 * z4 / (z4 + (j-1))
enddo ! j
if (mod(k,2) == 0) z2 = z2 - 0.5_8 * z3

z0 = sqrt(2.0_8)
do j = 1 ,k
   z0 = z0 * sqrt(1.0_8 - 0.25_8 / (j*j))
enddo ! j
ql = z0 * z2
return
end function ql
```



```
! ============
! FUNCTION QLD
! ============

real (kind=8) function qld(k,p)
implicit none
integer      , intent(IN) :: k
real (kind=8), intent(IN) :: p
real (kind=8) :: z
real (kind=8) :: ql

z = p * ql(k,p) - sqrt((k + k + 1.0_8) / (k + k - 1.0_8)) * ql(k-1,p)
qld = (p * p - 1.0_8) / (k * z)

return
end function qld
```

```
!     =============
!     MODULE FFTMOD
!     =============

      module fftmod
      parameter(NRES = 12)
      integer :: nallowed(NRES) = (/16,32,48,64,96,128,256,384,512,1024,2048,409
6/)
!     T3    - N16   : 8-2
!     T10   - N32   : 8-2-2
!     T15   - N48   : 8-3-2
!     T21   - N64   : 8-4-2
!     T31   - N96   : 8-4-3
!     T42   - N128  : 8-4-4
!     T85   - N256  : 8-4-4-2
!     T127  - N384  : 8-4-4-3
!     T170  - N512  : 8-4-4-4
!     T341  - N1024 : 8-4-4-4-2
!     T682  - N2048 : 8-4-4-4-4
!     T1365 - N4096 : 8-4-4-4-4-2

      integer :: lastn = 0
      real,allocatable :: trigs(:)
      end module fftmod

!     ================
!     SUBROUTINE GP2FC
!     ================

      subroutine gp2fc(a,n,lot)
      use fftmod
      real a(n,lot)

      if (n /= lastn) then
         if (allocated(trigs)) deallocate(trigs)
         allocate(trigs(n))
         lastn = n
         call fftini(n)
      endif

      call dfft8(a,a,n,lot)
      la = n / 8
      do while (la >= 4)
         call dfft4(a,trigs,n,lot,la)
      enddo

      if (la == 3) then
         do l = 1 , lot
            call dfft3(a(1,l),trigs,n)
         enddo
      endif

      if (la == 2) then
         do l = 1 , lot
            call dfft2(a(1,l),trigs,n)
         enddo
      endif
      return
      end subroutine gp2fc

!     ================
!     SUBROUTINE FC2GP
!     ================

      subroutine fc2gp(a,n,lot)
      use fftmod
      real a(n,lot)

      if (n /= lastn) then
```



```
         if (allocated(trigs)) deallocate(trigs)
         allocate(trigs(n))
         lastn = n
         call fftini(n)
      endif

      nf = n/8
      do while (nf >= 4)
         nf = nf/4
      enddo
      la = 1
      if (nf == 2) call ifft2(a,trigs,n,lot,la)
      if (nf == 3) call ifft3(a,trigs,n,lot,la)
      do while (la < n/8)
         call ifft4(a,trigs,n,lot,la)
      enddo
      call ifft8(a,a,n,lot)
      return
      end subroutine fc2gp

!     =================
!     SUBROUTINE FFTINI
!     =================

      subroutine fftini(n)
      use fftmod
      logical labort

!     check for allowed values of n

      labort = .true.
      do j = 1 , NRES
         if (n == nallowed(j)) labort = .false.
      enddo

      if (labort) then
         write (*,*) '*** FFT does not support n = ',n,' ***'
         write (*,*) 'Following resolutions may be used:'
         write (*,*) '-----------------------------------'
         do j = 1 , NRES
            write (*,1000) nallowed(j), nallowed(j)/2, nallowed(j)/3
         enddo
         stop
      endif
 1000 format(' NLON=',I5,' NLAT=',I5,' NTRU=',I5)

      del = 4.0 * asin(1.0) / n
      do k=0,n/2-1
        angle = k * del
        trigs(2*k+1) = cos(angle)
        trigs(2*k+2) = sin(angle)
      enddo
      return
      end subroutine fftini

!     ================
!     SUBROUTINE DFFT2
!     ================

      subroutine dfft2(a,trigs,n)
      dimension a(n),c(n),trigs(n)

      c(1) = a(1) + a(2)
      c(2) = 0.0

      ja = 3
      jb = n - 1

      do i=3,n-5,4
```

```
         c1 = trigs(ja  )
         s1 = trigs(ja+1)
         a1p3 = c1 * a(i+1) + s1 * a(i+3)
         a3m1 = c1 * a(i+3) - s1 * a(i+1)
         c(ja  ) = a(i) + a1p3
         c(jb  ) = a(i) - a1p3
         c(ja+1) = a3m1 + a(i+2)
         c(jb+1) = a3m1 - a(i+2)
         ja = ja + 2
         jb = jb - 2
      enddo

      c(ja  ) =  a(n-1)
      c(ja+1) = -a(n  )

      a = c
      return
      end subroutine dfft2

!     ================
!     SUBROUTINE DFFT3
!     ================

      subroutine dfft3(a,trigs,n)
      parameter(SIN60 = 0.866025403784438D0)
      dimension a(n),c(n),trigs(n)

      ja = 1              !  1
      jb = 2 * (n/3)  + 1 ! 65
      jc = jb             ! 65

      c(ja  ) = a(1) + a(2) + a(3)
      c(ja+1) = 0.0
      c(jb  ) = a(1) - 0.5 * (a(2) + a(3))
      c(jb+1) =      SIN60 * (a(3) - a(2))

      ja = 3          !  3, 5, 7, ... ,31
      jb = jb + 2     ! 67,69,71, ... ,95
      jc = jc - 2     ! 63,61,59, ... ,35

      do i = 4 , n-8 , 6 ! 88
         c1 = trigs(ja  )
         s1 = trigs(ja+1)
         c2 = trigs(ja+ja-1)
         s2 = trigs(ja+ja  )
         a1 = (c1*a(i+1)+s1*a(i+4))+(c2*a(i+2)+s2*a(i+5))
         b1 = (c1*a(i+4)-s1*a(i+1))+(c2*a(i+5)-s2*a(i+2))
         a2 = a(i  ) - 0.5 * a1
         b2 = a(i+3) - 0.5 * b1
         a3 = SIN60*((c1*a(i+1)+s1*a(i+4))-(c2*a(i+2)+s2*a(i+5)))
         b3 = SIN60*((c1*a(i+4)-s1*A(i+1))-(c2*a(i+5)-s2*a(i+2)))
         c(ja  ) = a(i  ) + a1
         c(ja+1) = a(i+3) + b1
         c(jb  ) = a2 + b3
         c(jb+1) = b2 - a3
         c(jc  ) = a2 - b3
         c(jc+1) =-b2 - a3
         ja = ja + 2
         jb = jb + 2
         jc = jc - 2
      enddo

      if (ja <= jc) then ! ja=33  jc=33
         c(ja  ) = a(n-2) + 0.5 * (a(n-1) - a(n)) ! 33
         c(ja+1) =       -SIN60 * (a(n-1) + a(n)) ! 34
      endif
      a(:) = c(:)
      return
      end subroutine dfft3
```



