CEE263B Final Project Report

Source Code for Fortran90-based Program

Yun Zhang

SUID: 05729191

1. Regionalmodel.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regionalmodel
! usage: generate a basic regional
! model that solves the equations of
! atompheric dynamics.
! Yun Zhang 04/24/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! MODEL DESCRIPTION
! The purpose of this project is to
! develop a basic regional- or global-scale
! model that solves the equations of atmospheric
! dynamics, except for the water-vapor continuity
! equation. Diabatic energy sources and sinks and
! eddy diffusion are ignored. Winds are driven
! primarily by pressure gradients.

program regionalmodel
    use iso_fortran_env
    use allocate_variable
    use phys
    use output
    implicit none
    integer:: nt
    real(dp):: tstart,tend

    ! initialize lat and long
    call initialize_lat_long_coriolis(lat_c,long_c,lat_uface,long_uface,lat_vface,long_vface,f_c)

    ! part 1)
    ! calculate dsigma
    ! assume mean surface altitude 0m
    ! from table B.1
    call initialize_vertical_sigima(zbot_test,pa_test,sigma_bot,dsigma)

    ! part 2)
    ! calculate initial surface pressure
    call initialize_air_pressure_surf(Pa_surf)

    ! calculate initial column pressure
    call
initialize_pressure_system(pi_c,pi_c_new,pi_c_tminus1,Pa_bot,P_bot,P_c,Pa_c,Pa_surf,sigma_bo
```

```
t)

    ! part 3)
    ! initialize water mass mixing ratio
    call initialize_watermass_mixingratio(qv_c,qv_c_new,Qqv)

    ! initialize temperature for each cell
    call initialize_temperature_system(Temp_c,Pa_c,qv_c,PVT_c,PVT_c_new,P_c,P_bot,Qheat)

    ! initialize gas concentration for each cell
    if(gasmodel==1)                                                                    call
initialize_gas_concentration(gas_c,gas_c_new,Pa_c,P_c,P_bot,lat_c,long_c,Qgas)

    ! part 4) initialize air density for each cell center
    call calculate_air_density(rhoa_c,Pa_c,qv_c,Temp_c)

    ! part 5) initialize velocity field, zero everywhere
    call initialize_velocity_field(u,u_new,v,v_new,w_sigma,w_sigma_new,lat_uface,lat_vface,&
        long_uface,long_vface)

    ! part 6) initialize turbulence term
    call calculate_eddy_viscousity_diffusivity(nu_t,K_t)

    ! part 7) initialize heat source
    if(heatsource==1) call calculate_heat_source(Pa_c,Qheat,lat_c,long_c,0)

    ! initialize specific humidity source
    if(qvsource==1) call calculate_qv_source(Pa_c,Qqv,lat_c,long_c,nt)

    ! gas source
    if(gasmodel==1) call calculate_gas_source(Pa_c,Qgas,lat_c,long_c,0)

    ! part 8) initialize geopotential
    call
calculate_geopotential(geopot_bot,geopot_c,geopot_c_tminus1,P_c,P_bot,PVT_c,lat_c,long_c,0)

    print *, "The initialization process is finished!"

    ! store cpu time
    call cpu_time(tstart)

    ! main loop for physical function
    do nt=1,Ndt
```

```
! calculate column pressure at flux face
call calculate_fluxface_column_pressure(pi_c,pi_uface,pi_vface)

! calcualte flux sum for each cell each layer
call
calculate_fluxsum(fluxsum,flux_uface,flux_vface,u,v,pi_uface,pi_vface,lat_c,lat_vface,dsigma)

! update column pressure for each cell center
call calculate_column_pressure(pi_c,pi_c_new,lat_c,fluxsum)

! update w_sigma
call calculate_w_sigma(w_sigma_new,pi_c,pi_c_new,lat_c,fluxsum,sigma_bot,dsigma)

if(qvmodel==1) then
   ! calculate qv at flux face
   call calculate_fluxface_qv(qv_c,qv_uface,qv_vface,qv_bot,P_c,P_bot,&
      lat_uface,long_uface,lat_vface,long_vface,nt)

   ! update specific humidity
   call calculate_scalar_field(qv_c, qv_c_new, qv_bot, &
      pi_c,pi_c_new,qv_uface, qv_vface,flux_uface,flux_vface,lat_c, rhoa_c,&
      dsigma, w_sigma_new,K_t,Pa_c,Qqv)
endif

if(PVTmodel==1) then
   ! calculate PVT at flux face
   call calculate_fluxface_PVT(PVT_c,PVT_uface,PVT_vface,PVT_bot,&
      P_c,P_bot,lat_uface,long_uface,lat_vface,long_vface,nt)
   ! update potential temperature
   call calculate_scalar_field(PVT_c, PVT_c_new, PVT_bot, &
      pi_c,pi_c_new,PVT_uface, PVT_vface,flux_uface,flux_vface,lat_c, rhoa_c,&
      dsigma, w_sigma_new,K_t,Pa_c,Qheat)
endif

if(gasmodel==1) then
   ! calculate gas at flux face
   call calculate_fluxface_gas(gas_c,gas_uface,gas_vface,gas_bot,&
      P_c,P_bot,lat_uface,long_uface,lat_vface,long_vface,nt)
   ! update potential temperature
   call calculate_scalar_field(gas_c, gas_c_new, gas_bot, &
      pi_c,pi_c_new,gas_uface, gas_vface,flux_uface,flux_vface,lat_c, rhoa_c,&
      dsigma, w_sigma_new,K_t,Pa_c,Qgas)
endif
```

```
! update velocity field
call calculate_velocity_field(pi_c,pi_c_new,pi_c,pi_c_tminus1,lat_c, lat_vface,&
   u,u_new,u,v,v_new,v,flux_uface,flux_vface,dsigma,&
   sigma_bot,w_sigma_new,f_c,geopot_c,geopot_c_tminus1,&
   PVT_c,P_bot,P_c,nu_t,rhoa_c)

! update air pressure
call calculate_pressure_field(pi_c_new,Pa_c_new,Pa_bot_new,&
   P_c_new,P_bot_new,sigma_bot)

! update air temperature
call calculate_air_temperature(PVT_c_new,Temp_c_new,qv_c_new,Pa_c_new)

! update air density
call calculate_air_density(rhoa_c_new,Pa_c_new,qv_c_new,Temp_c_new)

! update eddy diffusivity and viscousity
if(turbmodel==1) then
   call calculate_eddy_viscousity_diffusivity(nu_t,K_t)
endif

! update geopotential
call calculate_geopotential(geopot_bot,geopot_c,geopot_c_tminus1,P_c_new,P_bot_new,&
   PVT_c_new,lat_c,long_c,nt)

! for Matsuno method
if(matsuno==1) then
   ! calculate column pressure at flux face
   call calculate_fluxface_column_pressure(pi_c_new,pi_uface,pi_vface)
   ! calcualte flux sum for each cell each layer
   call
calculate_fluxsum(fluxsum,flux_uface,flux_vface,u_new,v_new,pi_uface,pi_vface,lat_c,lat_vface,
dsigma)
   ! store the first calculation of pi for velocity calculation
   pi_c_tmp=pi_c_new
   ! update column pressure for each cell center
   call calculate_column_pressure(pi_c,pi_c_new,lat_c,fluxsum)

   ! update w_sigma
   call calculate_w_sigma(w_sigma_new,pi_c,pi_c_new,lat_c,fluxsum,sigma_bot,dsigma)

   if(qvmodel==1) then
    ! calculate qv at flux face
    call calculate_fluxface_qv(qv_c_new,qv_uface,qv_vface,qv_bot,P_c_new,P_bot_new,&
```

```
    lat_uface,long_uface,lat_vface,long_vface,nt)
   ! update specific humidity
   call calculate_scalar_field(qv_c, qv_c_new, qv_bot, &
    pi_c,pi_c_new,qv_uface, qv_vface,flux_uface,flux_vface,lat_c, rhoa_c_new,&
    dsigma, w_sigma_new,K_t,Pa_c_new,Qqv)
  endif

  if(PVTmodel==1) then
    ! calculate PVT at flux face
    call calculate_fluxface_PVT(PVT_c_new,PVT_uface,PVT_vface,PVT_bot,&
      P_c_new,P_bot_new,lat_uface,long_uface,lat_vface,long_vface,nt)
    ! store the first calculation of pVT for velocity calculation
    PVT_c_tmp=PVT_c_new
    ! update potential temperature
    call calculate_scalar_field(PVT_c, PVT_c_new, PVT_bot, &
      pi_c,pi_c_new,PVT_uface, PVT_vface,flux_uface,flux_vface,lat_c, rhoa_c_new,&
      dsigma, w_sigma_new,K_t,Pa_c_new,Qheat)
  endif

  if(gasmodel==1) then
    ! calculate PVT at flux face
    call calculate_fluxface_gas(gas_c_new,gas_uface,gas_vface,gas_bot,&
      P_c_new,P_bot_new,lat_uface,long_uface,lat_vface,long_vface,nt)
    ! update potential temperature
    call calculate_scalar_field(gas_c, gas_c_new, gas_bot, &
      pi_c,pi_c_new,gas_uface, gas_vface,flux_uface,flux_vface,lat_c, rhoa_c_new,&
      dsigma, w_sigma_new,K_t,Pa_c_new,Qgas)
  endif

  ! update velocity field
  call calculate_velocity_field(pi_c,pi_c_new,pi_c_tmp,pi_c_tminus1,lat_c, lat_vface,&
    u,u_new,u_new,v,v_new,v_new,flux_uface,flux_vface,dsigma,&
    sigma_bot,w_sigma_new,f_c,geopot_c,geopot_c_tminus1,&
    PVT_c_tmp,P_bot_new,P_c_new,nu_t,rhoa_c_new)

  ! update air pressure
  call calculate_pressure_field(pi_c_new,Pa_c_new,Pa_bot_new,&
    P_c_new,P_bot_new,sigma_bot)

  ! update air temperature
  call calculate_air_temperature(PVT_c_new,Temp_c_new,qv_c_new,Pa_c_new)

  ! update eddy diffusivity and viscousity
  if(turbmodel==1) then
```

```
            call calculate_eddy_viscousity_diffusivity(nu_t,K_t)
         endif

         ! update geopotential
         call
calculate_geopotential(geopot_bot,geopot_c,geopot_c_tminus1,P_c_new,P_bot_new,&
            PVT_c_new,lat_c,long_c,nt)

      endif

      ! update heat source
      if(heatsource==1) call calculate_heat_source(Pa_c_new,Qheat,lat_c,long_c,nt)

      ! update specific humidity source
      if(qvsource==1) call calculate_qv_source(Pa_c,Qqv,lat_c,long_c,nt)

      ! update gas source
      if(gasmodel==1) call calculate_gas_source(Pa_c,Qgas,lat_c,long_c,nt)

      ! replace all variable with the new variables
      ! column pressure
      pi_c_tminus1=pi_c
      pi_c=pi_c_new
      ! pressure variable
      Pa_bot=Pa_bot_new
      P_bot=P_bot_new
      Pa_c=Pa_c_new
      Pa_bot=Pa_bot_new
      ! temperature
      Temp_c=Temp_c_new
      PVT_c=PVT_c_new
      gas_c=gas_c_new
      ! specific humidity
      qv_c=qv_c_new
      ! air density
      rhoa_c=rhoa_c_new
      ! velocity field
      w_sigma=w_sigma_new
      u=u_new
      v=v_new

      ! output results: pi_c,Pa_c,PTV_c,u,v,w_sigma,nu_t,K_t,geobot_c, Temp_c, rha_c, qv
      ! at first time step open all file
      if(outputswitch==1) then
```

```
        if(nt==1) call output_open_txt_files()
        if(nt==1 .or. mod(nt,Nout)==0) then
          call output_all_variables()
        endif
        if(nt==Ndt) call output_close_files()
      endif
  enddo

  ! calculate runtime
  call cpu_time(tend)
  print *, 'The program has been finished. The runtime is ',(tend-tstart),'sec'
end program regionalmodel
```

2. allocatevariable.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - allocate_variable
! usage: allocate all variables in the program
! Yun Zhang 04/24/2015
! @stanford
! unit clarification
! pressure: hpa
! temperature: K
! latitude,longitude: degree
! density: kg/m^3
! altitude, length: m
! gravity: m/s/s
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module allocate_variable
    use constant_parameter
    implicit none

    ! Initialize parameter
    ! lat and long for each cell
    real(dp),dimension(NLAT,NLONG):: lat_c, long_c ! latitude and longitude
    real(dp),dimension(NLAT,NLONG):: f_c ! coriolis coefficient
    real(dp),dimension(NLAT+1,NLONG):: lat_vface, long_vface ! latitude at v flux face
    real(dp),dimension(NLAT,NLONG+1):: lat_uface, long_uface ! longtitude at u flux face

    ! part 1)
    ! test column to calculate dsigma
    real(dp), dimension(0:NVERT):: zbot_test ! layer bottom elevation at test column
    real(dp), dimension(0:NVERT):: pa_test ! layer bottom pressure at test column
    real(dp), dimension(0:NVERT):: sigma_bot ! sigma value at each column, constant for all time
step
    real(dp), dimension(NVERT):: dsigma ! sigma(k+1)-sigma(k)

    ! part 2) pressure setup
    ! initialize surface pressure
    real(dp), dimension(NLAT,NLONG):: Pa_surf ! surface pressure
    real(dp), dimension(NLAT,NLONG):: pi_c_new, pi_c, pi_c_tminus1, pi_c_tmp! column pressure
at t+1 t t-1
    real(dp), dimension(NLAT+1,NLONG):: pi_vface ! column pressure at v face
    real(dp), dimension(NLAT,NLONG+1):: pi_uface ! column pressure at u face


    ! calculate all pressure at different layers for different cells
    real(dp), dimension(NLAT,NLONG,0:NVERT):: Pa_bot, Pa_bot_new ! layer bottom air pressure
```

at each layer at t t+1

real(dp), dimension(NLAT,NLONG,0:NVERT):: P_bot, P_bot_new ! layer bottom P at each layer at t t+!

real(dp), dimension(NLAT,NLONG,NVERT):: Pa_c, Pa_c_new ! cell center air pressure at t t+1

real(dp), dimension(NLAT,NLONG,NVERT):: P_c, P_c_new ! cell center P at t t+1

! part 3) initialize temperature, humidity and gas

real(dp), dimension(NLAT,NLONG,NVERT):: Temp_c, Temp_c_new ! cell center Temperature at t t+1

real(dp), dimension(NLAT,NLONG,NVERT):: PVT_c, PVT_c_new, PVT_c_tmp! cell center potential virtual temperature at t t+1

real(dp), dimension(NLAT,NLONG,0:NVERT):: PVT_bot, qv_bot, gas_bot! layer bottom potential virtual temperature and specific humidity

real(dp), dimension(NLAT,NLONG,NVERT):: qv_c, qv_c_new !cell center specific humidity at t t+1

real(dp), dimension(NLAT,NLONG,NVERT):: gas_c, gas_c_new !cell center gas concentration at t t+1

real(dp), dimension(NLAT+1,NLONG,NVERT):: PVT_vface, qv_vface, gas_vface! v face potential virtual temperature and specific humidity

real(dp), dimension(NLAT,NLONG+1,NVERT):: PVT_uface, qv_uface, gas_uface ! u face potential virtual temperature and specific humidity

! part 4) initialize air density

real(dp), dimension(NLAT,NLONG,NVERT):: rhoa_c, rhoa_c_new ! cell center air density at t t+1

! part 5) initialize velocity field

real(dp), dimension(NLAT,NLONG+1,NVERT):: u, u_new ! u at t t+1

real(dp), dimension(NLAT+1,NLONG,NVERT):: v, v_new ! v at t t+1

real(dp), dimension(NLAT,NLONG,0:NVERT):: w_sigma, w_sigma_new ! w_sigma at t t+!

