```
!*****************************************************************
! Copyright 1996, 1997, 2001, 2002, 2006, 2007, 2012, 2013, 2016      *
! Andreas Stohl, Bernard Legras, Ann'Sophe Tissier                    *
!                                                                     *
! 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/>.  *
!*****************************************************************
!
!#############################################################################
!------------------------------- READINTERP ----------------------------------
!#############################################################################

module readinterpN
use commons
implicit none
private locuv, locuv2, locw, locw2
private check_nx, check_ny, check_nuvz, check_nwz, printerror

logical, save :: ecmwf_data

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

      subroutine readpaths(pathfile,error)

!*****************************************************************************
!                                                                            *
!     Reads the pathnames, where input/output files are expected to be.      *
!     The file pathnames must be available in the current working directory. *
!                                                                            *
!     Author: A. Stohl                                                       *
!                                                                            *
!     1 February 1994                                                        *
!                                                                            *
!*****************************************************************************
!                                                                            *
! Variables:                                                                 *
! error               .true., if file pathnames does not exist               *
! len(numpath)        lengths of the path names                              *
! path(numpath)       pathnames of input/output files                        *
!                                                                            *
! Constants:                                                                 *
! numpath             number of pathnames to be read in                      *
!                                                                            *
!*****************************************************************************

      integer i
```



```
      logical, intent(out)::error

      character(len=*), intent(in):: pathfile

      error=.false.

! Read the pathname information stored in unitpath
!*********************************************

      open(unitpath,file=pathfile,status='old',err=999)
      print *,' opening pathfile ',pathfile

      do i=1,numpath
        read(unitpath,'(a)',err=998) path(i)
        len_path(i)=len_trim(path(i))
      enddo

      close(unitpath)
      return

998   write(*,*) ' #### TRAJECTORY MODEL ERROR! ERROR WHILE   #### '
      write(*,*) ' #### READING FILE PATHNAMES.           #### '

999   write(*,*) ' #### TRAJECTORY MODEL ERROR! FILE "pathnames"#### '
      write(*,*) ' #### CANNOT BE OPENED IN THE CURRENT WORKING #### '
      write(*,*) ' #### DIRECTORY.                #### '
      error=.true.

      return
      end subroutine readpaths

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

      subroutine readavailable(error)
!                                o
!*****************************************************************************
!                                                                            *
!   This routine reads the dates and times for which windfields are available. *
!                                                                            *
!     Authors: A. Stohl                                                      *
!                                                                            *
!     6 February 1994                                                        *
!     8 February 1999, Use of nested fields, A. Stohl
!     7 April    2002, Suppression of code related to nested fields, B. Legras
!                                                                            *
!*****************************************************************************
!                                                                            *
! Variables:                                                                 *
! bdate               beginning date as Julian date                          *
! beg                 beginning date for windfields                          *
! end                 ending date for windfields                             *
! error               .true., if error ocurred in subprogram, else .false.   *
! fname               filename of wind field, help variable                  *
! ideltas [s]         duration of modelling period                           *
! idiff               time difference between 2 wind fields                  *
! idiffnorm           normal time difference between 2 wind fields           *
! idiffmax [s]        maximum allowable time between 2 wind fields           *
! jul                 julian date, help variable                             *
! numbwf              actual number of wind fields                           *
! wfname(maxwf)       file names of needed wind fields                       *
```

```
! wfspec(maxwf)        file specifications of wind fields (e.g., if on disc)   *
! wftime(maxwf) [s]times of wind fields relative to beginning time             *
! wfname1,wfspec1,wftime1 = same as above, but only local (help variables)     *
!                                                                              *
! Constants:                                                                   *
! maxwf                 maximum number of wind fields                          *
! unitavailab           unit connected to file AVAILABLE                       *
!                                                                              *
!******************************************************************************

      use date
      logical error,toobig
      integer i,j,idiff,ldat,ltim
      integer year, month, day
      real (dbl) :: jul,beg,end
      character(len=16):: fname,spec
      character(len=16), allocatable:: wfname1(:),wfspec1(:)
      integer, allocatable:: wfldat1(:),wfltim1(:),wftime1(:)

      error=.false.
      allocate(wfname1(maxwf),wfspec1(maxwf), &
               wfldat1(maxwf),wfltim1(maxwf),wftime1(maxwf))

! Windfields are only used, if they are within the modelling period.
! However, 1 additional day at the beginning and at the end is used for
! interpolation. -> Compute beginning and ending date for the windfields.
!*****************************************************************

      if (ideltas.gt.0) then          ! forward trajectories
        beg=bdate-1.                  ! idiffmax should be used here too

        end=bdate+dble(float(ideltas)/86400.) &
                 +dble(float(idiffmax)/86400.)
      else                            ! backward trajectories
        beg=bdate+dble(float(ideltas)/86400.) &
                 -dble(float(idiffmax)/86400.)
        end=bdate+1.                  ! idiffmax shoukd be used here too
      endif

! Open the wind field availability file and read available wind fields
! within the modelling period.
!*****************************************************************

      open(unitavailab,file=path(4)(1:len_path(4)),status='old',err=999)

      do i=1,3
        read(unitavailab,*)
      enddo

      numbwf=0
100   read(unitavailab,'(i8,1x,i6,3x,a16,3x,a10)',end=99) &
           ldat,ltim,fname,spec
      jul=juldate(ldat,ltim)
      if ((jul.ge.beg).and.(jul.le.end)) then
        numbwf=numbwf+1
        if (numbwf.gt.maxwf) then      ! check exceedance of dimension
           write(*,*) 'Number of needed wind fields is too great.'
           write(*,*) 'actual, max ',numbwf,maxwf
           write(*,*) 'Reduce modelling period (file "COMMAND") or'
           write(*,*) 'reduce number of wind fields (file "AVAILABLE").'
           goto 1000
        endif
```



```
        wfname1(numbwf)=adjustl(fname)
        wfldat1(numbwf)=ldat
        wfltim1(numbwf)=ltim
        wfspec1(numbwf)=trim(spec)
        wftime1(numbwf)=nint((jul-bdate)*86400.)
      endif
      goto 100       ! next wind field

99    continue
      write(*,*) 'readavailable> numbwf',numbwf,maxwf
      write(*,*) 'readavailable> min, max ',&
        minval(wftime1(1:numbwf)),maxval(wftime1(1:numbwf))

      close(unitavailab)


! Check wind field times of file AVAILABLE (expected to be in temporal order)
!*****************************************************************************

      if (numbwf.eq.0) then
        write(*,*) ' #### FLEXPART MODEL ERROR! NO WIND FIELDS   #### '
        write(*,*) ' #### AVAILABLE FOR SELECTED TIME PERIOD.    #### '
        error=.TRUE.
        return
      endif

      do i=2,numbwf
        if (wftime1(i).le.wftime1(i-1)) then
          write(*,*) 'FLEXPART ERROR: FILE AVAILABLE IS CORRUPT.'
          write(*,*) 'THE WIND FIELDS ARE NOT IN TEMPORAL ORDER.'
          write(*,*) 'PLEASE CHECK FIELD ',wfname1(i)
          error=.TRUE.
          return
        endif
      enddo

! For backward trajectories, reverse the order of the windfields
!****************************************************************

      if (ideltas.ge.0) then
        wfname(1:numbwf)=wfname1(1:numbwf)
        wfspec(1:numbwf)=wfspec1(1:numbwf)
        wftime(1:numbwf)=wftime1(1:numbwf)
        wfldat(1:numbwf)=wfldat1(1:numbwf)
        wfltim(1:numbwf)=wfltim1(1:numbwf)
      else
        do i=1,numbwf
          wfname(numbwf-i+1)=wfname1(i)
          wfspec(numbwf-i+1)=wfspec1(i)
          wftime(numbwf-i+1)=wftime1(i)
          wfldat(numbwf-i+1)=wfldat1(i)
          wfltim(numbwf-i+1)=wfltim1(i)
        enddo
      endif

! Check the time difference between the wind fields. If it is big,
! write a warning message. If it is too big, warn that trajectory will
! terminate.
!*****************************************************************

      wftime1(:)=wftime(:)
```

```
      do i=2,numbwf
        idiff=abs(wftime(i)-wftime(i-1))
        toobig=.false.
        if (idiff.gt.idiffmax) then
!         detect end of february for bissextil year in perpetual run
!         shift all remaining times in such case to satisfy check done in
!         getfield
          if(perpetual) then
            ! check done on unshifted times to match successive bissextil years
            if(ideltas > 0) call caldate(wftime1(i-1)/86400.+bdate,ldat,ltim)
            if(ideltas < 0) call caldate(wftime1(i  )/86400.+bdate,ldat,ltim)
            year=int(ldat/10000)
            month=int((ldat-10000*year)/100)
            day=int(ldat-10000*year-100*month)
            if(month==2.and.day==28.and.mod(year,4)==0) then
              print *,'readavailable > jump due to bissextil year in perpetual run'
              print *,'readavailable > apply one-day shift to all remaining times'
              if(ideltas > 0) then
                do j=i,numbwf
                  wftime(j)=wftime(j)-86400
                enddo
              else
                do j=i,numbwf
                  wftime(j)=wftime(j)+86400
                enddo
              endif
            else
              toobig=.true.
            endif
          endif
          if(.not.perpetual.or.toobig) then
            write(*,*) 'FLEXPART WARNING: TIME DIFFERENCE BETWEEN TWO'
            write(*,*) 'WIND FIELDS IS TOO BIG FOR TRANSPORT CALCULATION.'
            write(*,*) 'THEREFORE, TRAJECTORIES HAVE TO BE SKIPPED.'
            print *, wfname(i),wfname(i-1),idiffmax
          endif
        else if (idiff.gt.idiffnorm) then
          write(*,*) 'FLEXPART WARNING: TIME DIFFERENCE BETWEEN TWO'
          write(*,*) 'WIND FIELDS IS BIG. THIS MAY CAUSE A DEGRADATION'
          write(*,*) 'OF SIMULATION QUALITY.'
        endif
      enddo


! Reset the times of the wind fields that are kept in memory to no time
!*****************************************************************

      do i=1,2
        memind(i)=i
        memtime(i)=999999999
      enddo

      print *,'readavailable> done'

      deallocate(wfname1,wfspec1,wfldat1,wftime1,wfltim1)

      return

999   write(*,*) ' #### FLEXPART MODEL ERROR! FILE #### '
      write(*,'(a)') '   '//path(4)(1:len_path(4))
      write(*,*) ' #### CANNOT BE OPENED       #### '
1000  error=.true.
```



```
      return
      end subroutine readavailable

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

   subroutine gridcheck(oronew,error)
!*****************************************************************************
!                                                                            *
!             TRAJECTORY MODEL SUBROUTINE GRIDCHECK                          *
!             FLEXPART VERSION -> DO NOT USE IN FLEXTRA, PRAEPRO             *
!                                                                            *
!*****************************************************************************
!                                                                            *
!             AUTHOR:      G. WOTAWA                                         *
!             DATE:        1997-08-06                                        *
!             LAST UPDATE: 1997-10-10                                        *
!                                                                            *
!             Update:      1999-02-08, global fields allowed, A. Stohl*
!
!             Rewritten    21-05-2013  copy new gridcheck
!                          from flexpart9 with grib2, B. Legras
!                          03-04-2016  improved gridcheck by
!                          reading parId and grid from GRIB2 messages,
!                          B. Legras
!                                                                            *
!*****************************************************************************
!                                                                            *
! DESCRIPTION:                                                               *
!                                                                            *
! THIS SUBROUTINE DETERMINES THE GRID SPECIFICATIONS (LOWER LEFT              *
! LONGITUDE, LOWER LEFT LATITUDE, NUMBER OF GRID POINTS, GRID DIST-          *
! ANCE AND VERTICAL DISCRETIZATION OF THE ECMWF MODEL) FROM THE              *
! GRIB HEADER OF THE FIRST INPUT FILE. THE CONSISTANCY (NO CHANGES           *
! WITHIN ONE FLEXPART RUN) IS CHECKED IN THE ROUTINE "READWIND" AT           *
! ANY CALL.                                                                  *
!                                                                            *
! OUTPUT       error .true.   - can not read grid specifications             *
!              error .false.  - normal                                       *
!              oronew .true.  - Terrain heights given in grib files          *
!              oronew .false. - Terrain heights not specified in the         *
!                               grib files (old file standard)               *
!                                                                            *
! XLON0               geographical longitude of lower left gridpoint         *
! YLAT0               geographical latitude of lower left gridpoint          *
! NX                  number of grid points x-direction                      *
! NY                  number of grid points y-direction                      *
! DX                  grid distance x-direction                              *
! DY                  grid distance y-direction                              *
! NUVZ                number of grid points for horizontal wind              *
!                     components in z direction                              *
! NWZ                 number of grid points for vertical wind                *
!                     component in z direction                               *
! sizesouth, sizenorth give the map scale (i.e. number of virtual grid*
!                     points of the polar stereographic grid):               *
!                     used to check the CFL criterion                        *
! UVHEIGHT(1)-        heights of gridpoints where u and v are                *
! UVHEIGHT(NUVZ)      given                                                  *
! WHEIGHT(1)-         heights of gridpoints where w is given                 *
! WHEIGHT(NWZ)                                                               *
```