```
!     ================
!     SUBROUTINE DFFT4
!     ================

      subroutine dfft4(a,trigs,n,lot,la)
      dimension a(n,lot),c(n,lot),trigs(n)
      la = la / 4

      i1 = la
      i2 = la + i1
      i3 = la + i2
      i4 = la + i3
      i5 = la + i4
      i6 = la + i5
      i7 = la + i6

      j1 = n/2 - la
      j2 = n - la
      j3 = j1
      j5 = j1 + la

      do i=1,la
         do l=1,lot
         a0p2 = a(i   ,l) + a(i2+i,l)
         a1p3 = a(i1+i,l) + a(i3+i,l)
         c(   i,l) = a0p2 + a1p3
         c(j2+i,l) = a0p2 - a1p3
         c(j1+i,l) = a(   i,l) - a(i2+i,l)
         c(j5+i,l) = a(i3+i,l) - a(i1+i,l)
         enddo
      enddo

      jink = 2 * la
      j0 = la
      j1 = j1 + jink
      j2 = j2 - jink
      j3 = j3 - jink
      j4 = j0 + la
      j5 = j1 + la
      j6 = j2 + la
      j7 = j3 + la

      ibase=4*la

      do 450 k=la,(n-4)/8,la
         kb=k+k
         kc=kb+kb
         kd=kc+kb
         c1=trigs(kb+1)
         s1=trigs(kb+2)
         c2=trigs(kc+1)
         s2=trigs(kc+2)
         c3=trigs(kd+1)
         s3=trigs(kd+2)

         i=ibase+1
         do j=1,la
            do l=1,lot
            a1p5 = c1 * a(i1+i,l) + s1 * a(i5+i,l)
            a2p6 = c2 * a(i2+i,l) + s2 * a(i6+i,l)
            a3p7 = c3 * a(i3+i,l) + s3 * a(i7+i,l)
            a5m1 = c1 * a(i5+i,l) - s1 * a(i1+i,l)
            a6m2 = c2 * a(i6+i,l) - s2 * a(i2+i,l)
            a7m3 = c3 * a(i7+i,l) - s3 * a(i3+i,l)
            a0 = a(i,l) + a2p6
            a2 = a(i,l) - a2p6
            a1 = a1p5 + a3p7
            a3 = a3p7 - a1p5
```

```
            b0 = a(i4+i,l) + a6m2
            b2 = a(i4+i,l) - a6m2
            b1 = a5m1 + a7m3
            b3 = a5m1 - a7m3
            c(j0+j,l) = a0+a1
            c(j2+j,l) = a0-a1
            c(j4+j,l) = b0+b1
            c(j6+j,l) = b1-b0
            c(j1+j,l) = a2+b3
            c(j3+j,l) = a2-b3
            c(j5+j,l) = a3+b2
            c(j7+j,l) = a3-b2
            enddo
            i=i+1
         enddo

        ibase=ibase+8*la
        j0 = j0 + jink
        j1 = j1 + jink
        j2 = j2 - jink
        j3 = j3 - jink
        j4 = j0 + la
        j5 = j1 + la
        j6 = j2 + la
        j7 = j3 + la
450   continue
      if (j1 <= j2) then
         sin45=sqrt(0.5)
         i=ibase+1
         do j=1,la
            do l=1,lot
            a1p3 = sin45 * (a(i1+i,l) + a(i3+i,l))
            a1m3 = sin45 * (a(i1+i,l) - a(i3+i,l))
            c(j0+j,l) =  a(   i,l) + a1m3
            c(j1+j,l) =  a(   i,l) - a1m3
            c(j4+j,l) = -a(i2+i,l) - a1p3
            c(j5+j,l) =  a(i2+i,l) - a1p3
            enddo
            i=i+1
         enddo
      endif
      if (la == 1) then
         do l=1,lot
         a(1,l) = c(1,l)
         a(2,l) = 0.0
         a(3:n,l) = c(2:n-1,l)
         enddo
      else
         a = c
      endif
      return
      end subroutine dfft4

!     ================
!     SUBROUTINE DFFT8
!     ================

      subroutine dfft8(a,c,n,lot)
      real a(n*lot),c(n*lot)
      la = n / 8
      z  = 1.0 / n
      zsin45 = z * sqrt(0.5)

      do i=0,la*lot-1
         i0 = (i/la) * n + mod(i,la) + 1
         i1 = i0 + la
         i2 = i1 + la
         i3 = i2 + la
         i4 = i3 + la
```



```
         i5 = i4 + la
         i6 = i5 + la
         i7 = i6 + la

         a0p4 =  a(i0) + a(i4)
         a1p5 =  a(i1) + a(i5)
         a2p6 =  a(i2) + a(i6)
         a3p7 =  a(i3) + a(i7)
         a5m1 =  a(i5) - a(i1)
         a7m3 =  a(i7) - a(i3)
         a0m4 = (a(i0) - a(i4)) * z
         a6m2 = (a(i6) - a(i2)) * z

         a0p4p2p6 = a0p4 + a2p6
         a1p5p3p7 = a1p5 + a3p7
         a7m3p5m1 = (a7m3 + a5m1) * zsin45
         a7m3m5m1 = (a7m3 - a5m1) * zsin45

         c(i0) = z * (a0p4p2p6 + a1p5p3p7)
         c(i7) = z * (a0p4p2p6 - a1p5p3p7)
         c(i3) = z * (a0p4 - a2p6)
         c(i4) = z * (a3p7 - a1p5)
         c(i1) = a0m4 + a7m3m5m1
         c(i5) = a0m4 - a7m3m5m1
         c(i2) = a7m3p5m1 + a6m2
         c(i6) = a7m3p5m1 - a6m2
      enddo
      return
      end subroutine dfft8

!     ================
!     SUBROUTINE IFFT4
!     ================

      subroutine ifft4(c,trigs,n,lot,la)
      dimension a(n,lot),c(n,lot),trigs(n)

      if (la == 1) then
         a(1,:) = 0.5 * c(1,:)
         a(n,:) = 0.0
         a(2:n-1,:) = c(3:n,:)
      else
         a = c
      endif

      m=n/4
      kstop=(n-4)/8

      i1 = n/2 - la
      i2 = n   - la
      i5 = i1  + la

      j1 = la
      j2 = la+j1
      j3 = la+j2
      j4 = la+j3
      j5 = la+j4
      j6 = la+j5
      j7 = la+j6

      do i=1,la
      do l=1,lot
         c(   i,l) = a(i,l) + a(i2+i,l) + a(i1+i,l)
         c(j1+i,l) = a(i,l) - a(i2+i,l) - a(i5+i,l)
         c(j2+i,l) = a(i,l) + a(i2+i,l) - a(i1+i,l)
         c(j3+i,l) = a(i,l) - a(i2+i,l) + a(i5+i,l)
      enddo
      enddo
```