! part 6) calculate column pressure at flux face

real(dp), dimension(NLAT,NLONG,NVERT):: fluxsum ! fluxsum(i,j,k) means the sum of flux from layer 1 to k at cell i,j

real(dp), dimension(NLAT,NLONG+1,NVERT):: flux_uface ! the sum of flux from u direction at each layer

real(dp), dimension(NLAT+1,NLONG,NVERT):: flux_vface ! the sum of flux from v direction at each layer

! part 7) turbulence variable

real(dp),dimension(NLAT,NLONG,NVERT):: nu_t,K_t ! cell center eddy viscousity and diffusity

```
    ! part 8) heat/gas source variable
    real(dp),dimension(NLAT,NLONG,NVERT):: Qheat ! cell center heat source
    real(dp),dimension(NLAT,NLONG,NVERT):: Qgas ! cell center gas source
    real(dp),dimension(NLAT,NLONG,NVERT):: Qqv ! cell center specific source

    ! part 9) geopotential
    real(dp), dimension(NLAT,NLONG,NVERT):: geopot_c,geopot_c_tminus1 ! geopotential at t t-
1
    real(dp), dimension(NLAT,NLONG,0:NVERT)::geopot_bot ! geopotential at layer bottom

end module allocate_variable
```

3. basic_state.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - basic_state
! usage: external procedure for all air
! state equation to transfer air, temp
! pressure, density and other related
! aspects
! Yun Zhang 04/24/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module basic_state
    use constant_parameter
    implicit none

contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! standard_atmophere_interp
! usage: use table B.1 to interpolate pressure, altitude,
! gravity, temperature, air density
! input: a vertical array for horizontal cell
! output: get the relevant data based on input
! input_mode: 'z'=altitude,'p'=pressure,'g'=gravity
! 'T'=temperature,'rho'=density
! output_mode is same as input_mode
! Yun Zhang @Stanford
! 04/25/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function standard_atmophere_interp(input,N,input_mode,output_mode) result(output)
    integer, intent(in)::N
    real(dp), dimension(1:N),intent(in)::input
    real(dp), dimension(1:N)::output
    character, intent(in)::input_mode,output_mode
    real(dp), dimension(5):: tmp_data
    real(dp), dimension(79):: alt, p, g, T, rho, basedata, outdata
    integer:: i,j,k,order1,order2

    ! read tableb1
    open (unit=99, file='tableb1.txt', status='old', action='read')
    do i=1,79
        read(99,*) tmp_data
        alt(i)=tmp_data(1)*1000
        p(i)=tmp_data(3)
        g(i)=tmp_data(2)
```

```
    T(i)=tmp_data(4)
    rho(i)=tmp_data(5)
enddo
close(99)
order1=1
order2=-1
select case(input_mode)
    case ('z')
        basedata=alt
        order1=-1
        order2=-1
    case ('p')
        basedata=p
    case ('g')
        basedata=g
    case ('T')
        basedata=T
    case ('rho')
        basedata=rho
end select

select case(output_mode)
    case ('z')
        outdata=alt
    case ('p')
        outdata=p
    case ('g')
        outdata=g
    case ('T')
        outdata=T
    case ('rho')
        outdata=rho
end select

! find section no. at basedata for each value in input data
! then interpolate data based on the section number
k=79
do i=1,N
    do j=1,k
        if((input(i)*order1)>(basedata(j)*order1)) then
            k=j
            output(i)=outdata(j+order2*1)+(input(i)-basedata(j+order2*1))&
                /(basedata(j)-basedata(j+order2*1))*(outdata(j)-outdata(j+order2*1))
            !print    *,    j,    input_mode,    input(i),    basedata(j),basedata(j+order2*1),
```

```
outdata(j),outdata(j+order2*1), output(i)

              exit
            endif
        enddo
    enddo
end function standard_atmophere_interp

end module     basic_state
```

4. boundary.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - boundary
! usage: include all the functions and
! subroutines to define boundary condition
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module boundary
   use constant_parameter
   implicit none

contains

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! boundary_potential_virtual_temp
! usage: provide the PVT value at domain
! boundary
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function boundary_potential_virtual_temp(lat,long,nlayer,t) result(output)
   real(dp),intent(in)::lat,long,t
   integer, intent(in):: nlayer
   real(dp)::output

   ! change boundary condition for differnt problem
   output=1.0_dp

end function boundary_potential_virtual_temp

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! boundary_specific_humidity
! usage: provide the qv value at domain
! boundary
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function boundary_specific_humidity(lat,long,nlayer,t) result(output)
   real(dp),intent(in)::lat,long,t
   integer, intent(in):: nlayer
   real(dp)::output

   ! change boundary condition for differnt problem
```

```
    output=1.0_dp

end function boundary_specific_humidity

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! boundary_gas
! usage: provide the gas concentration value at domain
! boundary
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function boundary_gas(lat,long,nlayer,t) result(output)
    real(dp),intent(in)::lat,long,t
    integer, intent(in):: nlayer
    real(dp)::output

    ! change boundary condition for differnt problem
    output=1.0_dp

end function boundary_gas

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! boundary_surf_geopotential
! usage: provide the geopotential boundary
! value at the surface layer
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function boundary_surf_geopotential(lat,long,t) result(output)
    real(dp),intent(in)::lat,long,t
    real(dp)::output

    ! change boundary condition for differnt problem
    output=0.0_dp

end function boundary_surf_geopotential

end module boundary
```

5. constant_parameter.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - constant_parameter
! usage: define all the paramter and
! precision setup
! Yun Zhang 04/24/2015
! @stanford
! unit clarification
! pressure: hpa
! temperature: K
! latitude,longitude: rad
! density: kg/m^3
! altitude, length: m
! gravity: m/s/s
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module constant_parameter
   use iso_fortran_env, only: real32, real64, int64
   ! basic setup
   integer, parameter:: sp=real32
   integer, parameter:: dp=real64
   integer, parameter:: li=int64
   real(dp), parameter:: pi = 4.0_dp*atan(1.0_dp) ! pi
   real(dp), parameter:: k_therm=0.286_dp
   real(dp), parameter:: R_prime=2.8704_dp !m^3 hpa kg^-1 K^-1
   real(dp), parameter:: Cp_d=1004.67 ! J kg^-1 K^-1
   real(dp), parameter:: Omega=7.2921e-5 ! rad/s rotation frequency for earth
   real(dp), parameter:: Re=6371000.0_dp ! Earth radius use unit in million of meters

   ! switch
   integer, parameter:: PVTmodel=1 ! whether the air is dry (0) or not(1)
   integer, parameter:: PTVbound=0 ! whether set exact boundary condition (1) or not (0)
   integer, parameter:: heatsource=0 ! whether there is heat source or sink in each cell
   integer, parameter:: gasmodel=1 ! whether to simulation passive gas transport
   integer, parameter:: gasbound=0 ! whether to use specified boundary condition
   integer, parameter:: gassource=0 ! whether there is a gas source within the domain
   integer, parameter:: qvmodel=1 ! whether the air is dry (0) or not(1)
   integer, parameter:: qvbound=0 ! whether set exact boundary condition (1) or not (0)
   integer, parameter:: qvsource=0 ! whether there is a specific humidity source within the domain
   integer, parameter:: turbmodel=0 ! whether to turn on turbulence
   integer, parameter:: outputswitch=1 ! whether output results or not
   integer, parameter:: periodicBC=0 ! whether to use periodic boundary condition
   integer, parameter:: matsuno=1 ! whether to use matsuno scheme
   integer, parameter:: coriolis=1 ! whether to consider coriolis effects
```

```
! numerical parameter
integer, parameter:: Ndt=500! the numbers of time steps
integer, parameter:: Nout=50 ! how often to output results
integer, parameter:: Nrep=1 ! how often to report progress
real(dp), parameter:: dt=5.0_dp! time step0

! grid parameter
real(dp), parameter:: lat_0=-1/180*pi ! southwest corner latitude
real(dp), parameter:: long_0=-1/180*pi ! southwest corner longitude

integer, parameter:: NLAT=40! the numbers of latitude cells
integer, parameter:: NLONG=40 ! the numbers of longitude cells
integer, parameter:: NVERT=15 ! the number of vertical layers

real(dp), parameter:: dlamda_e=0.05/180*pi ! differential longitude to represent cell size
real(dp), parameter:: dphi=0.05/180*pi ! differential latitude to represent cell size
real(dp), parameter:: phi_center=lat_0+((NLAT-1)/2+DBLE(mod((NLAT-1),2))/2)*dphi ! the latitude center in the domain
real(dp), parameter:: lamda_center=long_0+((NLONG-1)/2+DBLE(mod((NLONG-1),2))/2)*dlamda_e ! the longitude center in the domain
real(dp), parameter:: z_surf=0.0_dp ! surface elevation

! Phys parameter
real(dp), parameter:: Pa_top=250_dp    ! top pressure constant
real(dp), parameter:: Pa_below=265_dp, rhoa_below=0.414_dp, g_below=9.7764_dp, z_below=10000_dp ! values from table B.1
real(dp), parameter:: ztop_test=z_below+(Pa_below-Pa_top)*100/rhoa_below/g_below ! top elevation for test column
real(dp), parameter:: Pa_base=1000.0_dp, dpa_peak=10.0_dp

! output results: pi_c,Pa_c,PTV_c,u,v,w_sigma,nu_t,K_t,geobot_c, Temp_c, rhoa_c
! output file names
character(*), parameter:: resultfolder="/Users/zyaj/Documents/atomsphere-regional-model/results/"
character(*), parameter:: pi_file="pi.txt" ! file for column pressure results
character(*), parameter:: pi_format="(1xf9.4)"
integer, parameter:: pi_file_no=1
character(*), parameter:: Pa_file="Pa.txt" ! file for air pressure results
character(*), parameter:: Pa_format="(1xf9.4)"
integer, parameter:: Pa_file_no=2
character(*), parameter:: PTV_file="PVT.txt" ! file for potential virtual temperature results
integer, parameter:: PVT_file_no=3
character(*), parameter:: PVT_format="(1xf9.4)"
character(*), parameter:: u_file="u.txt" ! file for u results
```

