```fortran
module barotropic_dynamics_mod

use               fms_mod, only: open_namelist_file,   &
                                 open_restart_file,    &
                                 file_exist,           &
                                 check_nml_error,      &
                                 error_mesg,           &
                                 FATAL, WARNING,       &
                                 write_version_number, &
                                 mpp_pe,               &
                                 mpp_root_pe,          &
                                 read_data,            &
                                 write_data,           &
                                 set_domain,           &
                                 close_file,           &
                                 stdlog

use    time_manager_mod,  only : time_type,      &
                                 get_time,       &
                                 operator(==),   &
                                 operator(-)

use         constants_mod,  only: radius, omega

use         transforms_mod, only: transforms_init,         transforms_end,          &
                                  get_grid_boundaries,                              &
                                  trans_spherical_to_grid, trans_grid_to_spherical, &
                                  compute_laplacian,                                &
                                  get_sin_lat,             get_cos_lat,             &
                                  get_deg_lon,             get_deg_lat,             &
                                  get_grid_domain,         get_spec_domain,         &
                                  spectral_domain,         grid_domain,             &
                                  vor_div_from_uv_grid,    uv_grid_from_vor_div,    &
                                  horizontal_advection

use   spectral_damping_mod, only: spectral_damping_init,  &
                                  compute_spectral_damping

use           leapfrog_mod, only: leapfrog

use       fv_advection_mod, only: fv_advection_init, &
                                  a_grid_horiz_advection

use           stirring_mod, only: stirring_init, stirring, stirring_end


!
===================================================================================
implicit none
private
!
===================================================================================
logical :: used

public :: barotropic_dynamics_init, &
          barotropic_dynamics,      &
          barotropic_dynamics_end,  &
          dynamics_type,            &
          grid_type,                &
          spectral_type,            &
```

```
          tendency_type


! version information
!===================================================================
character(len=128) :: version = '$Id: barotropic_dynamics.f90,v 10.0 2003/10/24
22:00:57 fms Exp $'
character(len=128) :: tagname = '$Name: latest $'
!===================================================================

type grid_type
! Original model variables
   real, pointer, dimension(:,:,:) :: u=>NULL(), v=>NULL(), vor=>NULL(), trs=>NULL(),
tr=>NULL(), &
! Momentum, Vorticity budget variables
utend=>NULL(), vtend=>NULL(), vortend=>NULL(), vq=>NULL(), uq=>NULL(), utrunc=>NULL(),
vtrunc=>NULL(), & vortrunc=>NULL(), dt_vor=>NULL(), vlindamp=>NULL(), &
ulindamp_e=>NULL(), ulindamp_m=>NULL(), vorlindamp_e=>NULL(), vorlindamp_m=>NULL(),
uspecdamp=>NULL(), vspecdamp=>NULL(), vorspecdamp=>NULL(),ustir=>NULL(), vstir=>NULL(),
vorstir=>NULL(), &
urobdamp=>NULL(), vrobdamp=>NULL(), vorrobdamp=>NULL(), dx_u2_v2=>NULL(), dy_u2_v2=>NULL
(), rvor_advec=>NULL(), pvor_advec=>NULL(), beta=>NULL(), beta_star=>NULL(),
tester=>NULL(), &
! Energy
energy=>NULL(), energy_tend=>NULL(), energy_voradvec=>NULL(), energy_gradterm=>NULL(),
energy_trunc=>NULL(), energy_lindamp_m=>NULL(), energy_lindamp_e=>NULL(),
energy_specdamp=>NULL(), &
energy_stir=>NULL(), energy_robdamp=>NULL(), energy_tend_mean=>NULL(),
energy_voradvec_mean=>NULL(), energy_gradterm_mean=>NULL(), energy_trunc_mean=>NULL(),
energy_lindamp_m_mean=>NULL(), &
energy_lindamp_e_mean=>NULL(), energy_specdamp_mean=>NULL(), energy_stir_mean=>NULL(),
energy_robdamp_mean=>NULL(), energy_mean=>NULL(), &
! Enstrophy
enstrophy=>NULL(), enstrophy_tend=>NULL(), &
! Joe variables
vvor_beta=>NULL(), source=>NULL(), source_beta=>NULL(), sink=>NULL(), sink_beta=>NULL
(), dens_dt=>NULL(), robertssink=>NULL(), ensflux=>NULL(), ensflux_div=>NULL
(),ensflux_div_beta=>NULL()
   real, pointer, dimension(:,:)   :: pv=>NULL(), stream=>NULL()
end type
type spectral_type
   complex, pointer, dimension(:,:,:) :: vor=>NULL(), trs=>NULL(), roberts=>NULL()
end type
type tendency_type
   real, pointer, dimension(:,:) :: u=>NULL(), v=>NULL(), trs=>NULL(), tr=>NULL()
end type
type dynamics_type
   type(grid_type)     :: grid
   type(spectral_type) :: spec
   type(tendency_type) :: tend
   integer             :: num_lon, num_lat
   logical             :: grid_tracer, spec_tracer
end type

integer, parameter :: num_time_levels = 2

integer :: is, ie, js, je, ms, me, ns, ne, icnt

logical :: module_is_initialized = .false.

real,  allocatable, dimension(:)   :: sin_lat, cos_lat, rad_lat, rad_lon, &
```

```
                                     deg_lat, deg_lon, u_init, v_init, &
                                     coriolis, glon_bnd, glat_bnd

integer :: pe, npes

! namelist parameters with default values

logical  :: check_fourier_imag = .false.
logical  :: south_to_north     = .true.
logical  :: triang_trunc       = .true.

real     :: robert_coeff       = 0.04
real     :: longitude_origin   = 0.0

character(len=64) :: damping_option = 'resolution_dependent'
integer  :: damping_order      = 4
real     :: damping_coeff      = 1.e-04

real     :: zeta_0     = 8.e-05
real     :: tau        = 691200.
real     :: tau1       = 691200.
integer  :: m_0        = 4
integer  :: read_file  = 1
integer  :: damp       = 1.0
real     :: mult       = 0.0
real     :: linmult    = 0.0
real     :: eddy_width = 15.0
real     :: eddy_lat   = 45.0

logical  :: spec_tracer      = .true.
logical  :: grid_tracer      = .true.

integer  :: num_lat            = 128
integer  :: num_lon            = 256
integer  :: num_fourier        = 85
integer  :: num_spherical      = 86
integer  :: fourier_inc        = 1

real, dimension(2) :: valid_range_v = (/-1.e3,1.e3/)

namelist /barotropic_dynamics_nml/ check_fourier_imag, south_to_north, &
                                 triang_trunc,                         &
                                 num_lon, num_lat, num_fourier,        &
                                 num_spherical, fourier_inc,           &
                                 longitude_origin, damping_option,     &
                                 damping_order,     damping_coeff,     &
                                 robert_coeff,                         &
                                 spec_tracer, grid_tracer,             &
                                 eddy_lat, eddy_width, zeta_0, m_0, damp, mult,
linmult, tau, tau1, read_file, &
                                 valid_range_v

contains

!
==========================================================================================

subroutine barotropic_dynamics_init (Dyn,  Time, Time_init, dt_real, id_lon, id_lat,
id_lonb, id_latb)

type(dynamics_type), intent(inout)  :: Dyn
```

```
type(time_type)    , intent(in)     :: Time, Time_init
real, intent(in) :: dt_real
integer, intent(out) :: id_lon, id_lat, id_lonb, id_latb

integer :: i, j

real,    allocatable, dimension(:)   :: glon_bnd, glat_bnd
complex, allocatable, dimension(:,:) :: div
real :: xx, yy, dd

integer  :: ierr, io, unit, pe
logical  :: root

! < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > <
> < > < >

call write_version_number (version, tagname)

pe   =  mpp_pe()
root = (pe == mpp_root_pe())

if (file_exist('input.nml')) then
  unit = open_namelist_file ()
  ierr=1
  do while (ierr /= 0)
    read  (unit, nml=barotropic_dynamics_nml, iostat=io, end=10)
    ierr = check_nml_error (io, 'barotropic_dynamics_nml')
  enddo
  10 call close_file (unit)
endif


if (root) write (stdlog(), nml=barotropic_dynamics_nml)

call transforms_init(radius, num_lat, num_lon, num_fourier, fourier_inc, num_spherical,
&
         south_to_north=south_to_north,   &
         triang_trunc=triang_trunc,       &
         longitude_origin=longitude_origin       )

call get_grid_domain(is,ie,js,je)
call get_spec_domain(ms,me,ns,ne)

Dyn%num_lon      = num_lon
Dyn%num_lat      = num_lat
Dyn%spec_tracer  = spec_tracer
Dyn%grid_tracer  = grid_tracer

allocate (sin_lat   (js:je))
allocate (cos_lat   (js:je))
allocate (deg_lat   (js:je))
allocate (deg_lon   (is:ie))
allocate (rad_lat   (js:je))
allocate (rad_lon   (is:ie))
allocate (coriolis  (js:je))

allocate (glon_bnd  (num_lon + 1))
allocate (glat_bnd  (num_lat + 1))

 call get_deg_lon (deg_lon)
 call get_deg_lat (deg_lat)
```

```
 call get_sin_lat (sin_lat)
 call get_cos_lat (cos_lat)
 call get_grid_boundaries (glon_bnd, glat_bnd, global=.true.)

