```
!**********************************************************************
! Copyright 1996, 1997, 2001, 2002, 2006, 2007, 2012, 2013            *
! Andreas Stohl, G. Wotawa, Bernard Legras                            *
!                                                                     *
! This file is part of TRACZILLA which is derived from FLEXPART V6    *
!                                                                     *
! TRACZILLA is free software: you can redistribute it and/or modify   *
! it under the terms of the GNU General Public License as published by*
! the Free Software Foundation, either version 3 of the License, or   *
! (at your option) any later version.                                 *
!                                                                     *
! TRACZILLA is distributed in the hope that it will be useful,        *
! but WITHOUT ANY WARRANTY; without even the implied warranty of      *
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the       *
! GNU General Public License for more details.                        *
!                                                                     *
! You should have received a copy of the GNU General Public License   *
! along with TRACZILLA.  If not, see <http://www.gnu.org/licenses/>.  *
!**********************************************************************

!===============================================================================
!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ TRACZILLA @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
!=====|==1=========2=========3=========4=========5=========6=========7=========8

module merra

!*******************************************************************************
! This modules allow to use MERRA data.
! Must be combined with isentropic code.
!*******************************************************************************

use commons
use netcdf
use isentrop_h
implicit none
private :: check
private :: NbLon, NbLat, NbPress
logical, save :: merra_data, merra_diab
private :: PSId, UId, VId, TId,  OMEGAId, LWRId, SWRId, LWRCLRId, SWRCLRId
private :: grid_lat, grid_lon, grid_ver
private :: PressLev, facT, LogPressLev, pmc_merra, area_coefft_merra

! iso_mass   specify we are using merra data

integer, save :: NbTime, NbLon, NbLat, NbPress
integer, save :: PSId, UId, VId, TId,  OMEGAId, LWRId, SWRId, LWRCLRId, SWRCLRId
real(dp), save :: missing_value
real(dp), save, allocatable :: PressLev(:), facT(:), LogPressLev(:), pmc_merra(:
)
real(dbl), save, allocatable :: grid_lat(:), grid_lon(:),grid_ver(:)
real(dp), save, allocatable :: area_coefft_merra(:)
!real, allocatable :: theta_prof(:,:)
integer, save :: NPMass

! PressLev [Pa]         pressure levels
! LogPressLev           log(p0/PressLev)
! pmc_merra             delta p factor in the calculation of density sigma
! facT                  factor (p0/PressLev)**kappa
! NPMass                number of upper levels on which mass equilibration is pe
rformed
! area_coefft_merra     area coefft for the surface integral of the mass flux
! grid_lat (degree)     latitudes of the regular grid
```



```
! grid_lon (degree)     longitudes of the regular grid
! w_merra [X/s]         vertical velocity either in zlog coordinate (X=zlog)
!                       or potential temperature tendency (X=K)

contains

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

subroutine alloc_merra

print *,'alloc_merra uuh vvh uupol vvpol tth'
print *,nx,ny,nuvz
allocate (wwh(0:nx-1,-1:ny,nuvz,2))
allocate (uuh(0:nx-1,0:ny-1,nuvz,2),vvh(0:nx-1,0:ny-1,nuvz,2))
allocate (uupol(0:nx-1,-1:ny,nuvz,2),vvpol(0:nx-1,-1:ny,nuvz,2))
allocate (tth(0:nx-1,-1:ny,nuvz,2))
allocate (ps(0:nx-1,-1:ny,1,2))
if (merra_diab)  then
   print *,'alloc_merra theta_g theta_col theta_inv_col'
   if(upper_theta_level > nuvz) then
     print *,'WARNING: upper_theta_level is reduced to nuvz'
     upper_theta_level = nuvz
   endif
   print *,'upper lower theta_level ',lower_theta_level,upper_theta_level
   allocate (theta_g(lower_theta_level:upper_theta_level,0:nx-1,-1:ny,2))
   allocate (theta_col(0:nx-1,-1:ny,2),theta_inv_col(0:nx-1,-1:ny,2))
endif

end subroutine alloc_merra


!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

      subroutine gridcheck_merra(error)
!**********************************************************************
!                                                                     *
!             TRAJECTORY MODEL SUBROUTINE GRIDCHECK_MERRA
!                                                                     *
!**********************************************************************
!                                                                     *
!             AUTHOR:      B. LEGRAS
!             from version of gridcheck by G. Wotawa
!             DATE:                                         *
!             LAST UPDATE:                                 *
!                                                                     *
!                                                                     *
!**********************************************************************
!                                                                     *
! DESCRIPTION:        A MODIFIER                                      *
!                                                                     *
!
!**********************************************************************

  use coord
  integer :: ifn, ncid
  real(dp) :: sizesouth,sizenorth
  integer :: LonId, LatId, VerId, TIMEId
  integer :: LonVId,LatVId,VerVId

  logical error
```

```
  error=.false.

! Reads first or last field according to the direction of integration
  !if(ideltas.gt.0) then
  !  ifn=1
  !else
  !  ifn=numbwf
  !endif
  ifn=1

! Open the first file
  print *,ifn,path(3)(1:len_path(3))//wfname(ifn)
  call check(NF90_OPEN(path(3)(1:len_path(3))//wfname(ifn), &
      NF90_NOWRITE, ncid),1)
! get dimension Id
  call check(NF90_INQ_DIMID(ncid, 'XDim\:EOSGRID', LonId),2)
  call check(NF90_INQ_DIMID(ncid, 'YDim\:EOSGRID', LatId),2)
  call check(NF90_INQ_DIMID(ncid, 'Height\:EOSGRID', VerId),2)
  call check(NF90_INQ_DIMID(ncid, 'TIME\:EOSGRID', TimeId),2)
! get dimension length
  call check(NF90_INQUIRE_DIMENSION(ncid, LonId, LEN=NbLon),3)
  call check(NF90_INQUIRE_DIMENSION(ncid, LatId, LEN=NbLat),3)
  call check(NF90_INQUIRE_DIMENSION(ncid, VerId, LEN=NbPress),3)
  call check(NF90_INQUIRE_DIMENSION(ncid, TIMEId, LEN=NbTime),3)
  print *,'gridcheck_merra'
  write(*,'("LonId,LatId,VerId,TIMEId",4I5)') &
    LonId,LatId,VerId,TIMEId
  write(*,'("NbLon,NbLat,NbPress,NbTime",4I5)') NbLon,NbLat,NbPress,NbTime
! get coordinates
! longitude
  allocate(grid_lon(NbLon))
  call check(NF90_INQ_VARID(ncid, 'XDim\:EOSGRID', LonVId),4)
  call check(NF90_GET_VAR(ncid,LonVId,grid_lon),13)
! latitude
  allocate(grid_lat(NbLat))
  call check(NF90_INQ_VARID(ncid, 'YDim\:EOSGRID', LatVId),4)
  call check(NF90_GET_VAR(ncid,LatVId,grid_lat),13)
! pressures
  allocate(grid_ver(NbPress),PressLev(NbPress))
  call check(NF90_INQ_VARID(ncid, 'Height\:EOSGRID', VerVId),4)
  call check(NF90_GET_VAR(ncid,VerVId,grid_ver),13)
! variable Ids
  call check(NF90_INQ_VARID(ncid,'U',UId),23)
  call check(NF90_INQ_VARID(ncid,'V',VId),23)
  call check(NF90_INQ_VARID(ncid,'T',TId),23)
  call check(NF90_INQ_VARID(ncid,'OMEGA',OMEGAId),23)
  call check(NF90_INQ_VARID(ncid,'PS',PSId),23)
  call check(NF90_GET_ATT(ncid,TId,'missing_value',missing_value),53)
! Close the file
  call check(NF90_CLOSE(ncid),24)
! Proceed with the first diab file if this is needed
! to get variable Ids
! (assume same grid)
  if(merra_diab) then
    if(ideltas.gt.0) then
      ifn=1
    else
      ifn=numbwf_diab
    endif
    print *,path_diab(1)(1:len_diab(1))//wfname_diab(ifn)
    call check(NF90_OPEN(path_diab(1)(1:len_diab(1))//wfname_diab(ifn), &
```



```
      NF90_NOWRITE, ncid),1)
    call check(NF90_INQ_VARID(ncid,'DTDTLWR',LWRId),24)
    call check(NF90_INQ_VARID(ncid,'DTDTSWR',SWRId),25)
    call check(NF90_INQ_VARID(ncid,'DTDTLWRCLR',LWRCLRId),26)
    call check(NF90_INQ_VARID(ncid,'DTDTSWRCLR',SWRCLRId),27)
    call check(NF90_CLOSE(ncid),24)
  endif

! Fix grid parameters of the run
! assuming global grid at the moment
! and replication of the first longitude
! N_long and NH_lat in sphereharm do not need to be the same
! as when the spectral field has been generated
  nx = NbLon+1
  ny = NbLat
  nuvz=NbPress
  nwz=NbPress
! Allocate theta_prof needed in the interpolation
!  allocate(theta_prof(NbPress,2))
! Turn press levels into Pa (they are given in hPa)
  PressLev(:)=100*grid_ver(:)
! Conversion from T to theta tendencies
  allocate(facT(NbPress),LogPressLev(NbPress))
  facT(:)=(p0/PressLev(:))**Kappa
  LogPressLev(:)=log(p0/PressLev(:))
  xlon0=grid_lon(1)
  ylat0=grid_lat(1)
  xglobal=.true.