```
    integer, parameter:: u_file_no=4
    character(*), parameter:: u_format="(1xf9.6)"
    character(*), parameter:: v_file="v.txt" ! file for v results
    integer, parameter:: v_file_no=5
    character(*), parameter:: v_format="(1xf9.6)"
    character(*), parameter:: w_file="w.txt" ! file for w_sigma results
    integer, parameter:: w_file_no=6
    character(*), parameter:: w_format="(1xf9.6)"
    character(*), parameter:: nu_file="nu.txt" ! file for eddy viscousity results
    integer, parameter:: nu_file_no=7
    character(*), parameter:: nu_t_format="(1xf9.4)"
    character(*), parameter:: K_t_file="K_t.txt" ! file for eddy diffusivity results
    integer, parameter:: K_t_file_no=8
    character(*), parameter:: K_t_format="(1xf9.4)"
    character(*), parameter:: geopot_file="geopot.txt" ! file for geopotential results
    integer, parameter:: geopot_file_no=9
    character(*), parameter:: geopot_format="(1xf12.4)"
    character(*), parameter:: Temp_file="temp.txt" ! file for temperature results
    integer, parameter:: Temp_file_no=10
    character(*), parameter:: Temp_format="(1xf9.4)"
    character(*), parameter:: rhoa_file="rhoa.txt" ! file for air density results
    integer, parameter:: rhoa_file_no=11
    character(*), parameter:: rhoa_format="(1xf6.4)"
    character(*), parameter:: qv_file="qv.txt" ! file for specific humidity results
    integer, parameter:: qv_file_no=12
    character(*), parameter:: qv_format="(1xf6.4)"
    character(*), parameter:: gas_file="gas.txt" ! file for potential virtual temperature results
    integer, parameter:: gas_file_no=13
    character(*), parameter:: gas_format="(1xf10.8)"
end module constant_parameter
```

6. initialization.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - initialization
! usage: initialize all the variables
! Yun Zhang 04/29/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module initialization
   use constant_parameter
   use basic_state
   implicit none

contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_lat_long
! usage: initialize all lat and long for cell center and flux face
! Yun Zhang @Stanford
! 04/25/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine initialize_lat_long_coriolis(lat_c,long_c,lat_uface,long_uface,lat_vface,long_vface,f_c)
   integer::i,j,k
   real(dp),dimension(NLAT,NLONG),intent(inout):: lat_c, long_c,f_c
   real(dp),dimension(NLAT+1,NLONG),intent(inout):: lat_vface, long_vface
   real(dp),dimension(NLAT,NLONG+1),intent(inout):: lat_uface, long_uface
   ! calculate lat and long tidu for all cell
   do i=1,NLAT
      do j=1,NLONG
         lat_c(i,j)=lat_0+(i-1)*dphi
         long_c(i,j)=long_0+(j-1)*dlamda_e
      enddo
   enddo

   ! coriolis efficients
   if(coriolis==1) then
      f_c=2*Omega*sin(lat_c)
   else
      f_c=0.0_dp
   endif

   ! uface
   lat_uface(:,1:NLONG)=lat_c
   lat_uface(:,NLONG+1)=lat_c(:,NLONG)
   long_uface(:,1:NLONG)=long_c-0.5*dlamda_e
   long_uface(:,NLONG+1)=long_c(:,NLONG)+0.5*dlamda_e
```

```
    ! vface
    lat_vface(1:NLAT,:)=lat_c-0.5*dphi
    lat_vface(NLAT+1,:)=lat_c(NLAT,:)+0.5*dphi
    long_vface(1:NLAT,:)=long_c
    long_vface(NLAT+1,:)=long_c(NLAT,:)

end subroutine initialize_lat_long_coriolis

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_vertical_sigima
! usage: calculate and initialize sigma grid for test column
! Yun Zhang @Stanford
! 04/25/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine initialize_vertical_sigima(zbot_test,pa_test,sigma_bot,dsigma)
    integer::i,j,k
    real(dp), dimension(0:NVERT),intent(inout):: zbot_test, pa_test, sigma_bot ! from layer 0
    real(dp), dimension(NVERT),intent(inout):: dsigma
    do i=0,NVERT
        zbot_test(i)=z_surf+(ztop_test-z_surf)*(1-DBLE(i)/NVERT)
    enddo

    ! calculate pa_test
    pa_test=standard_atmophere_interp(zbot_test,NVERT+1,'z','p')
    pa_test(0)=Pa_top
    ! calculate sigma_bot
    sigma_bot=(pa_test-Pa_top)/(pa_test(NVERT)-Pa_top)
    dsigma=sigma_bot(1:NVERT)-sigma_bot(0:(NVERT-1))
end subroutine initialize_vertical_sigima

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_air_pressure_surf
! usage: initialize the air pressure at surface for each cell
! Yun Zhang @stanford
! 05/01/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine initialize_air_pressure_surf(Pa_surf)
    real(dp),dimension(NLAT,NLONG),intent(inout)::Pa_surf
    integer::i,j
    real(dp)::tmp
    do i=1,NLAT
        do j=1,NLONG
            tmp=((Re/1000000*cos((lat_0+(i-1)*dphi+phi_center)/2)*(long_0+(j-1)*dlamda_e-
```

```
lamda_center))**2)/2
            tmp=tmp+((Re/1000000*(lat_0+(i-1)*dphi-phi_center))**2)/2
            Pa_surf(i,j)=Pa_base+dpa_peak*exp(-tmp)
        enddo
    enddo
end subroutine initialize_air_pressure_surf

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_pressure_system
! usage: initialize the column pressure, air pressure
! at the bottom and center for each layer at each cell
! Yun Zhang @stanford
! 05/01/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine
initialize_pressure_system(pi_c_old,pi_c_new,pi_c_tminus1,Pa_bot,P_bot,P_c,Pa_c,Pa_surf,sigm
a_bot)
    real(dp),dimension(NLAT,NLONG),intent(inout)::pi_c_old,pi_c_new,pi_c_tminus1
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(inout)::Pa_bot,P_bot
    real(dp),dimension(0:NVERT),intent(in)::sigma_bot
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::Pa_c,P_c
    real(dp),dimension(NLAT,NLONG),intent(in)::Pa_surf
    integer::i,j

    pi_c_old=Pa_surf-Pa_top
    pi_c_tminus1=pi_c_old
    ! provide approximate value for pi_timus1 at boundaries
    pi_c_tminus1(1,:)=pi_c_old(1,:)+(pi_c_old(1,:)-pi_c_old(2,:))
    pi_c_tminus1(NLAT,:)=pi_c_old(NLAT,:)+(pi_c_old(NLAT,:)-pi_c_old(NLAT-1,:))
    pi_c_tminus1(:,1)=pi_c_old(:,1)+(pi_c_old(:,1)-pi_c_old(:,2))
    pi_c_tminus1(:,NLONG)=pi_c_old(:,NLONG)+(pi_c_old(:,NLONG)-pi_c_old(:,NLONG-1))
    pi_c_new=pi_c_old

    ! calculate pressure at different vertical layer
    ! layer bottom
    do i=1,NLAT
        do j=1,NLONG
            Pa_bot(i,j,:)=sigma_bot*pi_c_old(i,j)+Pa_top
            P_bot(i,j,:)=(Pa_bot(i,j,:)/1000)**k_therm
        enddo
    enddo

    ! layer center
    do i=1,NLAT
```

```
        do j=1,NLONG
            P_c(i,j,:)=1/(1+k_therm)*(P_bot(i,j,1:NVERT)*Pa_bot(i,j,1:NVERT)-
Pa_bot(i,j,0:(NVERT-1))*P_bot(i,j,0:(NVERT-1)))
            P_c(i,j,:)=P_c(i,j,:)/(Pa_bot(i,j,1:NVERT)-Pa_bot(i,j,0:NVERT-1))
            Pa_c(i,j,:)=1000*(P_c(i,j,:)**(1/k_therm))
        enddo
    enddo

end subroutine initialize_pressure_system

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_watermass_mixingratio
! usage: initialize qv for each cell each layer
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine initialize_watermass_mixingratio(qv_c,qv_c_new,Qqv)
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::qv_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::qv_c_new, Qqv
    qv_c=0.0_dp
    qv_c_new=qv_c
    Qqv=0.0_dp
end subroutine initialize_watermass_mixingratio

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_temperature_system
! usage: first initial the southwest corner of the domain
! then set all the value for all cells and layers
! Yun Zhang @stanford
! 05/01/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine
initialize_temperature_system(Temp_c,Pa_c,qv_c,PVT_c_old,PVT_c_new,P_c,P_bot,Qheat)

real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::Temp_c,PVT_c_old,PVT_c_new,Qhea
t
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::qv_c,Pa_c,P_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::P_bot
    real(dp)::tmp
    integer::i,j,k
    ! first choose one corner use the southwest cornder (1,1,:)
    Temp_c(1,1,:)=standard_atmophere_interp(Pa_c(1,1,:),NVERT,'p','T')

    ! set all other cells temperature and potential virtual temperature
```

```
    do i=1,NLAT
        do j=1,NLONG
            Temp_c(i,j,:)=Temp_c(1,1,:)
        enddo
    enddo

    ! set potential temperature at cell center
    PVT_c_old=Temp_c*(1+0.608*qv_c)*((1000/Pa_c)**k_therm)
    PVT_c_new=PVT_c_old
    Qheat=0.0_dp
end subroutine initialize_temperature_system

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_gas_concentration
! usage: set the initial value for passive gas concentration
! Yun Zhang @stanford
! 05/01/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine initialize_gas_concentration(gas_c_old,gas_c_new,Pa_c,P_c,P_bot,lat_c,long_c,Qgas)
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::gas_c_old,gas_c_new,Qgas
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::Pa_c,P_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::P_bot
    real(dp),dimension(NLAT,NLONG),intent(in)::lat_c,long_c
    real(dp)::tmp
    integer::i,j,k
    ! set all other cells temperature and potential virtual temperature
    do i=1,NLAT
        do j=1,NLONG
            if(j>18 .and. j<22 .and. i>18 .and. i<22) then
                gas_c_old(i,j,:)=1.0_dp
            else
                gas_c_old(i,j,:)=0.0_dp !Temp_c(1,1,:)
            endif
        enddo
    enddo

    ! set potential temperature at cell center
    gas_c_new=gas_c_old
    Qgas=0.0_dp
end subroutine initialize_gas_concentration

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! initialize_velocity_field
! usage: initialize field velocity field
```

```
! Yun Zhang @stanford
! 05/01/2015
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine
initialize_velocity_field(u,u_new,v,v_new,w_sigma_old,w_sigma_new,lat_uface,lat_vface,long_u
face,long_vface)
   real(dp), dimension(NLAT,NLONG+1,NVERT),intent(inout):: u,u_new
   real(dp), dimension(NLAT+1,NLONG,NVERT),intent(inout):: v,v_new
   real(dp), dimension(NLAT,NLONG,0:NVERT),intent(inout):: w_sigma_old, w_sigma_new
   real(dp), dimension(NLAT,NLONG+1),intent(in)::lat_uface,long_uface
   real(dp), dimension(NLAT+1,NLONG),intent(in)::long_vface,lat_vface
   u=0.0_dp
   v=0.0_dp
   u_new=u
   v_new=v
   w_sigma_old=0.0_dp
   w_sigma_new=0.0_dp
end subroutine initialize_velocity_field

end module initialization
```

7. output.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - output
! usage: output all results
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module output
    use allocate_variable
    use constant_parameter
    implicit none

contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_open_files
! usage: open all output files for output
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_open_bin_files()
    open (unit = pi_file_no, file = resultfolder//pi_file, status='new',form='unformatted',recl=2*10)
    open (unit = Pa_file_no, file = resultfolder//Pa_file, status='new',form='unformatted',recl=2*10)
    if(PVTmodel==1) open (unit = PVT_file_no, file = resultfolder//PTV_file, status='new',form='unformatted',recl=2*10)
    open (unit = u_file_no, file = resultfolder//u_file, status='new',form='unformatted',recl=2*10)
    open (unit = v_file_no, file = resultfolder//v_file, status='new',form='unformatted',recl=2*10)
    open (unit = w_file_no, file = resultfolder//w_file, status='new',form='unformatted',recl=2*10)
    open (unit = nu_file_no, file = resultfolder//nu_file, status='new',form='unformatted',recl=2*10)
    open (unit = K_t_file_no, file = resultfolder//K_t_file, status='new',form='unformatted',recl=2*10)
    open (unit = geopot_file_no, file = resultfolder//geopot_file, status='new',form='unformatted',recl=2*10)
    if(PVTmodel==1) open (unit = Temp_file_no, file = resultfolder//Temp_file, status='new',form='unformatted',recl=2*10)
    open (unit = rhoa_file_no, file = resultfolder//rhoa_file, status='new',form='unformatted',recl=2*10)
    if(qvmodel==1) open (unit = qv_file_no, file = resultfolder//qv_file, status='new',form='unformatted',recl=2*10)
    if(gasmodel==1) open (unit = gas_file_no, file = resultfolder//qv_file, status='new',form='unformatted',recl=2*10)

end subroutine output_open_bin_files
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_open_files
! usage: open all output files for output
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_open_txt_files()
   open (unit = pi_file_no, file = resultfolder//pi_file)
   open (unit = Pa_file_no, file = resultfolder//Pa_file)
   if(PVTmodel==1) open (unit = PVT_file_no, file = resultfolder//PTV_file)
   open (unit = u_file_no, file = resultfolder//u_file)
   open (unit = v_file_no, file = resultfolder//v_file)
   open (unit = w_file_no, file = resultfolder//w_file)
   open (unit = nu_file_no, file = resultfolder//nu_file)
   open (unit = K_t_file_no, file = resultfolder//K_t_file)
   open (unit = geopot_file_no, file = resultfolder//geopot_file)
   if(PVTmodel==1) open (unit = Temp_file_no, file = resultfolder//Temp_file)
   open (unit = rhoa_file_no, file = resultfolder//rhoa_file)
   if(qvmodel==1) open (unit = qv_file_no, file = resultfolder//qv_file)
   if(gasmodel==1) open (unit = gas_file_no, file = resultfolder//gas_file)
end subroutine output_open_txt_files

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_all_variables
! usage: output all variables
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_all_variables()

   ! column pressure
   call output_2Dcellcenter_variables(pi_c,pi_file_no,pi_format)

   ! air pressure
   call output_3Dcellcenter_variables(Pa_c,Pa_file_no,Pa_format)

   ! u
   call output_3Duface_variables(u,u_file_no,u_format)

   ! v
   call output_3Dvface_variables(v,v_file_no,v_format)

   ! w_sigma
   call output_3Dvertical_variables(w_sigma,w_file_no,w_format)
```

```
    ! eddy viscousity
    call output_3Dcellcenter_variables(nu_t,nu_file_no,nu_t_format)

    ! eddy diffusivity
    call output_3Dcellcenter_variables(K_t,K_t_file_no,K_t_format)

    ! geopotential
    call output_3Dcellcenter_variables(geopot_c,geopot_file_no,geopot_format)

    if(PVTmodel==1) then
        ! Temperature
        call output_3Dcellcenter_variables(Temp_c,Temp_file_no,Temp_format)
        ! potential virtual temperature
        call output_3Dcellcenter_variables(PVT_c,PVT_file_no,PVT_format)
    endif

    ! air density
    call output_3Dcellcenter_variables(rhoa_c,rhoa_file_no,rhoa_format)

    ! specific humidity
    if(qvmodel==1) call output_3Dcellcenter_variables(qv_c,qv_file_no,qv_format)

    ! gas concentration
    if(gasmodel==1) call output_3Dcellcenter_variables(gas_c,gas_file_no,gas_format)

end subroutine output_all_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_3Dcellcenter_variables
! usage: output 3D cellcenter all variables
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_3Dcellcenter_variables(Value,file_no,format)
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format
    do i=1,NLAT
        do j=1,NLONG
            do k=1,NVERT-1
                write(file_no, format, iostat=ios, advance="no") Value(i,j,k)
            enddo
```

```
            write(file_no, format, iostat=ios) Value(i,j,NVERT)
            if ( ios /= 0 ) then
              print *,file_no
              stop "Write error in output file"
            endif
        enddo
    enddo
end subroutine output_3Dcellcenter_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_3Duface_variables
! usage: output 3D variables at u flux face
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_3Duface_variables(Value,file_no,format)
    real(dp),dimension(NLAT,NLONG+1,NVERT),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format
    do i=1,NLAT
        do j=1,NLONG+1
            do k=1,NVERT-1
                write(file_no, format, iostat=ios,advance="no") Value(i,j,k)
            enddo
            write(file_no, format, iostat=ios) Value(i,j,NVERT)
            if ( ios /= 0 ) then
              print *,file_no
              stop "Write error in output file"
            endif
        enddo
    enddo
end subroutine output_3Duface_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_3Duface_variables
! usage: output 3D variables at v flux face
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_3Dvface_variables(Value,file_no,format)
    real(dp),dimension(NLAT+1,NLONG,NVERT),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
```

```
    character(*), intent(in)::format

    do i=1,NLAT+1
        do j=1,NLONG
            do k=1,NVERT-1
                write(file_no, format, iostat=ios,advance="no") Value(i,j,k)
            enddo
            write(file_no, format, iostat=ios) Value(i,j,NVERT)
            if ( ios /= 0 ) then
              print *,file_no
              stop "Write error in output file"
            endif
        enddo
    enddo
end subroutine output_3Dvface_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_3Dvertical_variables
! usage: output 3D variables at vertical face layer top-bottom
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_3Dvertical_variables(Value,file_no,format)
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format

    do i=1,NLAT
        do j=1,NLONG
            do k=0,NVERT-1
                write(file_no, format, iostat=ios,advance="no") Value(i,j,k)
            enddo
            write(file_no, format, iostat=ios) Value(i,j,NVERT)
            if ( ios /= 0 ) then
              print *,file_no
              stop "Write error in output file"
            endif
            enddo
    enddo
end subroutine output_3Dvertical_variables
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_2Dcellcenter_variables
! usage: output 2D cellcenter all variables
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_2Dcellcenter_variables(Value,file_no,format)
    real(dp),dimension(NLAT,NLONG),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format

    do i=1,NLAT
        do j=1,NLONG-1
            write(file_no, format, iostat=ios, advance="no") Value(i,j)
        enddo
        write(file_no, format, iostat=ios) Value(i,NLONG)
        if ( ios /= 0 ) then
          print *,file_no
          stop "Write error in output file"
        endif
    enddo
end subroutine output_2Dcellcenter_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_2Duface_variables
! usage: output 2D cellcenter at uface
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_2Duface_variables(Value,file_no,format)
    real(dp),dimension(NLAT,NLONG+1),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format
    do i=1,NLAT
        do j=1,NLONG
            write(file_no, format, iostat=ios,advance="no") Value(i,j)
        enddo
        write(file_no, format, iostat=ios) Value(i,NLONG+1)

        if ( ios /= 0 ) then
            print *,file_no
            stop "Write error in output file"
```