coriolis = 2*omega*sin_lat

rad_lat = deg_lat*atan(1.0)/45.0
rad_lon = deg_lon*atan(1.0)/45.0

 call spectral_damping_init(damping_coeff, damping_order, damping_option, num_fourier,
num_spherical, 1, 0., 0., 0.)

 call stirring_init(dt_real, Time, id_lon, id_lat, id_lonb, id_latb)

allocate (Dyn%spec%vor (ms:me, ns:ne, num_time_levels))
allocate (Dyn%grid%u   (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%v   (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vor (is:ie, js:je, num_time_levels))

allocate (Dyn%tend%u        (is:ie, js:je))
allocate (Dyn%tend%v        (is:ie, js:je))
allocate (Dyn%grid%stream   (is:ie, js:je))
allocate (Dyn%grid%pv       (is:ie, js:je))

!---------------------- Extra terms -------------------------------------------
! Momentum and Vorticity Budget Terms
allocate (Dyn%grid%utend                (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vtend                (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vortend              (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vq                   (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%uq                   (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%dt_vor               (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vlindamp             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ulindamp_e           (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ulindamp_m           (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vorlindamp_e         (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vorlindamp_m         (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%utrunc               (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vtrunc               (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vortrunc             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%dx_u2_v2             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%dy_u2_v2             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%uspecdamp            (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vspecdamp            (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vorspecdamp          (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ustir                (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vstir                (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vorstir              (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%urobdamp             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vrobdamp             (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%vorrobdamp           (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%rvor_advec           (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%pvor_advec           (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%beta                 (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%beta_star            (is:ie, js:je, num_time_levels))

allocate (Dyn%Grid%tester               (is:ie, js:je, num_time_levels))

! Energy Equation Terms
allocate (Dyn%Grid%energy               (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_tend          (is:ie, js:je, num_time_levels))
```

```
allocate (Dyn%Grid%energy_voradvec      (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_gradterm      (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_trunc         (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_lindamp_m     (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_lindamp_e     (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_specdamp      (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_stir          (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_robdamp       (is:ie, js:je, num_time_levels))

allocate (Dyn%Grid%energy_mean         (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_tend_mean     (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_voradvec_mean (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_gradterm_mean (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_trunc_mean    (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_lindamp_m_mean(is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_lindamp_e_mean(is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_specdamp_mean (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_stir_mean     (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%energy_robdamp_mean  (is:ie, js:je, num_time_levels))

! Enstrophy Equation Terms
allocate (Dyn%Grid%enstrophy            (is:ie, js:je, num_time_levels))
allocate (Dyn%Grid%enstrophy_tend       (is:ie, js:je, num_time_levels))

! Joe Terms
allocate (Dyn%grid%vvor_beta            (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%source               (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%source_beta          (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%sink                 (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%sink_beta            (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%robertssink          (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%dens_dt              (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ensflux              (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ensflux_div          (is:ie, js:je, num_time_levels))
allocate (Dyn%grid%ensflux_div_beta     (is:ie, js:je, num_time_levels))

allocate (Dyn%spec%roberts              (ms:me, ns:ne, num_time_levels))

! ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
allocate (div (ms:me, ns:ne))

 call fv_advection_init(num_lon, num_lat, glat_bnd, 360./float(fourier_inc))
if(Dyn%grid_tracer) then
  allocate(Dyn%Grid%tr (is:ie, js:je, num_time_levels))
  allocate(Dyn%Tend%tr (is:ie, js:je))
endif

if(Dyn%spec_tracer) then
  allocate(Dyn%Grid%trs (is:ie, js:je, num_time_levels))
  allocate(Dyn%Tend%trs (is:ie, js:je))
  allocate(Dyn%Spec%trs (ms:me, ns:ne, num_time_levels))
endif

if(Time == Time_init) then

if(read_file == 1) then
        open (unit=1, file='/scratch/z3410755/u.txt', status='old', action='read')
        read (1,*) icnt
        allocate(u_init(icnt))
                do i = 1, icnt
                read (1,*) u_init(i)
```

```
                end do
          do j = js, je
            Dyn%Grid%u(:,j,1) =   mult*u_init(j)
            Dyn%Grid%v(:,j,1) = 0.0 ! Dyn%Grid%v cannot be given any other value here
(by mass continuity). If another value is given it is ignored. To load a zonally
varying v profile (that still obeys                                            ! mass
continuity) use read_file = 4 initialisation
          end do
endif

if(read_file == 2) then
 call read_data('INPUT/u.nc', 'u',   Dyn%Grid%u  (:,:,1), grid_domain, timelevel=2)
   do j = js, je
Dyn%Grid%u  (:,j,1) = sum(Dyn%Grid%u(:,j,1))/128
Dyn%Grid%v(:,j,1) = 0.0
end do
endif


if(read_file == 3) then
 call read_data('INPUT/u.nc', 'ug',   Dyn%Grid%u  (:,:,1), grid_domain, timelevel=2)
   do j = js, je
Dyn%Grid%u  (:,j,1) = sum(Dyn%Grid%u(:,j,1))/128
Dyn%Grid%v(:,j,1) = 0.0
end do
endif


if(read_file == 4) then ! read full lon-lat u & v wind profile from .nc files
 call read_data('/srv/ccrc/data15/z3410755/init_data/u.nc', 'u',   Dyn%grid%u  (:,:,1),
grid_domain, timelevel=1)
 call read_data('/srv/ccrc/data15/z3410755/init_data/v.nc', 'v',   Dyn%Grid%v  (:,:,1),
grid_domain, timelevel=1)
endif

  call vor_div_from_uv_grid(Dyn%Grid%u(:,:,1), Dyn%Grid%v(:,:,1), Dyn%Spec%vor(:,:,1),
div)

  call trans_spherical_to_grid(Dyn%Spec%vor(:,:,1), Dyn%Grid%vor(:,:,1))

  do j = js, je
    do i = is, ie
      yy = (deg_lat(j)- eddy_lat)/eddy_width
      Dyn%Grid%vor(i,j,1) = Dyn%Grid%vor(i,j,1) + &
              0.5*zeta_0*cos_lat(j)*exp(-yy*yy)*cos(m_0*rad_lon(i))
    end do
  end do

 call trans_grid_to_spherical(Dyn%Grid%vor(:,:,1), Dyn%Spec%vor(:,:,1))

  div = (0.,0.)
  call uv_grid_from_vor_div   (Dyn%Spec%vor(:,:,1), div,        &
                              Dyn%Grid%u  (:,:,1), Dyn%Grid%v  (:,:,1))

if(Dyn%grid_tracer) then
    Dyn%Grid%tr = 0.0
    do j = js, je
      if(deg_lat(j) > 10.0 .and. deg_lat(j) < 20.0) Dyn%Grid%tr(:,j,1) =  1.0
      if(deg_lat(j) > 70.0 )                        Dyn%Grid%tr(:,j,1) = -1.0
    end do
  endif
```

```
  if(Dyn%spec_tracer) then
    Dyn%Grid%trs = 0.0
    do j = js, je
      if(deg_lat(j) > 10.0 .and. deg_lat(j) < 20.0) Dyn%Grid%trs(:,j,1) =  1.0
      if(deg_lat(j) > 70.0 )                        Dyn%Grid%trs(:,j,1) = -1.0
    end do
    call trans_grid_to_spherical(Dyn%Grid%trs(:,:,1), Dyn%Spec%trs(:,:,1))
  endif

else
  call read_restart(Dyn)
endif

module_is_initialized = .true.

return
end subroutine barotropic_dynamics_init

!
==============================================================================================

subroutine barotropic_dynamics(Time, Time_init, Dyn, previous, current, future, delta_t)

type(time_type)    , intent(in)     :: Time, Time_init
type(dynamics_type), intent(inout)  :: Dyn
integer, intent(in   )  :: previous, current, future
real,    intent(in   )  :: delta_t

! < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < 
> < > < >

complex, dimension(ms:me, ns:ne)  :: dt_vors, dt_vors_prev, damped, dt_divs,
dt_divs_throwaway, stream, zeros, spec_diss
complex, dimension(ms:me, ns:ne)  :: vors_rub, divs_rub, dt_vors_rub, dt_divs_rub,
rob_vors_diff
real,    dimension(is:ie, js:je)  :: dt_vorg, stir, umean_current, vmean_current,
uprime, vormean, vorprime, pvmean, energyprev, energymean_prev
real,    dimension(is:ie, js:je)  :: uprev, vprev, vorprev, zerog, rob_v_diff,
rob_u_diff, rob_vorg_diff, energytendcheck, energytendcheck_mean
real,    dimension(is:ie, js:je)  :: umean_prev, vmean_prev, umean_future,
vmean_future, utendmean, vtendmean, vqmean, uqmean, dx_u2_v2mean, dy_u2_v2mean,
utruncmean, vtruncmean, ulindamp_mmean, &
                                     ulindamp_emean, vlindampmean, uspecdampmean,
vspecdampmean, ustirmean, vstirmean, urobdampmean, vrobdampmean
real,    dimension(is:ie, js:je)  :: vormeanprev, vorprimeprev, enstrophyprev,
enstrophytendcheck, vortendmean, pvor_advecmean, rvor_advecmean, vortruncmean,
vorlindamp_emean, vorlindamp_mmean, &
                                     vorspecdampmean, vorstirmean, vorrobdampmean,
vortendprime, pvor_advecprime, rvor_advecprime, vortruncprime, vorlindamp_eprime,
vorlindamp_mprime, vorspecdampprime, &
                                     vorstirprime, vorrobdampprime
real,    dimension(is:ie, js:je)  :: f_array, pvmean_advec, beta_star
integer :: j

! < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < > < 
> < > < >

if(.not.module_is_initialized) then
  call error_mesg('barotropic_dynamics','dynamics has not been initialized ', FATAL)
endif
```

```
zeros = (0.,0.) ! Zero array with spectral dimensions
zerog = (0.,0.) ! Zero array with grid dimensions

uprev = Dyn%Grid%u(:,:,previous) ! Save these variable (u @ t = i-1) in order to
calculate tendancies later on (need to specify like this because the 212 time scheme
writes over it at step 6)
vprev = Dyn%Grid%v(:,:,previous)
vorprev = Dyn%Grid%vor(:,:,previous)
energyprev = Dyn%Grid%energy(:,:,previous) ! Havn't written this to a restart (don't
use it to calculate the tendency). Just here for check.