  dx=grid_lon(2)-grid_lon(1)
  dy=grid_lat(2)-grid_lat(1)
  dxconst=180._dp/(dx*r_earth*pi)
  dyconst=180._dp/(dy*r_earth*pi)
  zmax=LogPressLev(NbPress)
! CHECK WHAT IT GOING ON BELOW
! Imposes south pole
  sglobal=.true.               ! field contains south pole
! Enhance the map scale by factor 3 (*2=6) compared to north-south
! map scale
  sizesouth=6._dp*(switchsouth+90._dp)/dy
  call stlmbr(southpolemap,-90._dp,0._dp)
  call stcm2p(southpolemap,0._dp,0.,switchsouth,0._dp,sizesouth, &
  sizesouth,switchsouth,180._dp)
  switchsouthg=(switchsouth-ylat0)/dy
! Imposes north pole
  nglobal=.true.               ! field contains north pole
! Enhance the map scale by factor 3 (*2=6) compared to north-south
! map scale
  sizenorth=6._dp*(90._dp-switchnorth)/dy
  call stlmbr(northpolemap,90._dp,0._dp)
  call stcm2p(northpolemap,0._dp,0._dp,switchnorth,0._dp,sizenorth, &
  sizenorth,switchnorth,180._dp)
  switchnorthg=(switchnorth-ylat0)/dy
  write(*,*)
  write(*,'(a,2i5,3L3)')' gribcheck_merra> nx,ny,xglobal,nglobal,sglobal ', &
       nx, ny, xglobal, nglobal, sglobal
  write(*,'(a,2f10.2)')' gribcheck_merra> switchsouthg, switchnorthg ', &
       switchsouthg, switchnorthg
  write(*,'(a,2i7)') &
        ' gribcheck_merra> # of vertical levels in ECMWF data: ',   &
        nuvz,nwz
  write(*,'(a)') ' Mother domain:'
  write(*,'(a,f10.1,a1,f10.1,a,f10.1,a,i4)') ' Longitude range:', &
```

```fortran
         xlon0,' to ',xlon0+(nx-1)*dx,' Grid distance: ',dx,' # ',nx
  write(*,'(a,f10.1,a1,f10.1,a,f10.1,a,i4)') ' Latitude range: ',   &
         ylat0,' to ',ylat0+(ny-1)*dy,' Grid distance: ',dy,' # ',ny
  write(*,*)

  return
  end subroutine gridcheck_merra

  subroutine check(status,code)
  integer, intent ( in) :: status,code
  if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status)),code
      stop 2
  end if
  end subroutine check

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

      subroutine getfields_merra(itime,nstop)

!*******************************************************************************

!        B. Legras, March 2013 (from getfields)
!        Unified version that process both velocity and diabatic files
!        get U, V, T and w if required
!
!*******************************************************************************
!                                                                              *
! Variables:                                                                   *
! lwindinterval [s]    time difference between the two wind fields read in     *
! indj                 indicates the number of the wind field to be read in    *
! indmin               remembers the number of wind fields already treated     *
! memind(2)            pointer, on which place the wind fields are stored      *
! memtime(2) [s]       times of the wind fields, which are kept in memory      *
! itime [s]            current time since start date of trajectory calculation *
! nstop                > 0, if trajectory has to be terminated                 *
!                                                                              *
! Constants:                                                                   *
! idiffmax             maximum allowable time difference between 2 wind fields *
!                                                                             *
!*******************************************************************************

      integer :: indj,indmin,indmin_diab,itime,nstop,memaux
      save :: indmin,indmin_diab

      data indmin/1/,indmin_diab/1/

! 1st part
! Check, if wind fields are available for the current time step
!**************************************************************

      nstop=0

      if ((ldirect*wftime(1).gt.ldirect*itime).or.    &
        (ldirect*wftime(numbwf).lt.ldirect*itime)) then
        write(*,*) 'TRACZILLA WARNING: NO MERRA WINDS ARE AVAILABLE.'
        write(*,*) 'A TRAJECTORY HAS TO BE TERMINATED.'
        write(*,*) ldirect*wftime(1)
        write(*,*) ldirect*itime
        write(*,*) ldirect*wftime(numbwf)
        nstop=4
```



```fortran
        return
      endif

      if ((ldirect*memtime(1).le.ldirect*itime).and.   &
        (ldirect*memtime(2).gt.ldirect*itime)) then

! The right wind fields are already in memory -> don't do anything
!*****************************************************************

        continue

      else if ((ldirect*memtime(2).le.ldirect*itime).and. &
        (memtime(2).ne.999999999)) then

! Current time is after 2nd wind field
! -> Resort wind field pointers, so that current time is between 1st and 2nd
!***************************************************************************

        memaux=memind(1)
        memind(1)=memind(2)
        memind(2)=memaux
        memtime(1)=memtime(2)

! Read a new wind field and store it on place memind(2)
!******************************************************

        do indj=indmin,numbwf-1
           if (ldirect*wftime(indj+1).gt.ldirect*itime) then
              call read_merra(indj+1,memind(2))
              call verttransform_merra(memind(2))
              memtime(2)=wftime(indj+1)
              write(*,'(a,a,a,i11,a,i11)') &
                      ' getfields_merra> file ',trim(wfname(indj+1)),&
                      ' memtime ',memtime(2),' time ',itime
              nstop = 1
              goto 40
           endif
         enddo
 40    indmin=indj

      else

! No wind fields, which can be used, are currently in memory
! -> read both wind fields
!***********************************************************

         do indj=indmin,numbwf-1
            if ((ldirect*wftime(indj).le.ldirect*itime).and.   &
                  (ldirect*wftime(indj+1).gt.ldirect*itime)) then
               memind(1)=1
               call read_merra(indj,memind(1))
               call verttransform_merra(memind(1))
               memtime(1)=wftime(indj)
               write(*,'(a,a,a,i11,a,i11)') &
                      ' getfields_merra> file ',trim(wfname(indj)),&
                      ' memtime ',memtime(1),' time ',itime
               memind(2)=2
               call read_merra(indj+1,memind(2))
               call verttransform_merra(memind(2))
               memtime(2)=wftime(indj+1)
               write(*,'(a,a,a,i11,a,i11)') &
                      ' getfields_merra> file ',trim(wfname(indj+1)),&
```

```
                  ' memtime ',memtime(2),' time ',itime
              nstop = 1
              goto 60
           endif
        enddo
 60     indmin=indj

     endif

     lwindinterv=abs(memtime(2)-memtime(1))

     if (lwindinterv.gt.idiffmax) nstop=3

! 2nd part
! Check, if heating rates are available for the current time step
!***************************************************************

     if (merra_diab) then

       if ((ldirect*wftime_diab(1).gt.ldirect*itime).or.    &
           (ldirect*wftime_diab(numbwf_diab).lt.ldirect*itime)) then
         write(*,*) 'TRACZILLA WARNING: NO MERRA HEATINGS ARE AVAILABLE.'