```
      iink  = 2 * la
      jbase = 4 * la + 1
      i0    = la
      i1    = i0 + n/2
      i2    = n - 3 * la
      i3    = i2 - n/2
      i4    = i0 + la
      i5    = i1 + la
      i6    = i2 + la
      i7    = i3 + la

      do 450 k=la,kstop,la
        kb=k+k
        kc=kb+kb
        kd=kc+kb
        c1=trigs(kb+1)
        s1=trigs(kb+2)
        c2=trigs(kc+1)
        s2=trigs(kc+2)
        c3=trigs(kd+1)
        s3=trigs(kd+2)
        do i = 1 , la
           j = jbase
           do l=1,lot
           a0p2 = a(i0+i,l) + a(i2+i,l)
           a0m2 = a(i0+i,l) - a(i2+i,l)
           a1p3 = a(i1+i,l) + a(i3+i,l)
           a1m3 = a(i1+i,l) - a(i3+i,l)
           a4p6 = a(i4+i,l) + a(i6+i,l)
           a4m6 = a(i4+i,l) - a(i6+i,l)
           a5p7 = a(i5+i,l) + a(i7+i,l)
           a5m7 = a(i5+i,l) - a(i7+i,l)

           a0p2m1p3 = a0p2 - a1p3
           a4m6m5m7 = a4m6 - a5m7

           c(   j,l) = a0p2 + a1p3
           c(j4+j,l) = a4m6 + a5m7
           c(j2+j,l) = c2 * a0p2m1p3 - s2 * a4m6m5m7
           c(j6+j,l) = s2 * a0p2m1p3 + c2 * a4m6m5m7
           c(j1+j,l) = c1*(a0m2-a5p7)-s1*(a4p6+a1m3)
           c(j5+j,l) = s1*(a0m2-a5p7)+c1*(a4p6+a1m3)
           c(j3+j,l) = c3*(a0m2+a5p7)-s3*(a4p6-a1m3)
           c(j7+j,l) = s3*(a0m2+a5p7)+c3*(a4p6-a1m3)
           enddo
           jbase=jbase+1
        enddo
        i0 = i0 + iink
        i1 = i1 + iink
        i2 = i2 - iink
        i3 = i3 - iink
        i4 = i4 + iink
        i5 = i5 + iink
        i6 = i6 - iink
        i7 = i7 - iink
        jbase=jbase+7*la
450     continue

      if (i1 <= i2) then
         sin45=sqrt(0.5)
         do i=1,la
            j=jbase
            do l=1,lot
            c(   j,l)=a(i0+i,l)+a(i1+i,l)
            c(j1+j,l)=sin45*((a(i0+i,l)-a(i1+i,l))-(a(la+i0+i,l)+a(la+i1+i,l)))
            c(j2+j,l)=a(la+i1+i,l)-a(la+i0+i,l)
            c(j3+j,l)=-sin45*((a(i0+i,l)-a(i1+i,l))+(a(la+i0+i,l)+a(la+i1+i,l)))
            enddo
            jbase=jbase+1
```



```
         enddo
      endif
      la = la * 4
      return
      end subroutine ifft4

!     ================
!     SUBROUTINE IFFT2
!     ================

      subroutine ifft2(a,trigs,n,lot,la)
      dimension a(n,lot),c(n,lot),trigs(n)

      c(1,:) = 0.5 * a(1,:)
      c(2,:) = c(1,:)

      ia    =   3
      ib    = n-1

      do j = 3 , n-5 , 4
         c1 = trigs(ia  )
         s1 = trigs(ia+1)
         do l=1,lot
         amb = a(ia  ,l) - a(ib  ,l)
         apb = a(ia+1,l) + a(ib+1,l)
         c(j  ,l) = a(ia  ,l) + a(ib  ,l)
         c(j+2,l) = a(ia+1,l) - a(ib+1,l)
         c(j+1,l) = c1 * amb - s1 * apb
         c(j+3,l) = s1 * amb + c1 * apb
         enddo
         ia = ia + 2
         ib = ib - 2
      enddo
      c(n-1,:) =  a(ia  ,:)
      c(n  ,:) = -a(ia+1,:)

      a(:,:) = c(:,:)
      la = 2
      return
      end subroutine ifft2

!     ================
!     SUBROUTINE IFFT3
!     ================

      subroutine ifft3(a,trigs,n,lot,la)
      dimension a(n,lot),c(n,lot),trigs(n)
      parameter(SIN60 = 0.866025403784438D0)

      ib = 2 * (n/3) + 1

      c(1,:) = 0.5 * a(1,:) + a(ib,:)
      c(2,:) = 0.5 * a(1,:) - 0.5 * a(ib,:) - SIN60 * a(ib+1,:)
      c(3,:) = 0.5 * a(1,:) - 0.5 * a(ib,:) + SIN60 * a(ib+1,:)

      ia = 3
      ic = ib - 2
      ib = ib + 2

      do j = 4 , n-8 , 6
         c1 = trigs(ia  )
         s1 = trigs(ia+1)
         c2 = trigs(ia+ia-1)
         s2 = trigs(ia+ia  )

         do l = 1 , lot
            hbpc = a(ia  ,l) - 0.5 * (a(ib  ,l) + a(ic  ,l))
            hbmc = a(ia+1,l) - 0.5 * (a(ib+1,l) - a(ic+1,l))
            sbmc = SIN60 * (a(ib  ,l) - a(ic  ,l))
```

```
            sbpc = SIN60 * (a(ib+1,l) + a(ic+1,l))

            c(j  ,l) = a(ia  ,l) + a(ib  ,l) + a(ic  ,l)
            c(j+3,l) = a(ia+1,l) + a(ib+1,l) - a(ic+1,l)
            c(j+1,l) = c1 * (hbpc-sbpc) - s1 * (hbmc+sbmc)
            c(j+4,l) = s1 * (hbpc-sbpc) + c1 * (hbmc+sbmc)
            c(j+2,l) = c2 * (hbpc+sbpc) - s2 * (hbmc-sbmc)
            c(j+5,l) = s2 * (hbpc+sbpc) + c2 * (hbmc-sbmc)
         enddo
         ia = ia + 2
         ib = ib + 2
         ic = ic - 2
      enddo

      c(n-2,:) = a(ia,:)
      c(n-1,:) =   0.5 * a(ia,:) - SIN60 * a(ia+1,:)
      c(n  ,:) = - 0.5 * a(ia,:) - SIN60 * a(ia+1,:)

      a(:,:)  = c(:,:)
      la = 3
      return
      end subroutine ifft3

!     ================
!     SUBROUTINE IFFT8
!     ================

      subroutine ifft8(a,c,n,lot)
      parameter(SQRT2 = 1.414213562373095D0)
      dimension a(n*lot),c(n*lot)
      la = n / 8

      do i=0,la*lot-1
         i0 = (i/la) * n + mod(i,la) + 1
         i1 = i0 + la
         i2 = i1 + la
         i3 = i2 + la
         i4 = i3 + la
         i5 = i4 + la
         i6 = i5 + la
         i7 = i6 + la

         a0p7 = a(i0) + a(i7)
         a0m7 = a(i0) - a(i7)
         a1p5 = a(i1) + a(i5)
         a1m5 = a(i1) - a(i5)
         a2p6 = a(i2) + a(i6)
         a2m6 = a(i2) - a(i6)

         a0p7p3   = a0p7 + a(i3)
         a0p7m3   = a0p7 - a(i3)
         a0m7p4   = 2.0 * (a0m7 + a(i4))
         a0m7m4   = 2.0 * (a0m7 - a(i4))
         a1m5p2p6 = SQRT2 * (a1m5 + a2p6)
         a1m5m2p6 = SQRT2 * (a1m5 - a2p6)

         c(i0)  = 2.0 * (a0p7p3 + a1p5)
         c(i2)  = 2.0 * (a0p7m3 - a2m6)
         c(i4)  = 2.0 * (a0p7p3 - a1p5)
         c(i6)  = 2.0 * (a0p7m3 + a2m6)

         c(i1)  = a0m7m4 + a1m5m2p6
         c(i3)  = a0m7p4 - a1m5p2p6
         c(i5)  = a0m7m4 - a1m5m2p6
         c(i7)  = a0m7p4 + a1m5p2p6
      enddo
      return
      end
```