```
        endif
    enddo
end subroutine output_2Duface_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_2Dvface_variables
! usage: output 2D cellcenter at v flux face
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_2Dvface_variables(Value,file_no,format)
    real(dp),dimension(NLAT+1,NLONG),intent(in):: Value
    integer, intent(in):: file_no
    integer:: i,j,k,ios
    character(*), intent(in)::format
    do i=1,NLAT+1
        do j=1,NLONG-1
            write(file_no, format, iostat=ios,advance="no") Value(i,j)
        enddo
        write(file_no, format, iostat=ios) Value(i,NLONG)
        if ( ios /= 0 ) then
          print *,file_no
          stop "Write error in output file"
        endif
    enddo
end subroutine output_2Dvface_variables

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! output_close_files
! usage: close all output files for output
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine output_close_files()
    close (pi_file_no)
    close (Pa_file_no)
    close (PVT_file_no)
    close (u_file_no)
    close (v_file_no)
    close (w_file_no)
    close (nu_file_no)
    close (K_t_file_no)
    close (geopot_file_no)
    close (Temp_file_no)
```

```
    close (rhoa_file_no)
    close (qv_file_no)
end subroutine output_close_files

end module output
```

8. phys.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - phys
! usage: include all the subroutines
! and functions to calculate physical equations
! Yun Zhang 04/29/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module phys
    use constant_parameter
    use boundary
    use turbulence
    use source
    use initialization
    implicit none

contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_air_density
! usage: calculate air density for each cell
! each layer
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_air_density(rhoa_c,Pa_c,qv_c,Temp_c)
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::Pa_c,qv_c,Temp_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::rhoa_c
    rhoa_c=Pa_c/R_prime/(1+0.608*qv_c)/Temp_c
end subroutine calculate_air_density

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_mass_weighed_P
! usage: calculate P_c P_bot
! each layer
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_mass_weighed_P(Pa_bot,P_bot,P_c)
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(in)::Pa_bot
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(inout)::P_bot
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::P_c
    integer::i,j,k
    do i=1,NLAT
        do j=1,NLONG
```

```
            P_bot(i,j,:)=(Pa_bot(i,j,:)/1000)**k_therm
            P_c(i,j,:)=1.0_dp/(1+k_therm)*(P_bot(i,j,1:NVERT)*Pa_bot(i,j,1:NVERT)-
Pa_bot(i,j,0:(NVERT-1))*Pa_bot(i,j,0:(NVERT-1)))
            P_c(i,j,:)=P_c(i,j,:)/(Pa_bot(i,j,1:NVERT)-Pa_bot(i,j,0:NVERT-1))
        enddo
    enddo
end subroutine calculate_mass_weighed_P

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_fluxface_column_pressure
! usage: calculate column pressure at each edge
! of each cell. The values will be used to calculate
! cell-centered column pressure
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

subroutine calculate_fluxface_column_pressure(pi_c_old,pi_uface,pi_vface)
    integer::i,j
    real(dp), dimension(NLAT,NLONG), intent(in)::pi_c_old
    real(dp), dimension(NLAT,NLONG+1), intent(inout):: pi_uface
    real(dp), dimension(NLAT+1,NLONG), intent(inout):: pi_vface

    ! calculate uface
    ! use central differencing
    do i=1,NLAT
        if(periodicBC==1) then
            pi_uface(i,1)=0.5*(pi_c_old(i,1)+pi_c_old(i,NLONG))
            pi_uface(i,NLONG+1)=0.5*(pi_c_old(i,1)+pi_c_old(i,NLONG))
        else
            pi_uface(i,1)=pi_c_old(i,1)
            pi_uface(i,NLONG+1)=pi_c_old(i,NLONG)
        endif
        pi_uface(i,2:NLONG)=0.5*(pi_c_old(i,1:(NLONG-1))+pi_c_old(i,2:NLONG))
    enddo

    ! calculate vface
    ! use central differencing
    do i=1,NLONG
        if(periodicBC==1) then
            pi_vface(1,i)=0.5*(pi_c_old(1,i)+pi_c_old(NLAT,i))
            pi_vface(NLAT+1,i)=0.5*(pi_c_old(1,i)+pi_c_old(NLAT,i))
        else
            pi_vface(1,i)=pi_c_old(1,i)
```

```
            pi_vface(NLAT+1,i)=pi_c_old(NLAT,i)
        endif
        pi_vface(2:NLAT,i)=0.5*(pi_c_old(1:(NLAT-1),NLONG)+pi_c_old(2:NLAT,NLONG))
    enddo
end subroutine calculate_fluxface_column_pressure

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_fluxsum
! usage:calculate fluxsum for each cell and each layer
! fluxsum(i,j,k) means the sum of flux of 4 edges
! from layer 1 to layer k at cell i,j
! and also calculate flux_uface and flux_vface
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine
calculate_fluxsum(fluxsum,flux_uface,flux_vface,u,v,pi_uface,pi_vface,lat_c,lat_vface,dsigma)
    integer:: k
    real(dp), dimension(NVERT),intent(in):: dsigma
    real(dp), dimension(NLAT,NLONG,NVERT),intent(inout):: fluxsum
    real(dp), dimension(NLAT,NLONG+1,NVERT), intent(in):: u
    real(dp), dimension(NLAT+1,NLONG,NVERT), intent(in):: v
    real(dp), dimension(NLAT,NLONG+1), intent(in):: pi_uface
    real(dp), dimension(NLAT+1,NLONG), intent(in):: pi_vface,lat_vface
    real(dp), dimension(NLAT,NLONG),intent(in):: lat_c
    real(dp), dimension(NLAT,NLONG+1,NVERT),intent(inout):: flux_uface
    real(dp), dimension(NLAT+1,NLONG,NVERT),intent(inout):: flux_vface

    ! reset all flux as zero
    fluxsum=0.0_dp
    flux_vface=0.0_dp
    flux_uface=0.0_dp

    ! set for the first layer 7.15 7.16
    do k=1,NVERT
        flux_uface(:,:,k)=u(:,:,k)*pi_uface*Re*dphi
        flux_vface(:,:,k)=v(:,:,k)*pi_vface*Re*dlamda_e*cos(lat_vface)
    enddo

    k=1
    fluxsum(:,:,k)=(flux_uface(:,2:(NLONG+1),k)-flux_uface(:,1:NLONG,k))*dsigma(k)
    fluxsum(:,:,k)=fluxsum(:,:,k)+(flux_vface(2:(NLAT+1),:,k)-flux_vface(1:NLAT,:,k))*dsigma(k)

    ! for other layers
```

```
    do k=2,NVERT
        fluxsum(:,:,k)=fluxsum(:,:,k-1)+(flux_uface(:,2:(NLONG+1),k)-
flux_uface(:,1:NLONG,k))*dsigma(k)
        fluxsum(:,:,k)=fluxsum(:,:,k)+(flux_vface(2:(NLAT+1),:,k)-
flux_vface(1:NLAT,:,k))*dsigma(k)
    enddo
    !print                                                                    *,
fluxsum(1,1,NVERT),flux_uface(1,1,1),flux_uface(1,2,1),flux_vface(1,1,1),flux_vface(2,1,1)
end subroutine calculate_fluxsum

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_column_pressure
! usage: update column pressure for each cell
! and call by regional model every time step
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_column_pressure(pi_c_old,pi_c_new,lat_c,fluxsum)
    integer:: i,j,k
    real(dp), dimension(NLAT,NLONG,NVERT),intent(in):: fluxsum
    real(dp), dimension(NLAT,NLONG),intent(inout):: pi_c_new
    real(dp), dimension(NLAT,NLONG),intent(in):: pi_c_old
    real(dp), dimension(NLAT,NLONG),intent(in):: lat_c

    ! calculate new column pressure
    pi_c_new=pi_c_old-dt/(Re**2)/dphi/dlamda_e/cos(lat_c)*fluxsum(:,:,NVERT)
end subroutine calculate_column_pressure

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_w_sigma
! usage: update w_sigma after each time step
! for each layer using continuity equation for
! each layer
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_w_sigma(w_sigma, pi_c_old,pi_c_new,lat_c,fluxsum,sigma_bot,dsigma)
    integer:: i,j,k
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(inout)::w_sigma
    real(dp),dimension(NLAT,NLONG),intent(in):: pi_c_old,pi_c_new,lat_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: fluxsum
    real(dp),dimension(NVERT),intent(in):: dsigma
    real(dp),dimension(0:NVERT),intent(in):: sigma_bot
```

```
    ! assume the top and bottom w_sigma=0
    w_sigma(:,:,0)=0.0_dp
    w_sigma(:,:,NVERT)=0.0_dp

    ! calculate interior layers boundaries 7.21
    do i=1,NLAT
        do j=1,NLONG
            w_sigma(i,j,1:NVERT)=-
1.0_dp/(pi_c_new(i,j)*Re*Re*cos(lat_c(i,j))*dlamda_e*dphi)*fluxsum(i,j,1:NVERT)
            w_sigma(i,j,1:NVERT)=w_sigma(i,j,1:NVERT)-sigma_bot(1:NVERT)*(pi_c_new(i,j)&
                -pi_c_old(i,j))/dt/pi_c_new(i,j)
        enddo
    enddo

end subroutine calculate_w_sigma

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_fluxface_PVT
! usage: calculate the potential virtual temperature
! at flux face
! Yun Zhang 05/01/2015
! @STANFORD
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_fluxface_PVT(PVT_c_old,PVT_uface,PVT_vface,PVT_bot,&
    P_c,P_bot,lat_uface,long_uface,lat_vface,long_vface,nt)
    integer::i,j,k
    integer,intent(in):: nt
    real(dp), dimension(NLAT,NLONG,NVERT), intent(in):: PVT_c_old
    real(dp), dimension(NLAT,NLONG+1,NVERT), intent(inout):: PVT_uface
    real(dp), dimension(NLAT+1,NLONG,NVERT), intent(inout):: PVT_vface
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(inout)::PVT_bot
    real(dp),dimension(NLAT+1,NLONG),intent(in):: lat_vface, long_vface
    real(dp),dimension(NLAT,NLONG+1),intent(in):: lat_uface, long_uface
    real(dp), dimension(NLAT,NLONG,NVERT),intent(in)::P_c
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in)::P_bot
    ! calculate uface
    ! use central differencing
    do i=1,NLAT
        do k=1,NVERT
            if(periodicBC==1) then
                PVT_uface(i,1,k)=0.5*(PVT_c_old(i,1,k)+PVT_c_old(i,NLONG,k))
                PVT_uface(i,NLONG+1,k)=0.5*(PVT_c_old(i,1,k)+PVT_c_old(i,NLONG,k))
            else
                PVT_uface(i,1,k)=PVT_c_old(i,1,k)
```

```
            PVT_uface(i,NLONG+1,k)=PVT_c_old(i,NLONG,k)
         endif
         PVT_uface(i,2:NLONG,k)=0.5*(PVT_c_old(i,1:(NLONG-
1),k)+PVT_c_old(i,2:NLONG,k))
      enddo
   enddo

   ! calculate vface
   ! use central differencing
   do i=1,NLONG
      do k=1,NVERT
         if(periodicBC==1) then
            PVT_vface(1,i,k)=0.5*(PVT_c_old(1,i,k)+PVT_c_old(NLAT,i,k))
            PVT_vface(NLAT+1,i,k)=0.5*(PVT_c_old(1,i,k)+PVT_c_old(NLAT,i,k))
         else
            PVT_vface(1,i,k)=PVT_c_old(1,i,k)
            PVT_vface(NLAT+1,i,k)=PVT_c_old(NLAT,i,k)
         endif
         PVT_vface(2:NLAT,i,k)=0.5*(PVT_c_old(1:(NLAT-1),i,k)+PVT_c_old(2:NLAT,i,k))
      enddo
   enddo

   ! set boundary PTV value
   if(PTVbound==1) then
      ! vface value
      do j=1,NLONG
         do k=1,NVERT

PVT_vface(1,j,k)=boundary_potential_virtual_temp(lat_vface(1,j),long_vface(1,j),k,nt*dt)

PVT_vface(NLAT+1,j,k)=boundary_potential_virtual_temp(lat_vface(NLAT+1,j),long_vface(NL
AT+1,j),k,nt*dt)
         enddo
      enddo
      ! uface value
      do i=1,NLAT
         do k=1,NVERT

PVT_uface(i,1,k)=boundary_potential_virtual_temp(lat_vface(i,1),long_uface(i,1),k,nt*dt)

PVT_uface(i,NLONG+1,k)=boundary_potential_virtual_temp(lat_uface(i,NLONG+1),long_uface
(i,NLONG+1),k,nt*dt)
         enddo
      enddo
```

```
    endif

    ! layer bottom eqn 7.11
    do i=1,NLAT
        do j=1,NLONG
            ! for zero 0, P_bot(i,j,0)=P_c(i,j,0)
            PVT_bot(i,j,0)=PVT_c_old(i,j,1)
            PVT_bot(i,j,1:(NVERT-1))=(P_bot(i,j,1:(NVERT-1))-P_c(i,j,1:(NVERT-
1)))*PVT_c_old(i,j,1:(NVERT-1))
            PVT_bot(i,j,1:(NVERT-1))=PVT_bot(i,j,1:(NVERT-1))+(P_c(i,j,2:NVERT)-
P_bot(i,j,1:(NVERT-1)))*PVT_c_old(i,j,2:NVERT)
            PVT_bot(i,j,1:(NVERT-1))=PVT_bot(i,j,1:(NVERT-1))/(P_c(i,j,2:NVERT)-
P_c(i,j,1:(NVERT-1)))
            PVT_bot(i,j,NVERT)=PVT_c_old(i,j,NVERT)
        enddo
    enddo

end subroutine calculate_fluxface_PVT

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_fluxface_qv
! usage: calculate specific humidity
! at flux face
! Yun Zhang 05/01/2015
! @STANFORD
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_fluxface_qv(qv_c_old,qv_uface,qv_vface,qv_bot,P_c,P_bot,&