energymean_prev = Dyn%Grid%energy_mean(:,:,previous)

do j = js, je
  Dyn%grid%pv(:,j)  = Dyn%grid%vor(:,j,current)  + coriolis(j)
  f_array(:,j) = coriolis(j) ! needed below in advection terms calc
end do

Dyn%Tend%u = Dyn%Tend%u + Dyn%grid%pv*Dyn%Grid%v(:,:,current)
Dyn%Tend%v = Dyn%Tend%v - Dyn%grid%pv*Dyn%Grid%u(:,:,current)

do j = js, je
 umean_current(:,j)= sum(Dyn%Grid%u(:,j,current))/count(Dyn%Grid%u(:,j,current) >
-10000) ! Calculate zonal mean, zonal flow (needed for linear damping)
 vormeanprev(:,j)= sum(Dyn%Grid%vor(:,j,previous))/count(Dyn%Grid%u(:,j,current) >
-10000)
 vormean(:,j)= sum(Dyn%Grid%vor(:,j,current))/count(Dyn%Grid%u(:,j,current) > -10000) !
Calculate zonal mean, vorticity (needed for calculating beta_star)
 pvmean(:,j)= sum(Dyn%Grid%vor(:,j,current))/count(Dyn%Grid%u(:,j,current) > -10000) +
coriolis(j) ! Calculate zonal mean, pv (needed for calculating beta_star)
end do

vorprimeprev = Dyn%Grid%vor(:,:,previous)-vormeanprev(:,:)
enstrophyprev = 0.5*vorprimeprev*vorprimeprev

! --------------------------------- ADVECTION TERMS
---------------------------------------
Dyn%Grid%vq(:,:,future) = Dyn%grid%pv*Dyn%Grid%v(:,:,current) ! Full advection terms
Dyn%Grid%uq(:,:,future) = Dyn%grid%pv*Dyn%Grid%u(:,:,current)! Note the positive sign
!print *, "VQ", Dyn%Grid%vq(10,10,future)
!print *, "-UQ", -Dyn%Grid%uq(10,10,future)

! RELATIVE VORTICITY ADVECTION
 call vor_div_from_uv_grid (Dyn%grid%vor(:,:,current)*Dyn%Grid%v(:,:,current), -Dyn%grid
%vor(:,:,current)*Dyn%Grid%u(:,:,current), dt_vors_rub, dt_divs_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%Grid%rvor_advec(:,:,current))
! PLANETARY VORTICITY ADVECTION
 call vor_div_from_uv_grid (f_array*Dyn%Grid%v(:,:,current), -f_array*Dyn%Grid%u
(:,:,current), dt_vors_rub, dt_divs_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%Grid%pvor_advec(:,:,future))
! BETA
Dyn%grid%beta(:,:,future) = Dyn%Grid%pvor_advec(:,:,current)/Dyn%grid%v(:,:,current)
! PVMEAN_ADVEC & BETA_STAR
 call vor_div_from_uv_grid (pvmean*Dyn%Grid%v(:,:,current), -pvmean*Dyn%Grid%u
(:,:,current), dt_vors_rub, dt_divs_rub)
 call trans_spherical_to_grid(dt_vors_rub, pvmean_advec)
Dyn%grid%beta_star(:,:,future) = pvmean_advec/Dyn%grid%v(:,:,current)

!========= COMMENT ON horizontal_advection SUBROUTINE =============
! Both methods give (almost) the same result. I think the difference can be attributed
to some numerical errors in the different ways they are calculated  (small => not
```

```
important)
! METHOD 1
! call trans_grid_to_spherical(Dyn%grid%vor(:,:,current), vors_rub)
! call horizontal_advection   (vors_rub, Dyn%Grid%u(:,:,current), Dyn%Grid%v
(:,:,current), Dyn%Grid%tester(:,:,future))
!print *, "METHOD 1: U.GRAD(f)", Dyn%Grid%tester(10,10,future)
! METHOD 2
! call vor_div_from_uv_grid (Dyn%grid%vor(:,:,current)*Dyn%Grid%v(:,:,current), -Dyn%
grid%vor(:,:,current)*Dyn%Grid%u(:,:,current), dt_vors_rub, dt_divs_rub)
! call trans_spherical_to_grid(dt_vors_rub, Dyn%Grid%rvor_advec(:,:,future))
!print *, "METHOD 2: U.GRAD(VOR)", Dyn%Grid%rvor_advec(10,10,future)
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 call vor_div_from_uv_grid (Dyn%Tend%u, Dyn%Tend%v, dt_vors, dt_divs) ! Calculate
DT_VOR before linear damping is applied
 call trans_spherical_to_grid(dt_vors, Dyn%grid%dt_vor(:,:,future))
!print *, "DT_VOR", Dyn%grid%dt_vor(10,10,future)
! Further subdivide the advection term of q=(zeta + f) into a vorticity advection
component and a beta*v component
!vor_beta = beta*Dyn%Grid%v(:,:,current)
!vor_advec = Dyn%grid%dt_vor(:,:,future) - vor_beta

!===================== COMMENT ON DT_VOR ==========================
! The sum of planetary vorticity advection (pvor_advec) and relative vorticity
advection (rvor_advec) gives the absolute vorticity advection given in dt_vor
!print *, "ABSOLUTE ADVEC", Dyn%Grid%pvor_advec(10,10,current) + Dyn%Grid%rvor_advec
(10,10,current)
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ---------------------------- LINEAR DAMPING
--------------------------------------------
if(damp > 1) then ! do linear damping
 Dyn%Grid%vlindamp(:,:,future) =  Dyn%Grid%v(:,:,current)/tau
 Dyn%Grid%ulindamp_e(:,:,future) = (Dyn%Grid%u(:,:,current)-umean_current)/tau
 Dyn%Grid%ulindamp_m(:,:,future) = umean_current/tau1
 !print *, "V (EDDY) LINDAMP", Dyn%Grid%vlindamp(10,10,future)
 !print *, "U EDDY LINDAMP", Dyn%Grid%ulindamp_e(10,10,future)
 !print *, "U MEAN LINDAMP", Dyn%Grid%ulindamp_m(10,10,future)
 Dyn%Tend%v = Dyn%Tend%v - Dyn%Grid%vlindamp(:,:,future)  ! Damping on merdional (eddy)
flow
 Dyn%Tend%u = Dyn%Tend%u - Dyn%Grid%ulindamp_e(:,:,future) ! Damping on zonal eddy flow
 Dyn%Tend%u = Dyn%Tend%u - Dyn%Grid%ulindamp_m(:,:,future) ! Damping on zonal mean flow

 call vor_div_from_uv_grid (Dyn%Grid%ulindamp_e(:,:,future), Dyn%Grid%vlindamp
(:,:,future), dt_vors_rub, dt_divs_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%grid%vorlindamp_e(:,:,future)) !
Calculate the effect of eddy damping of u and v on vorticity
 call vor_div_from_uv_grid (Dyn%Grid%ulindamp_m(:,:,future), zerog, dt_vors_rub,
dt_divs_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%grid%vorlindamp_m(:,:,future)) !
Calculate the effect of mean flow damping of u on vorticity
 !print *, "VOR EDDY LINDAMP", Dyn%grid%vorlindamp_e(10,10,future)
 !print *, "VOR MEAN LINDAMP", Dyn%grid%vorlindamp_m(10,10,future)
endif
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------------------- TRUNCATION TERMS
---------------------------------------
```

```
! U & V
 call trans_grid_to_spherical(Dyn%Tend%u, dt_vors_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%grid%utrunc(:,:,current))
 call trans_grid_to_spherical(Dyn%Tend%v, dt_vors_rub)
 call trans_spherical_to_grid(dt_vors_rub, Dyn%grid%vtrunc(:,:,current))
Dyn%grid%utrunc(:,:,future) = Dyn%Tend%u - Dyn%grid%utrunc(:,:,current)
Dyn%grid%vtrunc(:,:,future) = Dyn%Tend%v - Dyn%grid%vtrunc(:,:,current)
!print *, "UTRUNC", Dyn%grid%utrunc(10,10,future)
!print *, "VTRUNC", Dyn%grid%vtrunc(10,10,future)

!============= COMMENT ON U^2 TRUNCATION TERM =====================
! call trans_grid_to_spherical(Dyn%Tend%u**2, dt_vors_rub)
! call trans_spherical_to_grid(dt_vors_rub, u2trunc)
!print *, "U^2 TRUNC", u2trunc(40,40)
! This truncation TRUNC(uvq) does not work for the dt_u^2 budget. Instead u*TRUNC(u)
works. On the one hand this makes sense, the dt_u^2 equation is just (up to a factor of
0.5) u*dt_u
!       u*dt_u = 0.5*dt_u^2 = u*vq - u*TRUNC
!On the other hand I would have thought TRUNC(uvq) should also have worked. It would be
good to understand why it does not.