```
      module restartmod
      integer, parameter :: nresdim  = 200   ! Max number of records
      integer, parameter :: nreaunit =  33   ! FORTRAN unit for reading
      integer, parameter :: nwriunit =  34   ! FORTRAN unit for writing
      integer            :: nexcheck =   1   ! Extended checks
      integer            :: nresnum  =   0   ! Actual number of records
      integer            :: nlastrec =   0   ! Last read record
      integer            :: nud      =   6   ! Standard output
      character (len=16) :: yresnam(nresdim) ! Array of record names
      end module restartmod

!     ======================
!     SUBROUTINE RESTART_INI
!     ======================

      subroutine restart_ini(lrestart,yrfile)
      use restartmod

      logical :: lrestart
      character (len=*)  :: yrfile
      character (len=16) :: yn ! variable name

      inquire(file=yrfile,exist=lrestart)
      if (lrestart) then
         open(nreaunit,file=yrfile,form='unformatted')
         do
            read (nreaunit,IOSTAT=iostat) yn
            if (iostat /= 0) exit
            nresnum = nresnum + 1
            yresnam(nresnum) = yn
            read (nreaunit,IOSTAT=iostat)
            if (iostat /= 0) exit
            if (nresnum >= nresdim) then
               write(nud,*) 'Too many variables in restart file'
               write(nud,*) 'Increase NRESDIM in module restartmod'
               write(nud,*) '*** Error Stop ***'
               stop
            endif
         enddo

         write(nud,'(a,i4,3a/)') 'Found ',nresnum, &
               ' variables in file <',trim(yrfile),'>'
         do j = 1 , nresnum
            write(nud,'(i4," : ",8x,1x,a)') j,yresnam(j)
         enddo
         nlastrec = nresnum
      endif ! (lrestart)

!     file must be left open for further access

      return
      end subroutine restart_ini


!     ==========================
!     SUBROUTINE RESTART_PREPARE
!     ==========================

      subroutine restart_prepare(ywfile)
      use restartmod

      character (len=*) :: ywfile

      open(nwriunit,file=ywfile,form='unformatted')

      return
      end subroutine restart_prepare
```



```
!     ======================
!     SUBROUTINE RESTART_STOP
!     ======================

      subroutine restart_stop
      use restartmod

      close (nreaunit)
      close (nwriunit)

      return
      end subroutine restart_stop


!     ============================
!     SUBROUTINE GET_RESTART_INTEGER
!     ============================

      subroutine get_restart_integer(yn,kv)
      use restartmod

      character (len=*) :: yn
      integer :: kv

      do j = 1 , nresnum
         if (trim(yn) == trim(yresnam(j))) then
            call fileseek(yn,j)
            read (nreaunit) kv
            nlastrec = nlastrec + 1
            return
         endif
      enddo
      if (nexcheck == 1) then
         write(nud,*) '*** Error in get_restart_integer ***'
         write(nud,*) 'Requested integer {',yn,'} was not found'
         stop
      endif
      return
      end subroutine get_restart_integer


!     ==========================
!     SUBROUTINE GET_RESTART_ARRAY
!     ==========================

      subroutine get_restart_array(yn,pa,k1,k2,k3)
      use restartmod

      character (len=*) :: yn
      real :: pa(k2,k3)

      do j = 1 , nresnum
         if (trim(yn) == trim(yresnam(j))) then
            call fileseek(yn,j)
            read (nreaunit) pa(1:k1,:)
            nlastrec = nlastrec + 1
            return
         endif
      enddo
      if (nexcheck == 1) then
         write(nud,*) '*** Error in get_restart_array ***'
         write(nud,*) 'Requested array {',yn,'} was not found'
         stop
      endif
      return
      end subroutine get_restart_array


!     ============================
```

```
!     SUBROUTINE PUT_RESTART_INTEGER
!     ==============================

      subroutine put_restart_integer(yn,kv)
      use restartmod

      character (len=*)  :: yn
      character (len=16) :: yy
      integer :: kv

      yy = yn
      write(nwriunit) yy
      write(nwriunit) kv
      return
      end subroutine put_restart_integer


!     ============================
!     SUBROUTINE PUT_RESTART_ARRAY
!     ============================

      subroutine put_restart_array(yn,pa,k1,k2,k3)
      use restartmod

      character (len=*)  :: yn
      character (len=16) :: yy
      integer :: k1,k2,k3
      real :: pa(k2,k3)

      yy = yn
      write(nwriunit) yy
      write(nwriunit) pa(1:k1,1:k3)
      return
      end subroutine put_restart_array


!     ==================
!     SUBROUTINE FILESEEK
!     ==================

      subroutine fileseek(yn,k)
      use restartmod

      character (len=*)  :: yn
      character (len=16) :: yy

!     write(nud,*) 'Pos:',nlastrec,'   Want:',k
      if (k <= nlastrec) then
!        write(nud,*) 'Rewinding'
         rewind nreaunit
         nlastrec = 0
      endif

      do
         read (nreaunit,iostat=iostat) yy
         if (iostat /= 0) exit
         if (trim(yn) == trim(yy)) return ! success
         read (nreaunit,iostat=iostat)    ! skip data
         if (iostat /= 0) exit
         nlastrec = nlastrec + 1
      enddo
      write(nud,*) 'Variable <',trim(yn),'> not in restart file'
      return
      end


!     ========================
!     SUBROUTINE CHECK_EQUALITY
!     ========================
```



```
      subroutine check_equality(yn,pa,pb,k1,k2)
      character (len=*) :: yn
      real :: pa(k1,k2)
      real :: pb(k1,k2)

      do j2 = 1 , k2
      do j1 = 1 , k1
         if (pa(j1,j2) /= pb(j1,j2)) then
            write(nud,*) 'No Equality on ',yn,'(',j1,',',j2,')',pa(j1,j2),pb(j1,j2)
            return
         endif
      enddo
      enddo
      write(nud,*) 'Array {',yn,'} is OK'
      return
      end


!     ==================
!     SUBROUTINE VARSEEK
!     ==================

      subroutine varseek(yn,knum)
      use restartmod

      character (len=*)  :: yn
      character (len=16) :: ytmp
      integer :: k, knum

      knum = 0
      do k = 1,nresdim
         ytmp = yresnam(k)
         if (trim(yn) == trim(ytmp)) then
            knum = k
         endif
      enddo
      return
      end
```

```
      module mpimod
      use pumamod
      use mpi

      integer :: mpi_itype = MPI_INTEGER4
      integer :: mpi_rtype = MPI_REAL4
      integer :: mpi_ltype = MPI_LOGICAL

      end module mpimod
!
!     interface routines to MPI:
!