         write(*,*) 'A TRAJECTORY HAS TO BE TERMINATED.'
         write(*,*) ldirect*wftime_diab(1)
         write(*,*) ldirect*itime
         write(*,*) ldirect*wftime_diab(numbwf_diab)
         nstop=4
         return
       endif

       if ((ldirect*memtime_diab(1).le.ldirect*itime).and.   &
           (ldirect*memtime_diab(2).gt.ldirect*itime)) then

! The right heating rates are already in memory -> don't do anything
!******************************************************************

         continue

       else if ((ldirect*memtime_diab(2).le.ldirect*itime).and. &
          (memtime_diab(2).ne.999999999)) then

! Current time is after 2nd wind field
! -> Resort heating rate pointers, so that current time is between 1st and 2nd
!*****************************************************************************

         memaux=memind_diab(1)
         memind_diab(1)=memind_diab(2)
         memind_diab(2)=memaux
         memtime_diab(1)=memtime_diab(2)

! Read a new heating rate and store it on place memind(2)
!*******************************************************

         do indj=indmin_diab,numbwf_diab-1
            if (ldirect*wftime_diab(indj+1).gt.ldirect*itime) then
               call read_merra_diab(indj+1,memind_diab(2))
               call verttransform_merra_diab(memind_diab(2),2)
               memtime_diab(2)=wftime_diab(indj+1)
               write(*,'(a,a,a,i11,a,i11)') &
                     ' getfields_merra> file ',trim(wfname_diab(indj+1)),&
                     ' memtime ',memtime_diab(2),' time ',itime
```



```
                  nstop = 1
                  goto 45
               endif
            enddo
 45         indmin_diab=indj

          else

! No heating rates, which can be used, are currently in memory
! -> read both heating rates
!************************************************************

            do indj=indmin_diab,numbwf_diab-1
               if ((ldirect*wftime_diab(indj).le.ldirect*itime).and.   &
                   (ldirect*wftime_diab(indj+1).gt.ldirect*itime)) then
                 memind_diab(1)=1
                 call read_merra_diab(indj,memind_diab(1))
                 call verttransform_merra_diab(memind_diab(1),1)
                 memtime_diab(1)=wftime_diab(indj)
                 write(*,'(a,a,a,i11,a,i11)') &
                        ' getfields_merra> file ',trim(wfname_diab(indj)),&
                        ' memtime ',memtime_diab(1),' time ',itime
                 memind_diab(2)=2
                 call read_merra_diab(indj+1,memind_diab(2))
                 call verttransform_merra_diab(memind_diab(2),2)
                 memtime_diab(2)=wftime_diab(indj+1)
                 write(*,'(a,a,a,i11,a,i11)') &
                        ' getfields_merra> file ',trim(wfname_diab(indj+1)),&
                        ' memtime ',memtime_diab(2),' time ',itime
                 nstop = 1

                 goto 65
               endif
             enddo
 65          indmin_diab=indj

          endif

          lwindinterv=abs(memtime_diab(2)-memtime_diab(1))

          if (lwindinterv.gt.idiffmax) nstop=3

        endif

! memind_diab set to memind because of the interpolation using memind_diab for w
! even in the case of z_motion
        if(z_motion) memind_diab(1:2)=memind(1:2)

        return
        end subroutine getfields_merra

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

   subroutine read_merra(indj,n)
!*******************************************************************************
!                                                                              *
!             TRAJECTORY MODEL SUBROUTINE READ_MERRA                           *
!                                                                              *
!*******************************************************************************
!                                                                              *
!             AUTHOR:      B. Legras
```

```
!            from FLEXPART version by G. Wotawa                              *
!                                                                            *
! DESCRIPTION:                                                               *
!                                                                            *
! READING OF ECMWF METEOROLOGICAL FIELDS INTERPOLATED ON THETA LEVELS       *
! INPUT DATA FILES ARE EXPECTED TO BE AVAILABLE IN SPECTRAL FORMAT           *
!                                                                            *
! INPUT:                                                                     *
! indj               indicates number of the wind field to be read in       *
! n                  temporal index for meteorological fields (1 to 3)*
!                                                                            *
! IMPORTANT VARIABLES FROM SHARED VARIABLES:                                 *
!                                                                            *
! wfname           File name of data to be read in                        *
! nx,ny,nuvz,nwz     expected field dimensions                               *
! nlev_ec            number of vertical levels ecmwf model                   *
! w_iso             temperature tendency over 3h                           *
!                                                                            *
!*****************************************************************************

  integer, intent(in) :: indj,n
  integer :: ncid,it,ll
  character (len=2) :: hour
! Finds the index within the daily file from termination in AVAILABLE
  ll=len_trim(wfname(indj))
  hour=wfname(indj)(ll-1:ll)
  select case(hour)
  case('00')
    it=1
  case('03')
    it=2
  case('06')
    it=3
  case('09')
    it=4
  case('12')
    it=5
  case('15')
    it=6
  case('18')
    it=7
  case('21')
    it=8
  case default
    it=16
  end select

! Open file
  call check(NF90_OPEN(path(3)(1:len_path(3))//wfname(indj), &
      NF90_NOWRITE, ncid),1)
  ! Read winds and temperature, wind in m/s, temperature in K, PS in Pa
  call check(NF90_GET_VAR(ncid,UId,uuh(0:NbLon-1,0:NbLat-1,1:NbPress,n), &
    start=(/1,1,1,it/),count=(/NbLon,NbLat,NbPress,1/)),25)
  call check(NF90_GET_VAR(ncid,VId,vvh(0:Nblon-1,0:NbLat-1,1:NbPress,n), &
    start=(/1,1,1,it/),count=(/NbLon,NbLat,NbPress,1/)),25)
  call check(NF90_GET_VAR(ncid,TId,tth(0:Nblon-1,0:NbLat-1,1:NbPress,n), &
    start=(/1,1,1,it/),count=(/NbLon,NbLat,NbPress,1/)),25)
  call check(NF90_GET_VAR(ncid,PSId,ps(0:Nblon-1,0:NbLat-1,1,n), &
    start=(/1,1,1,it/),count=(/NbLon,NbLat,1,1/)),25)
  if(z_motion) then
    call check(NF90_GET_VAR(ncid,OMEGAId,wwh(0:Nblon-1,0:NbLat-1,1:NbPress,n), &
      start=(/1,1,1,it/),count=(/NbLon,NbLat,NbPress,1/)),25)
```



```
  endif
! Close file
  call check(NF90_CLOSE(ncid),26)

! adding extra column for global fields
  if (xglobal) then
    uuh(nx-1,0:NbLat-1,:,n) = uuh(0,0:NbLat-1,:,n)
    vvh(nx-1,0:NbLat-1,:,n) = vvh(0,0:Nblat-1,:,n)
    tth(nx-1,0:Nblat-1,:,n) = tth(0,0:Nblat-1,:,n)