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

! note !! is for statements related to variables which exist in flexpart9
! but not in traczilla   B. Legras
! these variables are nxshift, nxminl, nyminl

   use coord
   use grib_api
   logical, intent(out) :: error, oronew
   integer :: ix,jy,i,ifn,ifield,j,k,numskip
   integer :: kumin,kumax,kwmin,kwmax
   real (kind=4) :: xaux1,xaux2,yaux1,yaux2
   real (kind=8) :: xaux1in,xaux2in,yaux1in,yaux2in
   real :: sizesouth,sizenorth,xauxa,pint

! VARIABLES AND ARRAYS NEEDED FOR GRIB DECODING

   integer :: ifile, iret, igrib, gotGrid
   integer :: gribVer,typSurf,parId,parLev,nbVert,pvsize
   integer, allocatable :: inbuff(:)
   real (kind=4), allocatable:: zsec4(:), pv(:)
   character(len=24) :: gribErrorMsg = 'Error reading grib file'
   character(len=20) :: gribFunction = 'gridcheck'

   error=0
!
   if(ideltas.gt.0) then
     ifn=1
   else
     ifn=numbwf
   endif

   allocate (zsec4(jpunp))
!
! OPENING OF DATA FILE (GRIB CODE)
!

   print *,path(3)(1:len_path(3))//trim(wfname(ifn))
   call grib_open_file(ifile,path(3)(1:len_path(3))//trim(wfname(ifn)),'r',iret)
   print *,'gridcheck> open file ',wfname(ifn)
   if(iret.ne.GRIB_SUCCESS) call printerror(ifn,error)
   if(error) return

   ifield=0
   gotGrid=0
   iret=0

!  Initialize ku and kw max and min
   kumax=0
   kwmax=0
   kumin=9999
   kwmin=9999

!  get first field
   call grib_new_from_file(ifile,igrib,iret)

   do while (iret/=GRIB_END_OF_FILE)  ! start loop on fields
      ifield=ifield+1   ! return point of the loop on fields
      parId=-1

      !first see if we read GRIB1 or GRIB2
```



```
      call grib_get_int(igrib,'editionNumber',gribVer,iret)
      call grib_check(iret,gribFunction,gribErrorMsg)

      if (gribVer.eq.1) then ! GRIB Edition 1
        !read the grib1 identifiers
        call grib_get_int(igrib,'indicatorOfParameter',parId,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'level',parLev,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)

      else  ! GRIB Edition 2
        !read the grib2 identifiers
        call grib_get_int(igrib,'paramId',parId,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'typeOfFirstFixedSurface',typSurf,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'level',parLev,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)

      endif

      !change code for etadot to code for omega
      if (parId.eq.77) parId=135

      !get the size and data of the values array
      if (parId.ne.-1) then
        call grib_get_real4_array(igrib,'values',zsec4,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
      endif

      if (ifield.eq.1) then
        print *,'gridcheck> processing grid'
        ! get the required fields from section 2 in a gribex compatible manner
        call grib_get_int(igrib,'numberOfPointsAlongAParallel', &
           nxfield,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'numberOfPointsAlongAMeridian', &
           ny,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_real8(igrib,'longitudeOfFirstGridPointInDegrees', &
           xaux1in,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'numberOfVerticalCoordinateValues', &
           nbVert,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_real8(igrib,'longitudeOfLastGridPointInDegrees', &
          xaux2in,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_real8(igrib,'latitudeOfLastGridPointInDegrees', &
          yaux1in,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_real8(igrib,'latitudeOfFirstGridPointInDegrees', &
          yaux2in,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        !  get the size and data of the vertical coordinate array
        call grib_get_size(igrib,'pv',pvsize,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
        allocate (pv(pvsize))
        call grib_get_real4_array(igrib,'pv',pv,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)

        nlev_ec=nbVert/2-1
```

```
        xaux1=xaux1in
        xaux2=xaux2in
        yaux1=yaux1in
        yaux2=yaux2in
        if (xaux1.gt.180.) xaux1=xaux1-360.0
        if (xaux2.gt.180.) xaux2=xaux2-360.0
        if (xaux1.lt.-180.) xaux1=xaux1+360.0
        if (xaux2.lt.-180.) xaux2=xaux2+360.0
        if (xaux2.lt.xaux1) xaux2=xaux2+360.0
        xlon0=xaux1
        ylat0=yaux1
        dx=(xaux2-xaux1)/real(nxfield-1)
        dy=(yaux2-yaux1)/real(ny-1)
        dxconst=180./(dx*r_earth*pi)
        dyconst=180./(dy*r_earth*pi)
        gotGrid=1