!     ================
!     SUBROUTINE MPBCI
!     ================

      subroutine mpbci(k) ! broadcast 1 integer
      use mpimod
      integer :: k(*)

      call mpi_bcast(k,1,mpi_itype,NROOT,myworld,mpinfo)

      return
      end subroutine mpbci

!     =================
!     SUBROUTINE MPBCIN
!     =================

      subroutine mpbcin(k,n) ! broadcast n integer
      use mpimod

      integer :: k(n)

      call mpi_bcast(k,n,mpi_itype,NROOT,myworld,mpinfo)

      return
      end subroutine mpbcin

!     ================
!     SUBROUTINE MPBCR
!     ================

      subroutine mpbcr(p) ! broadcast 1 real
      use mpimod
      real :: p(*)

      call mpi_bcast(p,1,mpi_rtype,NROOT,myworld,mpinfo)

      return
      end subroutine mpbcr

!     =================
!     SUBROUTINE MPBCRN
!     =================

      subroutine mpbcrn(p,n) ! broadcast n real
      use mpimod

      real :: p(n)

      call mpi_bcast(p,n,mpi_rtype,NROOT,myworld,mpinfo)

      return
      end subroutine mpbcrn

!     ================
```



```
!     SUBROUTINE MPBCL
!     ================

      subroutine mpbcl(l) ! broadcast 1 logical
      use mpimod
      logical :: l(*)

      call mpi_bcast(l,1,mpi_ltype,NROOT,myworld,mpinfo)

      return
      end subroutine mpbcl

!     =================
!     SUBROUTINE MPSCIN
!     =================

      subroutine mpscin(k,n) ! scatter n integer
      use mpimod

      integer :: k(*)

      call mpi_scatter(k,n,mpi_itype,k,n,mpi_itype                  &
     &                ,NROOT,myworld,mpinfo)

      return
      end subroutine mpscin

!     =================
!     SUBROUTINE MPSCRN
!     =================

      subroutine mpscrn(p,n) ! scatter n real
      use mpimod

      real :: p(*)

      call mpi_scatter(p,n,mpi_rtype,p,n,mpi_rtype,NROOT,myworld,mpinfo)

      return
      end subroutine mpscrn

!     =================
!     SUBROUTINE MPSCDN
!     =================

      subroutine mpscdn(p,n) ! scatter n double precision
      use mpimod

      real (kind=8) :: p(*)

      call mpi_scatter(p,n,MPI_REAL8,p,n,MPI_REAL8,NROOT,myworld,mpinfo)

      return
      end subroutine mpscdn

!     =================
!     SUBROUTINE MPSCGP
!     =================

      subroutine mpscgp(pf,pp,klev) ! scatter gridpoint fields
      use mpimod

      real :: pf(NUGP,klev)
      real :: pp(NHOR,klev)

      do jlev = 1 , klev
         call mpi_scatter(pf(:,jlev),NHOR,mpi_rtype,                &
     &                    pp(:,jlev),NHOR,mpi_rtype,                &
     &                    NROOT,myworld,mpinfo)
```

```
      enddo

      return
      end subroutine mpscgp

!     ==================
!     SUBROUTINE MPGAGP
!     ==================

      subroutine mpgagp(pf,pp,klev) ! gather gridpoint fields
      use mpimod

      real :: pf(NLON*NLAT,klev)
      real :: pp(NHOR,klev)

      do jlev = 1 , klev
         call mpi_gather(pp(:,jlev),NHOR,mpi_rtype,                    &
     &                   pf(:,jlev),NHOR,mpi_rtype,                    &
     &                   NROOT,myworld,mpinfo)
      enddo

      return
      end subroutine mpgagp

!     ====================
!     SUBROUTINE MPGALLGP
!     ====================

      subroutine mpgallgp(pf,pp,klev) ! gather gritpoint to all
      use mpimod

      real :: pf(NLON*NLAT,klev)
      real :: pp(NHOR,klev)

      do jlev = 1 , klev
         call mpi_allgather(pp(:,jlev),NHOR,mpi_rtype,                 &
     &                      pf(:,jlev),NHOR,mpi_rtype,                 &
     &                      myworld,mpinfo)
      enddo

      return
      end subroutine mpgallgp

!     ==================
!     SUBROUTINE MPSCSP
!     ==================

      subroutine mpscsp(pf,pp,klev) ! scatter spectral fields
      use mpimod

      real :: pf(NESP,klev)
      real :: pp(NSPP,klev)

      do jlev = 1 , klev
         call mpi_scatter(pf(:,jlev),NSPP,mpi_rtype                    &
     &                   ,pp(:,jlev),NSPP,mpi_rtype                    &
     &                   ,NROOT,myworld,mpinfo)
      enddo

      return
      end subroutine mpscsp

!     ==================
!     SUBROUTINE MPGASP
!     ==================

      subroutine mpgasp(pf,pp,klev) ! gather spectral fields
      use mpimod
```



```
      real :: pf(NESP,klev)
      real :: pp(NSPP,klev)

      do jlev = 1 , klev
         call mpi_gather(pp(:,jlev),NSPP,mpi_rtype                     &
     &                   ,pf(:,jlev),NSPP,mpi_rtype                    &
     &                   ,NROOT,myworld,mpinfo)
      enddo

      return
      end subroutine mpgasp

!     ==================
!     SUBROUTINE MPGACS
!     ==================

      subroutine mpgacs(pcs) ! gather cross sections
      use mpimod

      real :: pcs(NLAT,NLEV)

      do jlev = 1 , NLEV
         call mpi_gather(pcs(:,jlev),NLPP,mpi_rtype                    &
     &                   ,pcs(:,jlev),NLPP,mpi_rtype                   &
     &                   ,NROOT,myworld,mpinfo)

      enddo
      return
      end subroutine mpgacs

!     ====================
!     SUBROUTINE MPGALLSP
!     ====================

      subroutine mpgallsp(pf,pp,klev) ! gather spectral to all
      use mpimod

      real :: pf(NESP,klev)
      real :: pp(NSPP,klev)

      do jlev = 1 , klev
         call mpi_allgather(pp(:,jlev),NSPP,mpi_rtype                  &
     &                      ,pf(:,jlev),NSPP,mpi_rtype                 &
     &                      ,myworld,mpinfo)
      enddo

      return
      end subroutine mpgallsp

!     ================
!     SUBROUTINE MPSUM
!     ================

      subroutine mpsum(psp,klev) ! sum spectral fields
      use mpimod

      real :: psp(NESP*klev)
      real :: tmp(NESP*klev)

      call mpi_reduce(psp,tmp,NESP*klev,mpi_rtype,MPI_SUM              &
     &               ,NROOT,myworld,mpinfo)
      if (mypid == NROOT) psp = tmp

      return
      end subroutine mpsum

!     ==================
!     SUBROUTINE MPSUMSC
!     ==================
```

```
      subroutine mpsumsc(psf,psp,klev) ! sum & scatter spectral
      use mpimod

      real :: psf(NESP,klev)
      real :: psp(NSPP,klev)

      do jlev = 1 , klev
         call mpi_reduce_scatter(psf(:,jlev),psp(:,jlev),nscatsp        &
     &                          ,mpi_rtype,MPI_SUM,myworld,mpinfo)
      enddo

      return
      end subroutine mpsumsc

!     ====================
!     SUBROUTINE MPSUMR
!     ====================

      subroutine mpsumr(pr,kdim) ! sum kdim reals
      use mpimod

      real pr(kdim)
      real tmp(kdim)

      call mpi_reduce(pr,tmp,kdim,mpi_rtype,MPI_SUM,NROOT,myworld,mpinfo)
      if (mypid == NROOT) pr = tmp

      return
      end subroutine mpsumr

!     ====================
!     SUBROUTINE MPSUMBCR
!     ====================

      subroutine mpsumbcr(pr,kdim) ! sum & broadcast kdim reals
      use mpimod

      real :: pr(kdim)
      real :: tmp(kdim)

      call mpi_allreduce(pr,tmp,kdim,mpi_rtype,MPI_SUM,myworld,mpinfo)
      pr = tmp

      return
      end subroutine mpsumbcr

!     ==================
!     SUBROUTINE MPSTART
!     ==================

      subroutine mpstart ! initialization
      use mpimod
      integer :: itest = 0
      real    :: rtest = 0.0

      if (kind(itest) == 8) mpi_itype = MPI_INTEGER8
      if (kind(rtest) == 8) mpi_rtype = MPI_REAL8

      call mpi_init(mpinfo)
      myworld=MPI_COMM_WORLD

      call mpi_comm_size(myworld,npro ,mpinfo)
      call mpi_comm_rank(myworld,mypid,mpinfo)
      allocate(ympname(npro)) ; ympname(:) = ' '
      call mpi_get_processor_name(ympname(1),ilen,mpinfo)

      call mpi_gather(ympname,80,MPI_CHARACTER,   &
                      ympname,80,MPI_CHARACTER,   &
```