    ps(nx-1,0:Nblat-1,1,n) = ps(0,0:Nblat-1,1,n)
    if (z_motion) wwh(nx-1,:,:,n) = wwh(0,:,:,n)
  endif

! Check the number of missing_values above surface pressure
!  call check_merra_data(n)

  return

  end subroutine read_merra

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

  subroutine read_merra_diab(indj,n)
!*****************************************************************************
!                                                                            *
!             TRAJECTORY MODEL SUBROUTINE READ_MERRA_DIAB                    *
!                                                                            *
!*****************************************************************************
!                                                                            *
!             AUTHOR:      B. Legras
!             from FLEXPART version by G. Wotawa                             *
!                                                                            *

!*****************************************************************************
  integer, intent(in):: indj,n
  real(dp), allocatable :: buffer(:,:,:)
  integer :: ncid,it,ll
  character (len=2) :: hour
  ll=len_trim(wfname_diab(indj))
  hour=wfname_diab(indj)(ll-1:ll)
  select case(hour)
  case('00')
    it=1
  case('03')
    it=2
  case('06')
    it=3
  case('09')
    it=4
  case('12')
    it=5
  case('15')
    it=6
  case('18')
    it=7
  case('21')
    it=8
  case default
    it=16
  end select
  allocate(buffer(0:NbLon-1,0:NbLat-1,NbPress))
```

```
! Open file
  call check(NF90_OPEN(path_diab(1)(1:len_diab(1))//wfname_diab(indj), &
      NF90_NOWRITE, ncid),1)
  ! Read radiative velocities (in K/s))
  call check(NF90_GET_VAR(ncid,LWRId,wwh(0:NbLon-1,0:NbLat-1,1:NbPress,n), &
    start=(/1,1,1,it/),count=(/NbLon,NbLat,NbPress,1/)),25)
  call check(NF90_GET_VAR(ncid,SWRId,buffer(0:Nblon-1,0:NbLat-1,1:NbPress), &
    start=(/1,1,1,it /),count=(/NbLon,NbLat,NbPress,1/)),25)
  wwh(0:NbLon-1,0:NbLat-1,1:NbPress,n) = wwh(0:NbLon-1,0:NbLat-1,1:NbPress,n) +
buffer
! Close file
  call check(NF90_CLOSE(ncid),26)
  deallocate(buffer)
  ! adding extra column for global fields
  if (xglobal) then
    wwh(nx-1,0:NbLat-1,:,n) = wwh(0,0:NbLat-1,:,n)
  endif
  return
  end subroutine read_merra_diab

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

  subroutine verttransform_merra(n)

  use coord
  integer, intent(in):: n
  integer :: ix, jy, iz, i, j
  real(dp) :: xlon, ylat

! north pole region
  if (nglobal) then
!$OMP PARALLEL DEFAULT(SHARED)
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(ix,jy,iz,xlon,ylat)
        do ix=0,nx-1
          xlon=xlon0+float(ix)*dx
          do jy=int(switchnorthg)-2,ny-1
            ylat=ylat0+float(jy)*dy
            do iz=1,nuvz
              call cc2gll(northpolemap,ylat,xlon,  &
                uuh(ix,jy,iz,n), vvh(ix,jy,iz,n),  &
                uupol(ix,jy,iz,n), vvpol(ix,jy,iz,n))
            enddo
          enddo
        enddo
!$OMP END DO
!      calculation at the north pole from an average over the closest circle
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(iz)
        do iz=1,nuvz
          uupol(0:nx-1,ny,iz,n)=sum(uupol(0:nx-2,ny-1,iz,n))/(nx-1)
          vvpol(0:nx-1,ny,iz,n)=sum(vvpol(0:nx-2,ny-1,iz,n))/(nx-1)
          tth(0:nx-1,ny,iz,n)=sum(tth(0:nx-2,ny-1,iz,n))/(nx-1)
          if(z_motion) wwh(0:nx-1,ny,iz,n)=sum(wwh(0:nx-2,ny-1,iz,n))/(nx-1)
        enddo
!$OMP ENDDO
!$OMP END PARALLEL
        ps(0:nx-1,ny,1,n)=sum(ps(0:nx-2,ny-1,1,n))/(nx-1)
  endif

!    south pole region
  if (sglobal) then
!$OMP PARALLEL DEFAULT(SHARED)
```



```
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(ix,jy,iz,xlon,ylat)
        do ix=0,nx-1
          xlon=xlon0+float(ix)*dx
          do jy=0,int(switchsouthg)+3
            ylat=ylat0+float(jy)*dy
            do iz=1,nuvz
              call cc2gll(southpolemap,ylat,xlon, &
                uuh(ix,jy,iz,n), vvh(ix,jy,iz,n), &
                uupol(ix,jy,iz,n),vvpol(ix,jy,iz,n))
            enddo
          enddo
        enddo
!$OMP ENDDO
!       calculation at the south pole from the average of the closest circle
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(iz)
        do iz=1,nuvz
          uupol(0:nx-1,-1,iz,n)=sum(uupol(0:nx-2,0,iz,n))/(nx-1)
          vvpol(0:nx-1,-1,iz,n)=sum(vvpol(0:nx-2,0,iz,n))/(nx-1)
          tth(0:nx-1,-1,iz,n)=sum(tth(0:nx-2,0,iz,n))/(nx-1)
          if (z_motion) wwh(0:nx-1,-1,iz,n)=sum(wwh(0:nx-2,0,iz,n))/(nx-1)
        enddo
!$OMP ENDDO
!$OMP END PARALLEL
        ps(0:nx-1,-1,1,n)=sum(ps(0:nx-2,0,1,n))/(nx-1)
  endif

  ! Initialize theta control (and theta, if needed)
!----------------------------------------------

  if(merra_diab) then
!       Notice that a global calculation of theta (without sorting) is already
!       done if mass correction is activated

!        if(num_threads==1) then
!          theta_col(:,:,n)=.false.
!          theta_inv_col(:,:,n)=.false.
!          print *,'NOCOL'
!        else
#if defined(PAR_RUN)
!$OMP PARALLEL DO DEFAULT(SHARED) SCHEDULE(DYNAMIC) PRIVATE(i,j)
         do j=-1,ny
             do i=0,nx-1
                call calc_col_theta_merra(i,j,n)
             enddo
         enddo
!$OMP END PARALLEL DO
         print *,'COL_THETA'
#else
         theta_col(:,:,n)=.false.
         theta_inv_col(:,:,n)=.false.
#endif
!        endif
  endif

  return
  end subroutine verttransform_merra

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

  subroutine verttransform_merra_diab(n,m)
```

```fortran
  integer, intent(in):: n,m
  integer :: iz

!     Conversion in degree of theta per second
!$OMP PARALLEL DO DEFAULT(SHARED) SCHEDULE(DYNAMIC) PRIVATE(iz)
        do iz=1,nuvz
           wwh(:,:,iz,n)=wwh(:,:,iz,n)*facT(iz)
           if(nglobal) wwh(0:nx-1,ny,iz,n)=sum(wwh(0:nx-2,ny-1,iz,n))/(nx-1)
           if(sglobal) wwh(0:nx-1,-1,iz,n)=sum(wwh(0:nx-2,0,iz,n))/(nx-1)
        enddo
!$OMP END PARALLEL DO

!     call mass correction when required
      if(mass_correction) call diab_mass_merra(n,m)

  return
  end subroutine verttransform_merra_diab

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
=#=#=#

  subroutine diab_mass_merra(n,m)

!*******************************************************************************
! This subroutine calculates the mass flux across pure pressure levels in the
! stratosphere from heating rates, pressure and temperature.
! The isentropic density sigma is calculated and the mass flux is calculated
! by taking its product with the heating rate (in d theta/dt) and integrated
! over the sphere.
! The integral is then divided by the integral of sigma to get the the mass
! averaged heating rate to be removed on each vertical in order to achieve the
! mass conservation acrosss the pressure surface.
! Vertical derivate involved in sigma is calculated by centered differences
! except on the last upper level.
! (note that the derivative on the first pure pressure level neglects the bkz
! coefficient on the layer just below)
!
! B. Legras
! 28/02/2013
!