        ! Check whether fields are global
        ! If they contain the poles, specify polar stereographic map
        ! projections using the stlmbr- and stcm2p-calls
        !*****************************************************
        xauxa=abs(xaux2+dx-360.-xaux1)
        if (xauxa.lt.0.001) then
          nx=nxfield+1                 ! field is cyclic
          xglobal=.true.
          !!if (abs(nxshift).ge.nx) &
          !!   stop 'nxshift in file par_mod is too large'
          !!xlon0=xlon0+real(nxshift)*dx
        else
          nx=nxfield
          xglobal=.false.
          !!if (nxshift.ne.0) &
          !!    stop 'nxshift (par_mod) must be zero for non-global domain'
        endif

        if (xlon0.gt.180.) xlon0=xlon0-360.
        xauxa=abs(yaux1+90.)
        if (xglobal.and.xauxa.lt.0.001) then
          sglobal=.true.               ! field contains south pole
          ! Enhance the map scale by factor 3 (*2=6) compared to north-south
          ! map scale
          sizesouth=6.*(switchsouth+90.)/dy
          call stlmbr(southpolemap,-90.,0.)
          call stcm2p(southpolemap,0.,0.,switchsouth,0.,sizesouth, &
            sizesouth,switchsouth,180.)
          switchsouthg=(switchsouth-ylat0)/dy
        else
          sglobal=.false.
          switchsouthg=999999.
        endif
        xauxa=abs(yaux2-90.)
        if (xglobal.and.xauxa.lt.0.001) then
          nglobal=.true.               ! field contains north pole
          ! Enhance the map scale by factor 3 (*2=6) compared to north-south
          ! map scalecall
          sizenorth=6.*(90.-switchnorth)/dy
          call stlmbr(northpolemap,90.,0.)
          call stcm2p(northpolemap,0.,0.,switchnorth,0.,sizenorth, &
             sizenorth,switchnorth,180.)
          switchnorthg=(switchnorth-ylat0)/dy
        else
          nglobal=.false.
```



```
          switchnorthg=999999.
        endif
        call check_nx(error)
        call check_ny(error)
        if (error) return

      endif ! gotGrid

      k=parLev  ! level number
      select case (parId)
      case (129)
        oronew=.true.
        allocate (oro(0:nx-1,0:ny-1))
        do j=0,ny-1
          do i=0,nxfield-1
            oro(i,j)=zsec4(nxfield*(ny-j-1)+i+1)/ga
          enddo
          if (xglobal) oro(nx-1,j)=oro(0,j)
        enddo
      case (131)
        kumax=max(k,kumax)
        kumin=min(k,kumin)
        !iumax=max(iumax,nlev_ec-k+1) ! do we expect to get
      case (135)
        kwmax=max(k,kwmax)
        kwmin=min(k,kwmin)
        !iwmax=max(iwmax,nlev_ec-k+1) ! iumax or iwmax != nlev_ec
      case (172)
        allocate (lsm(0:nx-1,0:ny-1))
        do j=0,ny-1
          do i=0,nxfield-1
            lsm(i,j)=zsec4(nxfield*(ny-j-1)+i+1)
          enddo
          if (xglobal) lsm(nx-1,j)=lsm(0,j)
        enddo
      case (160)
        allocate (excessoro(0:nx-1,0:ny-1))
        do j=0,ny-1
          do i=0,nxfield-1
            excessoro(i,j)=zsec4(nxfield*(ny-j-1)+i+1)
          enddo
          if (xglobal) excessoro(nx-1,j)=excessoro(0,j)
          enddo
      end select

!     get next field
      call grib_release(igrib)
      call grib_new_from_file(ifile,igrib,iret)

    enddo   ! end loop on fields

    call grib_close_file(ifile)

    !error message if no fields found with correct first longitude in it
    if (gotGrid.eq.0) then
      print*,'***ERROR: input file needs to contain grid data'
      stop
    endif

    ! Determines vertical index boundaries
    nuvz=nlev_ec-kumin+1
    nuvz_b=nlev_ec-kumax+1
```

```
   nwz=nlev_ec-kwmin+1
   nwz_b=nlev_ec-kwmax+1
   ! One more point at the top for the vertical velocity with w=0
   if((nuvz.eq.nlev_ec).and.(nwz.eq.nlev_ec)) then
     nwz=nlev_ec+1
     w_top=.true.
   else
         w_top=.false.
   endif
   ! One more point at the bottom
   if(nuvz_b.eq.1) then
     nuvz=nuvz+1
     u_bot=.true.
   else
     u_bot=.false.
   endif

   call check_nuvz(error)
   call check_nwz(error)
   if (error) return

   ! If desired, shift all grids by nxshift grid cells
   !**************************************************

   !!if (xglobal) then
   !!  call shift_field_0(oro,nxfield,ny)
   !!  call shift_field_0(lsm,nxfield,ny)
   !!  call shift_field_0(excessoro,nxfield,ny)
   !!endif

! Output of grid info
!*******************

   write(*,*)
   !write(*,'(a,2i7)') &
   !     ' gribcheck> # of vertical levels in ECMWF data: ',  &
   !      nuvz_b,nuvz,nwz_b,nwz
   write(*,'(a,2i5,3L3)')' gribcheck> nx,ny,xglobal,nglobal,sglobal ', &
           nx, ny, xglobal, nglobal, sglobal
   write(*,'(a,2f10.2)')' gribcheck> switchsouthg, switchnorthg ', &
           switchsouthg, switchnorthg
   write(*,'(a,5i5,2L3)')' gridcheck> nlev_ec,nuvz_b,nuvz,nwz_b,nwz,u_bot,w_top', &
      nlev_ec,nuvz_b,nuvz,nwz_b,nwz,u_bot,w_top
   write(*,*)
   write(*,'(a)')  ' Mother domain:'
   write(*,'(a,f10.1,a1,f10.1,a,f10.1,a,i4)')  ' Longitude range: ',  &
         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(*,*)


! CALCULATE VERTICAL DISCRETIZATION OF ECMWF MODEL
! PARAMETER akm,bkm DESCRIBE THE HYBRID "ETA" COORDINATE SYSTEM

   !numskip=nlev_ec-nuvz  ! number of ecmwf model layers not used
                          ! by trajectory model
   numskip=0
   allocate(akm(nlev_ec+1),bkm(nlev_ec+1))
   do i=1,nlev_ec+1
        j=i
        k=nlev_ec+1+i
```



```
        akm(nlev_ec-i+2)=pv(j)
        bkm(nlev_ec-i+2)=pv(k)
   enddo
     !do i=1,nlev_ec+1
     !   print*,'gridcheck> akm, bkm ',akm(i),bkm(i)
     !enddo

! Assign the 10 m winds to an artificial model level with akz=0 and bkz=1.0,
! i.e. ground level
!*****************************************************************************
   if (u_bot) then
     allocate(akz(nlev_ec+1),bkz(nlev_ec+1))
     akz(1)=0.
     bkz(1)=1.0
     do i=1,nlev_ec
       akz(i+1)=0.5*(akm(i+1)+akm(i))
       bkz(i+1)=0.5*(bkm(i+1)+bkm(i))
     enddo
   else
     allocate(akz(nlev_ec),bkz(nlev_ec))
     do i=1,nlev_ec
       akz(i)=0.5*(akm(i+1)+akm(i))
       bkz(i)=0.5*(bkm(i+1)+bkm(i))
     enddo
   endif

   !print *,'gridcheck> nwz, nuvz ',nwz,nuvz
      print *,'min max U,V,T levels'
      do i=nuvz_b,nuvz,nuvz-nuvz_b
        write (*,'(i3,3g15.5)')i,akz(i),bkz(i),akz(i)+p0*bkz(i)
      enddo
      print *,'min max W levels'
      do i=nwz_b,nwz,nwz-nwz_b
        write (*,'(i3,3g15.5)')i,akm(i),bkm(i),akm(i)+p0*bkm(i)
      enddo

!  In order to avoid log(0), we pull down the upper boundary from
!  zero pressure to the top layer for (u,v)
!  vertical velocity at this level is defined in readwinds.f
!  This feature is not in the flexpart9 version: B. Legras

   if (w_top) then
     akm(nwz) = akz(nuvz)
     bkm(nwz) = bkz(nuvz)
   endif

   deallocate (zsec4,pv)
   return
   end subroutine gridcheck

!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


   subroutine check_nx (error)
   logical, intent(out) :: error
   if (nx.gt.nxmax) then
   write(*,*) 'Trajectory  model error: Too many grid points'// &
              ' in x direction.'
   write(*,*) 'Reduce resolution of wind fields.'
   error=.true.
   endif
   end subroutine check_nx
```

```
!-------------------------------------------------------------------------------
   subroutine check_ny (error)
   logical, intent(out) :: error
   if (ny.gt.nymax) then
   write(*,*) 'FLEXPART error: Too many grid points in y direction.'
   write(*,*) 'Reduce resolution of wind fields.'
   error=.true.
   endif
   end subroutine check_ny
!-------------------------------------------------------------------------------

   subroutine check_nuvz (error)
   logical, intent(out) :: error
   if (nuvz.gt.nuvzmax) then
   write(*,*) 'FLEXPART error: Too many u,v grid points in z '// &
      'direction.'
   write(*,*) 'Reduce resolution of wind fields.'
   error=.true.
   endif
   end subroutine check_nuvz
!-------------------------------------------------------------------------------

   subroutine check_nwz (error)
   logical, intent(out) :: error
   if (nwz.gt.nwzmax) then
   write(*,*) 'FLEXPART error: Too many w grid points in z '// &
     'direction.'
   write(*,*) 'Reduce resolution of wind fields.'
   error=.true.
   endif
   end subroutine check_nwz
!-------------------------------------------------------------------------------

   subroutine printerror (ifn,error)
   integer, intent(in) :: ifn
   logical, intent(out) :: error
   write(*,*)
   write(*,*) '############################################'// &
          '###### '
   write(*,*) '    TRAJECTORY MODEL SUBROUTINE GRIDCHECK:'
   write(*,*) ' CAN NOT OPEN INPUT DATA FILE '//wfname(ifn)
   write(*,*) '############################################'// &
          '###### '
   error=.true.
   end subroutine printerror