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
! VORTICITY
if(previous == current) then
 call trans_grid_to_spherical(Dyn%grid%vor(:,:,current), vors_rub)
 call trans_spherical_to_grid(vors_rub, Dyn%grid%vortrunc(:,:,future))
Dyn%grid%vortrunc(:,:,future) = (Dyn%grid%vor(:,:,current)-Dyn%grid%vortrunc
(:,:,future))/delta_t
!print *, "VORTRUNC", Dyn%grid%vortrunc(10,10,future)
else
 call trans_grid_to_spherical(Dyn%grid%vor(:,:,previous), vors_rub)
 call trans_spherical_to_grid(vors_rub, Dyn%grid%vortrunc(:,:,future))
Dyn%grid%vortrunc(:,:,future) = (Dyn%grid%vor(:,:,previous)-Dyn%grid%vortrunc
(:,:,future))/delta_t
!print *, "VORTRUNC", Dyn%grid%vortrunc(10,10,future)
endif
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------------------------- GRAD(U^2 + V^2)
-----------------------------------------
 call vor_div_from_uv_grid (Dyn%Tend%u, Dyn%Tend%v, dt_vors, dt_divs) ! 3. Moving into
spectral space, recast equ. of motion (EoF) in vor-div; compute spectral divergence of
{u_t,v_t}={(f+zeta)u,(f+zeta)v}
 call uv_grid_from_vor_div(dt_vors,  zeros, Dyn%grid%dx_u2_v2(:,:,current), Dyn%grid%
dy_u2_v2(:,:,current))
Dyn%grid%dx_u2_v2(:,:,future) = Dyn%Tend%u - Dyn%grid%dx_u2_v2(:,:,current) - Dyn%grid%
utrunc(:,:,future)
Dyn%grid%dy_u2_v2(:,:,future) = Dyn%Tend%v - Dyn%grid%dy_u2_v2(:,:,current) - Dyn%grid%
vtrunc(:,:,future)
!print *, "DX_U2_V2", Dyn%grid%dx_u2_v2(10,10,future)
!print *, "DY_U2_V2", Dyn%grid%dy_u2_v2(10,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------  JOE CODE (SNIPPET FOR LINEARISED
EQUATIONS)------------------------
!pvmean, for linearized equations:
!do j = js, je
!  Dyn%grid%pv(:,j)  = sum(Dyn%grid%vor(:,j,current))/count(Dyn%Grid%u(:,j,current) >
-10000)   + coriolis(j)
!end do
```

```
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -----------------  JOE CODE (SNIPPET FOR QUASILINEAR EQUATIONS)
-----------------------
! for QL integrations:
!Dyn%Tend%u=Dyn%Tend%u+linmult*dt_vorg
!make a linear model - product of eddies removed.
!Dyn%Tend%u = Dyn%Tend%u + Dyn%grid%pv*Dyn%Grid%v(:,:,current)
!zonal mean vorticity time eddy u
!Dyn%Tend%v = Dyn%Tend%v - Dyn%grid%pv*(Dyn%Grid%u(:,:,current)-umean)
!do j = js, je
!  Dyn%grid%pv(:,j)  = Dyn%grid%vor(:,j,current)  + coriolis(j)
!end do
! add total vorticity times zonal mean u, so still missing products of eddies (u'zeta')
!Dyn%Tend%v = Dyn%Tend%v - Dyn%grid%pv*umean
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ------------------------------- SPECTRAL DAMPING
---------------------------------------
if(damp > 0) then
  dt_vors_rub = dt_vors
  call compute_spectral_damping(Dyn%Spec%vor(:,:,previous), dt_vors, delta_t) ! 4.
Compute spectral damping, the subroutine (located in src/atmos_spectral/
spectral_damping.f90 outputs dt_vors)
  dt_vors_rub = dt_vors - dt_vors_rub
  call uv_grid_from_vor_div(dt_vors_rub,  zeros, Dyn%grid%uspecdamp(:,:,future),  Dyn%
grid%vspecdamp(:,:,future))
  call trans_spherical_to_grid(dt_vors_rub, Dyn%grid%vorspecdamp(:,:,future))
endif
!print *, "USPECDAMP", Dyn%grid%uspecdamp(10,10,future)
!print *, "VSPECDAMP", Dyn%grid%vspecdamp(10,10,future)
!print *, "VORSPECDAMP", Dyn%grid%vorspecdamp(10,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ------------------------------ STIRRING
------------------------------------------------
dt_vors_rub = dt_vors
 call stirring(Time, dt_vors)
dt_vors_rub = dt_vors - dt_vors_rub
 call uv_grid_from_vor_div(dt_vors_rub,  zeros, Dyn%grid%ustir(:,:,future),  Dyn%grid%
vstir(:,:,future))
 call trans_spherical_to_grid(dt_vors_rub,Dyn%Grid%vorstir(:,:,future))
!print *, "U STIR", Dyn%grid%ustir(10,10,future)
!print *, "V STIR", Dyn%grid%vstir(10,10,future)
!print *, "VOR STIR", Dyn%grid%vorstir(10,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ------------------  JOE CODE (SNIPPET FOR ENSTROPHY
EQUATION)---------------------------
!calc the enstrophy flux in grid space, take the divergence later.
!Dyn%Grid%vtend(:,:,current)=Dyn%Grid%vor(:,:,current)*Dyn%Grid%v(:,:,current)
!
!do j = js, je
!vormean(:,j)= sum(Dyn%Grid%vor(:,j,current))/count(Dyn%Grid%u(:,j,current) > -10000)
!end do
!vorprime = Dyn%Grid%vor(:,:,current)-vormean(:,:)
!Dyn%grid%source(:,:,current) = stir*vorprime
```

```
!Dyn%grid%sink(:,:,current) = Dyn%Grid%vortend(:,:,current)*vorprime
!Dyn%grid%ensflux(:,:,current) = 0.5*vorprime*vorprime*Dyn%Grid%v(:,:,current)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ---------------------------------- ROBERTS DAMPING
--------------------------------------
Dyn%spec%roberts(:,:,future) = Dyn%spec%vor(:,:,current)
 call leapfrog(Dyn%Spec%vor , dt_vors  , previous, current, future, delta_t,
robert_coeff) ! 5. Apply Roberts damping/filter using leapfrog scheme, again in
spectral space
Dyn%spec%roberts(:,:,future) = Dyn%spec%vor(:,:,current)-Dyn%spec%roberts(:,:,future) !
This is the roberts damping term: rob_coeff*(phi_{n-1} - 2*phi_n + phi_{n+1})

!print *, "ROBO CURRENT", Dyn%spec%roberts(10,10,current)
rob_vors_diff = Dyn%spec%vor(:,:,future)-Dyn%spec%roberts(:,:,current) ! Gives the time-
stepped spec%vor that results from using an unfiltered spec%vor(previous)
 call uv_grid_from_vor_div(rob_vors_diff,  zeros, rob_u_diff,  rob_v_diff)
 call trans_spherical_to_grid(rob_vors_diff, rob_vorg_diff)

 call trans_spherical_to_grid(Dyn%Spec%vor(:,:,future), Dyn%Grid%vor(:,:,future)) ! 6.
Transforms from spherical harmonics to grid space and then from vor-div to u-v EoF
 call uv_grid_from_vor_div(Dyn%Spec%vor(:,:,future),  zeros, Dyn%Grid%u(:,:,future),
Dyn%Grid%v(:,:,future))

Dyn%grid%urobdamp(:,:,future) = (Dyn%grid%u(:,:,future)-rob_u_diff(:,:))/delta_t
Dyn%grid%vrobdamp(:,:,future) = (Dyn%grid%v(:,:,future)-rob_v_diff(:,:))/delta_t
Dyn%grid%vorrobdamp(:,:,future) = (Dyn%grid%vor(:,:,future)-rob_vorg_diff(:,:))/delta_t
!print *, "UROBDAMP", Dyn%grid%urobdamp(10,10,future)
!print *, "VROBDAMP", Dyn%grid%vrobdamp(10,10,future)
!print *, "VORROBDAMP", Dyn%grid%vorrobdamp(10,10,future)

!======= COMMENT ON CALCULATION OF ROBERTS FILTER TERM =============
! I have written an extensive comment on the roberts filter operation and calculation,
see src/atmos_spectral/model/Roberts_damping.pdf
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
! AFTER THIS POINT THERE IS NO MORE "ORGINAL" CODE (BESIDES A SMALL SNIPPET FOR THE
CALCULATION OF THE STREAMFUNCTION AND TRACER ADVECTION). THE REST IS CALCULATIONS OF
TENDENCIES, OTHER FIELDS THAT ARE
! DERIVED FROM U,V AND VOR FIELDS (EG. ENERGY & ENSTROPHY) AND BUDGETS FOR ALL THESE
VARIABLES.