!*******************************************************************************

  integer, intent(in) :: n,m
  real(dp), allocatable :: theta(:,:,:), sigma(:,:,:), flux(:,:), flux_lat(:),de
lta_theta(:,:)
  real(dp), allocatable :: mass_flux(:), mean_sigma(:), mean_sigma_lat(:)
  real(dp), allocatable :: mean_w(:)
  integer :: k,i
  real(dp) :: buff
!  integer :: OMP_GET_THREAD_NUM

!  print *,'enter diab_mass_merra'

  allocate(theta(0:nx-1,0:ny-1,nuvz),sigma(0:nx-1,0:ny-1,nuvz))
  allocate(mean_sigma(nuvz),mass_flux(nuvz),mean_w(nuvz))

!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(mean_sigma_lat,flux,flux_lat,delta_theta,
buff)
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(k)
! Calculation of the potential temperature
```



```fortran
  do k=nuvz-NPmass,nuvz
     theta(:,:,k)=0.5_dp*(tth(:,:,k,1)+tth(:,:,k,2))*facT(k)
  enddo
!$OMP END DO
!  print *,'mass before sigma ',OMP_GET_THREAD_NUM()
!  allocate(delta_theta(0:nx-1,0:ny-1))
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(k)
  do k=nuvz-NPMass+1,nuvz-1
!     delta_theta=theta(:,:,k+1)-theta(:,:,k-1)
!     print *,k,minval(minval(abs(delta_theta),dim=1),dim=1)
!     do j=0,ny-1
!       buff=sum(delta_theta(0:nx-2,j))/(nx-1)
!       delta_theta(0:nx-1,j)=buff
!     enddo
!     print *,k,minval(minval(abs(delta_theta),dim=1),dim=1)
!     sigma(:,:,k)=pmc_merra(k)/delta_theta(:,:)
     sigma(:,:,k)=pmc_merra(k)/(theta(:,:,k+1)-theta(:,:,k-1))
  enddo
!$OMP END DO
!  print *,'mass after sigma ',OMP_GET_THREAD_NUM()
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(i)
  do i=0,nx-1
    sigma(i,:,nuvz)=pmc_merra(nuvz)/(theta(i,:,nuvz)-theta(i,:,nuvz-1))
!    sigma(:,:,1)=pmc_merra(1)/(theta(:,:,2)-theta(:,:,1))
  enddo
!$OMP END DO
!  deallocate(delta_theta)
  allocate(mean_sigma_lat(0:ny-1))

! Calculation of the mass flux across the surface and correction
  allocate(flux(0:nx-1,0:ny-1))
  allocate(flux_lat(0:ny-1))
!$OMP DO SCHEDULE(DYNAMIC) PRIVATE(k)
  do k=nuvz-NPMass+1,nuvz
     flux(:,:)=wwh(:,:,k,n)*sigma(:,:,k)
     flux_lat(:)=sum(flux(0:nx-2,:),DIM=1)
     mean_sigma_lat(:)=sum(sigma(0:nx-2,:,k),DIM=1)
     mass_flux(k)=0.5_dp*dot_product(flux_lat(:),area_coefft_merra)
     mean_sigma(k)=0.5_dp*dot_product(mean_sigma_lat(:),area_coefft_merra)
     mean_w(k)=mass_flux(k)/mean_sigma(k)
     wwh(:,:,k,n)=wwh(:,:,k,n)-mean_w(k)
  enddo
!$OMP END DO
  deallocate(flux,flux_lat,mean_sigma_lat)
!$OMP END PARALLEL

! Output of the mass averaged diab heating
  if(mean_diab_output) then
    write(unitflux) memtime_diab(m),mean_w(nuvz-NPMass+1:nuvz)
    flush(unitflux)
  endif

  deallocate(theta,sigma,mean_sigma)
  deallocate(mass_flux,mean_w)

  return
  end subroutine diab_mass_merra

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
=#=#=#

  subroutine diab_mass_merra_init
```

```fortran
!*******************************************************************************
!
! This routine initializes coefficients used in the calculation of the masss cor
rection
! The coefficients are p delta log(p) for the calculation of the vertical deriva
tive
! and sin(phi + dphi) - sin(phi - dphi) = 2 sin(dphi) cos(phi) for the surface i
ntegral
! where phi si a grid latitude and dphi=0.5*dy. This quantity is the diffference

! (up to factor 2 pi which is not accounted) of area
! between to caps bouded by phi-dphi and phi+dphi latitudes. The first and last
values
! are the area of polar caps of angle dphi/2, that is 2 sin^2(dphi/4)
! Conversion to radian is applied
!*******************************************************************************
*

  integer :: i

  if(.not.(xglobal.and.nglobal.and.sglobal)) then
      write(*,*) '#### TRACZILLA MODEL ERROR! DIAB_MASS_MERRA  ####'
      write(*,*) '#### mass flux cannot be balanced         ####'
      write(*,*) '#### on not global grid                   ####'
      stop
  endif

! calculate area coefficient assuming pole to pole regular grid
! (poles on half grid)
  allocate(area_coefft_merra(0:ny-1))
! it is assumed that dy = 180/ny
  do i=0,ny-1
     area_coefft_merra(i)=2*sin(pi/ny/2)*cos(pi*(2*i-ny+1)/ny/2)
  enddo
! test : the sum should be equal to 2
  write(*,'(" diab_mass_merra_init > check sum area  ",g10.3 )') &
       sum(area_coefft_merra)

! This value must not be larger than 19 because then delta theta may vanish loca
lly
! with consequences on the ass flux correction
  NPMass=19
! calculate weighting pressure factors in the vertical
  allocate(pmc_merra(NbPress))
  do i=2,NbPress-1
     pmc_merra=PressLev(i)*(LogPressLev(i+1)-LogPressLev(i-1))
  enddo
  pmc_merra(NbPress)=PressLev(NbPress)*(LogPressLev(NbPress)-LogPressLev(NbPress
-1))
  pmc_merra(1)=PressLev(1)*(LogPressLev(2)-LogPressLev(1))

! Create the output file of the mas averaged heating flux
  open(unitflux,file=trim(path(2))//'MassMeanDiab2.dat',&
     form='unformatted',position='append')

  return
  end subroutine diab_mass_merra_init

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
```



```fortran
subroutine interpol_wind_merra &
         (itime,xt,yt,zt, dxdt,dydt,dzdt, ngrid, &
          theta_inf,theta_sup,psaver,z_factor,tint,nstop)

!*******************************************************************************
!                                                                             *
!  This subroutine interpolates the wind data to current trajectory position.  *
!                                                                             *
!    Author: B. Legras (derived from original interpol_wind)
!                                                                             *
!
!                                                                             *
!*******************************************************************************
!                                                                             *
! Variables:                                                                  *
! dxdt,dydt          horizontal wind components in grid units per second      *
! itime [s]          current temporal position                                *
! memtime(3) [s]     times of the wind fields in memory                       *
! xt,yt,theta        coordinates position for which wind data shall be calculat*
!                                                                             *
! Constants:                                                                  *
!                                                                             *
!*******************************************************************************

      integer, intent(in) :: itime,ngrid
      integer, intent(inout):: nstop
      real(dp), intent(in) :: xt,yt,zt
      real(dp), intent(out) :: tint
      real(dp), intent(out) :: dxdt,dydt,dzdt,z_factor
      real(dp), intent(out) :: theta_inf,theta_sup,psaver

! Auxiliary variables needed for interpolation
      real(dp) :: zlog(2)
      real(dp) :: u1(2),v1(2),w1(2),dt1,dt2,dtt,tp1(2),dt1_diab,dt2_diab,dtt_dia
b
      real(dp) :: tr(2),trp(2),u(4,2),v(4,2),w(4,2),tp(4,2)
      integer :: m,indexh,indexh_diab,indz(2)
      integer :: ix,jy,ixp,jyp
      real(dp) :: ddx,ddy,rddx,rddy,p1,p2,p3,p4
      !real, allocatable :: theta_prof(:,:)
      real(dp) :: theta_prof(250,2)

!********************************************
! Multilinear interpolation in time and space
!********************************************

! Determine the lower left corner and its distance to the current position
!*******************************************************************

      ! min and max required for the points just falling on the boundary
      ! as it may happen typically as a result of initialization
      if(xglobal) ix=modulo(floor(xt),nx-1)
      jy=max(min(floor(yt),ny-1),-1)
      ixp=ix+1         ;  jyp=jy+1
      ddx=modulo(xt-float(ix),1.)