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

   subroutine getfields(itime,nstop)

!*****************************************************************************
!                                                                            *
!  This subroutine manages the 3 data fields to be kept in memory.           *
!  During the first time step of petterssen it has to be fulfilled that the  *
!  first data field must have |wftime|<itime, i.e. the absolute value of wftime*
!  must be smaller than the absolute value of the current time in [s].       *
!  The other 2 fields are the next in time after the first one.              *
!  Pointers (memind) are used, because otherwise one would have to resort the*
!  wind fields, which costs a lot of computing time. Here only the pointers are*
!  resorted.                                                                 *
```



```
!                                                                            *
!     Author: A. Stohl                                                       *
!                                                                            *
!     29 April 1994                                                          *
!                                                                            *
!*****************************************************************************
!  Changes, Bernd C. Krueger, Feb. 2001:
!        Variables tth,qvh,tthn,qvhn (on eta coordinates) in common block.
!        Function of nstop extended.
!           B. Legras, April 2002:
!        Nested operations cancelled.
!        uuh, vvh, wwh on two times and in common
!        Message printed when reading a field
!
!*****************************************************************************
!                                                                            *
! 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               *
! nx,ny,nuvz,nwz       field dimensions in x,y and z direction               *
! uuh(0:nxmax,0:nymax,nuvzmax,2)   wind components in x-direction [m/s]
*
! vvh(0:nxmax,0:nymax,nuvzmax,2)   wind components in y-direction [m/s]
*
! wwh(0:nxmax,0:nymax,nwzmax,2)    wind components in z-direction [deltaeta/s]
*
!                                                                            *
! Constants:                                                                 *
! idiffmax             maximum allowable time difference between 2 wind fields *
!                                                                             *
!*****************************************************************************

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

   data indmin/1/

! 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(*,*) 'FLEXPART WARNING: NO WIND FIELDS ARE AVAILABLE.'
     write(*,*) 'A TRAJECTORY HAS TO BE TERMINATED.'
     nstop=4
     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)
!*****************************************************
! to do: rewrite this with a WHILE

     do indj=indmin,numbwf-1
        if (ldirect*wftime(indj+1).gt.ldirect*itime) then
           call readwind(indj+1,memind(2))
!          call calcpar(memind(2))
           memtime(2)=wftime(indj+1)
           call verttransformB(memind(2))
           write(*,'(a,a,a,i11,a,i11)') &
                  ' getfields> file ',trim(wfname(indj+1)),&
                  ' memtime ',memtime(2),' time ',itime
           nstop = 1
           goto 40
        endif
     enddo
     go to 90
 40  indmin=indj ! memorize last value read

   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 readwind(indj,memind(1))
!          call calcpar(memind(1))
           memtime(1)=wftime(indj)
           write(*,'(a,a,a,i11,a,i11)') &
                  ' getfields> 1st ',trim(wfname(indj)),&
                  ' memtime ',memtime(1),' time ',itime
!          write(*,'(a,a,a,i11,i10,i8)') &
!                  ' getfields> 1st  ',trim(wfname(indj)),&
!                  '  time ',memtime(1),wfldat(indj),wfltim(indj)
           memind(2)=2
           call readwind(indj+1,memind(2))
!          call calcpar(memind(2))
           memtime(2)=wftime(indj+1)
           call verttransformB(memind(1)) ! needs Ps next time
           call verttransformB(memind(2))
           write(*,'(a,a,a,i11,a,i11)') &
                  ' getfields> 2nd ',trim(wfname(indj+1)),&
                  ' memtime ',memtime(2),' time ',itime
```



```
!          write(*,'(a,a,a,i11,i10,i8)') &
!                  ' getfields> 2nd  ',trim(wfname(indj+1)),&
!                  '  time ',memtime(2),wfldat(indj+1),wfltim(indj+1)
           nstop = 1
           goto 60
        endif
     enddo
     go to 90
 60  indmin=indj

   endif

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

   if (lwindinterv.gt.idiffmax) nstop=3

   return

90 print *,'getfields > ERROR '
   print *,'TRY TO GET FIELD BEYOND AVAILABLE LIST'
   print *,'last ',wfname(numbwf),wfldat(numbwf), &
                   wfltim(numbwf),wftime(numbwf)
   print *,'called ',itime
   stop

   end subroutine getfields

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

   subroutine readwind(indj,n)
!*****************************************************************************
!                                                                            *
!              TRAJECTORY MODEL SUBROUTINE READWIND                          *
!                                                                            *
!*****************************************************************************
!                                                                            *
!              AUTHOR:      G. WOTAWA                                        *
!              DATE:        1997-08-05                                       *
!              LAST UPDATE: 2000-10-17, Andreas Stohl                        *
!
!              REWRITTEN  : 21-05-2013  from new version in flexpart9
!                           handling grib2 fields, B. Legras
!                           03-04-2016  improved as gridcheck
!                           removed many checks, B. Legras
!                                                                            *
!*****************************************************************************
!  Changes, Bernd C. Krueger, Feb. 2001:
!        Variables tth and qvh (on eta coordinates) in common block
!           B. Legras, Apr. 2002:
!        Two times for uuh, vvh, wwh, in common
!        select case + loop inside
!*****************************************************************************
!                                                                            *
! DESCRIPTION:                                                               *
!                                                                            *
! READING OF ECMWF METEOROLOGICAL FIELDS FROM INPUT DATA FILES. THE          *
! INPUT DATA FILES ARE EXPECTED TO BE AVAILABLE IN GRIB CODE                 *
!                                                                            *
! INPUT:                                                                     *
! indj              indicates number of the wind field to be read in         *
```

```
! n                   temporal index for meteorological fields (1 to 3)*
!                                                                       *
! IMPORTANT VARIABLES FROM COMMON BLOCK:                                *
!                                                                       *
! wfname              File name of data to be read in                   *
! nx,ny,nuvz,nwz      expected field dimensions                         *
! nlev_ec             number of vertical levels ecmwf model             *
! uu,vv,ww            wind fields  m/s and Pa/s                         *
! tt,qv               temperature and specific humidity                 *
! ps                  surface pressure                                  *
!                                                                       *
!***********************************************************************
!
   use isentrop_h, only: isentropic_motion,theta_col,theta_inv_col
   use isentrop_m
   use ecmwf_diab, only: diabatic_w,mass_correction
   use grib_api

   integer, intent(in):: indj,n
   integer :: i,j,k,levdiff2,ifield,lunit

! VARIABLES AND ARRAYS NEEDED FOR GRIB DECODING

   integer :: gribVer, parLev, parId, typSurf
   integer :: nxf, nyf, nlevf
   !integer :: discipl, parCat, valSurf, parNum, gotGrid
   integer :: ifile, igrib, iret, off_bot
   !integer, allocatable :: isec2(:)
   real(kind=4), allocatable :: zsec4(:)
   !real(kind=4) :: xaux,yaux,xaux0,yaux0
   !real(kind=8) :: xauxin,yauxin
   !real,parameter :: eps=1.e-4
   !real :: plev1,pmean,tv,fu,hlev1,ff10m,fflev1
   !logical :: hflswitch,strswitch
   character(len=24) :: gribErrorMsg = 'Error reading grib file'
   character(len=20) :: gribFunction = 'readwind'
   character(len=256) :: fname, lock_file
   logical :: exfile,exlock
   integer :: cc

   integer :: OMP_GET_NUM_THREADS

   !hflswitch=.false.
   !strswitch=.false.
   levdiff2=nlev_ec-nwz+1
   allocate (zsec4(jpunp))
   !allocate (isec2(22+nx+ny))
!  allocate (ewss(0:nx-1,0:ny-1),nsss(0:nx-1,0:ny-1))

!
! OPENING OF DATA FILE (GRIB CODE)
! check first that the file is available

   fname=path(3)(1:len_path(3))//trim(wfname(indj))
   lock_file=path(3)(1:len_path(3))//'.'//trim(wfname(indj))//'.lock'
   inquire(file=fname,exist=exfile)
   inquire(file=lock_file,exist=exlock)
   cc=0
   do while (((.not.exfile).or.(exlock)).and.(cc<200))
      call sleep(15)
      print *,'wait ',fname
      flush 6
```



```
      cc=cc+1
      inquire(file=fname,exist=exfile)
      inquire(file=lock_file,exist=exlock)
   enddo

   call grib_open_file(ifile,fname,'r',iret)
   if(iret.ne.GRIB_SUCCESS) goto 999

!  print *,'readwind> file ',wfname(indj)

   ifield=0
   !gotGrid=0
   if (u_bot) then
     off_bot=1
   else
     off_bot=0
   endif

   call grib_new_from_file(ifile,igrib,iret)

   do while (iret/=GRIB_END_OF_FILE)  ! start loop on fields
      ifield=ifield+1
      parId=-1