!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

! -----------------  JOE CODE (EFFECT OF KILLING CERTAIN
WAVENUMBERS)------------------------
! try killing the affect of waves 1:3
!dt_vors(1:3,:)=0;
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -----------------  JOE CODE (SNIPPET FOR ENSTOPHY
EQUATION)-----------------------------
!Dyn%Grid%robertssink(:,:,current)=roberts*vorprime
!Dyn%Grid%dens_dt(:,:,current)=dt_vorg - Dyn%Grid%vor(:,:,current)
!
```

```
! call trans_grid_to_spherical  (vorprime, dt_divs_throwaway)
! call horizontal_advection     (dt_divs_throwaway, Dyn%Grid%u(:,:,current)-umean, Dyn%
Grid%v(:,:,current), Dyn%grid%ensflux_div(:,:,current))
!Dyn%grid%ensflux_div(:,:,current) = Dyn%grid%ensflux_div(:,:,current)*vorprime
!
!Zonal mean absolute vorticity
!do j = js, je
!  vormean(:,j)  = sum(Dyn%Grid%vor(:,j,current))/count(Dyn%Grid%u(:,j,current) >
-10000)   + coriolis(j)
!end do
!Zonal mean absolute vorticity in spherical, just for advection later
!call trans_grid_to_spherical  (vormean, dt_divs_throwaway)
!get v'beta*, on the way to getting vvor'beta*
!call horizontal_advection     (dt_divs_throwaway, 0*Dyn%Grid%u(:,:,current), Dyn%Grid%v
(:,:,current), Dyn%Grid%vvor_beta(:,:,current))
!get beta, to use for calculating source and sink of vorticity flux budget
!beta = Dyn%Grid%vvor_beta(:,:,current)/Dyn%Grid%v(:,:,current)
!Dyn%Grid%vvor_beta(:,:,current) = Dyn%Grid%vvor_beta(:,:,current)*vorprime
!vvor_beta
!Dyn%grid%ensflux_div_beta(:,:,current) = Dyn%grid%ensflux_div(:,:,current)/beta
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ------------------------------ ENERGY MEAN_ENERGY & ENSTROPHY
----------------------------
Dyn%Grid%energy(:,:,future) = (Dyn%Grid%u(:,:,future)**2+Dyn%Grid%v(:,:,future)**2)
!print *, "ENERGY", Dyn%Grid%energy(10,10,future)

! Need these for the product rule (see documentation) and for calculating energy_mean
and enstrophy
do j = js, je
 umean_prev(:,j)= sum(uprev(:,j))/count(uprev(:,j) > -10000)
 vmean_prev(:,j)= sum(vprev(:,j))/count(vprev(:,j) > -10000)
 umean_future(:,j)= sum(Dyn%Grid%u(:,j,future))/count(Dyn%Grid%u(:,j,future) > -10000)
 vmean_future(:,j)= sum(Dyn%Grid%v(:,j,future))/count(Dyn%Grid%v(:,j,future) > -10000)
 vormean(:,j)= sum(Dyn%Grid%vor(:,j,future))/count(Dyn%Grid%u(:,j,current) > -10000)
end do
vorprime = Dyn%Grid%vor(:,:,future)-vormean(:,:)

Dyn%Grid%energy_mean(:,:,future) = (umean_future**2+vmean_future**2)
Dyn%Grid%enstrophy(:,:,future) = vorprime*vorprime
!print *, "ENSTROPHY", Dyn%Grid%enstrophy(10,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------------- U,V,VOR, ENERGY & ENSTROPHY TENDENCIES
------------------------------
if(previous == current) then
Dyn%Grid%utend(:,:,future) = (Dyn%Grid%u(:,:,future)-Dyn%Grid%u(:,:,current))/delta_t
Dyn%Grid%vtend(:,:,future) = (Dyn%Grid%v(:,:,future)-Dyn%Grid%v(:,:,current))/delta_t
Dyn%Grid%vortend(:,:,future) = (Dyn%Grid%vor(:,:,future)-Dyn%Grid%vor(:,:,current))/
delta_t
energytendcheck(:,:) = (Dyn%Grid%energy(:,:,future)-Dyn%Grid%energy(:,:,current))/
delta_t
energytendcheck_mean(:,:) = (Dyn%Grid%energy_mean(:,:,future)-Dyn%Grid%energy_mean
(:,:,current))/delta_t
enstrophytendcheck(:,:) = (Dyn%Grid%enstrophy(:,:,future)-Dyn%Grid%enstrophy
(:,:,current))/delta_t
else
Dyn%Grid%utend(:,:,future) = (Dyn%Grid%u(:,:,future)-uprev)/delta_t
Dyn%Grid%vtend(:,:,future) = (Dyn%Grid%v(:,:,future)-vprev)/delta_t
```

```
Dyn%Grid%vortend(:,:,future) = (Dyn%Grid%vor(:,:,future)-vorprev)/delta_t
energytendcheck(:,:) = (Dyn%Grid%energy(:,:,future)-energyprev)/delta_t ! These two
variables act as checks (confirm that the tendency calculated using the product rule is
the same as central differences)
energytendcheck_mean(:,:) = (Dyn%Grid%energy_mean(:,:,future)-energymean_prev)/
delta_t ! These won't be correct for first two time steps (didn't put energyprev in
restart file). Thats ok, just a check.
enstrophytendcheck(:,:) = (Dyn%Grid%enstrophy(:,:,future)-enstrophyprev)/delta_t ! See
above comment
endif
!print *, "UTEND", Dyn%Grid%utend(10,10,future)
!print *, "VTEND", Dyn%Grid%vtend(10,10,future)
!print *, "VORTEND", Dyn%Grid%vortend(10,10,future)
!print *, "UTEND*(U(PREV)+U(FUT))", Dyn%Grid%utend(10,10,future)*(uprev(10,10)+Dyn%Grid%
u(10,10,future)) ! This is the central differences analogy to the product rule.
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------------------- ENERGY BUDGET TERMS
--------------------------------------------
!============= COMMENT ON ENERGY CALCULATIONS =====================
! The U^2 and V^2 budget are calculated using the product rule (this way we can
leverage all the work/code that went into creating the U and V budgets).
! The central difference anlogy to the product rule is
!       D(a*b)_n = D(a)_n*b_{n+1} + a_n*D(b)_n
! where D(a)_n = a_{n+1} - a_{n-1} is the central difference operator.
! For a = b this simplifies to
!       D(a^2)_n = D(a)_n*(a_{n+1} + a_{n-1})
! Calculating u2tend in these two different ways confirms they are exact.
! Because the above product rule demands knowledge of u @ t=n+1 I calculate all the
terms at the end of the subroutine, when this value is known.
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Dyn%Grid%energy_tend(:,:,future) = Dyn%Grid%utend(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) + Dyn%Grid%vtend(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_TEND", Dyn%Grid%energy_tend(10,10,future)
!print *, "ENERGYTENDCHECK", energytendcheck(10,10)
Dyn%Grid%energy_voradvec(:,:,future) = Dyn%Grid%vq(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) - Dyn%Grid%uq(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_VORADVEC", Dyn%Grid%energy_voradvec(10,10,future)
Dyn%Grid%energy_gradterm(:,:,future) = Dyn%Grid%dx_u2_v2(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%dy_u2_v2(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_GRADTERM", Dyn%Grid%energy_gradterm(10,10,future)
Dyn%Grid%energy_trunc(:,:,future) = Dyn%Grid%utrunc(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) + Dyn%Grid%vtrunc(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_TRUNC", Dyn%Grid%energy_trunc(10,10,future)
Dyn%Grid%energy_lindamp_m(:,:,future) = Dyn%Grid%ulindamp_m(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future))
!print *, "ENERGY_LINDAMP_M", Dyn%Grid%energy_lindamp_m(10,10,future)
Dyn%Grid%energy_lindamp_e(:,:,future) = Dyn%Grid%ulindamp_e(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%vlindamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_LINDAMP_E", Dyn%Grid%energy_lindamp_e(10,10,future)
Dyn%Grid%energy_specdamp(:,:,future) = Dyn%Grid%uspecdamp(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%vspecdamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_SPECDAMP", Dyn%Grid%energy_specdamp(10,10,future)
Dyn%Grid%energy_stir(:,:,future) = Dyn%Grid%ustir(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) + Dyn%Grid%vstir(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_STIR", Dyn%Grid%energy_stir(10,10,future)
Dyn%Grid%energy_robdamp(:,:,future) = Dyn%Grid%urobdamp(:,:,future)*(uprev(:,:)+Dyn%Grid
%u(:,:,future)) + Dyn%Grid%vrobdamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_ROBDAMP", Dyn%Grid%energy_robdamp(10,10,future)
!