      ! accounts for the two polar regions
      if (yt<0) then
         ddy=2*(yt+0.5_dp)
      else if (yt>ny-1) then
         ddy=2*(yt+1._dp-ny)
      else
         ddy=yt-float(jy)
```

```
      endif
      rddx=1.-ddx      ;  rddy=1.-ddy
      p1=rddx*rddy     ;  p2=ddx*rddy
      p3=rddx*ddy      ;  p4=ddx*ddy

! Calculate coefficients for temporal interpolation
!*****************************************************

      dt1=float(itime-memtime(1))
      dt2=float(memtime(2)-itime)
      dtt=1./(dt1+dt2)
      dt1=dt1*dtt
      dt2=dt2*dtt
      if (merra_diab) then
        dt1_diab=float(itime-memtime_diab(1))
        dt2_diab=float(memtime_diab(2)-itime)
        dtt_diab=1/(dt1_diab+dt2_diab)
        dt1_diab=dt1_diab*dtt_diab
        dt2_diab=dt2_diab*dtt_diab
      endif

! Determine the level below the current position for u,v
!*******************************************************

!  z pressure coordinates

      if (z_motion) then

        zlog(:)=zt

!       Locates the pressure in the grid
!       Equal indz since it is a pure pressure grid

        indz(1) = locp(zt,1,NbPress)
        indz(2) = indz(1)

        psaver=dt2*(p1*ps(ix,jy,1,memind(1))+p2*ps(ixp,jy,1,memind(1))    &
                   +p3*ps(ix,jyp,1,memind(1))+p4*ps(ixp,jyp,1,memind(1))) &
              +dt1*(p1*ps(ix,jy,1,memind(2))+p2*ps(ixp,jy,1,memind(2))    &
                   +p3*ps(ix,jyp,1,memind(2))+p4*ps(ixp,jyp,1,memind(2)))

!   theta coordinates

      else if (merra_diab) then

        !allocate (theta_prof(NbPress,2))

!       Calculate the values on the four adjacent corners if required

#if defined(PAR_RUN)

#else
        if(.not.theta_col(ix,jy,memind(1)))   call calc_col_theta_merra(ix,jy,memind
(1))
        if(.not.theta_col(ix,jy,memind(2)))   call calc_col_theta_merra(ix,jy,memind
(2))
        if(.not.theta_col(ix,jyp,memind(1)))  call calc_col_theta_merra(ix,jyp,memin
d(1))
        if(.not.theta_col(ix,jyp,memind(2)))  call calc_col_theta_merra(ix,jyp,memin
d(2))
        if(.not.theta_col(ixp,jy,memind(1)))  call calc_col_theta_merra(ixp,jy,memin
d(1))
```



```
        if(.not.theta_col(ixp,jy,memind(2)))  call calc_col_theta_merra(ixp,jy,memin
d(2))
        if(.not.theta_col(ixp,jyp,memind(1))) call calc_col_theta_merra(ixp,jyp,memi
nd(1))
        if(.not.theta_col(ixp,jyp,memind(2))) call calc_col_theta_merra(ixp,jyp,memi
nd(2))
#endif

!       Calculate the mean theta profile at the location of the parcel

        theta_prof(lower_theta_level:upper_theta_level,:)= &
                        p1*theta_g(lower_theta_level:upper_theta_level,ix,jy,:)
&
                      + p2*theta_g(lower_theta_level:upper_theta_level,ixp,jy,:)
 &
                      + p3*theta_g(lower_theta_level:upper_theta_level,ix,jyp,:)
 &
                      + p4*theta_g(lower_theta_level:upper_theta_level,ixp,jyp,:
)

!       Provides clip if required

        if (theta_bounds) then
           theta_inf=dt2_diab*theta_prof(lower_theta_level,memind(1)) &
                    +dt1_diab*theta_prof(lower_theta_level,memind(2))
           theta_sup=dt2_diab*theta_prof(upper_theta_level,memind(1)) &
                    +dt1_diab*theta_prof(upper_theta_level,memind(2))
        endif

!       Find the surrounding levels

        indz(1) = locisent_merra(zt,lower_theta_level,upper_theta_level,memind(1
),theta_prof)
        indz(2) = locisent2_merra(zt,indz(1),memind(2),theta_prof)

!       if (debug_out) print *,'interpol indz memind ',indz,memind
!       if (debug_out) print *,ix,jy,ixp,jyp
!       if (debug_out) print *,ddx,ddy
!       if (debug_out) print *,p1,p2,p3,p4
!       if (debug_out) print *,dt1,dt2,dt1_diab,dt2_diab
!       if (debug_out) write(*,'(5g12.4)') tth(ix,jy,indz(1)-2:indz(1)+2,memind(
1))
!       if (debug_out) write(*,'(5g12.4)') theta_g(indz(1)-2:indz(1)+2,ix,jy,mem
ind(1))
!        if (debug_out) write(*,'(5g12.4)') tth(ix,jyp,indz(1)-2:indz(1)+2,memin
d(1))
!        if (debug_out) write(*,'(5g12.4)') theta_g(indz(1)-2:indz(1)+2,ix,jyp,m
emind(1))
!        if (debug_out) write(*,'(5g12.4)') tth(ixp,jy,indz(1)-2:indz(1)+2,memin
d(1))
!        if (debug_out) write(*,'(5g12.4)') theta_g(indz(1)-2:indz(1)+2,ixp,jy,m
emind(1))
!        if (debug_out) write(*,'(5g12.4)') tth(ixp,jyp,indz(1)-2:indz(1)+2,memi
nd(1))
!        if (debug_out) write(*,'(5g12.4)') theta_g(indz(1)-2:indz(1)+2,ixp,jyp,
memind(1))
!       if (debug_out) write(*,'(5g12.4)') theta_prof(indz(1)-2:indz(1)+2,memind
(1))
!        if (debug_out) write(*,'(5g12.4)') theta_prof(indz(2)-2:indz(2)+2,memin
d(2))

        zlog(1)=(log(theta_prof(indz(1)+1,memind(1))/zt)*LogPressLev(indz(1)) &
```

```fortran
                +log(zt/theta_prof(indz(1),memind(1)))*LogPressLev(indz(1)+1)) &
                / log(theta_prof(indz(1)+1,memind(1))/theta_prof(indz(1),memind(
1)))
        zlog(2)=(log(theta_prof(indz(2)+1,memind(2))/zt)*LogPressLev(indz(2)) &
                +log(zt/theta_prof(indz(2),memind(2)))*LogPressLev(indz(2)+1)) &
                / log(theta_prof(indz(2)+1,memind(2))/theta_prof(indz(2),memind(
2)))

!        if (debug_out) then
!           print *,'interpol merra > indz ',indz
!           print *,'interpol_merra > zlog ',zlog
!           print *,'interpol_merra > pres ',1000*exp(-zlog)
!           print *,'interpol_merra > preslev ',presslev(indz(1)),presslev(indz(
2))
!           print *,'interpol_merra > PS ',ps(ix,jy,1,1),ps(ix,jy,1,2)
!           print *,'interpol_merra > ix, jy ',ix,jy
!           print *,'interpol_merra > lon,lat ',xlon0+ix*dx,ylat0+jy*dy
!           print *,'theta_prof'
!           print *,'wwh(ix,jy,...  > ',wwh(ix,jy,indz(1),memind(1))*86400, &
!                                       wwh(ix,jy,indz(1)+1,memind(1))*86400
!           write(*,'(8g12.5)'),theta_prof(:,1)
!           print *,'uuh'
!           write(*,'(8g12.5)'),uuh(ix,jy,:,1)
!           print *,'tth'
!           write(*,'(8g12.5)'),tth(ix,jy,:,1)
!           print *,'theta_g'
!           write(*,'(8g12.5)'),theta_g(:,ix,jy,1)
!        endif

        !deallocate(theta_prof)

      else
        stop 999
      endif

!  Defines pressure at the 8 nearby meshpoints for
!  the two times
!  Actually only 2 values since the levels are pure pressure
!  which makes 4 when the next level is added

      tr (1) = LogPressLev(indz(1))
      tr (2) = LogPressLev(indz(2))
      trp(1) = LogPressLev(indz(1)+1)
      trp(2) = LogPressLev(indz(2)+1)

! Halt trajectories which are too close from lower boundaries
!*************************************************************
! parcels cannot exceed the max altitude due to bound in
! advanceB

      if((minval(indz)==1)) then
         nstop=2
         dxdt=0. ; dydt=0.; dzdt=0.; tint=275.;
         z_factor=1.