      !first see if we read GRIB1 or GRIB2
      call grib_get_int(igrib,'editionNumber',gribVer,iret)
      !call grib_check(iret,gribFunction,gribErrorMsg)

      if (gribVer.eq.1) then ! GRIB Edition 1

        !print*,'GRiB Edition 1'
        !read the grib1 identifiers
        call grib_get_int(igrib,'indicatorOfParameter',parId,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'level',parLev,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)

      else

        !print*,'GRiB Edition 2'
        !read the grib2 identifiers
        call grib_get_int(igrib,'paramId',parId,iret)
        !call grib_get_int(igrib,'typeOfFirstFixedSurface',typSurf,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'level',Parlev,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)

      endif

      !change code for etadot to code for omega
      if (parId.eq.77) parId=135

      ! get the size and data of the values array
      if (parId.ne.-1) then
        call grib_get_real4_array(igrib,'values',zsec4,iret)
        call grib_check(iret,gribFunction,gribErrorMsg)
      endif

      ! get the required fields from section 2 in a gribex compatible manner
      if (ifield.eq.1) then
        call grib_get_int(igrib,'numberOfPointsAlongAParallel',nxf,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)
```

```
        call grib_get_int(igrib,'numberOfPointsAlongAMeridian',nyf,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)
        call grib_get_int(igrib,'numberOfVerticalCoordinateValues',nlevf,iret)
        !call grib_check(iret,gribFunction,gribErrorMsg)
         ! CHECK GRID SPECIFICATIONS
        if(nxf.ne.nxfield) stop 'READWIND: NX NOT CONSISTENT'
        if(nyf.ne.ny) stop 'READWIND: NY NOT CONSISTENT'
        if(nlevf/2-1.ne.nlev_ec) &
             stop 'READWIND: VERTICAL DISCRETIZATION NOT CONSISTENT'
      endif ! ifield

      k=parLev
      field_identifier: select case (parId)
! 3D fields
      case (130)              !! TEMPERATUR
        do j=0,ny-1 ; do i=0,nxfield-1
          tth(i,j,nlev_ec-k+1+off_bot,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ; enddo
      case (131)               !! U VELOCIT
        do j=0,ny-1 ; do i=0,nxfield-1
          uuh(i,j,nlev_ec-k+1+off_bot,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ; enddo
      case (132)              !! V VELOCITY
        do j=0,ny-1 ; do i=0,nxfield-1
          vvh(i,j,nlev_ec-k+1+off_bot,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ; enddo
      case (133)              !! SPEC. HUMIDITY
        if(TTLactiv .OR. CLAUSactiv) then
          do j=0,ny-1 ; do i=0,nxfield-1
            qvh(i,j,nlev_ec-k+1+off_bot,n) = max(zsec4(nxfield*(ny-j-1)+i+1),0.)
          enddo     ; enddo
        endif
      case (135)              !! W VELOCITY
        if(z_motion) then
          do j=0,ny-1 ; do i=0,nxfield-1
             wwh(i,j,nlev_ec-k+1,n) = zsec4(nxfield*(ny-j-1)+i+1)
          enddo       ; enddo
        endif
! surface fields
      case (134)              !! SURF. PRESS.
        do j=0,ny-1 ; do i=0,nxfield-1
           ps(i,j,1,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ; enddo
      case (152)              !! LN SURF. PRESS.
        do j=0,ny-1 ; do i=0,nxfield-1
           ps(i,j,1,n) = exp(zsec4(nxfield*(ny-j-1)+i+1))
        enddo       ; enddo
      case (165)              !! 10 M U VELOCITY
        do j=0,ny-1 ; do i=0,nxfield-1
           u10(i,j,1,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ;  enddo
      case (166)              !! 10 M V VELOCITY
        do j=0,ny-1 ; do i=0,nxfield-1
           v10(i,j,1,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ;  enddo
      case (167)              !! 2 M TEMPERATURE
        do j=0,ny-1 ; do i=0,nxfield-1
           tt2(i,j,1,n) = zsec4(nxfield*(ny-j-1)+i+1)
        enddo       ;  enddo
      end select field_identifier

!     get next field
```



```
      call grib_release(igrib)
      call grib_new_from_file(ifile,igrib,iret)
   enddo

! CLOSING OF INPUT DATA FILE
!
   call grib_close_file(ifile)

!     print *,'readwind> ',uuh(186,136,24,n),vvh(186,136,24,n),wwh(186,136,24,n)

!* Defines vertical velocity on the last level to be zero or to be half
!* of the last-1, according to the type of vertical interpolation

   if(z_motion.and.(levdiff2.eq.0).and.w_top) then
     select case (vert_interpol)
       case ('lin')
         wwh(:,:,nwz,n)=0.                        ! linear interpol
       case ('log')
         wwh(:,:,nwz,n)=0.5*wwh(:,:,nwz-1,n)    ! log    interpol
       case default
         stop 'ERROR 1 in readwind'
     end select
   endif

! For global fields, assign rightmost grid point the value of the
! leftmost point
!*****************************************************************

   if (xglobal) then
     ps(nx-1,:,1,n)=ps(0,:,1,n)
     if(u_bot) then
       tt2(nx-1,:,1,n)=tt2(0,:,1,n)
       u10(nx-1,:,1,n)=u10(0,:,1,n)
       u10(nx-1,:,1,n)=u10(0,:,1,n)
     endif
     uuh(nx-1,:,:,n)=uuh(0,:,:,n)
     vvh(nx-1,:,:,n)=vvh(0,:,:,n)
     tth(nx-1,:,:,n)=tth(0,:,:,n)
     if(TTLactiv .OR. CLAUSactiv) qvh(nx-1,:,:,n)=qvh(0,:,:,n)
     if(z_motion) wwh(nx-1,:,:,n)=wwh(0,:,:,n)
   endif

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

   if(diabatic_w.or.isentropic_motion) then
!       correct that as it is not defined outside a // region
!       have a variable which lets the code know the number of processors
!       declared in the PBS preamble
!       This is just inducing a waste of resources which can be significant
!       when parcels are within a localized cloud
!$OMP PARALLEL
!$OMP MASTER
     num_threads=OMP_GET_NUM_THREADS()
!$OMP END MASTER
!$OMP END PARALLEL
     if(num_threads==1) then
       theta_col(:,:,n)=.false.
       theta_inv_col(:,:,n)=.false.
       print *,'readwind > NOCOL'
     else
!$OMP PARALLEL DO DEFAULT(SHARED) SCHEDULE(DYNAMIC) PRIVATE(i,j)
```

```
      do j=0,ny-1
        do i=0,nx-1
          call calc_col_theta(i,j,n)
        enddo
      enddo
!$OMP END PARALLEL DO
    endif
  endif

! Assign 10 m wind to model level at eta=1.0 to have one additional model
! level at the ground
! Specific humidity is taken the same as at one level above
! Temperature is taken as 2 m temperature
!*********************************************************************

  if (u_bot) then
    uuh(:,:,1,n)=u10(:,:,1,n)
    vvh(:,:,1,n)=v10(:,:,1,n)
    if(TTLactiv .OR. CLAUSactiv) qvh(:,:,1,n)=qvh(:,:,2,n)
    tth(:,:,1,n)=tt2(:,:,1,n)
  endif

  !if(iumax/=nuvz-1) stop 'READWIND: NUVZ NOT CONSISTENT'
  !if(z_motion.and.(iwmax/=nwz))    stop 'READWIND: NWZ NOT CONSISTENT'

  deallocate (zsec4)

  ! diag min pressure at grid bottom
  print*, 'min pressure at bottom ', &
    akm(nwz_b)+bkm(nwz_b)*minval(minval(ps(:,:,1,n),DIM=2),DIM=1)

  return
  stop 'Execution terminated'
999 write(*,*) ' #### FLEXPART MODEL ERROR! WINDFIELD      #### '
  write(*,*) ' #### ',wfname(indj),'          #### '
  write(*,*) ' #### CANNOT BE OPENED !!!          #### '
  stop 'Execution terminated'

  end subroutine readwind

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


  subroutine verttransformB(n)
!                              i  i   i   i
!-cv     subroutine verttransformB(n,uuh,vvh,tth,qvh,wwh)
!*****************************************************************************
!                                                                            *
!     This subroutine calculates the pressure vertical velocity on           *
!     the eta grid by adding the correction due to horizontal advection      *
!     and dPs/dt to the eta vertical velocity.                               *
!     It also calculates the velocities in the polar region in stereographic *
!     map.                                                                   *
!     It will calculate the density and its vertical derivative when it is   *
!     necessary in the future.                                               *
!                                                                            *
!     Author: B. Legras                                                      *
!     date:   6 April 2002                                                   *
!                                                                            *
!     from a previous version of verttransform.f                             *
```