    lat_uface,long_uface,lat_vface,long_vface,nt)
    integer::i,j,k
    integer,intent(in):: nt
    real(dp), dimension(NLAT,NLONG,NVERT), intent(in):: qv_c_old
    real(dp), dimension(NLAT,NLONG+1,NVERT), intent(inout):: qv_uface
    real(dp), dimension(NLAT+1,NLONG,NVERT), intent(inout):: qv_vface
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(inout)::qv_bot
    real(dp), dimension(NLAT+1,NLONG),intent(in):: lat_vface, long_vface
    real(dp), dimension(NLAT,NLONG+1),intent(in):: lat_uface, long_uface
    real(dp), dimension(NLAT,NLONG,NVERT),intent(in)::P_c
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in)::P_bot

    ! calculate uface
    ! use central differencing
    do i=1,NLAT
        do k=1,NVERT
            if(periodicBC==1) then
```

```
            qv_uface(i,1,k)=0.5*(qv_c_old(i,1,k)+qv_c_old(i,NLONG,k))
            qv_uface(i,NLONG+1,k)=0.5*(qv_c_old(i,1,k)+qv_c_old(i,NLONG,k))
         else
            qv_uface(i,1,k)=qv_c_old(i,1,k)
            qv_uface(i,NLONG+1,k)=qv_c_old(i,NLONG,k)
         endif
         qv_uface(i,2:NLONG,k)=0.5*(qv_c_old(i,1:(NLONG-1),k)+qv_c_old(i,2:NLONG,k))
      enddo
   enddo

   ! calculate vface
   ! use central differencing
   do i=1,NLONG
      do k=1,NVERT
         if(periodicBC==1) then
            qv_vface(1,i,k)=0.5*(qv_c_old(1,i,k)+qv_c_old(NLAT,i,k))
            qv_vface(NLAT+1,i,k)=0.5*(qv_c_old(1,i,k)+qv_c_old(NLAT,i,k))
         else
            qv_vface(1,i,k)=qv_c_old(1,i,k)
            qv_vface(NLAT+1,i,k)=qv_c_old(NLAT,i,k)
         endif
         qv_vface(2:NLAT,i,k)=0.5*(qv_c_old(1:(NLAT-1),i,k)+qv_c_old(2:NLAT,i,k))
      enddo
   enddo

   ! set boundary PTV value
   if(qvbound==1) then
      ! vface value
      do j=1,NLONG
         do k=1,NVERT
            qv_vface(1,j,k)=boundary_specific_humidity(lat_vface(1,j),long_vface(1,j),k,nt*dt)

qv_vface(NLAT+1,j,k)=boundary_specific_humidity(lat_vface(NLAT+1,j),long_vface(NLAT+1,j
),k,nt*dt)
         enddo
      enddo
      ! uface value
      do i=1,NLAT
         do k=1,NVERT
            qv_uface(i,1,k)=boundary_specific_humidity(lat_vface(i,1),long_uface(i,1),k,nt*dt)

qv_uface(i,NLONG+1,k)=boundary_specific_humidity(lat_uface(i,NLONG+1),long_uface(i,NL
ONG+1),k,nt*dt)
         enddo
```

```
        enddo
    endif

    ! layer bottom 7.25
    do i=1,NLAT
        do j=1,NLONG
            ! for zero 0, P_bot(i,j,0)=P_c(i,j,0)
            qv_bot(i,j,0)=qv_c_old(i,j,1)
            do k=1,(NVERT-1)
                if(qv_c_old(i,j,k)==qv_c_old(i,j,k+1)) then
                    qv_bot(i,j,k)=qv_c_old(i,j,k)
                else if(qv_c_old(i,j,k)==0 .or. qv_c_old(i,j,k+1)==0) then
                    qv_bot(i,j,k)=0.5*(qv_c_old(i,j,k)+qv_c_old(i,j,k+1))
                else
                    qv_bot(i,j,k)=(log(qv_c_old(i,j,k))-log(qv_c_old(i,j,k+1)))&
                        /(1/qv_c_old(i,j,k+1)-1/qv_c_old(i,j,k))
                endif
            enddo
            qv_bot(i,j,NVERT)=qv_c_old(i,j,NVERT)
        enddo
    enddo

end subroutine calculate_fluxface_qv

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_fluxface_gas
! usage: calculate gas concentration
! at flux face
! Yun Zhang 05/01/2015
! @STANFORD
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_fluxface_gas(gas_c_old,gas_uface,gas_vface,gas_bot,P_c,P_bot,&
    lat_uface,long_uface,lat_vface,long_vface,nt)
    integer::i,j,k
    integer,intent(in):: nt
    real(dp), dimension(NLAT,NLONG,NVERT), intent(in):: gas_c_old
    real(dp), dimension(NLAT,NLONG+1,NVERT), intent(inout):: gas_uface
    real(dp), dimension(NLAT+1,NLONG,NVERT), intent(inout):: gas_vface
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(inout)::gas_bot
    real(dp), dimension(NLAT+1,NLONG),intent(in):: lat_vface, long_vface
    real(dp), dimension(NLAT,NLONG+1),intent(in):: lat_uface, long_uface
    real(dp), dimension(NLAT,NLONG,NVERT),intent(in)::P_c
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in)::P_bot
```

```
    ! calculate uface
    ! use central differencing
    do i=1,NLAT
       do k=1,NVERT
          if(periodicBC==1) then
             gas_uface(i,1,k)=0.5*(gas_c_old(i,1,k)+gas_c_old(i,NLONG,k))
             gas_uface(i,NLONG+1,k)=0.5*(gas_c_old(i,1,k)+gas_c_old(i,NLONG,k))
          else
             gas_uface(i,1,k)=gas_c_old(i,1,k)
             gas_uface(i,NLONG+1,k)=gas_c_old(i,NLONG,k)
          endif
          gas_uface(i,2:NLONG,k)=0.5*(gas_c_old(i,1:(NLONG-1),k)+gas_c_old(i,2:NLONG,k))
       enddo
    enddo

    ! calculate vface
    ! use central differencing
    do i=1,NLONG
       do k=1,NVERT
          if(periodicBC==1) then
             gas_vface(1,i,k)=0.5*(gas_c_old(1,i,k)+gas_c_old(NLAT,i,k))
             gas_vface(NLAT+1,i,k)=0.5*(gas_c_old(1,i,k)+gas_c_old(NLAT,i,k))
          else
             gas_vface(1,i,k)=gas_c_old(1,i,k)
             gas_vface(NLAT+1,i,k)=gas_c_old(NLAT,i,k)
          endif
          gas_vface(2:NLAT,i,k)=0.5*(gas_c_old(1:(NLAT-1),i,k)+gas_c_old(2:NLAT,i,k))
       enddo
    enddo

    ! set boundary PTV value
    if(gasbound==1) then
       ! vface value
       do j=1,NLONG
          do k=1,NVERT
             gas_vface(1,j,k)=boundary_gas(lat_vface(1,j),long_vface(1,j),k,nt*dt)

 gas_vface(NLAT+1,j,k)=boundary_gas(lat_vface(NLAT+1,j),long_vface(NLAT+1,j),k,nt*dt)
          enddo
       enddo
       ! uface value
       do i=1,NLAT
          do k=1,NVERT
             gas_uface(i,1,k)=boundary_gas(lat_vface(i,1),long_uface(i,1),k,nt*dt)
```

```
gas_uface(i,NLONG+1,k)=boundary_gas(lat_uface(i,NLONG+1),long_uface(i,NLONG+1),k,nt*
dt)
            enddo
        enddo
    endif

    ! layer bottom 7.25
    do i=1,NLAT
        do j=1,NLONG
            gas_bot(i,j,0)=gas_c_old(i,j,1)
            gas_bot(i,j,NVERT)=gas_c_old(i,j,NVERT)
            do k=1,(NVERT-1)
                gas_bot(i,j,k)=0.5*(gas_c_old(i,j,k)+gas_c_old(i,j,k+1))
            enddo
        enddo
    enddo

end subroutine calculate_fluxface_gas


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_heat_source
! usage: calculate heat source value for each time
! step for each cell at each layer
! the heat source is treated as explicit
! and defined in the heat_source function
! in source.f90
! Yun Zhang 05/01/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_heat_source(Pa_c,Q,lat_c,long_c,nt)
    real(dp),dimension(NLAT,NLONG),intent(in)::lat_c,long_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout):: Q
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: Pa_c
    integer, intent(in):: nt
    integer:: i,j,k
    do i=1,NLAT
        do j=1,NLONG
            do k=1,NVERT
                Q(i,j,k)=heat_source(lat_c(i,j),long_c(i,j),k,nt*dt)
                Q(i,j,k)=((1000/Pa_c(i,j,k))**k_therm)/Cp_d*Q(i,j,k)
            enddo
        enddo
```

```
    enddo
end subroutine calculate_heat_source


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_gas_source
! usage: calculate gas source value for each time
! step for each cell at each layer
! the heat source is treated as explicit
! and defined in the heat_source function
! in source.f90
! Yun Zhang 05/01/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_gas_source(Pa_c,Q,lat_c,long_c,nt)
    real(dp),dimension(NLAT,NLONG),intent(in)::lat_c,long_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout):: Q
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: Pa_c
    integer, intent(in):: nt
    integer:: i,j,k
    do i=1,NLAT
        do j=1,NLONG
            do k=1,NVERT
                Q(i,j,k)=gas_source(lat_c(i,j),long_c(i,j),k,nt*dt)
            enddo
        enddo
    enddo
end subroutine calculate_gas_source

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_gas_source
! usage: calculate gas source value for each time
! step for each cell at each layer
! the heat source is treated as explicit
! and defined in the heat_source function
! in source.f90
! Yun Zhang 05/01/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_qv_source(Pa_c,Q,lat_c,long_c,nt)
    real(dp),dimension(NLAT,NLONG),intent(in)::lat_c,long_c
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout):: Q
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: Pa_c
    integer, intent(in):: nt
```

```
    integer:: i,j,k
    do i=1,NLAT
        do j=1,NLONG
            do k=1,NVERT
                Q(i,j,k)=qv_source(lat_c(i,j),long_c(i,j),k,nt*dt)
            enddo
        enddo
    enddo
end subroutine calculate_qv_source

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_scalar_field
! usage: update scalar transport for each cell
! at every time step
! eqn 7.24 7.27
! use for PTV, qv and other components
! code use PTV as example
! Yun Zhang    04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_scalar_field(PVT_c_old, PVT_c_new, PVT_bot, &
    pi_c_old,pi_c_new,PVT_uface, PVT_vface,flux_uface,flux_vface,lat_c, rhoa_c,&
    dsigma, w_sigma_new,K_t,Pa_c,Q)
    real(dp),dimension(NLAT,NLONG,NVERT),intent(in):: PVT_c_old, K_t, Q
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout):: PVT_c_new
    real(dp),dimension(NLAT,NLONG),intent(in)::pi_c_old, pi_c_new, lat_c
    real(dp), dimension(NLAT+1,NLONG,NVERT),intent(in):: PVT_vface, flux_vface
    real(dp), dimension(NLAT,NLONG+1,NVERT),intent(in):: PVT_uface, flux_uface
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in):: w_sigma_new
    real(dp), dimension(NVERT),intent(in):: dsigma
    real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in):: PVT_bot
    real(dp), dimension(NLAT,NLONG,NVERT),intent(in):: Pa_c, rhoa_c
    integer:: i,j,k

    ! old value
    do k=1,NVERT
        PVT_c_new(:,:,k)=pi_c_old*PVT_c_old(:,:,k)/pi_c_new
    enddo
    ! horizontal flux at boundary
    do k=1,NVERT
        PVT_c_new(:,:,k)=PVT_c_new(:,:,k)+dt/(pi_c_new*Re*Re*cos(lat_c)*dlamda_e*dphi)&
        *(flux_uface(:,1:NLONG,k)*PVT_uface(:,1:NLONG,k)-
flux_uface(:,2:(NLONG+1),k)*PVT_uface(:,2:(NLONG+1),k)&
            +flux_vface(1:NLAT,:,k)*PVT_vface(1:NLAT,:,k)-
```

```
flux_vface(2:(NLAT+1),:,k)*PVT_vface(2:(NLAT+1),:,k))
   enddo

   ! vertical flux
   do k=1,NVERT
      PVT_c_new(:,:,k)=PVT_c_new(:,:,k)+dt/dsigma(k)*(w_sigma_new(:,:,k-1)*PVT_bot(:,:,k-
1)&
         -w_sigma_new(:,:,k)*PVT_bot(:,:,k))
   enddo

   ! turbulence vertical flux
   ! not finished
   if(turbmodel==1) then
      do k=1,NVERT
         PVT_c_new(:,:,k)=PVT_c_new(:,:,k)
      enddo
   endif

   ! heat source
   if(heatsource==1) then
      do k=1,NVERT
         PVT_c_new(:,:,k)=PVT_c_new(:,:,k)+dt*pi_c_old/pi_c_new*Q(:,:,k)
      enddo
   endif

end subroutine calculate_scalar_field

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_geopotential
! usage: calculate geopotential for each cell center
! and layer bottom
! need to specify the boundary geopotential at bottom
! use boundary_surf_geopotential
! Yun 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine
calculate_geopotential(geopot_bot,geopot_c,geopot_c_tminus1,P_c,P_bot,PVT_c_old,lat_c,long_
c,nt)
   real(dp), dimension(NLAT,NLONG,NVERT),intent(in)::P_c,PVT_c_old
   real(dp), dimension(NLAT,NLONG,NVERT),intent(inout)::geopot_c,geopot_c_tminus1
   real(dp), dimension(NLAT,NLONG,0:NVERT),intent(inout):: geopot_bot
   real(dp), dimension(NLAT,NLONG,0:NVERT),intent(in):: P_bot
   real(dp), dimension(NLAT,NLONG), intent(in):: lat_c,long_c
```

```
    integer,intent(in):: nt
    integer::i,j,k

    if(nt>0) geopot_c_tminus1=geopot_c
    do i=1,NLAT
        do j=1,NLONG
            geopot_bot(i,j,NVERT)=boundary_surf_geopotential(lat_c(i,j),long_c(i,j),nt*dt)