         return
      endif

!***********************************************************************
! 1.) Bilinear horizontal interpolation
! This has to be done separately for 4 fields (Temporal(2)*Vertical(2))
!***********************************************************************
```



```fortran
! Loop over 2 time steps and 2 levels
!************************************

      if (ngrid < 0) then  ! polar region

        do m=1,2
          indexh=memind(m)
          indexh_diab=memind_diab(m)

            u(1,m)=(uupol(ix ,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + uupol(ix ,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            v(1,m)=(vvpol(ix ,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + vvpol(ix ,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            w(1,m)=(wwh(ix ,jy ,indz(m)  ,indexh_diab)*(zlog(m)-trp(m))  &
                  + wwh(ix ,jy ,indz(m)+1,indexh_diab)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            u(2,m)=(uupol(ixp,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + uupol(ixp,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            v(2,m)=(vvpol(ixp,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + vvpol(ixp,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            w(2,m)=(wwh(ixp,jy ,indz(m)  ,indexh_diab)*(zlog(m)-trp(m))  &
                  + wwh(ixp,jy ,indz(m)+1,indexh_diab)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            u(3,m)=(uupol(ix ,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + uupol(ix ,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            v(3,m)=(vvpol(ix ,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + vvpol(ix ,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            w(3,m)=(wwh(ix ,jyp,indz(m)  ,indexh_diab)*(zlog(m)-trp(m))  &
                  + wwh(ix ,jyp,indz(m)+1,indexh_diab)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            u(4,m)=(uupol(ixp,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + uupol(ixp,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            v(4,m)=(vvpol(ixp,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + vvpol(ixp,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            w(4,m)=(wwh(ixp,jyp,indz(m)  ,indexh_diab)*(zlog(m)-trp(m))  &
                  + wwh(ixp,jyp,indz(m)+1,indexh_diab)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))

          u1(m)=p1*u(1,m)+p2*u(2,m)+p3*u(3,m)+p4*u(4,m)
          v1(m)=p1*v(1,m)+p2*v(2,m)+p3*v(3,m)+p4*v(4,m)
          w1(m)=p1*w(1,m)+p2*w(2,m)+p3*w(3,m)+p4*w(4,m)

        enddo

      else     ! non polar region

        do m=1,2
          indexh=memind(m)
          indexh_diab=memind_diab(m)

            u(1,m)=(uuh(ix ,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                  + uuh(ix ,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                  / (tr(m)-trp(m))
            v(1,m)=(vvh(ix ,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
```

```fortran
                 + vvh(ix ,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           w(1,m)=(wwh(ix ,jy ,indz(m)  ,indexh_diab)*(zlog(m)-trp(m)) &
                 + wwh(ix ,jy ,indz(m)+1,indexh_diab)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           u(2,m)=(uuh(ixp,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + uuh(ixp,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           v(2,m)=(vvh(ixp,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + vvh(ixp,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           w(2,m)=(wwh(ixp,jy ,indz(m)  ,indexh_diab)*(zlog(m)-trp(m)) &
                 + wwh(ixp,jy ,indz(m)+1,indexh_diab)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           u(3,m)=(uuh(ix ,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + uuh(ix ,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           v(3,m)=(vvh(ix ,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + vvh(ix ,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           w(3,m)=(wwh(ix ,jyp,indz(m)  ,indexh_diab)*(zlog(m)-trp(m)) &
                 + wwh(ix ,jyp,indz(m)+1,indexh_diab)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           u(4,m)=(uuh(ixp,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + uuh(ixp,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           v(4,m)=(vvh(ixp,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m)) &
                 + vvh(ixp,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))
           w(4,m)=(wwh(ixp,jyp,indz(m)  ,indexh_diab)*(zlog(m)-trp(m)) &
                 + wwh(ixp,jyp,indz(m)+1,indexh_diab)*(tr(m)-zlog(m))) &
                 / (tr(m)-trp(m))

         u1(m)=p1*u(1,m)+p2*u(2,m)+p3*u(3,m)+p4*u(4,m)
         v1(m)=p1*v(1,m)+p2*v(2,m)+p3*v(3,m)+p4*v(4,m)
         w1(m)=p1*w(1,m)+p2*w(2,m)+p3*w(3,m)+p4*w(4,m)

       enddo

!       if (debug_out) then
!          print *,'interpol merra > u v'
!          print *,u(:,1)
!          print *,u(:,2)
!          print *,w(:,1)*86400
!          print *,w(:,2)*86400
!          print *,uuh(ix,jy,indz(1),memind(1)),uuh(ix,jyp,indz(1),memind(1)), &
!                  uuh(ixp,jy,indz(1),memind(1)),uuh(ixp,jyp,indz(1),memind(1))
!          print *,uuh(ix,jy,indz(1)+1,memind(1)),uuh(ix,jyp,indz(1)+1,memind(1)
), &
!                  uuh(ixp,jy,indz(1)+1,memind(1)),uuh(ixp,jyp,indz(1)+1,memind(
1))
!          print *,uuh(ix,jy,indz(2),memind(2)),uuh(ix,jyp,indz(2),memind(2)), &
!                  uuh(ixp,jy,indz(2),memind(2)),uuh(ixp,jyp,indz(2),memind(2))
!          print *,uuh(ix,jy,indz(2)+1,memind(2)),uuh(ix,jyp,indz(2)+1,memind(2)
), &
!                  uuh(ixp,jy,indz(2)+1,memind(2)),uuh(ixp,jyp,indz(2)+1,memind(
2))
!       endif

      endif

 ! Accurate calculation of the temperature if needed
```



```fortran

      if (AccurateTemp) then

        do m=1,2
          indexh=memind(m)

            tp(1,m)=(tth(ix ,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + tth(ix ,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            tp(2,m)=(tth(ixp,jy ,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + tth(ixp,jy ,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            tp(3,m)=(tth(ix ,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + tth(ix ,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))
            tp(4,m)=(tth(ixp,jyp,indz(m)  ,indexh)*(zlog(m)-trp(m))  &
                  + tth(ixp,jyp,indz(m)+1,indexh)*(tr(m)-zlog(m)))  &
                  / (tr(m)-trp(m))

          tp1(m)=p1*tp(1,m)+p2*tp(2,m)+p3*tp(3,m)+p4*tp(4,m)

        enddo

      endif

!************************************
! 3.) Temporal interpolation (linear)
!************************************

      dxdt=u1(1)*dt2+u1(2)*dt1
      dydt=v1(1)*dt2+v1(2)*dt1
      if (z_motion)    dzdt=w1(1)*dt2+w1(2)*dt1
      if (merra_diab)  dzdt=w1(1)*dt2_diab+w1(2)*dt1_diab
      if(AccurateTemp) tint=tp1(1)*dt2+tp1(2)*dt1


!**************************************************
! 4.) Calculation of z_factor for vertical diffusion
!**************************************************

      select case (diftype)
      case (1)          ! diffusion in z (cf p.46, book C, part2)
!       d theta / dz = - (g/theta) d theta / d Pi = -g d Log(theta) / d Pi
!       where Pi = Cp (p/p0)**kappa = Cp T/theta
!       estimated from the data on the lower left corner at first time
!       of the interval, for a better estimate using the closest point
!       activate the first following line and deactivate the second one
!       call sort_hor_distance
!        i0=ix; j0=jy; idxy=1
!        pisup0 = cpa * tth(i0,j0,indz(idxy,1)+1)/trp(idxy,1)
!        piinf0 = cpa * tth(i0,j0,indz(idxy,1))/tr(idxy,1)
!        z_factor = -ga*(trp(idxy,1)-tr(idxy,1))/(pisup0-piinf0)
!        TO BE REACTIVATED
        z_factor=0.