```
!
!     Modifications
!     (...)
!     5/12/07 Calculation of the velocity at the pole for southern
!     hemisphere. Previous calculation (in flexpart)
!     was much too complicated, though correct, and was inducing
!     occasionnaly spurious floating point exceptions
!                                                                            *
! Variables:                                                                 *
! nx,ny,nz                        field dimensions in x,y and z direction    *
! uuh(0:nxmax,0:nymax,nuvz,2)     wind components in x-direction [m/s]       *
! vvh(0:nxmax,0:nymax,nuvz,2)     wind components in y-direction [m/s]       *
! wwh(0:nxmax,0:nymax,nwz,2)      wind components in z-direction [deltaeta/s]*
! tth(0:nxmax,0:nymax,nuvz,2)     temperature [K]                            *
! ps(0:nxmax,0:nymax,2)           surface pressure [Pa]                      *
!                                                                            *
!*****************************************************************************

  use coord
  integer :: ix,jy,iz,n,ix1,jy1,ixp,jyp,ixi,nold
  real :: pint,pb,pt,pih,ut,vt,ub,vb
  real :: xlon,ylat,xlonr
  real :: uupolaux,vvpolaux,wdummy
  real :: dPsdx,dPsdy,dPsdt,uint,vint

  parameter(pih=pi/180.)

  logical init
  save init
  data init/.true./

! During the first call, the estimate of dPs/dt is done with the next time
! and the maximum height of the model is set (assuming constant pressure
! level at the top)

  if(init) then
    nold=2 ; init=.false. ; zmax=-log(akm(nwz)/p0)
  else
    nold=3-n
  endif

!   print *,'verttran> ',uuh(186,136,24,n),vvh(186,136,24,n),wwh(186,136,24,n)

! Loop over the whole grid
!*************************

!      do jy=0,ny-1
!        do ix=0,nx-1

!       Nothing to be done for the horizontal wind

!       Calculation of rho and drhodz to be done here when necessary
!        enddo
!      enddo

!*****************************************************************
! Compute slope of eta levels in windward direction and resulting
! vertical wind correction
!*****************************************************************
  if( z_motion.and.correct_vertwind ) then
    do jy=1,ny-2
      if(xglobal) then
```

```
          ixi=0
        else
          ixi=1
        endif
        do ix=ixi,nx-2
!         velocities in grid units per s
          ub=uuh(ix,jy,1,n)*dxconst/cos((float(jy)*dy+ylat0)*pih)
          vb=vvh(ix,jy,1,n)*dyconst
          pb=akz(1)+bkz(1)*ps(ix,jy,1,n)
          do iz=nwz_b,nwz-1
!           Wind and pressure on adjacent levels
            ut=uuh(ix,jy,iz+1,n)*dxconst/cos((float(jy)*dy+ylat0)*pih)
            vt=vvh(ix,jy,iz+1,n)*dyconst
            pt=akz(iz+1)+bkz(iz+1)*ps(ix,jy,1,n)
            pint=akm(iz)+bkm(iz)*ps(ix,jy,1,n)
!----------- temporary test
!            if((pint.lt.pt).or.(pint.gt.pb)) then
!              print *,'verttransform> ALARM!!: bad vertical ordering'
!              print *,ix,jy,iz,ps(ix,jy,1,n)
!              print *,pb,pint,pt
!              print *,akz(iz+1),akm(iz),akz(iz)
!              print *,bkz(iz+1),bkm(iz),bkz(iz)
!              print *,wwh(ix,jy,iz,n)
!            endif
!-----------
!           Interpolation of horizontal wind
            uint=(log(pint/pt)*ub + log(pb/pint)*ut)/log(pb/pt)
            vint=(log(pint/pt)*vb + log(pb/pint)*vt)/log(pb/pt)
!           Horizontal derivative of Ps
            ix1=modulo(ix-1,nx-1)
!----------- temporary test
!            if((ix1.lt.0).or.(ix1.gt.nx-1)) then
!              print *,'verttransform> ALARM!!: ix1 <0'
!            endif
!-----------
            jy1=jy-1
            ixp=ix+1
            jyp=jy+1
!           horizontal Ps gradient in Pa per grid unit
            dPsdx = (Ps(ixp,jy,1,n)-Ps(ix1,jy,1,n))/2.
            dPsdy = (Ps(ix,jyp,1,n)-Ps(ix,jy1,1,n))/2.

!           Estimation of temporal derivative of Ps at constant eta
!           unit Pa per s
            dPsdt = (Ps(ix,jy,1,n)-Ps(ix,jy,1,nold)) / &
                    (memtime(2) - memtime(1))

! TEST debut
!            if((ix==186).and.(jy==136).and.(iz==24)) then
!              print *,uint,vint,bkm(iz)
!              print *,dPsdx,dPsdy,dPsdt
!              print *,memtime(2),memtime(1)
!            endif
! TEST fin

!           Correction of vertical velocity
!           by horizontal advection and  dPs/dt
            wwh(ix,jy,iz,n) = wwh(ix,jy,iz,n)          &
              + (dPsdx*uint + dPsdy*vint + dPsdt) *bkm(iz)

            ub=ut ; vb=vt ; pb=pt
          enddo
```



```
        enddo
      enddo
    endif
!***  For the moment, the vertical velocities are not corrected
!***  on the first and last latitude and on the first and last
!***  longitude when xglobal=.false.
!***  TODO: apply first order estimate of spatial derivatives
!***  on the edges


! If north pole is in the domain, calculate wind velocities in polar
! stereographic coordinates
!*****************************************************************

    if (nglobal) then
      do jy=int(switchnorthg)-2,ny-1
        ylat=ylat0+float(jy)*dy
        do ix=0,nx-1
          xlon=xlon0+float(ix)*dx
          do iz=nuvz_b,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

! CALCULATE FFPOL, DDPOL FOR CENTRAL GRID POINT
! modified because the original version, though correct, was totally crazy
      do iz=nuvz_b,nuvz
        xlon=xlon0+float(nx/2-1)*dx
        xlonr=xlon*pi/180.
        jy=ny-1
        call cc2gll(northpolemap,90.,xlon,             &
             uuh(nx/2-1,jy,iz,n),vvh(nx/2-1,ny-1,iz,n), &
             uupolaux,vvpolaux)
        do ix=0,nx-1
          uupol(ix,jy,iz,n)=uupolaux
          vvpol(ix,jy,iz,n)=vvpolaux
        enddo
      enddo

! Fix: Set W at pole to the zonally averaged W of the next equator-
! ward parallel of latitude

      if(z_motion) then
        do iz=nwz_b,nwz
          wdummy=0.
          do ix=0,nx-1
            wdummy=wdummy+wwh(ix,ny-2,iz,n)
          enddo
          wdummy=wdummy/float(nx)
          do ix=0,nx-1
            wwh(ix,ny-1,iz,n)=wdummy
          enddo
        enddo
      endif
    endif