```
                      NROOT,myworld,mpinfo)

      return
      end subroutine mpstart

!     ==================
!     SUBROUTINE MPSTOP
!     ==================

      subroutine mpstop
      use mpimod

      call mpi_barrier(myworld,mpinfo)
      call mpi_finalize(mpinfo)

      return
      end subroutine mpstop

!     ====================
!     SUBROUTINE MPREADGP
!     ====================

      subroutine mpreadgp(ktape,p,kdim,klev)
      use mpimod

      real p(kdim,klev)
      real z(NLON*NLAT,klev)

      z = 0.0

      if (mypid == NROOT) read (ktape) z(1:NLON*NLAT,:)

      if (kdim == NHOR) then
       call mpscgp(z,p,klev)
      else
       if (mypid == NROOT) p = z
      endif

      return
      end subroutine mpreadgp

!     ====================
!     SUBROUTINE MPWRITEGP
!     ====================

      subroutine mpwritegp(ktape,p,kdim,klev)
      use mpimod

      real p(kdim,klev)
      real z(NLON*NLAT,klev)

      if (kdim == NHOR) then
       call mpgagp(z,p,klev)
       if (mypid == NROOT) write(ktape) z(1:NLON*NLAT,:)
      else
       if (mypid == NROOT) write(ktape) p(1:NLON*NLAT,:)
      endif

      return
      end subroutine mpwritegp


!     ====================
!     SUBROUTINE MPWRITEGPH
!     ====================

      subroutine mpwritegph(ktape,p,kdim,klev,ihead)
      use mpimod
```

```
      real p(kdim,klev)
      real z(NLON*NLAT,klev)
!
      real(kind=4) :: zp(kdim,klev)
      real(kind=4) :: zz(NLON*NLAT,klev)
!

      integer ihead(8)

      if (kdim == NHOR) then
       call mpgagp(z,p,klev)
       if (mypid == NROOT) then
          write(ktape) ihead
          zz(:,:)=z(:,:)
          write(ktape) zz(1:NLON*NLAT,:)
       endif
      else
       if (mypid == NROOT) then
          write(ktape) ihead
          zp(:,:)=p(:,:)
          write(ktape) zp(1:NLON*NLAT,:)
       endif
      endif

      return
      end subroutine mpwritegph

!     ====================
!     SUBROUTINE MPREADSP
!     ====================

      subroutine mpreadsp(ktape,p,kdim,klev)
      use mpimod

      real p(kdim,klev)
      real z(NESP,klev)

      z = 0.0
      if (mypid == NROOT) read(ktape) z(1:NRSP,:)
      if (kdim == NSPP) then
         call mpscsp(z,p,klev)
      else
         if (mypid == NROOT) p = z
      endif

      return
      end subroutine mpreadsp

!     ====================
!     SUBROUTINE MPWRITESP
!     ====================

      subroutine mpwritesp(ktape,p,kdim,klev)
      use mpimod

      real p(kdim,klev)
      real z(NESP,klev)

      if (kdim == NSPP) then
         call mpgasp(z,p,klev)
         if (mypid == NROOT) write(ktape) z(1:NRSP,:)
      else
         if (mypid == NROOT) write(ktape) p(1:NRSP,:)
      endif

      return
      end subroutine mpwritesp
```



```
!     ====================
!     SUBROUTINE MPI_INFO
!     ====================

      subroutine mpi_info(nprocess,npid)     ! get nproc and pid
      use mpimod

      myworld=MPI_COMM_WORLD

      call mpi_comm_size(myworld,nprocess,mpinfo)
      call mpi_comm_rank(myworld,npid,mpinfo)

      return
      end subroutine mpi_info


!     ==================
!     SUBROUTINE MPGETSP
!     ==================

      subroutine mpgetsp(yn,p,kdim,klev)
      use mpimod

      character (len=*) :: yn
      real :: p(kdim,klev)
      real :: z(NESP,klev)

      z(:,:) = 0.0
      if (mypid == NROOT) call get_restart_array(yn,z,NRSP,NESP,klev)
      call mpscsp(z,p,klev)

      return
      end subroutine mpgetsp


!     ==================
!     SUBROUTINE MPGETGP
!     ==================

      subroutine mpgetgp(yn,p,kdim,klev)
      use mpimod

      character (len=*) :: yn
      real :: p(kdim,klev)
      real :: z(NUGP,klev)

      if (mypid == NROOT) call get_restart_array(yn,z,NUGP,NUGP,klev)
      call mpscgp(z,p,klev)

      return
      end subroutine mpgetgp


!     ==================
!     SUBROUTINE MPPUTSP
!     ==================

      subroutine mpputsp(yn,p,kdim,klev)
      use mpimod

      character (len=*) :: yn
      real :: p(kdim,klev)
      real :: z(NESP,klev)

      call mpgasp(z,p,klev)
      if (mypid == NROOT) call put_restart_array(yn,z,NRSP,NESP,klev)

      return
      end subroutine mpputsp
```

```
!     ==================
!     SUBROUTINE MPPUTGP
!     ==================

      subroutine mpputgp(yn,p,kdim,klev)
      use mpimod

      character (len=*) :: yn
      real :: p(kdim,klev)
      real :: z(NUGP,klev)

      call mpgagp(z,p,klev)
      if (mypid == NROOT) call put_restart_array(yn,z,NUGP,NUGP,klev)

      return
      end subroutine mpputgp


!!    ===================
!!    SUBROUTINE MPSURFGP
!!    ===================
!
!     subroutine mpsurfgp(yn,p,kdim,klev)
!     use mpimod
!
!     character (len=*) :: yn
!     real :: p(kdim,klev)
!     real :: z(NUGP,klev)
!
!     if (mypid == NROOT) call get_surf_array(yn,z,NUGP,NUGP,klev,iread)
!     call mpbci(iread)
!     if (iread == 1) call mpscgp(z,p,klev)
!
!     return
!     end subroutine mpsurfgp
!
!
!!    ====================
!!    SUBROUTINE MPSURFYEAR
!!    ====================
!
!     subroutine mpsurfyear(yn,p,kdim,kmon)
!     use mpimod
!
!     character (len=*) :: yn
!     real :: p(kdim,kmon)
!     real :: z(NUGP,kmon)
!
!     if (mypid == NROOT) call get_surf_year(yn,z,NUGP,kmon,iread)
!     call mpbci(iread)
!     if (iread == 1) call mpscgp(z,p,kmon)
!
!     return
!     end subroutine mpsurfyear
!
!
!!    ====================
!!    SUBROUTINE MP3DYEAR
!!    ====================
!
!     subroutine mp3dyear(yn,p,kdim,klev,kmon)
!     use mpimod
!
!     character (len=*) :: yn
!     real :: p(kdim,klev,kmon)
!     real :: z(NUGP,klev,kmon)
!
```