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! ----------------------------- MEAN ENERGY BUDGET TERMS
-----------------------------------------
! Take the mean for mean budget
do j = js, je
 utendmean(:,j) = sum(Dyn%Grid%utend(:,j,future))/count(Dyn%Grid%utend(:,j,future) >
-10000)
 vtendmean(:,j) = sum(Dyn%Grid%vtend(:,j,future))/count(Dyn%Grid%vtend(:,j,future) >
-10000)
 vqmean(:,j) = sum(Dyn%Grid%vq(:,j,future))/count(Dyn%Grid%vq(:,j,future) > -10000)
 uqmean(:,j) = sum(Dyn%Grid%uq(:,j,future))/count(Dyn%Grid%uq(:,j,future) > -10000)
 dx_u2_v2mean(:,j) = sum(Dyn%Grid%dx_u2_v2(:,j,future))/count(Dyn%Grid%dx_u2_v2
(:,j,future) > -10000) ! Term shoud be zero (d/dx)
 dy_u2_v2mean(:,j) = sum(Dyn%Grid%dy_u2_v2(:,j,future))/count(Dyn%Grid%dy_u2_v2
(:,j,future) > -10000)
 utruncmean(:,j) = sum(Dyn%Grid%utrunc(:,j,future))/count(Dyn%Grid%utrunc(:,j,future) >
-10000) ! Both truncation terms should have a zonal mean of ~ zero (composed entirely
of high frequency "eddies")
 vtruncmean(:,j) = sum(Dyn%Grid%vtrunc(:,j,future))/count(Dyn%Grid%vtrunc(:,j,future) >
-10000)
 ulindamp_mmean(:,j) = sum(Dyn%Grid%ulindamp_m(:,j,future))/count(Dyn%Grid%ulindamp_m
(:,j,future) > -10000)
 ulindamp_emean(:,j) = sum(Dyn%Grid%ulindamp_e(:,j,future))/count(Dyn%Grid%ulindamp_e
(:,j,future) > -10000) ! should be zero
 vlindampmean(:,j) = sum(Dyn%Grid%vlindamp(:,j,future))/count(Dyn%Grid%vlindamp
(:,j,future) > -10000) ! should be zero
 uspecdampmean(:,j) = sum(Dyn%Grid%uspecdamp(:,j,future))/count(Dyn%Grid%uspecdamp
(:,j,future) > -10000)
 vspecdampmean(:,j) = sum(Dyn%Grid%vspecdamp(:,j,future))/count(Dyn%Grid%vspecdamp
(:,j,future) > -10000)
 ustirmean(:,j) = sum(Dyn%Grid%ustir(:,j,future))/count(Dyn%Grid%ustir(:,j,future) >
-10000)
 vstirmean(:,j) = sum(Dyn%Grid%vstir(:,j,future))/count(Dyn%Grid%vstir(:,j,future) >
-10000)
 urobdampmean(:,j) = sum(Dyn%Grid%urobdamp(:,j,future))/count(Dyn%Grid%urobdamp
(:,j,future) > -10000)
 vrobdampmean(:,j) = sum(Dyn%Grid%vrobdamp(:,j,future))/count(Dyn%Grid%vrobdamp
(:,j,future) > -10000)
end do
! ======== Compute energy_tend_mean in a different way =============
!energymean_prev = (umean_prev**2+vmean_prev**2)
!Dyn%Grid%energy_mean(:,:,future) = (umean_future**2+vmean_future**2)
!print *, "ENERGY_MEAN", Dyn%Grid%energy_mean(10,10,future)
!if(previous == current) then
!energytendcheck_mean(:,:) = (Dyn%Grid%energy_mean(:,:,future)-Dyn%Grid%energy_mean
(:,:,current))/delta_t
!else
!energytendcheck_mean(:,:) = (Dyn%Grid%energy_mean(:,:,future)-energymean_prev)/delta_t
!endif
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
! Compute \bar{E} = \bar{u}^2 + \bar{v}^2 budget terms using product rule
Dyn%Grid%energy_tend_mean(:,:,future) = utendmean(:,:)*(umean_prev(:,:)+umean_future
(:,:)) + vtendmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYTEND_MEAN", Dyn%Grid%energy_tend_mean(1,10,future)
!print *, "ENERGYTENDCHECK_MEAN", energytendcheck_mean(1,10)
Dyn%Grid%energy_voradvec_mean(:,:,future) = vqmean(:,:)*(umean_prev(:,:)+umean_future
(:,:)) - uqmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYVORADVEC_MEAN", Dyn%Grid%energy_voradvec_mean(1,10,future)
Dyn%Grid%energy_gradterm_mean(:,:,future) = dx_u2_v2mean(:,:)*(umean_prev(:,:)
+umean_future(:,:)) + dy_u2_v2mean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
```

```
!print *, "ENERGYGRADTERM_MEAN", Dyn%Grid%energy_gradterm_mean(1,10,future)
Dyn%Grid%energy_trunc_mean(:,:,future) = utruncmean(:,:)*(umean_prev(:,:)+umean_future
(:,:)) + vtruncmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYTRUNC_MEAN", Dyn%Grid%energy_trunc_mean(1,10,future)
Dyn%Grid%energy_lindamp_m_mean(:,:,future) = ulindamp_mmean(:,:)*(umean_prev(:,:)
+umean_future(:,:))
!print *, "ENERGYLINDAMP_M_MEAN", Dyn%Grid%energy_lindamp_m_mean(1,10,future)
Dyn%Grid%energy_lindamp_e_mean(:,:,future) = ulindamp_emean(:,:)*(umean_prev(:,:)
+umean_future(:,:)) + vlindampmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYLINDAMP_E_MEAN", Dyn%Grid%energy_lindamp_e_mean(1,10,future)
Dyn%Grid%energy_specdamp_mean(:,:,future) = uspecdampmean(:,:)*(umean_prev(:,:)
+umean_future(:,:)) + vspecdampmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYSPECDAMP_MEAN", Dyn%Grid%energy_specdamp_mean(1,10,future)
Dyn%Grid%energy_stir_mean(:,:,future) = ustirmean(:,:)*(umean_prev(:,:)+umean_future
(:,:)) + vstirmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYSTIR_MEAN", Dyn%Grid%energy_stir_mean(1,10,future)
Dyn%Grid%energy_robdamp_mean(:,:,future) = urobdampmean(:,:)*(umean_prev(:,:)
+umean_future(:,:)) + vrobdampmean(:,:)*(vmean_prev(:,:)+vmean_future(:,:))
!print *, "ENERGYROBDAMP_MEAN", Dyn%Grid%energy_robdamp_mean(1,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

! -------------------------------- ENSTROPHY BUDGET
-----------------------------------------------
!============= COMMENT ON ENSTROPHY CALCULATIONS =====================
! The VOR'^2 budget are calculated using the product rule (this way we can leverage all
the work/code that went into creating the VOR budget).
! The central difference anlogy to the product rule is
!       D(a*b)_n = D(a)_n*b_{n+1} + a_n*D(b)_n
! where D(a)_n = a_{n+1} - a_{n-1} is the central difference operator.
! For a = b this simplifies to
!       D(a^2)_n = D(a)_n*(a_{n+1} + a_{n-1})
! Calculating u2tend in these two different ways confirms they are exact.
! Because the above product rule demands knowledge of u @ t=n+1 I calculate all the
terms at the end of the subroutine, when this value is known.