! If south pole is in the domain, calculate wind velocities in polar
! stereographic coordinates
```

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

   if (sglobal) then
     do jy=0,int(switchsouthg)+3
       ylat=ylat0+float(jy)*dy
       do ix=0,nx-1
         xlon=xlon0+float(ix)*dx
         do iz=nuvz_b,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

     do iz=nuvz_b,nuvz
! CALCULATE FFPOL, DDPOL FOR CENTRAL GRID POINT
       xlon=xlon0+float(nx/2-1)*dx
       xlonr=xlon*pi/180.
       jy=0
       call cc2gll(southpolemap,-90.,xlon,      &
         uuh(nx/2-1,jy,iz,n),vvh(nx/2-1,0,iz,n), &
            uupolaux, vvpolaux)
       do ix=0,nx-1
         uupol(ix,jy,iz,n)=uupolaux
         vvpol(ix,jy,iz,n)=vvpolaux
       enddo
     enddo

! Fix: Set W at pole to the zonally averaged W of the next equator-
! ward parallel of latitude

     if(z_motion) then
       do iz=nwz_b,nwz
         wdummy=0.
         do ix=0,nx-1
           wdummy=wdummy+wwh(ix,1,iz,n)
         enddo
         wdummy=wdummy/float(nx)
         do ix=0,nx-1
           wwh(ix,0,iz,n)=wdummy
         enddo
       enddo
     endif
   endif

!     print *,'verttran> ',uuh(186,136,24,n),vvh(186,136,24,n),wwh(186,136,24,n)

   return
   end subroutine verttransformB

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

   subroutine interpol_windB    &
         (itime,xt,yt,zt,dxdt,dydt,dzdt,ngrid,psaver,z_factor,tint)

!***********************************************************************
!                                                                      *
!  This subroutine interpolates the wind data to current trajectory position.  *
!                                                                      *
!    Author: A. Stohl                                                  *
```



```
!                                                                      *
!    16 December 1997                                                  *
!                                                                      *
!    Changes: B. Legras, April 2002                                    *
!             interpolation from eta winds                             *
!             variance calculation cancelled                           *
!             B. Legras, June 2002                                     *
!             optimisation of log calculations                         *
!             new calculation of z_factor for z and theta diffusion    *
!             B. Legras, December 2007                                 *
!             patch to avoid extrapolations at the top and bottom      *
!                                                                      *
!***********************************************************************
!                                                                      *
! Variables:                                                           *
! dxdt,dydt,dzdt     wind components in grid units per second          *
! itime [s]          current temporal position                         *
! memtime(3) [s]     times of the wind fields in memory                *
! xt,yt,zt           coordinates position for which wind data shall be calculat*
!                                                                      *
! Constants:                                                           *
!                                                                      *
!***********************************************************************

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

! Auxiliary variables needed for interpolation
   real :: u1(2),v1(2),w1(2),dt1,dt2,dtt,pint,dTdLogp,tp1(2)
   real :: psl(4,2),pr(4,2),prp(4,2),u(4,2),v(4,2),w(4,2),tp(4,2)
   real :: pint1, pint2, pint3, pint4
   integer :: i,m,indexh,indz(4,2)
   integer :: ix,jy,ixp,jyp
   real :: ddx,ddy,rddx,rddy,p1,p2,p3,p4

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

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

   ix=min(floor(xt),nx-2)       ;  jy=min(floor(yt),ny-2)
   ixp=ix+1          ;  jyp=jy+1
   ddx=modulo(xt-float(ix),1.) ;  ddy=yt-float(jy)
   rddx=1.-ddx       ;  rddy=1.-ddy
   p1=rddx*rddy      ;  p2=ddx*rddy
   p3=rddx*ddy       ;  p4=ddx*ddy
   psl(1,1)=ps(ix ,jy ,1,memind(1))
   psl(1,2)=ps(ix ,jy ,1,memind(2))
   psl(2,1)=ps(ixp,jy ,1,memind(1))
   psl(2,2)=ps(ixp,jy ,1,memind(2))
   psl(3,1)=ps(ix ,jyp,1,memind(1))
   psl(3,2)=ps(ix ,jyp,1,memind(2))
   psl(4,1)=ps(ixp,jyp,1,memind(1))
   psl(4,2)=ps(ixp,jyp,1,memind(2))
   !if (debug_out) then
   !  print *,xt,yt,ix,jy
   ! ! print *,ddx,ddy
   !  print *,'pi ',p1,p2,p3,p4
   !  !print *,psl
```

```
   !endif

! Calculate variables for time interpolation
!****************************************

   dt1=float(itime-memtime(1))
   dt2=float(memtime(2)-itime)
   dtt=1./(dt1+dt2)

! Ground pressure at the location of the particle
!************************************************

   psaver = (dt2*(p1*psl(1,1)+p2*psl(2,1)+p3*psl(3,1)+p4*psl(4,1))   &
          +  dt1*(p1*psl(1,2)+p2*psl(2,2)+p3*psl(3,2)+p4*psl(4,2)))  &
          *  dtt

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

!  Locates lower left corner
   pint = p0*exp(-zt)
   indz(1,1) = locuv(nuvz_b,nuvz,psl(1,1),pint)

!  Locates other points by assuming there are close to the first

   do i=2,4
     indz(i,1)=locuv2(indz(1,1),psl(i,1),pint)
   enddo
   do i=1,4
     indz(i,2)=locuv2(indz(1,1),psl(i,2),pint)
   enddo

   do i=1,4
     pr (i,1)=akz(indz(i,1)  )+bkz(indz(i,1)  )*psl(i,1)
     pr (i,2)=akz(indz(i,2)  )+bkz(indz(i,2)  )*psl(i,2)
     prp(i,1)=akz(indz(i,1)+1)+bkz(indz(i,1)+1)*psl(i,1)
     prp(i,2)=akz(indz(i,2)+1)+bkz(indz(i,2)+1)*psl(i,2)
   enddo

!     Patch to avoid extrapolation in the boundary layer near orography
!     and at the top of the atmosphere
   pint1=min(pint,min(pr(1,1),pr(1,2)))
   pint2=min(pint,min(pr(2,1),pr(2,2)))
   pint3=min(pint,min(pr(3,1),pr(3,2)))
   pint4=min(pint,min(pr(4,1),pr(4,2)))
   if(maxval(indz)==nuvz-1) pint=max(pint,maxval(prp)) ! partial patch

!*********************************************************************
! 1.) Bilinear horizontal interpolation
! This has to be done separately for 6 fields (Temporal(2)*Vertical(3))
!*********************************************************************

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

   if (ngrid < 0) then

     select case(vert_interpol)

      case('log')

        do m=1,2
```



```
          indexh=memind(m)

          u(1,m)=(uupol(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
                + uupol(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
                / (log(pr(1,m))-log(prp(1,m)))
          v(1,m)=(vvpol(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
                + vvpol(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
                / (log(pr(1,m))-log(prp(1,m)))
          u(2,m)=(uupol(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
                + uupol(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
                / (log(pr(2,m))-log(prp(2,m)))
          v(2,m)=(vvpol(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
                + vvpol(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
                / (log(pr(2,m))-log(prp(2,m)))
          u(3,m)=(uupol(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
                + uupol(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
                / (log(pr(3,m))-log(prp(3,m)))
          v(3,m)=(vvpol(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
                + vvpol(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
                / (log(pr(3,m))-log(prp(3,m)))
          u(4,m)=(uupol(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
                + uupol(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
                / (log(pr(4,m))-log(prp(4,m)))
          v(4,m)=(vvpol(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
                + vvpol(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
                / (log(pr(4,m))-log(prp(4,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)

        enddo

      case('lin')

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

          u(1,m)=(uupol(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m))  &
                + uupol(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1))  &
                / (pr(1,m)-prp(1,m))
          v(1,m)=(vvpol(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m))  &
                + vvpol(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1))  &
                / (pr(1,m)-prp(1,m))
          u(2,m)=(uupol(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m))  &
                + uupol(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2))  &
                / (pr(2,m)-prp(2,m))
          v(2,m)=(vvpol(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m))  &
                + vvpol(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2))  &
                / (pr(2,m)-prp(2,m))
          u(3,m)=(uupol(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m))  &
                + uupol(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3))  &
                / (pr(3,m)-prp(3,m))
          v(3,m)=(vvpol(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m))  &
                + vvpol(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3))  &
                / (pr(3,m)-prp(3,m))
          u(4,m)=(uupol(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m))  &
                + uupol(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4))  &
                / (pr(4,m)-prp(4,m))
          v(4,m)=(vvpol(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m))  &
                + vvpol(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4))  &
                / (pr(4,m)-prp(4,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)

      enddo

    end select

  else

    select case(vert_interpol)

     case('log')

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

        u(1,m)=(uuh(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
              + uuh(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
              / (log(pr(1,m))-log(prp(1,m)))
        v(1,m)=(vvh(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
              + vvh(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
              / (log(pr(1,m))-log(prp(1,m)))
        u(2,m)=(uuh(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
              + uuh(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
              / (log(pr(2,m))-log(prp(2,m)))
        v(2,m)=(vvh(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
              + vvh(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
              / (log(pr(2,m))-log(prp(2,m)))
        u(3,m)=(uuh(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
              + uuh(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
              / (log(pr(3,m))-log(prp(3,m)))
        v(3,m)=(vvh(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