            geopot_c(i,j,NVERT)=geopot_bot(i,j,NVERT)-
Cp_d*(PVT_c_old(i,j,NVERT)*(P_c(i,j,NVERT)-P_bot(i,j,NVERT)))
            do k=NVERT-1,0,-1
                geopot_bot(i,j,k)=geopot_c(i,j,k+1)-Cp_d*(PVT_c_old(i,j,k+1)*(P_bot(i,j,k)-
P_c(i,j,k+1)))
                if(k/=0) then
                    geopot_c(i,j,k)=geopot_bot(i,j,k)-Cp_d*(PVT_c_old(i,j,k)*(P_c(i,j,k)-P_bot(i,j,k)))
                endif
            enddo
        enddo
    enddo
    if(nt==0) geopot_c_tminus1=geopot_c

end subroutine calculate_geopotential

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_velocity_field
! usage: update velocity field for each flux face
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_velocity_field(pi_c_old,pi_c_new,pi_c_tmp,pi_c_tminus1,lat_c,lat_vface,&
    u_old,u_new,u_tmp,v_old,v_new,v_tmp,flux_uface,flux_vface,dsigma,&
    sigma_bot,w_sigma_new,f_c,geopot_c,geopot_c_tminus1,&
    PVT_c_old,P_bot,P_c,nu_t,rhoa_c)

real(dp),dimension(NLAT,NLONG),intent(in)::pi_c_old,pi_c_new,pi_c_tmp,pi_c_tminus1,lat_c,f
_c
    real(dp),dimension(NLAT,NLONG+1,NVERT),intent(in)::u_old,flux_uface,u_tmp
    real(dp),dimension(NLAT,NLONG+1,NVERT),intent(inout)::u_new
    real(dp),dimension(NLAT+1,NLONG,NVERT),intent(in)::v_old,flux_vface,v_tmp
    real(dp),dimension(NLAT+1,NLONG,NVERT),intent(inout)::v_new
    real(dp),dimension(0:NVERT),intent(in)::sigma_bot
    real(dp),dimension(NVERT),intent(in)::dsigma
    real(dp),dimension(NLAT,NLONG,0:NVERT),intent(in)::w_sigma_new,P_bot

real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::geopot_c,geopot_c_tminus1,PVT_c_old,P
```

```
_c,nu_t,rhoa_c
    real(dp),dimension(NLAT+1,NLONG),intent(in)::lat_vface
    real(dp),dimension(NLAT,NLONG+1,0:NVERT)::u_bot_old
    real(dp),dimension(NLAT+1,NLONG,0:NVERT)::v_bot_old

    ! u field
    real(dp),dimension(NLAT,NLONG+1)::A_old,A_new
    real(dp),dimension(NLAT,NLONG+1,0:NVERT)::F
    real(dp),dimension(NLAT,0:NLONG+1,NVERT)::B
    real(dp),dimension(NLAT+1,NLONG+1,NVERT)::C
    real(dp),dimension(NLAT+1,0:NLONG+1,NVERT)::D,E

    ! v field
    real(dp),dimension(NLAT+1,NLONG)::P_old,P_new
    real(dp),dimension(NLAT+1,NLONG,0:NVERT)::O
    real(dp),dimension(0:NLAT+1,NLONG,NVERT)::R
    real(dp),dimension(NLAT+1,NLONG+1,NVERT)::Q
    real(dp),dimension(0:NLAT+1,NLONG+1,NVERT)::S,T

    integer::i,j,k

    ! prepare for the boundary values
    ! use new virtual matrix to calculate velocity to simplify equation
    real(dp),dimension(0:NLAT+1,0:NLONG+1)::v_f_c,v_pi_c_new,v_pi_c_old,v_lat_c
    real(dp),dimension(0:NLAT+1,0:NLONG+1,0:NVERT)::v_F_new
    real(dp),dimension(0:NLAT+1,0:NLONG+1,0:NVERT)::v_w_sigma_new
    !real(dp),dimension(0:NLAT+1,0:NLONG+1,NVERT):: v_PVT_c_old, v_P_c, v_geopot_c
    !real(dp),dimension(0:NLAT+1,0:NLONG+1,0:NVERT):: v_P_bot
    real(dp),dimension(0:NLAT+1,0:NLONG+2,NVERT)::v_u_old, v_flux_uface
    real(dp),dimension(0:NLAT+2,0:NLONG+1,NVERT)::v_v_old, v_flux_vface

    v_f_c(1:NLAT,1:NLONG)=f_c
    v_pi_c_old(1:NLAT,1:NLONG)=pi_c_old
    v_pi_c_new(1:NLAT,1:NLONG)=pi_c_new
    v_lat_c(1:NLAT,1:NLONG)=lat_c
    v_w_sigma_new(1:NLAT,1:NLONG,0:NVERT)=w_sigma_new

    if(periodicBC==1) then
        v_pi_c_old(0,1:NLONG)=pi_c_old(NLAT,:)
        v_pi_c_old(NLAT+1,1:NLONG)=pi_c_old(1,:)
        v_pi_c_old(1:NLAT,0)=pi_c_old(:,NLONG)
        v_pi_c_old(1:NLAT,NLONG+1)=pi_c_old(:,1)
        v_pi_c_old(0,0)=0.5*(pi_c_old(1,NLONG)+pi_c_old(NLAT,1))
        v_pi_c_old(NLAT,0)=0.5*(pi_c_old(1,NLONG)+pi_c_old(1,1))
```

```
    v_pi_c_old(0,NLONG+1)=0.5*(pi_c_old(1,1)+pi_c_old(NLAT,NLONG))
    v_pi_c_old(NLAT,NLONG+1)=0.5*(pi_c_old(1,NLONG)+pi_c_old(NLAT,1))
else
    v_pi_c_old(0,1:NLONG)=pi_c_old(1,:)
    v_pi_c_old(NLAT+1,1:NLONG)=pi_c_old(NLAT,:)
    v_pi_c_old(:,0)=v_pi_c_old(:,1)
    v_pi_c_old(:,NLONG+1)=v_pi_c_old(:,NLONG)
endif

if(periodicBC==1) then
    v_pi_c_new(0,1:NLONG)=pi_c_new(NLAT,:)
    v_pi_c_new(NLAT+1,1:NLONG)=pi_c_new(1,:)
    v_pi_c_new(1:NLAT,0)=pi_c_new(:,NLONG)
    v_pi_c_new(1:NLAT,NLONG+1)=pi_c_new(:,1)
    v_pi_c_new(0,0)=0.5*(pi_c_new(1,NLONG)+pi_c_new(NLAT,1))
    v_pi_c_new(NLAT,0)=0.5*(pi_c_new(1,NLONG)+pi_c_new(1,1))
    v_pi_c_new(0,NLONG+1)=0.5*(pi_c_new(1,1)+pi_c_new(NLAT,NLONG))
    v_pi_c_new(NLAT,NLONG+1)=0.5*(pi_c_new(1,NLONG)+pi_c_new(NLAT,1))
else
    v_pi_c_new(0,1:NLONG)=pi_c_new(1,:)
    v_pi_c_new(NLAT+1,1:NLONG)=pi_c_new(NLAT,:)
    v_pi_c_new(:,0)=v_pi_c_new(:,1)
    v_pi_c_new(:,NLONG+1)=v_pi_c_new(:,NLONG)
endif

v_lat_c(0,:)=lat_c(1,1)-dphi
v_lat_c(NLAT+1,:)=lat_c(NLAT,1)+dphi
v_lat_c(1:NLAT,0)=lat_c(1:NLAT,1)
v_lat_c(1:NLAT,NLONG+1)=lat_c(1:NLAT,NLONG)

if(coriolis==1) then
    v_f_c=2*Omega*sin(v_lat_c)
else
    v_f_c=0.0_dp
endif


if(periodicBC==1) then
    v_w_sigma_new(0,1:NLONG,:)=w_sigma_new(NLAT,:,:)
    v_w_sigma_new(NLAT+1,1:NLONG,:)=w_sigma_new(1,:,:)
    v_w_sigma_new(1:NLAT,0,:)=w_sigma_new(:,NLONG,:)
    v_w_sigma_new(1:NLAT,NLONG+1,:)=w_sigma_new(:,1,:)
    v_w_sigma_new(0,0,:)=0.5*(w_sigma_new(1,NLONG,:)+w_sigma_new(NLAT,1,:))
    v_w_sigma_new(NLAT,0,:)=0.5*(w_sigma_new(1,NLONG,:)+w_sigma_new(1,1,:))
```

```
v_w_sigma_new(0,NLONG+1,:)=0.5*(w_sigma_new(1,1,:)+w_sigma_new(NLAT,NLONG,:))

v_w_sigma_new(NLAT,NLONG+1,:)=0.5*(w_sigma_new(1,NLONG,:)+w_sigma_new(NLAT,1,
:))
  else
    v_w_sigma_new(0,1:NLONG,:)=w_sigma_new(1,:,:)
    v_w_sigma_new(NLAT+1,1:NLONG,:)=w_sigma_new(NLAT,:,:)
    v_w_sigma_new(:,0,:)=v_w_sigma_new(:,1,:)
    v_w_sigma_new(:,NLONG+1,:)=v_w_sigma_new(:,NLONG,:)
  endif

  v_u_old(1:NLAT,1:NLONG+1,:)=u_tmp
  v_flux_uface(1:NLAT,1:NLONG+1,:)=flux_uface
  v_v_old(1:NLAT+1,1:NLONG,:)=v_tmp
  v_flux_vface(1:NLAT+1,1:NLONG,:)=flux_vface

  v_u_old(0,1:NLONG+1,:)=u_tmp(1,:,:)
  v_u_old(NLAT+1,1:NLONG+1,:)=u_tmp(NLAT,:,:)
  v_u_old(:,0,:)=v_u_old(:,1,:)
  v_u_old(:,NLONG+2,:)=v_u_old(:,NLONG+1,:)

  v_flux_uface(0,1:NLONG+1,:)=flux_uface(1,:,:)
  v_flux_uface(NLAT+1,1:NLONG+1,:)=flux_uface(NLAT,:,:)
  v_flux_uface(:,0,:)=v_flux_uface(:,1,:)
  v_flux_uface(:,NLONG+2,:)=v_flux_uface(:,NLONG+1,:)

  v_v_old(1:NLAT+1,0,:)=v_tmp(:,1,:)
  v_v_old(1:NLAT+1,NLONG+1,:)=v_tmp(:,NLONG,:)
  v_v_old(0,:,:)=v_v_old(1,:,:)
  v_v_old(NLAT+2,:,:)=v_v_old(NLAT+1,:,:)

  v_flux_vface(1:NLAT+1,0,:)=flux_vface(:,1,:)
  v_flux_vface(1:NLAT+1,NLONG+1,:)=flux_vface(:,NLONG,:)
  v_flux_vface(0,:,:)=v_flux_vface(1,:,:)
  v_flux_vface(NLAT+2,:,:)=v_flux_vface(NLAT+1,:,:)

  ! Update u field
  ! calculate column pressure multiplied by grid-cell area 7.38
  ! interior points

A_old=1.0_dp/8.0_dp*Re*Re*dphi*dlamda_e*(v_pi_c_old(2:NLAT+1,0:NLONG)*cos(v_lat_c(
2:NLAT+1,0:NLONG))&
```

```
+v_pi_c_old(2:NLAT+1,1:NLONG+1)*cos(v_lat_c(2:NLAT+1,1:NLONG+1))+2*v_pi_c_old(1:
NLAT,0:NLONG)*cos(v_lat_c(1:NLAT,0:NLONG))&

+2*v_pi_c_old(1:NLAT,1:NLONG+1)*cos(v_lat_c(1:NLAT,1:NLONG+1))+v_pi_c_old(0:NLAT-
1,0:NLONG)*cos(v_lat_c(0:NLAT-1,0:NLONG))&
        +v_pi_c_old(0:NLAT-1,1:NLONG+1)*cos(v_lat_c(0:NLAT-1,1:NLONG+1)))


A_new=1.0_dp/8.0_dp*Re*Re*dphi*dlamda_e*(v_pi_c_new(2:NLAT+1,0:NLONG)*cos(v_lat_c
(2:NLAT+1,0:NLONG))&
        +v_pi_c_new(2:NLAT+1,1:NLONG+1)*cos(v_lat_c(2:NLAT+1,1:NLONG+1))&
        +2*v_pi_c_new(1:NLAT,0:NLONG)*cos(v_lat_c(1:NLAT,0:NLONG))&
        +2*v_pi_c_new(1:NLAT,1:NLONG+1)*cos(v_lat_c(1:NLAT,1:NLONG+1))&
        +v_pi_c_new(0:NLAT-1,0:NLONG)*cos(v_lat_c(0:NLAT-1,0:NLONG))&
        +v_pi_c_new(0:NLAT-1,1:NLONG+1)*cos(v_lat_c(0:NLAT-1,1:NLONG+1)))

    ! equation 7.41
    B=1.0_dp/12.0_dp*(v_flux_uface(0:NLAT-1,0:NLONG+1,:)+v_flux_uface(0:NLAT-
1,1:NLONG+2,:)&
        +2*v_flux_uface(1:NLAT,0:NLONG+1,:)+2*v_flux_uface(1:NLAT,1:NLONG+2,:)&
        +v_flux_uface(2:NLAT+1,0:NLONG+1,:)+v_flux_uface(2:NLAT+1,1:NLONG+2,:))

    ! equation 7.42

C=1.0_dp/12.0_dp*(v_flux_vface(0:NLAT,0:NLONG,:)+v_flux_vface(0:NLAT,1:NLONG+1,:)&
        +2*v_flux_vface(1:NLAT+1,0:NLONG,:)+2*v_flux_vface(1:NLAT+1,1:NLONG+1,:)&
        +v_flux_vface(2:NLAT+2,0:NLONG,:)+v_flux_vface(2:NLAT+2,1:NLONG+1,:))

    ! equation 7.43

D=1.0_dp/24.0_dp*(v_flux_vface(0:NLAT,0:NLONG+1,:)+2*v_flux_vface(1:NLAT+1,0:NLON
G+1,:)+v_flux_vface(2:NLAT+2,0:NLONG+1,:)&

+v_flux_uface(0:NLAT,0:NLONG+1,:)+v_flux_uface(1:NLAT+1,0:NLONG+1,:)+v_flux_uface(
0:NLAT,1:NLONG+2,:)&
        +v_flux_uface(1:NLAT+1,1:NLONG+2,:))

    ! equation 7.44

E=1.0_dp/24.0_dp*(v_flux_vface(0:NLAT,0:NLONG+1,:)+2*v_flux_vface(1:NLAT+1,0:NLON
G+1,:)+v_flux_vface(2:NLAT+2,0:NLONG+1,:)&
        -v_flux_uface(0:NLAT,0:NLONG+1,:)-v_flux_uface(1:NLAT+1,0:NLONG+1,:)-
v_flux_uface(0:NLAT,1:NLONG+2,:)&
        -v_flux_uface(1:NLAT+1,1:NLONG+2,:))
```

```
    ! equation 7.45 get velocity at layer bottom
    ! the top most and bottom most u_bot are not important because w_sigma is zero
    do i=1,NLAT
        do j=1,NLONG+1
            u_bot_old(i,j,1:NVERT-1)=(u_tmp(i,j,1:NVERT-1)*dsigma(1:NVERT-
1)+u_tmp(i,j,2:NVERT)*dsigma(2:NVERT))&
            /(dsigma(1:NVERT-1)+dsigma(2:NVERT))
            u_bot_old(i,j,0)=u_tmp(i,j,1)
            u_bot_old(i,j,NVERT)=u_tmp(i,j,NVERT)
        enddo
    enddo

    ! equation 7.40
    do i=0,NLAT+1
        do j=0,NLONG+1

v_F_new(i,j,:)=v_pi_c_new(i,j)*Re*Re*dphi*dlamda_e*cos(v_lat_c(i,j))*v_w_sigma_new(i,j,:)
        enddo
    enddo

    F=1.0_dp/8.0_dp*(v_F_new(2:NLAT+1,0:NLONG,:)+v_F_new(2:NLAT+1,1:NLONG+1,:)&
        +2*v_F_new(1:NLAT,0:NLONG,:)+2*v_F_new(1:NLAT,1:NLONG+1,:)&
        +v_F_new(0:NLAT-1,0:NLONG,:)+v_F_new(0:NLAT-1,1:NLONG+1,:))

    ! equation 7.32 time difference term
    do k=1,NVERT
        u_new(:,:,k)=A_old/A_new*u_old(:,:,k)
    enddo

    ! equation 7.33 horizontal advection B
    do k=1,NVERT
        u_new(:,:,k)=u_new(:,:,k)+dt/A_new*(B(:,0:NLONG,k)&
            *0.5*(v_u_old(1:NLAT,0:NLONG,k)+v_u_old(1:NLAT,1:NLONG+1,k))&
            -
B(:,1:NLONG+1,k)*0.5*(v_u_old(1:NLAT,1:NLONG+1,k)+v_u_old(1:NLAT,2:NLONG+2,k)))
    enddo

    ! equation 7.33 horizontal advection C
    do k=1,NVERT
        u_new(:,:,k)=u_new(:,:,k)+dt/A_new*(C(1:NLAT,1:NLONG+1,k)&
            *0.5*(v_u_old(0:NLAT-1,1:NLONG+1,k)+v_u_old(1:NLAT,1:NLONG+1,k))&
            -
C(2:NLAT+1,1:NLONG+1,k)*0.5*(v_u_old(1:NLAT,1:NLONG+1,k)+v_u_old(2:NLAT+1,1:NL
```

```
ONG+1,k)))
   enddo

   ! equation 7.33 horizontal advection D
   do k=1,NVERT
      u_new(:,:,k)=u_new(:,:,k)+dt/A_new*(D(1:NLAT,0:NLONG,k)&
         *0.5*(v_u_old(0:NLAT-1,0:NLONG,k)+v_u_old(1:NLAT,1:NLONG+1,k))&
         -
D(2:NLAT+1,1:NLONG+1,k)*0.5*(v_u_old(1:NLAT,1:NLONG+1,k)+v_u_old(2:NLAT+1,2:NL
ONG+2,k)))
   enddo

   ! equation 7.33 horizontal advection E
   do k=1,NVERT
      u_new(:,:,k)=u_new(:,:,k)+dt/A_new*(E(1:NLAT,1:NLONG+1,k)&
         *0.5*(v_u_old(0:NLAT-1,2:NLONG+2,k)+v_u_old(1:NLAT,1:NLONG+1,k))&
         -
E(2:NLAT+1,0:NLONG,k)*0.5*(v_u_old(1:NLAT,1:NLONG+1,k)+v_u_old(2:NLAT+1,0:NLON
G,k)))
   enddo

   ! equation 7.34 vertical transport F
   do k=1,NVERT
      u_new(:,:,k)=u_new(:,:,k)+dt/A_new/dsigma(k)*(F(:,:,k-1)*u_bot_old(:,:,k-1)&
         -F(:,:,k)*u_bot_old(:,:,k))
   enddo

   ! equation 7.35 coriolis and sperical grid conversion
   do k=1,NVERT
      u_new(:,:,k)=u_new(:,:,k)+0.5*dt/A_new*Re*dlamda_e*dphi&

*(v_pi_c_old(1:NLAT,0:NLONG)*0.5*(v_v_old(1:NLAT,0:NLONG,k)+v_v_old(2:NLAT+1,0:N
LONG,k))&
         *(v_f_c(1:NLAT,0:NLONG)*Re*cos(v_lat_c(1:NLAT,0:NLONG))&

+0.5*(v_u_old(1:NLAT,0:NLONG,k)+v_u_old(1:NLAT,1:NLONG+1,k))*sin(v_lat_c(1:NLAT,0:
NLONG)))&

+v_pi_c_old(1:NLAT,1:NLONG+1)*0.5*(v_v_old(1:NLAT,1:NLONG+1,k)+v_v_old(2:NLAT+1,
1:NLONG+1,k))&
         *(v_f_c(1:NLAT,1:NLONG+1)*Re*cos(v_lat_c(1:NLAT,1:NLONG+1))&

+0.5*(v_u_old(1:NLAT,1:NLONG+1,k)+v_u_old(1:NLAT,2:NLONG+2,k))*sin(v_lat_c(1:NLAT,