```
!     if (mypid == NROOT) call get_3d_year(yn,z,NUGP,klev,kmon,iread)
!     call mpbci(iread)
!     if (iread == 1) call mpscgp(z,p,klev*kmon)
!
!     return
!     end subroutine mp3dyear


!     ===================
!     SUBROUTINE MPMAXVAL
!     ===================

      subroutine mpmaxval(p,kdim,klev,pmax)
      use mpimod

      real :: p(kdim,klev)
      real :: pmax(1)
      real :: zmax(1)

      zmax = maxval(p(:,:))
      call mpi_allreduce(zmax,pmax,1,mpi_rtype,MPI_MAX,myworld,mpinfo)

      return
      end subroutine mpmaxval


!     ===================
!     SUBROUTINE MPSUMVAL
!     ===================

      subroutine mpsumval(p,kdim,klev,psum)
      use mpimod

      real :: p(kdim,klev)
      real :: psum(1)
      real :: zsum(1)

      zsum = sum(p(:,:))
      call mpi_allreduce(zsum,psum,1,mpi_rtype,MPI_SUM,myworld,mpinfo)

      return
      end subroutine mpsumval

!     Some dummy declarations that are use in multirun mode only

      subroutine mrdiff(p,d,n,l)
      real :: p(n)
      real :: d(n)
      return
      end

      subroutine mrsum(k) ! sum up 1 integer
      return
      end

      subroutine mrbci(k) ! broadcast 1 integer
      return
      end

      subroutine mrdimensions
      return
      end
```

```
!     ======================================
!     mpimod_dummy.f90
!     ----------------
!     This module replaces <mpimod.f90> for
!     single CPU runs
!
!     The module is shared by PUMA and PlaSim
!     ======================================

      subroutine mrdimensions
      return
      end

      subroutine mrdiff(p,d,n,l)
      real :: p(n)
      real :: d(n)
      return
      end

      subroutine mrsum(k) ! sum up 1 integer
      return
      end

      subroutine mrbci(k) ! broadcast 1 integer
      return
      end

      subroutine mpbci(k) ! broadcast 1 integer
      return
      end

      subroutine mpbcin(k,n) ! broadcast n integer
      integer :: k(n)
      return
      end

      subroutine mpbcr(p) ! broadcast 1 real
      return
      end

      subroutine mpbcrn(p,n) ! broadcast n real
      real :: p(n)
      return
      end

      subroutine mpbcl(k) ! broadcast 1 logical
      logical :: k
      return
      end

      subroutine mpscin(k,n) ! scatter n integer
      integer :: k(n)
      return
      end

      subroutine mpscrn(p,n) ! scatter n real
      real :: p(n)
      return
      end

      subroutine mpscdn(p,n) ! scatter n double precision
      real (kind=8) :: p(n)
      return
      end

      subroutine mpscsp(pf,pp,klev) ! scatter spectral fields
      use pumamod
      real pf(NESP,klev)
      real pp(NSPP,klev)
```



```
      pp(1:NSPP,1:klev) = pf(1:NSPP,1:klev)
      return
      end

      subroutine mpscgp(pf,pp,klev) ! scatter gridpoint fields
      use pumamod
      real pf(NLON*NLAT,klev)
      real pp(NHOR,klev)
      pp(1:NHOR,1:klev) = pf(1:NHOR,1:klev)
      return
      end

      subroutine mpgasp(pf,pp,klev) ! gather spectral fields
      use pumamod
      real pf(NESP,klev)
      real pp(NSPP,klev)
      pf(1:NSPP,1:klev) = pp(1:NSPP,1:klev)
      return
      end

      subroutine mpgagp(pf,pp,klev) ! gather gridpoint fields
      use pumamod
      real pf(NHOR,klev)
      real pp(NHOR,klev)
      pf = pp
      return
      end

      subroutine mpgacs(pcs) ! gather cross sections
      return
      end

      subroutine mpgallsp(pf,pp,klev) ! gather spectral to all
      use pumamod
      real pf(NESP,klev)
      real pp(NSPP,klev)
      pf(1:NSPP,1:klev) = pp(1:NSPP,1:klev)
      return
      end

      subroutine mpsum(psp,klev) ! sum spectral fields
      return
      end

      subroutine mpsumsc(psf,psp,klev) ! sum & scatter spectral
      use pumamod
      real psf(NESP,klev)
      real psp(NSPP,klev)
      psp(1:NSPP,1:klev) = psf(1:NSPP,1:klev)
      return
      end

      subroutine mpsumr(pr,kdim) ! sum kdim reals
      return
      end subroutine mpsumr

      subroutine mpsumbcr(pr,kdim) ! sum & broadcast kdim reals
      return
      end

      subroutine mpstart ! initialization
      use pumamod
      npro = 1
      return
      end

      subroutine mpstop
      return
      end
```

```
      subroutine mpreadsp(ktape,p,kdim,klev)
      real p(kdim,klev)
      read (ktape) p
      return
      end

      subroutine mpreadgp(ktape,p,kdim,klev)
      real p(kdim,klev)
      read (ktape) p
      return
      end

      subroutine mpwritesp(ktape,p,kdim,klev)
      real p(kdim,klev)
      write (ktape) p
      return
      end

      subroutine mpwritegp(ktape,p,kdim,klev)
      real p(kdim,klev)
      write (ktape) p
      return
      end

      subroutine mpwritegph(ktape,p,kdim,klev,ihead)
      real :: p(kdim,klev)
      integer :: ihead(8)
      write (ktape) ihead
      write (ktape) p

      return
      end


      subroutine mpi_info(nprocess,pid)     ! get nproc and pid
      integer nprocess, pid
      nprocess = 1
      pid = 0
      return
      end subroutine mpi_info


      subroutine mpgetsp(yn,p,kdim,klev)
      character (len=*) :: yn
      real :: p(kdim,klev)
      call get_restart_array(yn,p,kdim,kdim,klev)
      return
      end subroutine mpgetsp


      subroutine mpgetgp(yn,p,kdim,klev)
      character (len=*) :: yn
      real :: p(kdim,klev)
      call get_restart_array(yn,p,kdim,kdim,klev)
      return
      end subroutine mpgetgp


      subroutine mpputsp(yn,p,kdim,klev)
      character (len=*) :: yn
      real :: p(kdim,klev)
      call put_restart_array(yn,p,kdim,kdim,klev)
      return
      end subroutine mpputsp


      subroutine mpputgp(yn,p,kdim,klev)
      character (len=*) :: yn
```



```
      real :: p(kdim,klev)
      call put_restart_array(yn,p,kdim,kdim,klev)
      return
      end subroutine mpputgp