              + vvh(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
              / (log(pr(3,m))-log(prp(3,m)))
        u(4,m)=(uuh(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
              + uuh(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
              / (log(pr(4,m))-log(prp(4,m)))
        v(4,m)=(vvh(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
              + vvh(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
              / (log(pr(4,m))-log(prp(4,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)

      enddo
     case('lin')

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

        u(1,m)=(uuh(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m)) &
              + uuh(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1)) &
              / (pr(1,m)-prp(1,m))
        v(1,m)=(vvh(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m)) &
              + vvh(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1)) &
              / (pr(1,m)-prp(1,m))
        u(2,m)=(uuh(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m)) &
              + uuh(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2)) &
              / (pr(2,m)-prp(2,m))
        v(2,m)=(vvh(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m)) &
              + vvh(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2)) &
              / (pr(2,m)-prp(2,m))
```



```
        u(3,m)=(uuh(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m)) &
              + uuh(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3)) &
              / (pr(3,m)-prp(3,m))
        v(3,m)=(vvh(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m)) &
              + vvh(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3)) &
              / (pr(3,m)-prp(3,m))
        u(4,m)=(uuh(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m)) &
              + uuh(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4)) &
              / (pr(4,m)-prp(4,m))
        v(4,m)=(vvh(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m)) &
              + vvh(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4)) &
              / (pr(4,m)-prp(4,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)

      enddo

    end select

  endif

! Calculation of z_factor
! Estimation of temperature near the particle
  tint =                                                          &
        (tth(ix,jy,indz(1,1),1)*(log(pint)-log(prp(1,1 )))   &
      + tth(ix ,jy ,indz(1,1)+1,1)*(log(pr(1,1))-log(pint))) &
      / (log(pr(1,1))-log(prp(1,1)))
  select case (diftype)
    case (1)          ! diffusion in z
      z_factor = ga/(r_air*tint)
    case (2)          ! diffusion in theta
      dTdLogp = (tth(ix,jy,indz(1,1),1)-tth(ix,jy,indz(1,1)+1,1))/ &
                (log(pr(1,1))-log(prp(1,1)))
      z_factor = abs((pint/p0)**kappa / (kappa*tint - dTdLogp))
      z_factor = min(z_factor,0.5)   ! bound z_factor to 50 times the
    case default                     ! tropospheric value
      z_factor = 0.
  end select

! Accurate calculation of the temperature if needed

  if (AccurateTemp) then

    select case(vert_interpol)

     case('log')

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

        tp(1,m)=(tth(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
              + tth(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
              / (log(pr(1,m))-log(prp(1,m)))
        tp(2,m)=(tth(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
              + tth(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
              / (log(pr(2,m))-log(prp(2,m)))
        tp(3,m)=(tth(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
              + tth(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
              / (log(pr(3,m))-log(prp(3,m)))
        tp(4,m)=(tth(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
              + tth(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
```

```
                / (log(pr(4,m))-log(prp(4,m)))

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

        enddo

      case('lin')

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

          tp(1,m)=(tth(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m))  &
                + tth(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1))  &
                / (pr(1,m)-prp(1,m))
          tp(2,m)=(tth(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m))  &
                + tth(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2))  &
                / (pr(2,m)-prp(2,m))
          tp(3,m)=(tth(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m))  &
                + tth(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3))  &
                / (pr(3,m)-prp(3,m))
          tp(4,m)=(tth(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m))  &
                + tth(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4))  &
                / (pr(4,m)-prp(4,m))

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

        enddo

     end select
     tint=(tp1(1)*dt2+tp1(2)*dt1)*dtt
 !     if (debug_out)  then
 !       print *,'T ',(tth(ix ,jy ,indz(1,m),memind(m)),tth(ixp,jy ,indz(2,m),me
mind(m)),&
 !         tth(ix,jyp,indz(3,m),memind(m)),tth(ixp,jyp,indz(4,m),memind(m)),m=1,
1)
 !       print *,'T ',(tth(ix ,jy ,indz(1,m)+1,memind(m)),tth(ixp,jy ,indz(2,m)+
1,memind(m)),&
 !         tth(ix,jyp,indz(3,m)+1,memind(m)),tth(ixp,jyp,indz(4,m)+1,memind(m)),
m=1,1)
 !       print *, 'P ',pint1,pint2,pint3,pint4
 !       print *, (pr(i,1),i=1,4)
 !       print *, (prp(i,1),i=1,4)
 !       print *,'TP ',(tp(i,1),i=1,4)
 !       print *,tp1
 !       print *,p1,p2,p3,p4
 !       print *,tint
 !     endif
   endif

   !if(debug_out) &
   !  print "('interpol>',i3,' P ',3f7.0,' T ',3f7.2,' TH ',3f7.2)", &
   !     indz(1,1),pint,pr(1,1),prp(1,1), &
   !     tint,tth(ix,jy,indz(1,1),memind(1)),tth(ix,jy,indz(1,1)+1,memind(1)), &
   !     tint*(p0/pint)**kappa,tth(ix,jy,indz(1,1),memind(1))*(p0/pr(1,1))**kapp
a, &
   !                           tth(ix,jy,indz(1,1)+1,memind(1))*(p0/prp(1,1))**k
appa

! Localisation in w levels

   indz(1,1) = locw2(indz(1,1),psl(1,1),pint)
   do i=2,4
```



```
     indz(i,1)=locw2(indz(1,1),psl(i,1),pint)
   enddo
   do i=1,4
     indz(i,2)=locw2(indz(1,1),psl(i,2),pint)
   enddo

   do i=1,4
     pr (i,1)=akm(indz(i,1)  )+bkm(indz(i,1)  )*psl(i,1)
     prp(i,1)=akm(indz(i,1)+1)+bkm(indz(i,1)+1)*psl(i,1)
     pr (i,2)=akm(indz(i,2)  )+bkm(indz(i,2)  )*psl(i,2)
     prp(i,2)=akm(indz(i,2)+1)+bkm(indz(i,2)+1)*psl(i,2)
   enddo

!  Patch to avoid extrapolation in the boundary layer near orography
!  and at the top of the atmosphere
   pint1=min(pint,min(pr(1,1),pr(1,2)))
   pint2=min(pint,min(pr(2,1),pr(2,2)))
   pint3=min(pint,min(pr(3,1),pr(3,2)))
   pint4=min(pint,min(pr(4,1),pr(4,2)))
   if(maxval(indz)==nwz-1) pint=max(pint,maxval(prp)) ! partial patch

   select case(vert_interpol)

      case('log')

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

          w(1,m)=(wwh(ix ,jy ,indz(1,m)  ,indexh)*(log(pint1)-log(prp(1,m))) &
                + wwh(ix ,jy ,indz(1,m)+1,indexh)*(log(pr(1,m))-log(pint1))) &
                / (log(pr(1,m))-log(prp(1,m)))
          w(2,m)=(wwh(ixp,jy ,indz(2,m)  ,indexh)*(log(pint2)-log(prp(2,m))) &
                + wwh(ixp,jy ,indz(2,m)+1,indexh)*(log(pr(2,m))-log(pint2))) &
                / (log(pr(2,m))-log(prp(2,m)))
          w(3,m)=(wwh(ix ,jyp,indz(3,m)  ,indexh)*(log(pint3)-log(prp(3,m))) &
                + wwh(ix ,jyp,indz(3,m)+1,indexh)*(log(pr(3,m))-log(pint3))) &
                / (log(pr(3,m))-log(prp(3,m)))
          w(4,m)=(wwh(ixp,jyp,indz(4,m)  ,indexh)*(log(pint4)-log(prp(4,m))) &
                + wwh(ixp,jyp,indz(4,m)+1,indexh)*(log(pr(4,m))-log(pint4))) &
                / (log(pr(4,m))-log(prp(4,m)))

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

       enddo

      case('lin')

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

          w(1,m)=(wwh(ix ,jy ,indz(1,m)  ,indexh)*(pint1-prp(1,m))  &
                + wwh(ix ,jy ,indz(1,m)+1,indexh)*(pr(1,m)-pint1))  &
                / (pr(1,m)-prp(1,m))
          w(2,m)=(wwh(ixp,jy ,indz(2,m)  ,indexh)*(pint2-prp(2,m))  &
                + wwh(ixp,jy ,indz(2,m)+1,indexh)*(pr(2,m)-pint2))  &
                / (pr(2,m)-prp(2,m))
          w(3,m)=(wwh(ix ,jyp,indz(3,m)  ,indexh)*(pint3-prp(3,m))  &
                + wwh(ix ,jyp,indz(3,m)+1,indexh)*(pr(3,m)-pint3))  &
                / (pr(3,m)-prp(3,m))
          w(4,m)=(wwh(ixp,jyp,indz(4,m)  ,indexh)*(pint4-prp(4,m))  &
                + wwh(ixp,jyp,indz(4,m)+1,indexh)*(pr(4,m)-pint4))  &
                / (pr(4,m)-prp(4,m))
```

```
        w1(m)=p1*w(1,m)+p2*w(2,m)+p3*w(3,m)+p4*w(4,m)

      enddo

   end select

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

   dxdt=(u1(1)*dt2+u1(2)*dt1)*dtt
   dydt=(v1(1)*dt2+v1(2)*dt1)*dtt
   dzdt=(w1(1)*dt2+w1(2)*dt1)*dtt
   !---- Temporary test
   !if(debug_out) write(*,'(a,3g12.3)') 'interpol> u,v,w ',dxdt,dydt,dzdt
   !----

   return

   end subroutine interpol_windB

!@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


   function locuv(ib1,iu1,psu,pint)
   integer, intent(in) :: ib1,iu1
   real, intent(in) :: psu,pint
   integer :: locuv,ib,iu,im
   real :: pm
   ib=ib1 ; iu=iu1
   do while(iu-ib>1)
     im = (iu+ib)/2
     pm = akz(im) + bkz(im)*psu
     if( pm >= pint ) then
       ib=im
     else
       iu=im
     endif
   enddo
   locuv = ib
   end function locuv
!-------------------------------------------------------------------------------

   function locuv2(ib1,psu,pint)
   integer, intent(in) :: ib1
   real, intent(in) :: psu,pint
   integer :: locuv2,ib
   real :: pm
   ib=ib1
   pm = akz(ib) + bkz(ib)*psu
   if (pm < pint) then
     do while(ib > nuvz_b)
       ib = ib-1
       pm = akz(ib) + bkz(ib)*psu
       if (pm >= pint) exit
     enddo
   else
     do while(ib < nuvz-1)
       pm = akz(ib+1) + bkz(ib+1)*psu
       if (pm < pint) exit
       ib = ib+1
```



```
     enddo
   endif
   locuv2 = ib
   end function locuv2
!-------------------------------------------------------------------------------
   function locw(ib1,iu1,psu,pint)
   integer, intent(in) :: ib1, iu1
   real, intent(in) :: psu,pint
   integer :: locw,ib,iu,im
   real :: pm
   ib=ib1 ; iu=iu1
   do while(iu-ib>1)
     im = (iu+ib)/2
     pm = akm(im) + bkm(im)*psu
     if( pm >= pint ) then
       ib=im
     else
       iu=im
     endif
   enddo
   locw = ib
   end function locw
!-------------------------------------------------------------------------------

   function locw2(ib1,psu,pint)
   integer, intent(in) :: ib1
   real, intent(in) :: psu, pint
   integer :: locw2,ib
   real :: pm
   ib=ib1
   pm = akm(ib) + bkm(ib)*psu
   if (pm < pint) then
     do while(ib > nwz_b)
       ib = ib-1
       pm = akm(ib) + bkm(ib)*psu
       if (pm >= pint) exit
     enddo
   else
     do while(ib < nwz-1)
       pm = akm(ib+1) + bkm(ib+1)*psu
       if (pm < pint) exit
       ib = ib+1
     enddo
   endif
   locw2 = ib
   end function locw2

end module readinterpN
!
!$Log:
```