1:NLONG+1))))
```

```
    enddo

    ! equation 7.36 pressure gradient
    do k=1,NVERT
        u_new(:,2:NLONG,k)=u_new(:,2:NLONG,k)-
dt/A_new(:,2:NLONG)*Re*dphi*((geopot_c(:,2:NLONG,k)&
            -geopot_c(:,1:NLONG-1,k))*0.5*(pi_c_tmp(:,1:NLONG-1)+pi_c_tmp(:,2:NLONG))&
            +0.5*(-pi_c_tmp(:,1:NLONG-1)+pi_c_tmp(:,2:NLONG))*Cp_d&
            *(PVT_c_old(:,1:NLONG-1,k)/dsigma(k)*(sigma_bot(k)&
            *(P_bot(:,1:NLONG-1,k)-P_c(:,1:NLONG-1,k))+sigma_bot(k-1)&
            *(P_c(:,1:NLONG-1,k)-P_bot(:,1:NLONG-1,k-
1)))+PVT_c_old(:,2:NLONG,k)/dsigma(k)&
            *(sigma_bot(k)*(P_bot(:,2:NLONG,k)-P_c(:,2:NLONG,k))&
            +sigma_bot(k-1)*(P_c(:,2:NLONG,k)-P_bot(:,2:NLONG,k-1)))))

        ! west boundary
        u_new(:,1,k)=u_new(:,1,k)-dt/A_new(:,1)*Re*dphi*((-
geopot_c_tminus1(:,1,k)+geopot_c(:,1,k))*pi_c_tmp(:,1)&
            +(-
pi_c_tminus1(:,1)+pi_c_tmp(:,1))*Cp_d*(PVT_c_old(:,1,k)/dsigma(k)*(sigma_bot(k)*(P_bot(:,1,
k)&
            -P_c(:,1,k))+sigma_bot(k-1)*(P_c(:,1,k)-P_bot(:,1,k-1)))))

        ! east boundary
        if(periodicBC==1) then
            u_new(:,NLONG+1,k)=u_new(:,1,k);
        else
            u_new(:,NLONG+1,k)=u_new(:,NLONG+1,k)-
dt/A_new(:,NLONG+1)*Re*dphi*((geopot_c_tminus1(:,NLONG,k)&
                -geopot_c(:,NLONG,k))*pi_c_tmp(:,NLONG)&
                +(pi_c_tminus1(:,NLONG)-
pi_c_tmp(:,NLONG))*Cp_d*(PVT_c_old(:,NLONG,k)/dsigma(k)*(sigma_bot(k)*(P_bot(:,NLO
NG,k)&
                -P_c(:,NLONG,k))+sigma_bot(k-1)*(P_c(:,NLONG,k)-P_bot(:,NLONG,k-1)))))
        endif
    enddo

    ! equation 7.37 eddy vicousity not finished!
    if(turbmodel==1) then
        do k=1,NVERT
            u_new(:,:,k)=u_new(:,:,k)
        enddo
    endif
```

```
    ! Update v field
    ! column pressure multiplied by the area eqn 7.53

P_old=1.0_dp/8.0_dp*Re*Re*dphi*dlamda_e*(v_pi_c_old(0:NLAT,2:NLONG+1)*cos(v_lat_c(0
:NLAT,2:NLONG+1))&
        +v_pi_c_old(1:NLAT+1,2:NLONG+1)*cos(v_lat_c(1:NLAT+1,2:NLONG+1))&
        +2*v_pi_c_old(0:NLAT,1:NLONG)*cos(v_lat_c(0:NLAT,1:NLONG))&
        +2*v_pi_c_old(1:NLAT+1,1:NLONG)*cos(v_lat_c(1:NLAT+1,1:NLONG))&
        +v_pi_c_old(0:NLAT,0:NLONG-1)*cos(v_lat_c(0:NLAT,0:NLONG-1))&
        +v_pi_c_old(1:NLAT+1,0:NLONG-1)*cos(v_lat_c(1:NLAT+1,0:NLONG-1)))


P_new=1.0_dp/8.0_dp*Re*Re*dphi*dlamda_e*(v_pi_c_new(0:NLAT,2:NLONG+1)*cos(v_lat_c
(0:NLAT,2:NLONG+1))&
        +v_pi_c_new(1:NLAT+1,2:NLONG+1)*cos(v_lat_c(1:NLAT+1,2:NLONG+1))&
        +2*v_pi_c_new(0:NLAT,1:NLONG)*cos(v_lat_c(0:NLAT,1:NLONG))&
        +2*v_pi_c_new(1:NLAT+1,1:NLONG)*cos(v_lat_c(1:NLAT+1,1:NLONG))&
        +v_pi_c_new(0:NLAT,0:NLONG-1)*cos(v_lat_c(0:NLAT,0:NLONG-1))&
        +v_pi_c_new(1:NLAT+1,0:NLONG-1)*cos(v_lat_c(1:NLAT+1,0:NLONG-1)))

     ! equation 7.56
    R=1.0_dp/12.0_dp*(v_flux_vface(0:NLAT+1,0:NLONG-
1,:)+v_flux_vface(1:NLAT+2,0:NLONG-1,:)&
        +2*v_flux_vface(0:NLAT+1,1:NLONG,:)+2*v_flux_vface(1:NLAT+2,1:NLONG,:)&
        +v_flux_vface(0:NLAT+1,2:NLONG+1,:)+v_flux_vface(1:NLAT+2,2:NLONG+1,:))

    ! equation 7.55

Q=1.0_dp/12.0_dp*(v_flux_uface(0:NLAT,0:NLONG,:)+v_flux_uface(1:NLAT+1,0:NLONG,:)&
        +2*v_flux_uface(0:NLAT,1:NLONG+1,:)+2*v_flux_uface(1:NLAT+1,1:NLONG+1,:)&
        +v_flux_uface(0:NLAT,2:NLONG+2,:)+v_flux_uface(1:NLAT+1,2:NLONG+2,:))

    ! equation 7.57

S=1.0_dp/24.0_dp*(v_flux_uface(0:NLAT+1,0:NLONG,:)+2*v_flux_uface(0:NLAT+1,1:NLON
G+1,:)+v_flux_uface(0:NLAT+1,2:NLONG+2,:)&

+v_flux_vface(0:NLAT+1,0:NLONG,:)+v_flux_vface(0:NLAT+1,1:NLONG+1,:)+v_flux_vface(
1:NLAT+2,0:NLONG,:)&
        +v_flux_vface(1:NLAT+2,1:NLONG+1,:))

    ! equation 7.58
    T=1.0_dp/24.0_dp*(-v_flux_uface(0:NLAT+1,0:NLONG,:)-
2*v_flux_uface(0:NLAT+1,1:NLONG+1,:)-v_flux_uface(0:NLAT+1,2:NLONG+2,:)&
```

```
+v_flux_vface(0:NLAT+1,0:NLONG,:)+v_flux_vface(0:NLAT+1,1:NLONG+1,:)+v_flux_vface(
1:NLAT+2,0:NLONG,:)&
        +v_flux_vface(1:NLAT+2,1:NLONG+1,:))

    ! equation 7.54
    O=1.0_dp/8.0_dp*(v_F_new(0:NLAT,2:NLONG+1,:)+v_F_new(1:NLAT+1,2:NLONG+1,:)&
        +2*v_F_new(0:NLAT,1:NLONG,:)+2*v_F_new(1:NLAT+1,1:NLONG,:)&
        +v_F_new(0:NLAT,0:NLONG-1,:)+v_F_new(1:NLAT+1,0:NLONG-1,:))

    ! equation 7.45 get velocity at layer bottom
    ! the top most and bottom most u_bot are not important because w_sigma is zero
    do i=1,NLAT+1
        do j=1,NLONG
            v_bot_old(i,j,1:NVERT-1)=(v_tmp(i,j,1:NVERT-1)*dsigma(1:NVERT-
1)+v_tmp(i,j,2:NVERT)*dsigma(2:NVERT))&
            /(dsigma(1:NVERT-1)+dsigma(2:NVERT))
            v_bot_old(i,j,0)=v_tmp(i,j,1)
            v_bot_old(i,j,NVERT)=v_tmp(i,j,NVERT)
        enddo
    enddo

    ! equation 7.47 time difference term
    do k=1,NVERT
        v_new(:,:,k)=P_old/P_new*v_old(:,:,k)
    enddo

    ! equation 7.48 horizontal advection R
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)+dt/P_new*(R(0:NLAT,:,k)&
            *0.5*(v_v_old(0:NLAT,1:NLONG,k)+v_v_old(1:NLAT+1,1:NLONG,k))&
            -
R(1:NLAT+1,:,k)*0.5*(v_v_old(1:NLAT+1,1:NLONG,k)+v_v_old(2:NLAT+2,1:NLONG,k)))
    enddo

    ! equation 7.48 horizontal advection Q
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)+dt/P_new*(Q(1:NLAT+1,1:NLONG,k)&
            *0.5*(v_v_old(1:NLAT+1,0:NLONG-1,k)+v_v_old(1:NLAT+1,1:NLONG,k))&
            -
Q(1:NLAT+1,2:NLONG+1,k)*0.5*(v_v_old(1:NLAT+1,1:NLONG,k)+v_v_old(1:NLAT+1,2:NL
ONG+1,k)))
    enddo
```

```
    ! equation 7.48 horizontal advection S
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)+dt/P_new*(S(0:NLAT,1:NLONG,k)&
            *0.5*(v_v_old(0:NLAT,0:NLONG-1,k)+v_v_old(1:NLAT+1,1:NLONG,k))&
            -
S(1:NLAT+1,2:NLONG+1,k)*0.5*(v_v_old(1:NLAT+1,1:NLONG,k)+v_v_old(2:NLAT+2,2:NL
ONG+1,k)))
    enddo

    ! equation 7.48 horizontal advection T
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)+dt/P_new*(-T(1:NLAT+1,1:NLONG,k)&
            *0.5*(v_v_old(2:NLAT+2,0:NLONG-1,k)+v_v_old(1:NLAT+1,1:NLONG,k))&

+T(0:NLAT,2:NLONG+1,k)*0.5*(v_v_old(1:NLAT+1,1:NLONG,k)+v_v_old(0:NLAT,2:NLON
G+1,k)))
    enddo

    ! equation 7.49 vertical transport O
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)+dt/P_new/dsigma(k)*(O(:,:,k-1)*v_bot_old(:,:,k-1)&
            -O(:,:,k)*v_bot_old(:,:,k))
    enddo

    ! equation 7.50 coriolis and sperical grid conversion
    do k=1,NVERT
        v_new(:,:,k)=v_new(:,:,k)-0.5*dt/P_new*Re*dlamda_e*dphi&

*(v_pi_c_old(0:NLAT,1:NLONG)*0.5*(v_u_old(0:NLAT,1:NLONG,k)+v_u_old(0:NLAT,2:NLO
NG+1,k))&
            *(v_f_c(0:NLAT,1:NLONG)*Re*cos(v_lat_c(0:NLAT,1:NLONG))&

+0.5*(v_u_old(0:NLAT,1:NLONG,k)+v_u_old(0:NLAT,2:NLONG+1,k))*sin(v_lat_c(0:NLAT,1:
NLONG)))&

+v_pi_c_old(1:NLAT+1,1:NLONG)*0.5*(v_u_old(1:NLAT+1,1:NLONG,k)+v_u_old(1:NLAT+1
,2:NLONG+1,k))&
            *(v_f_c(1:NLAT+1,1:NLONG)*Re*cos(v_lat_c(1:NLAT+1,1:NLONG))&

+0.5*(v_u_old(1:NLAT+1,1:NLONG,k)+v_u_old(1:NLAT+1,2:NLONG+1,k))*sin(v_lat_c(1:NL
AT+1,1:NLONG))))
    enddo

    ! equation 7.51 pressure gradient
```

```
   do k=1,NVERT
      v_new(2:NLAT,:,k)=v_new(2:NLAT,:,k)-
dt/P_new(2:NLAT,:)*Re*dlamda_e*cos(lat_vface(2:NLAT,:))&
         *((geopot_c(2:NLAT,:,k)-geopot_c(1:NLAT-1,:,k))*0.5*(pi_c_old(1:NLAT-
1,:)+pi_c_old(2:NLAT,:))&
         +0.5*(-pi_c_old(1:NLAT-1,:)+pi_c_old(2:NLAT,:))*Cp_d&
         *(PVT_c_old(1:NLAT-1,:,k)/dsigma(k)*(sigma_bot(k)&
         *(P_bot(1:NLAT-1,:,k)-P_c(1:NLAT-1,:,k))+sigma_bot(k-1)&
         *(P_c(1:NLAT-1,:,k)-P_bot(1:NLAT-1,:,k-1)))+PVT_c_old(2:NLAT,:,k)/dsigma(k)&
         *(sigma_bot(k)*(P_bot(2:NLAT,:,k)-P_c(2:NLAT,:,k))&
         +sigma_bot(k-1)*(P_c(2:NLAT,:,k)-P_bot(2:NLAT,:,k-1)))))

      ! south boundary
      v_new(1,:,k)=v_new(1,:,k)-dt/P_new(1,:)*Re*dlamda_e*cos(lat_vface(1,:))&
         *((-geopot_c_tminus1(1,:,k)+geopot_c(1,:,k))*pi_c_old(1,:)&
         +(-
pi_c_tminus1(1,:)+pi_c_old(1,:))*Cp_d*(PVT_c_old(1,:,k)/dsigma(k)*(sigma_bot(k)*(P_bot(1,:,
k)&
         -P_c(1,:,k))+sigma_bot(k-1)*(P_c(1,:,k)-P_bot(1,:,k-1)))))
      ! north boundary
      if(periodicBC==1) then
         v_new(NLAT+1,:,k)=v_new(1,:,k)
      else
         v_new(NLAT+1,:,k)=v_new(NLAT+1,:,k)-
dt/P_new(NLAT+1,:)*Re*dlamda_e*cos(lat_vface(NLAT+1,:))&
            *((geopot_c_tminus1(NLAT,:,k)-geopot_c(NLAT,:,k))*pi_c_old(NLAT,:)&
            +(pi_c_tminus1(NLAT,:)-
pi_c_old(NLAT,:))*Cp_d*(PVT_c_old(NLAT,:,k)/dsigma(k)*(sigma_bot(k)*(P_bot(NLAT,:,k)&
            -P_c(NLAT,:,k))+sigma_bot(k-1)*(P_c(NLAT,:,k)-P_bot(NLAT,:,k-1)))))
      endif
   enddo

   ! equation 7.52 eddy vicousity not finished!
   if(turbmodel==1) then
      do k=1,NVERT
         v_new(:,:,k)=v_new(:,:,k)
      enddo
   endif

end subroutine calculate_velocity_field

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_pressure_field
! usage: update pressure field using pi_c_new
```

```
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_pressure_field(pi_c,Pa_c,Pa_bot,P_c,P_bot,sigma_bot)
   real(dp),dimension(NLAT,NLONG),intent(in)::pi_c
   real(dp),dimension(NLAT,NLONG,0:NVERT),intent(inout)::Pa_bot,P_bot
   real(dp),dimension(0:NVERT),intent(in)::sigma_bot
   real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::Pa_c,P_c
   integer::i,j

   ! calculate pressure at different vertical layer
   ! layer bottom
   do i=1,NLAT
      do j=1,NLONG
         Pa_bot(i,j,:)=sigma_bot*pi_c(i,j)+Pa_top
         P_bot(i,j,:)=(Pa_bot(i,j,:)/1000)**k_therm
      enddo
   enddo

   ! layer center
   do i=1,NLAT
      do j=1,NLONG
         P_c(i,j,:)=1.0_dp/(1+k_therm)*(P_bot(i,j,1:NVERT)*Pa_bot(i,j,1:NVERT)-
P_bot(i,j,0:(NVERT-1))*Pa_bot(i,j,0:(NVERT-1)))
         P_c(i,j,:)=P_c(i,j,:)/(Pa_bot(i,j,1:NVERT)-Pa_bot(i,j,0:NVERT-1))
         Pa_c(i,j,:)=1000*(P_c(i,j,:)**(1/k_therm))
      enddo
   enddo

end subroutine calculate_pressure_field

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_air_temperature
! usage: update air temperature with the new
! Potential virtual temperature
! Yun Zhang 05/03/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_air_temperature(PVT_c,Temp_c,qv_c,Pa_c)
   real(dp),dimension(NLAT,NLONG,NVERT),intent(in)::PVT_c,qv_c,Pa_c
   real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::Temp_c
   Temp_c=PVT_c/(1+0.608*qv_c)*((Pa_c/1000)**k_therm)
end subroutine calculate_air_temperature
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_horizontal_bottom_value
! usage: calculate the horizontal value (u,v)
! at the layer bottom of each edege
! Yun Zhang 05/22/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_horizontal_bottom_value(u,u_bot,N1,N2,dsigma)
    real(dp),dimension(N1,N2,NVERT),intent(in)::u
    real(dp),dimension(N1,N2,0:NVERT),intent(inout)::u_bot
    integer,intent(in)::N1,N2
    real(dp),dimension(NVERT),intent(in)::dsigma
    integer:: i,j
    do i=1,N1
        do j=1,N2
            u_bot(i,j,1:NVERT-1)=(u(i,j,1:NVERT-1)*dsigma(1:NVERT-
1)+u(i,j,2:NVERT)*dsigma(2:NVERT))&
            /(dsigma(1:NVERT-1)+dsigma(2:NVERT))
            u_bot(i,j,0)=u(i,j,1)
            u_bot(i,j,NVERT)=u(i,j,NVERT)
        enddo
    enddo
end subroutine calculate_horizontal_bottom_value

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_horizontal_center_value
! usage: calculate the horizontal value (u,v)
! at the layer bottom of each edege
! Yun Zhang 05/22/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_horizontal_center_value(u,u_c,N1,N2)
    integer,intent(in)::N1,N2
    real(dp),dimension(N1,N2,NVERT),intent(in)::u
    real(dp),dimension(NLAT,NLONG,NVERT),intent(inout)::u_c
    if(N2>NLONG) u_c=0.5*(u(:,1:NLONG,:)+u(:,2:NLONG+1,:))
    if(N1>NLAT) u_c=0.5*(u(1:NLAT,:,:)+u(2:NLAT+1,:,:))
end subroutine calculate_horizontal_center_value

end module phys
```