!++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
! Find the eddy quantities to calculate vorticity budget
do j = js, je
 vortendmean(:,j) = sum(Dyn%Grid%utend(:,j,future))/count(Dyn%Grid%utend(:,j,future) >
-10000)
 pvor_advecmean(:,j) = sum(Dyn%Grid%dx_u2_v2(:,j,future))/count(Dyn%Grid%dx_u2_v2
(:,j,future) > -10000) ! Term shoud be zero (d/dx)
 rvor_advecmean(:,j) = sum(Dyn%Grid%dx_u2_v2(:,j,future))/count(Dyn%Grid%dx_u2_v2
(:,j,future) > -10000) ! Term shoud be zero (d/dx)
 vortruncmean(:,j) = sum(Dyn%Grid%utrunc(:,j,future))/count(Dyn%Grid%utrunc(:,j,future)
> -10000) ! Both truncation terms should have a zonal mean of ~ zero (composed entirely
of high frequency "eddies")
 vorlindamp_emean(:,j) = sum(Dyn%Grid%vtrunc(:,j,future))/count(Dyn%Grid%vtrunc
(:,j,future) > -10000)
 vorlindamp_mmean(:,j) = sum(Dyn%Grid%ulindamp_m(:,j,future))/count(Dyn%Grid%ulindamp_m
(:,j,future) > -10000)
 vorspecdampmean(:,j) = sum(Dyn%Grid%uspecdamp(:,j,future))/count(Dyn%Grid%uspecdamp
(:,j,future) > -10000)
 vorstirmean(:,j) = sum(Dyn%Grid%ustir(:,j,future))/count(Dyn%Grid%ustir(:,j,future) >
-10000)
 vorrobdampmean(:,j) = sum(Dyn%Grid%urobdamp(:,j,future))/count(Dyn%Grid%urobdamp
(:,j,future) > -10000)
end do
 vortendprime = Dyn%Grid%vortend(:,:,future)-vortendmean(:,:)
 pvor_advecprime = Dyn%Grid%pvor_advec(:,:,future)-pvor_advecmean(:,:)
 rvor_advecprime = Dyn%Grid%rvor_advec(:,:,future)-rvor_advecmean(:,:)
```

```
 vortruncprime = Dyn%Grid%vortrunc(:,:,future)-vortruncmean(:,:)
 vorlindamp_eprime = Dyn%Grid%vorlindamp_e(:,:,future)-vorlindamp_emean(:,:)
 vorlindamp_mprime = Dyn%Grid%vorlindamp_m(:,:,future)-vorlindamp_mmean(:,:)
 vorspecdampprime = Dyn%Grid%vorspecdamp(:,:,future)-vorspecdampmean(:,:)
 vorstirprime = Dyn%Grid%vorstir(:,:,future)-vorstirmean(:,:)
 vorrobdampprime = Dyn%Grid%vorrobdamp(:,:,future)-vorrobdampmean(:,:)
! Enstrophy budget terms
Dyn%Grid%enstrophy_tend(:,:,future) = vortendprime*(vorprimeprev+vorprime)
print *, "ENSTROPHY_TEND", Dyn%Grid%enstrophy_tend(10,10,future)
print *, "ENSTROPHYTENDCHECK", enstrophytendcheck(10,10)
!Dyn%Grid%energy_voradvec(:,:,future) = Dyn%Grid%vq(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) - Dyn%Grid%uq(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_VORADVEC", Dyn%Grid%energy_voradvec(10,10,future)
!Dyn%Grid%energy_gradterm(:,:,future) = Dyn%Grid%dx_u2_v2(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%dy_u2_v2(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_GRADTERM", Dyn%Grid%energy_gradterm(10,10,future)
!Dyn%Grid%energy_trunc(:,:,future) = Dyn%Grid%utrunc(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) + Dyn%Grid%vtrunc(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_TRUNC", Dyn%Grid%energy_trunc(10,10,future)
!Dyn%Grid%energy_lindamp_m(:,:,future) = Dyn%Grid%ulindamp_m(:,:,future)*(uprev(:,:)+Dyn
%Grid%u(:,:,future))
!print *, "ENERGY_LINDAMP_M", Dyn%Grid%energy_lindamp_m(10,10,future)
!Dyn%Grid%energy_lindamp_e(:,:,future) = Dyn%Grid%ulindamp_e(:,:,future)*(uprev(:,:)+Dyn
%Grid%u(:,:,future)) + Dyn%Grid%vlindamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_LINDAMP_E", Dyn%Grid%energy_lindamp_e(10,10,future)
!Dyn%Grid%energy_specdamp(:,:,future) = Dyn%Grid%uspecdamp(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%vspecdamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_SPECDAMP", Dyn%Grid%energy_specdamp(10,10,future)
!Dyn%Grid%energy_stir(:,:,future) = Dyn%Grid%ustir(:,:,future)*(uprev(:,:)+Dyn%Grid%u
(:,:,future)) + Dyn%Grid%vstir(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_STIR", Dyn%Grid%energy_stir(10,10,future)
!Dyn%Grid%energy_robdamp(:,:,future) = Dyn%Grid%urobdamp(:,:,future)*(uprev(:,:)+Dyn%
Grid%u(:,:,future)) + Dyn%Grid%vrobdamp(:,:,future)*(vprev(:,:)+Dyn%Grid%v(:,:,future))
!print *, "ENERGY_ROBDAMP", Dyn%Grid%energy_robdamp(10,10,future)
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if(minval(Dyn%Grid%v) < valid_range_v(1) .or. maxval(Dyn%Grid%v) > valid_range_v(2))
then
  call error_mesg('barotropic_dynamics','meridional wind out of valid range balls',
FATAL)
endif

if(Dyn%spec_tracer) call update_spec_tracer(Dyn%Spec%trs, Dyn%Grid%trs, Dyn%Tend%trs, &
                         Dyn%Grid%u, Dyn%Grid%v, previous, current, future, delta_t)

if(Dyn%grid_tracer) call update_grid_tracer(Dyn%Grid%tr, Dyn%Tend%tr, &
                         Dyn%Grid%u, Dyn%Grid%v, previous, current, future, delta_t)

stream = compute_laplacian(Dyn%Spec%vor(:,:,current), -1)
 call trans_spherical_to_grid(stream, Dyn%grid%stream)

! -------------------------------- BUDGET EQUATIONS
----------------------------------------
! CHECK TO MAKE SURE BUDGET IS BEING BALANCED CORRECTLY (RESIDUAL SHOULD BE ~<= 10e-19)
! ZONAL MOMENTUM
!print *, "BUDGET: UTEND_AFTER = VQ - UTRUNC", Dyn%Grid%utend(40,40,future) - (Dyn%Grid%
vq(40,40,future) - Dyn%grid%utrunc(40,40,future)) ! BUDGET WITH NO DAMPING (SET
COEFFICENTS TO ZERO IN NAMELIST)
!print *, "BUDGET: UTEND_AFTER = VQ - UTRUNC - LIND + SPECD + STIR + ROBD", Dyn%Grid%
utend(10,10,future) - (Dyn%Grid%vq(10,10,future) - Dyn%grid%utrunc(10,10,future) - Dyn%
```

```
Grid%ulindamp_m(10,10,future) &
!- Dyn%Grid%ulindamp_e(10,10,future) + Dyn%grid%uspecdamp(10,10,future) + Dyn%grid%ustir
(10,10,future) + Dyn%grid%urobdamp(10,10,future))
! MERIDIONAL MOMENTUM
!print *, "BUDGET: VTEND = -UQ - VTRUNC", Dyn%Grid%vtend(10,10,future) - (-Dyn%Grid%uq
(10,10,future) - Dyn%grid%vtrunc(10,10,future))
!print *, "BUDGET: VTEND = -UQ - VTRUNC + VSPECDAMP + VROBDAMP", Dyn%Grid%vtend
(27,50,future) - (-Dyn%grid%uq(27,50,future) - Dyn%grid%vtrunc(27,50,future) + Dyn%grid%
vspecdamp(27,50,future) &
! + Dyn%grid%vrobdamp(27,50,future)) ! BUDGET WITH SPECTRAL DAMPING & ROBERTS DAMPING
! VORTICTY
!print *, "BUDGET: VORTEND = DT_VOR - VORTRUNC", Dyn%Grid%vortend(10,10,future) - (Dyn%
Grid%dt_vor(10,10,future) - Dyn%grid%vortrunc(10,10,future))
!print *, "BUDGET: VORTEND = DT_VOR - VORTRUNC - LIND + SPECD + STIR + ROBD", Dyn%Grid%
vortend(10,10,future) - (Dyn%Grid%dt_vor(10,10,future) - Dyn%grid%vortrunc
(10,10,future) - &
!Dyn%grid%vorlindamp_e(10,10,future) - Dyn%grid%vorlindamp_m(10,10,future) + Dyn%Grid%
vorspecdamp(10,10,future) + Dyn%Grid%vorstir(10,10,future) + Dyn%Grid%vorrobdamp
(10,10,future))
! ENERGY
!print *, "ENERGY_TEND = VORADVEC - GRADTERM - TRUNC - LINDAMP + SPECDAMP + STIR +
ROBDAMP", Dyn%Grid%energy_tend(10,10,future) - (Dyn%Grid%energy_voradvec(10,10,future)
&
! - Dyn%Grid%energy_gradterm(10,10,future) - Dyn%Grid%energy_lindamp_m(10,10,future) -
Dyn%Grid%energy_lindamp_e(10,10,future) - Dyn%Grid%energy_trunc(10,10,future) + Dyn%Grid
%energy_specdamp(10,10,future)&
! + Dyn%Grid%energy_stir(10,10,future) + Dyn%Grid%energy_robdamp(10,10,future))
! MEAN ENERGY
!print *, "ENERGY_TEND_MEAN = VORADVEC - GRADTERM - TRUNC - LINDAMP + SPECDAMP + STIR +
ROBDAMP", Dyn%Grid%energy_tend_mean(10,10,future) - (Dyn%Grid%energy_voradvec_mean
(10,10,future) &
! - Dyn%Grid%energy_gradterm_mean(10,10,future) - Dyn%Grid%energy_lindamp_m_mean
(10,10,future) - Dyn%Grid%energy_lindamp_e_mean(10,10,future) - Dyn%Grid%
energy_trunc_mean(10,10,future) &
! + Dyn%Grid%energy_specdamp_mean(10,10,future) + Dyn%Grid%energy_stir_mean
(10,10,future) + Dyn%Grid%energy_robdamp_mean(10,10,future))
!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

return
end subroutine barotropic_dynamics

!=================================================================================

subroutine update_spec_tracer(tr_spec, tr_grid, dt_tr, ug, vg, &
                              previous, current, future, delta_t)

complex, intent(inout), dimension(ms:me, ns:ne, num_time_levels) :: tr_spec
real   , intent(inout), dimension(is:ie, js:je, num_time_levels) :: tr_grid
real   , intent(inout), dimension(is:ie, js:je                 ) :: dt_tr