      case (2)          ! diffusion in theta
        z_factor = 1.
      case default
        z_factor = 0.
      end select

      return
```

```fortran
      end subroutine interpol_wind_merra

!*******************************************************************************
      function locp(zlog,ib1,iu1)
      integer, intent(in) :: ib1,iu1
      real(dp), intent(in) :: zlog
      integer :: locp,ib,iu,im
      ib=ib1 ; iu=iu1
      do while(iu-ib>1)
        im = (iu+ib)/2
        if( zlog >= LogPressLev(im) ) then
          ib=im
        else
          iu=im
        endif
      enddo
      locp = ib
      end function locp
!*******************************************************************************

      function locp2(zlog,ib1)
      integer, intent(in) :: ib1
      real(dp), intent(in) :: zlog
      integer :: locp2,ib
      ib=ib1
      if (zlog < LogPressLev(ib)) then
        do while(ib>1)
          ib = ib-1
          if ( zlog >= LogPressLev(ib) ) exit
        enddo
      else
        do while(ib<NbPress-1)
          if (zlog < LogPressLev(ib+1)) exit
          ib = ib+1
        enddo
      endif
      locp2 = ib
      end function locp2
!*******************************************************************************

      function locisent_merra(theta,ib1,iu1,ind,theta_prof)
      integer, intent(in) :: ib1,iu1,ind
      real(dp), intent(in) :: theta, theta_prof(:,:)
      integer :: locisent_merra,ib,iu,im
      ib=ib1 ; iu=iu1
      do while(iu-ib>1)
        im = (iu+ib)/2
        if( theta >= theta_prof(im,ind) ) then
          ib=im
        else
          iu=im
        endif
      enddo
      locisent_merra = ib
      end function locisent_merra
!*******************************************************************************

      function locisent2_merra(theta,ib1,ind,theta_prof)
      integer, intent(in) :: ib1,ind
      real(dp), intent(in) :: theta, theta_prof(:,:)
      integer :: locisent2_merra,ib
      ib=ib1
```



```fortran
      if (theta < theta_prof(ib,ind)) then
        do while(ib>lower_theta_level)
          ib = ib-1
          if ( theta >= theta_prof(ib,ind) ) exit
        enddo
      else
        do while(ib<upper_theta_level-1)
          if (theta < theta_prof(ib+1,ind)) exit
          ib = ib+1
        enddo
      endif
      locisent2_merra = ib
      end function locisent2_merra

!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
!#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#


      subroutine calc_col_theta_merra(x,y,ind)
      integer, intent(in) :: x, y, ind
!     theta_diff  must be defined here to avoid // conflicts
      real(dp), allocatable :: theta_diff(:) !
      real(dp) :: tol
      integer k, id, id1(1), nbunmix
!     Basic calculation of the potential temperature
      do k= lower_theta_level,upper_theta_level
        if(tth(x,y,k,ind)<missing_value) then
          theta_g(k,x,y,ind) = facT(k) * tth(x,y,k,ind)
        else
          theta_g(k,x,y,ind) = k
        endif
      enddo
!     Tolerance on the difference of theta between two successive levels
      tol = 0.001
      nbunmix=0
      theta_col(x,y,ind) = .true.
      allocate (theta_diff(1:upper_theta_level-lower_theta_level))
      theta_diff = theta_g(lower_theta_level+1:upper_theta_level,x,y,ind) &
                 - theta_g(lower_theta_level:upper_theta_level-1,x,y,ind)
      if(minval(theta_diff) < 0.) then
        theta_inv_col(x,y,ind) = .true.
        call quicksort(theta_g(lower_theta_level,x,y,ind),0, &
          upper_theta_level-lower_theta_level)
          ! quicksort written in C, for which indices start at 0, not 1
        theta_diff = theta_g(lower_theta_level+1:upper_theta_level,x,y,ind) &
                   - theta_g(lower_theta_level:upper_theta_level-1,x,y,ind)
      endif
      ! Recalculation of the profile in case of mixing with equal theta over
      ! two or more successive levels
      ! The algorithm is borrowed from unmix with the important difference that
      ! it does not aim at smoothing the profile but only at removing spurious
      ! singularities.
      ! Hence min are replaced by max and conversely, with respect to unmix.
      ! Interpolation is here only used to ensure monotonicity. (to be checked)
      do while ((minval(theta_diff) < tol) .and. nbunmix < 30)
        nbunmix=nbunmix+1
        id1=minloc(theta_diff(:))
        id=id1(1)+1
        if(id==2) then
          theta_g(lower_theta_level,x,y,ind)= &
                  theta_g(lower_theta_level+1,x,y,ind)-2.2*tol
        else
```

```
          theta_g(lower_theta_level+id-1,x,y,ind)= &
                  theta_g(lower_theta_level+id-1,x,y,ind)-1.1*tol
!              min((2*theta_g(lower_theta_level+id-3,x,y,ind) &
!                    +theta_g(lower_theta_level+id,x,y,ind))/3., &
!                  theta_g(lower_theta_level+id-1,x,y,ind)-2*tol)
          theta_g(lower_theta_level+id,x,y,ind)= &
                  theta_g(lower_theta_level+id,x,y,ind)+1.1*tol
!              max((theta_g(lower_theta_level+id-3,x,y,ind) &
!                +2*theta_g(lower_theta_level+id,x,y,ind))/3., &
!                  theta_g(lower_theta_level+id-2,x,y,ind)+2*tol)
          call quicksort(theta_g(lower_theta_level,x,y,ind),0, id)
        endif
        theta_diff = theta_g(lower_theta_level+1:upper_theta_level,x,y,ind) &
                   - theta_g(lower_theta_level:upper_theta_level-1,x,y,ind)
        !print *,id,x,y,ind,nbunmix
      enddo
      deallocate(theta_diff)
      return
      end subroutine calc_col_theta_merra

      subroutine check_merra_data(n)
      integer, intent(in) :: n
      integer :: ix,jy,k,indz,um,count_c,count_miss

!      print *,'check_merra_data'
      count_c=0
      count_miss=0
      do ix=0,nx-2
         do jy=0,ny-1
           if (ps(ix,jy,1,n)>PressLev(1)) then
              indz=0
           else
              indz=locp(log(p0/ps(ix,jy,1,n)),1,20)
           endif
           um=0
           do k=indz+3,NbPress
             if (uuh(ix,jy,k,n)==missing_value) um=um+1
           enddo
!           um=int(sum(uuh(ix,jy,indz+2:,n),mask=uuh(ix,jy,indz+2:,n)==missing_v
alue)/missing_value)
!           vm=int(sum(vvh(ix,jy,indz+2:,n),mask=vvh(ix,jy,indz+2:,n)==missing_v
alue)/missing_value)
!           tm=int(sum(tth(ix,jy,indz+2:,n),mask=tth(ix,jy,indz+2:,n)==missing_v
alue)/missing_value)
           count_c=count_c+1
           if (um>0) then
!              write(*,'("overmiss ",6I8,)')indz,ix,jy,um
              count_miss=count_miss+1
           endif
         enddo
      enddo
      write(*,'("check_merra_data",f6.2,"%")')100.*count_miss/count_c
      return
      end subroutine check_merra_data

end module merra

!
!=====|==1=========2=========3=========4=========5=========6=========7==
```