!      subroutine mpsurfgp(yn,p,kdim,klev)
!      character (len=*) :: yn
!      real :: p(kdim,klev)
!      call get_surf_array(yn,p,kdim,kdim,klev,iread)
!      return
!      end subroutine mpsurfgp
!
!
!      subroutine mpsurfyear(yn,p,kdim,kmon)
!      character (len=*) :: yn
!      real :: p(kdim,kmon)
!      call get_surf_year(yn,p,kdim,kmon,iread)
!      return
!      end subroutine mpsurfyear
!
!
!      subroutine mp3dyear(yn,p,kdim,klev,kmon)
!      character (len=*) :: yn
!      real :: p(kdim,klev,kmon)
!      call get_3d_year(yn,p,kdim,klev,kmon,iread)
!      return
!      end subroutine mp3dyear


      subroutine mpmaxval(p,kdim,klev,pmax)
      real :: p(kdim,klev)
      pmax = maxval(p(:,:))
      return
      end subroutine mpmaxval


      subroutine mpsumval(p,kdim,klev,psum)
      real :: p(kdim,klev)
      psum = sum(p(:,:))
      return
      end subroutine mpsumval
```

```
# Makefile for PUMA without MPI and GUI
# Xinyu Wen, Peking Univ, Mar/23/2017

F90=gfortran
MPIMOD=mpimod_stub
FFTMOD=fftmod

# Makefile for PUMA with MPI, without GUI
# Xinyu Wen, Peking Univ, Mar/27/2017

#F90=mpif90
#MPIMOD=mpimod
#FFTMOD=fftmod

OBJ=${MPIMOD}.o ${FFTMOD}.o puma.o legsym.o restartmod.o gaussmod.o

%.o : %.f90
        ${F90} -c -O3 $<

puma.x: $(OBJ)
        ${F90} -o puma.x $(OBJ)

puma.o:          puma.f90
${FFTMOD}.o:     ${FFTMOD}.f90
${MPIMOD}.o:     ${MPIMOD}.f90 puma.o
legsym.o:        legsym.f90
restartmod.o:    restartmod.f90
gaussmod.o:      gaussmod.f90

clean:
        rm -f *.o *.mod
```

```
# Change Log

## 2017-Mar-29: Make "doc" directory and produce PDF for all sources

Creat "doc" directory and move LICENSE and readme.md into it. Make "make_srcpdf" s
cript to produce a PDF book for all sources in ../src and readme.md.

## 2017-Mar-28: Rename directory puma to src

Rename "puma", the directory of all fortran source code, into "src", to make the
directory structure neater.

## 2017-Mar-27: Add MPI function

Add mpimod.f90 and make minor modification on makefile, so that I can deliver MP
I runs using 4 CPUs. Seems the excutable with MPI still can be run with 1 single
 CPU, like this: "./puma_mpi.x 32 10". Normally, the MPI run can be done by "mpiexec -
np 4 puma_mpi.x 32 10".

The running speed, under the unif of "simulate model years per day",  was tested on my I
ntel i5 CPU with 4 processors, with gfortran as the compilor and -O3 as the opti
mal argument.

| `gfortran -O3` |   T21 |  T42 |
|:---------------|------:|-----:|
| 1 CPU (No MPI) |  4373 |  299 |
| 4 CPU (MPI)    | 13918 |  782 |

Note that the 1st number here (4373) is 1755 tested on pkuclimate.club, an 5-yea
r-old Intel i3 Xeon CPU.

## 2017-Mar-25: Add Post-Processing (pp)

Add directory "pp" for the post-processing excutable "burn7.x" and 2 associated na
melist, one for 3 surface variables, another for 10 multi-level variables. To co
mpile burn7.x you should apt install libnetcdf-cxx-legacy-dev first.

## 2017-Mar-25: Remove GUI-related code
Remove guimod_stub.f90, and comment out all the GUI-related code in puma.f90 wit
h a description "XW(Mar/25/2017) to remove GUI:". Please note I did not delete any lin
es, just comment out. The number of source code were shinked from 7 to 6.

## 2017-Mar-23: Make PUMA running

Successfully make PUMA running on 1CPU, with fake MPI and GUI interface (stub).
I changed "nresources" line in subroutine "epilog" of puma.f90, which is related to
retrieving process time, memory, and disk info used in pumax.c. They are just pr
inted into puma_diag file at the closing procedure of a run. No big deal. The co
de are marked with "XW".

## 2017-Mar-17: Make PLASIM17 running with MOST

### PUMA 4core 1year T21 run

        wensir@himalaya ~/model/puma17 $ ./most.x
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  puma.f90
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  mpimod.f90
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  fftmod.f90
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  guimod.f90
        mpicc -c -O3 pumax.c
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  legini.f90
        as -o legfast32.o legfast32.s
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  restartmod.f90
        mpif90 -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -finit
-real=snan  gaussmod.f90
        mpif90 -o puma.x -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none
 -finit-real=snan
                mpimod.o fftmod.o guimod.o pumax.o legini.o legfast32.o puma.o re
startmod.o gaussmod.o
                -L/usr/lib/X11 -lX11

        === Success: Launched process most_puma_run ===


### PUMA 1core 1year T21 run

        wensir@himalaya ~/model/puma17 $ ./most.x
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  puma.f90
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  mpimod_stub.f90
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  fftmod.f90
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  guimod.f90
        gcc   -c -O3 -I /usr/lib/X11/include pumax.c
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  legini.f90
        as -o legfast32.o legfast32.s
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  restartmod.f90
        gfortran -c -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=none -fin
it-real=snan  gaussmod.f90
        gfortran -o puma.x -O3 -ffpe-trap=invalid,zero,overflow -ffpe-summary=no
ne -finit-real=snan
                mpimod_stub.o fftmod.o guimod.o pumax.o legini.o legfast32.o pu
ma.o restartmod.o gaussmod.o
                -L/usr/lib/X11 -lX11

        === Success: Launched process most_puma_run ===


### Configure and Compile Source Code

        wensir@himalaya ~/model/puma17 $ ./configure.sh
        Found FORTRAN-90 compiler     at: /usr/bin/gfortran
        gfortran version  5.4
        Found Xlib (X11)              at: /usr/lib/X11
        Found C compiler              at: /usr/bin/gcc
        Found C++ compiler            at: /usr/bin/c++
        Found MPI FORTRAN-90 compiler at: /usr/bin/mpif90
        Found MPI C compiler          at: /usr/bin/mpicc
        Found MPI C++ compiler        at: /usr/bin/mpicxx
        Found GNU assembler           at: /usr/bin/as
        Fast Legendre Transformation    : ACTIVE (Linux)
        gfortran -o f90check.x f90check.f90
        gcc -o cc_check.x cc_check.c

        System info for <himalaya>
        Architecture: Linux himalaya 4.4.0-53-generic #74-Ubuntu SMP Fri Dec 2 1
5:59:10 UTC 2016 x86_64 GNU/Linux
        Endian format              : little endian
        FORTRAN control word size : 4 bytes
        FORTRAN integer size      : 4 bytes
        FORTRAN real    size      : 4 bytes
        C       int     size      : 4 bytes
        C       float   size      : 4 bytes
        C       long    size      : 8 bytes
        C  long long    size      : 8 bytes
        FORTRAN Compiler: gfortran
        C       Compiler: gcc
        gcc -o most.x most.c -I/usr/lib/X11/include -lm -L/usr/lib/X11 -lX11
```

```
       configuration complete - run <most.x>


### Install required library following README_UBUNTU

- apt install libx11-dev
- apt install openmpi-bin openmpi-common openmpi-doc libopenmpi-dev
```

Table of Contents