9. source.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - source
! usage: include all the functions and
! subroutines to define source and sinks
! for heat, momentum and scalar
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module source
   use constant_parameter
   implicit none
contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! heat_source
! usage: provide the value of heat sinks
! and source to calculate potential virtual
! temperature
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function heat_source(lat,long,nlayer,t) result(output)
   real(dp),intent(in)::lat,long,t
   integer, intent(in):: nlayer
   real(dp)::output
   output=0.0_dp

end function heat_source

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! gas_source
! usage: provide the value of gas sinks
! and source to calculate gas concentration
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function qv_source(lat,long,nlayer,t) result(output)
   real(dp),intent(in)::lat,long,t
   integer, intent(in):: nlayer
   real(dp)::output
   output=0.0_dp

end function qv_source
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! gas_source
! usage: provide the value of gas sinks
! and source to calculate gas concentration
! Yun Zhang 04/30/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function gas_source(lat,long,nlayer,t) result(output)
   real(dp),intent(in)::lat,long,t
   integer, intent(in):: nlayer
   real(dp)::output
   output=0.0_dp

end function gas_source
end module source
```

10. turbulence.f90

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! regional model module - turbulence
! usage: include all the subroutines
! and functions to calculate eddy diffusivity
! and viscousity
! Yun Zhang 05/01/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
module turbulence
   use constant_parameter
   implicit none

! not finished assume Kb is zero
contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate_eddy_viscousity_diffusivity
! usage: use turbulence model to calculate
! eddy diffusivty and viscousity for momentum
! and transport equations
! Yun Zhang 05/02/2015
! @stanford
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine calculate_eddy_viscousity_diffusivity(nu_t,K_t)
   real(dp),dimension(NLAT,NLONG,NVERT),intent(inout):: nu_t,K_t
   nu_t=0.0_dp
   K_t=0.0_dp
end subroutine calculate_eddy_viscousity_diffusivity
end module turbulence
```

11. Makefile

```
FC = gfortran
LD = gfortran
#LDFLAGS = -framework Accelerate
LDFLAGS = -lblas

FFLAGS = -O

.PHONY: clean

%.o : %.f90
	$(FC) $(FFLAGS) -c $<
%.mod : %.f90
	$(FC) $(FFLAGS) -c $<

EXE0 = regionalmodel
MOD0 = constant_parameter.mod allocate_variable.mod basic_state.mod \
	initialization.mod boundary.mod turbulence.mod source.mod output.mod phys.mod
OBJ0 = regionalmodel.o constant_parameter.o allocate_variable.o \
	basic_state.o initialization.o boundary.o turbulence.o source.o output.o phys.o

$(EXE0): $(MOD0) $(OBJ0)
	$(LD) $(OBJ0) $(LDFLAGS) -o $@

ALLEXE = $(EXE0)

all: $(ALLEXE)

cleanall:
	rm -f $(ALLEXE) *.mod *.o
	rm -rf results

clean:
	rm -f $(ALLEXE) *.mod *.o
	rm -f results/*.bin results/*.txt

test:
	./$(ALLEXE)
```

12. Matlab Library

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ReadResult.m
% usage: read the results from the output
% of regional model for cee263B
% save as result.mat which can be used for
% further discussion
% Yun Zhang 05/06/2015
% @Stanford
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% result directory
datadir='../results';

NLAT=40;
NLONG=40;
NVERT=15;
lat_0=-1;
long_0=-1;
dlat=0.05;
dlong=0.05;
Pa_top=250;
for i=1:NLAT
	for j=1:NLONG
		lat(i,j)=lat_0+(i-1)*dlat;
		long(i,j)=long_0+(j-1)*dlong;
	end
end

latuface=zeros(NLAT,NLONG+1);
longuface=zeros(NLAT,NLONG+1);
latuface(:,1:NLONG)=lat;
latuface(:,NLONG+1)=lat(:,1);
longuface(:,1:NLONG)=long-0.5*dlong;
longuface(:,NLONG+1)=long(:,NLONG)+0.5*dlong;

latvface=zeros(NLAT+1,NLONG);
longvface=zeros(NLAT+1,NLONG);
longvface(1:NLAT,:)=long;
longvface(NLAT+1,:)=long(NLAT,:);
latvface(1:NLAT,:)=lat-0.5*dlong;
latvface(NLAT+1,:)=lat(NLAT,:)+0.5*dlong;

% load in all results
```

```
filename=[datadir,'/pi.txt'];
pi_tmp=load(filename);

filename=[datadir,'/Pa.txt'];
Pa_tmp=load(filename);

filename=[datadir,'/geopot.txt'];
geopot_tmp=load(filename);

filename=[datadir,'/K_t.txt'];
K_t_tmp=load(filename);

filename=[datadir,'/nu.txt'];
nu_t_tmp=load(filename);

filename=[datadir,'/PVT.txt'];
PVT_tmp=load(filename);

filename=[datadir,'/u.txt'];
u_tmp=load(filename);

filename=[datadir,'/v.txt'];
v_tmp=load(filename);

filename=[datadir,'/w.txt'];
w_tmp=load(filename);

filename=[datadir,'/qv.txt'];
qv_tmp=load(filename);

filename=[datadir,'/temp.txt'];
temp_tmp=load(filename);

filename=[datadir,'/rhoa.txt'];
rhoa_tmp=load(filename);

filename=[datadir,'/gas.txt'];
gas_tmp=load(filename);

% get Nt from output
Nt=length(pi_tmp')/NLAT;

% seperate different time step
for i=1:Nt
```

```
    base1=(i-1)*NLAT;
    base2=(i-1)*NLAT*NLONG;
    base3=(i-1)*(NLAT+1)*NLONG;
    base4=(i-1)*(NLONG+1)*NLAT;
    pi{i}=pi_tmp((base1+1):(base1+NLAT),:);
    Pa{i}=Pa_tmp((base2+1):(base2+NLAT*NLONG),:);
    geopot{i}=geopot_tmp((base2+1):(base2+NLAT*NLONG),:);
    K_t{i}=K_t_tmp((base2+1):(base2+NLAT*NLONG),:);
    nu_t{i}=nu_t_tmp((base2+1):(base2+NLAT*NLONG),:);
    PVT{i}=PVT_tmp((base2+1):(base2+NLAT*NLONG),:);
    u{i}=u_tmp((base4+1):(base4+NLAT*(NLONG+1)),:);
    v{i}=v_tmp((base3+1):(base3+(NLAT+1)*NLONG),:);
    w{i}=w_tmp((base2+1):(base2+NLAT*NLONG),:);
    qv{i}=qv_tmp((base2+1):(base2+NLAT*NLONG),:);
    temp{i}=temp_tmp((base2+1):(base2+NLAT*NLONG),:);
    rhoa{i}=rhoa_tmp((base2+1):(base2+NLAT*NLONG),:);
    gas{i}=gas_tmp((base2+1):(base2+NLAT*NLONG),:);
end

% save results
save('results.mat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plotvelocityquiver.m
% usage: plot velocity quiver for specific layer
% Yun Zhang 05/07/2015
% @Stanford
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotvelocityquiver(u,v,k,x,y,xtype,ytype,titleinfo)
[a,b]=size(x);
uu=reshape(u(:,k),b+1,a);
vv=reshape(v(:,k),b,a+1);
vv=vv';
uu=uu';
uc=(uu(:,1:b)+uu(:,2:b+1))/2;
vc=(vv(1:a,:)+vv(2:a+1,:))/2;
quiver(x,y,uc,vc,1.4)
xlabel(xtype);
ylabel(ytype);
title(titleinfo);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot3Dsliceresults.m
% usage: plot 3D results for a specific layer from regional model
% Yun Zhang 05/07/2015
% @Stanford
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plot3Dsliceresults(input,k,x,y,xtype,ytype,titleinfo)
[a,b]=size(x);
plotdata=reshape(input(:,k),b,a);
plotdata=plotdata';
pcolor(x,y,plotdata)
xlabel(xtype);
ylabel(ytype);
title(titleinfo);
colorbar;
colormap('Jet');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot2Dresults.m
% usage: plot 2D results from regional model
% only for cell centered 2D data
% Yun Zhang 05/07/2015
% @Stanford
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plot2Dresults(input,x,y,xtype,ytype,titleinfo)
pcolor(x,y,input)
xlabel(xtype);
ylabel(ytype);
title(titleinfo);
colorbar;
colormap('Jet');
end
```