real   , intent(in   ), dimension(is:ie, js:je, num_time_levels) :: ug, vg
real   , intent(in   )  :: delta_t
integer, intent(in   )  :: previous, current, future

complex, dimension(ms:me, ns:ne) :: dt_trs

call horizontal_advection     (tr_spec(:,:,current), ug(:,:,current), vg(:,:,current),
dt_tr)
call trans_grid_to_spherical  (dt_tr, dt_trs)
call compute_spectral_damping (tr_spec(:,:,previous), dt_trs, delta_t)
call leapfrog                 (tr_spec, dt_trs, previous, current, future, delta_t,
```

```
robert_coeff)
call trans_spherical_to_grid  (tr_spec(:,:,future), tr_grid(:,:,future))

return
end subroutine update_spec_tracer
!==========================================================================

subroutine update_grid_tracer(tr_grid, dt_tr_grid, ug, vg, &
                              previous, current, future, delta_t)

real   , intent(inout), dimension(is:ie, js:je, num_time_levels) :: tr_grid
real   , intent(inout), dimension(is:ie, js:je                 ) :: dt_tr_grid
real   , intent(in   ), dimension(is:ie, js:je, num_time_levels) :: ug, vg

real   , intent(in   )  :: delta_t
integer, intent(in   )  :: previous, current, future

real, dimension(size(tr_grid,1),size(tr_grid,2)) :: tr_current, tr_future

tr_future = tr_grid(:,:,previous) + delta_t*dt_tr_grid
dt_tr_grid = 0.0
call a_grid_horiz_advection (ug(:,:,current), vg(:,:,current), tr_future, delta_t,
dt_tr_grid)
tr_future = tr_future + delta_t*dt_tr_grid
tr_current = tr_grid(:,:,current) + &
    robert_coeff*(tr_grid(:,:,previous) + tr_future - 2.0*tr_grid(:,:,current))
tr_grid(:,:,current) = tr_current
tr_grid(:,:,future)  = tr_future

return
end subroutine update_grid_tracer

!==========================================================================

subroutine read_restart(Dyn)

type(dynamics_type), intent(inout)  :: Dyn

integer :: unit, m, n, nt
real, dimension(ms:me, ns:ne) :: real_part, imag_part
if(file_exist('INPUT/barotropic_dynamics.res.nc')) then
  do nt = 1, 2
    call read_data('INPUT/barotropic_dynamics.res.nc', 'vors_real', real_part,
spectral_domain, timelevel=nt)
    call read_data('INPUT/barotropic_dynamics.res.nc', 'vors_imag', imag_part,
spectral_domain, timelevel=nt)
    do n=ns,ne
      do m=ms,me
        Dyn%Spec%vor(m,n,nt) = cmplx(real_part(m,n),imag_part(m,n))
      end do
    end do
    call read_data('INPUT/barotropic_dynamics.res.nc', 'roberts_real', real_part,
spectral_domain, timelevel=nt) ! Read restart for Dyn%spec%roberts variable
    call read_data('INPUT/barotropic_dynamics.res.nc', 'roberts_imag', imag_part,
spectral_domain, timelevel=nt)
    do n=ns,ne
      do m=ms,me
        Dyn%Spec%roberts(m,n,nt) = cmplx(real_part(m,n),imag_part(m,n))
      end do
    end do
    if(Dyn%spec_tracer) then
```

```
      call read_data('INPUT/barotropic_dynamics.res.nc', 'trs_real', real_part,
spectral_domain, timelevel=nt)
      call read_data('INPUT/barotropic_dynamics.res.nc', 'trs_imag', imag_part,
spectral_domain, timelevel=nt)
      do n=ns,ne
        do m=ms,me
           Dyn%Spec%trs(m,n,nt) = cmplx(real_part(m,n),imag_part(m,n))
        end do
      end do
    endif
    call read_data('INPUT/barotropic_dynamics.res.nc', 'u',   Dyn%Grid%u  (:,:,nt),
grid_domain, timelevel=nt)
    call read_data('INPUT/barotropic_dynamics.res.nc', 'v',   Dyn%Grid%v  (:,:,nt),
grid_domain, timelevel=nt)
    call read_data('INPUT/barotropic_dynamics.res.nc', 'vor', Dyn%Grid%vor(:,:,nt),
grid_domain, timelevel=nt)
    call read_data('INPUT/barotropic_dynamics.res.nc', 'energy', Dyn%Grid%energy
(:,:,nt), grid_domain, timelevel=nt)
    if(Dyn%spec_tracer) then
      call read_data('INPUT/barotropic_dynamics.res.nc', 'trs', Dyn%Grid%trs(:,:,nt),
grid_domain, timelevel=nt)
    endif
    if(Dyn%grid_tracer) then
      call read_data('INPUT/barotropic_dynamics.res.nc', 'tr', Dyn%Grid%tr(:,:,nt),
grid_domain, timelevel=nt)
    endif
  end do


else if(file_exist('INPUT/barotropic_dynamics.res')) then
  unit = open_restart_file(file='INPUT/barotropic_dynamics.res',action='read')

  do nt = 1, 2
    call set_domain(spectral_domain)
    call read_data(unit,Dyn%Spec%vor(:,:, nt))
    if(Dyn%spec_tracer) call read_data(unit,Dyn%Spec%trs(:,:, nt))

    call set_domain(grid_domain)
    call read_data(unit,Dyn%Grid%u   (:,:, nt))
    call read_data(unit,Dyn%Grid%v   (:,:, nt))
    call read_data(unit,Dyn%Grid%vor (:,:, nt))
    if(Dyn%spec_tracer) call read_data(unit,Dyn%Grid%trs(:,:, nt))
    if(Dyn%grid_tracer) call read_data(unit,Dyn%Grid%tr (:,:, nt))

  end do
  call close_file(unit)


else
  call error_mesg('read_restart', 'restart does not exist', FATAL)
endif


return
end subroutine read_restart



!====================================================================

subroutine write_restart(Dyn, previous, current)
```

```
type(dynamics_type), intent(in)  :: Dyn
integer, intent(in) :: previous, current

integer :: unit, nt, nn

do nt = 1, 2
  if(nt == 1) nn = previous
  if(nt == 2) nn = current
  call write_data('RESTART/barotropic_dynamics.res.nc', 'vors_real',  real(Dyn%Spec%vor
(:,:,nn)), spectral_domain)
  call write_data('RESTART/barotropic_dynamics.res.nc', 'vors_imag', aimag(Dyn%Spec%vor
(:,:,nn)), spectral_domain)
  call write_data('RESTART/barotropic_dynamics.res.nc', 'roberts_real',  real(Dyn%Spec%
roberts(:,:,nn)), spectral_domain) ! Write roberts into restart file
  call write_data('RESTART/barotropic_dynamics.res.nc', 'roberts_imag', aimag(Dyn%Spec%
roberts(:,:,nn)), spectral_domain)
  if(Dyn%spec_tracer) then
    call write_data('RESTART/barotropic_dynamics.res.nc', 'trs_real',  real(Dyn%Spec%trs
(:,:,nn)), spectral_domain)
    call write_data('RESTART/barotropic_dynamics.res.nc', 'trs_imag', aimag(Dyn%Spec%trs
(:,:,nn)), spectral_domain)
  endif
  call write_data('RESTART/barotropic_dynamics.res.nc', 'u',   Dyn%Grid%u  (:,:,nn),
grid_domain)
  call write_data('RESTART/barotropic_dynamics.res.nc', 'v',   Dyn%Grid%v  (:,:,nn),
grid_domain)
  call write_data('RESTART/barotropic_dynamics.res.nc', 'vor', Dyn%Grid%vor(:,:,nn),
grid_domain)
  call write_data('RESTART/barotropic_dynamics.res.nc', 'energy', Dyn%Grid%energy
(:,:,nn), grid_domain)
  if(Dyn%spec_tracer) then
    call write_data('RESTART/barotropic_dynamics.res.nc', 'trs', Dyn%Grid%trs(:,:,nn),
grid_domain)
  endif
  if(Dyn%grid_tracer) then
    call write_data('RESTART/barotropic_dynamics.res.nc', 'tr', Dyn%Grid%tr(:,:,nn),
grid_domain)
  endif
enddo

!unit = open_restart_file(file='RESTART/barotropic_dynamics.res', action='write')

!do nt = 1, 2
!  if(nt == 1) nn = previous
!  if(nt == 2) nn = current

!  call set_domain(spectral_domain)
!  call write_data(unit,Dyn%Spec%vor(:,:, nn))
!  if(Dyn%spec_tracer) call write_data(unit,Dyn%Spec%trs(:,:, nn))

!  call set_domain(grid_domain)
!  call write_data(unit,Dyn%Grid%u   (:,:, nn))
!  call write_data(unit,Dyn%Grid%v   (:,:, nn))
!  call write_data(unit,Dyn%Grid%vor (:,:, nn))
!  if(Dyn%spec_tracer) call write_data(unit,Dyn%Grid%trs(:,:, nn))
!  if(Dyn%grid_tracer) call write_data(unit,Dyn%Grid%tr (:,:, nn))
!end do

!call close_file(unit)
```

```
end subroutine write_restart

!=====================================================================

subroutine barotropic_dynamics_end (Dyn, previous, current)

type(dynamics_type), intent(inout)  :: Dyn
integer, intent(in) :: previous, current

if(.not.module_is_initialized) then
  call error_mesg('barotropic_dynamics','dynamics has not been initialized ', FATAL)
endif

call transforms_end()
call stirring_end()

call write_restart (Dyn, previous, current)

module_is_initialized = .false.

return
end subroutine barotropic_dynamics_end
!==================================================================================

end module barotropic_dynamics_mod
```