浅水波方程隐式求解串行代码阅读笔记

刘群*

地球系统科学研究中心

Center for Earth System Science

2015 年 3 月 19 日

下面是我的浅水波方程隐式求解串行代码的阅读笔记。在对各个程序分别进行介绍时，首先对程序的整体功能和特点做一个介绍，然后将程序源码贴出，对代码进行详细的注释 (当然有些没有看懂的地方没法给出注释了)。最后对阅读代码的收获和感受进行了总结。

1 module_para.f90

这是一个 Module，主要是定义了一些在程序中用到的各种参数。在定义时，含有 parameter 属性的变量在程序运行过程中是不可以改变的，相当于常量。定义的变量中，有些是关于网格信息的变量，比如说 p 与纬向的分辨率相关，当 p = 720 时，此时纬向分辨率为 $360°/720 = 0.5°$，当 $q = 2q_0 = 360$ 时，经向的分辨率为 $180°/360 = 0.5°$。所以在 comment.txt 中说如果令 $p = 80, q_0 = 20(q = 2q_0 = 40)$，则有分辨率为 $4.5° \times 4.5°$，即纬向分辨率为 $360°/80 = 4.5°$, 经向分辨率为 $180°/40 = 4.5°$. 其中 np 是纬向的格点个数，n 是经向的格点个数。

tlo 是时间步长 200s，t0 是初始时间，t1 为模拟终止的时间 (100 天)。

n0 是一个整数标志位，当 n0 = 0 时，使用 Rossby-Haurwitz waves 作为初始条件，当 n0=1 时，从文件读入初始条件。thalf 是输出中间结果的时间。

fu, fv 和 fh(长度为 7 个字节的字符串)，分别表示的是纬向风、经向风和位势高度的初始场，可以用来作自定义的试验。由于前面 n0 已经设为 0，因此实际上并不从这些文件中读取任何初始场信息。ffu, ffv, ffh (长度为 7 个字节的字符串) 表示的是纬向风，经向风和位势高度输出结果的文件名。

```
module module_para                    ! 参数定义模块
   implicit none                      ! 变量必须显式声明，不允许隐式声明
```

*电子邮件: liu-q14@mails.tsinghua.edu.cn，学号: 2014211591

```
    real*8,  parameter      :: omg0 = 7.292d-5 ! angular velocity of
                                               ! the earth rotation
    real*8,  parameter      :: a = 6371000d0   ! radius of the earth

    integer, parameter      :: p  = 720  ! to define zonal resolution
    integer, parameter      :: kn = p/2  ! to define zonal resolution
    integer, parameter      :: np = p+1  ! zonal grid nmuber
    integer, parameter      :: n1 = np+1

    integer, parameter      :: q0 = 180  ! to define meridional resolution
    integer, parameter      :: nq = q0+1
    integer, parameter      :: q  = q0*2
    integer, parameter      :: n  = q+1  ! meridional grid number

    integer, parameter      :: nm = (np-1)*n ! total grid number

    integer, parameter      :: tlo = 200    ! time stepsize
    integer, parameter      :: t0 = 0       ! initial time
    integer, parameter      :: t1 = 8640000 ! final time

    integer, parameter      :: n0 = 0 ! 0: using rh waves as initial conditon
                                      ! (ic);  1: read ic from files
    integer, parameter      :: nyn = 0    ! screen output 0: at each step;
                                          ! 1: once 12 hours
    integer, parameter      :: thalf = 432000! the time for saving the
                                             ! intermediate result

    character*7, parameter  :: fu='uui.dat' ! initial field of zonal wind
                                            ! for self-defined experiment
    character*7, parameter  :: fv='vvi.dat' ! initial field of meridional
                                          ! wind for self-defined experiment
    character*7, parameter  :: fh='hhi.dat' ! initial field of geopotential
                                        ! height for self-defined experiment

    character*6, parameter  :: ffu='ui.dat' ! result of zonal wind for self
                                            ! -defined experiment
    character*6, parameter  :: ffv='vi.dat' ! result of meridional wind for
                                            ! self-defined experiment
    character*6, parameter  :: ffh='hi.dat' ! result of geopotential height
                                            ! for self-defined experiment

    real*8                  :: pi           ! 圆周率
    real*8                  :: deta,deta1,deta2,detb

    real*8,  dimension(1:n) :: c1, s1  ! c1(j)=cos(theta(j)), s1(j)=sin
                                       ! (theta(j)), where theta is latitude
    real*8,  dimension(1:n) :: c2, c11, c12, c13, c14  !
    real*8,  dimension(1:n) :: f1, f2      ! f1是科氏参数
end module module_para
```

2 module_array.f90

module_array 这个模块主要是定义了程序中用到的数组的信息。当然这个模块也利用了 module_para 中的网格数目的参数信息，定义了纬向风 u(浮点数，8 个字节，维度为 $n1 \times n, n1 = 722, n = 361$), 经向风 v, 位势高度 wh(公式中的 φ) 以及作变换之后的变量 wu(公式中的 $U = uh = u\sqrt{\varphi}$) 和 wv(公式中的 $V = vh = v\sqrt{\varphi}$).

```
module module_array                          ! 数组定义模块

    use module_para                         ! 调用module_para模块

    implicit none                           ! 变量必须显式声明，不允许隐式声明
```

```
    real*8,  dimension(1:n1,1:n) ::    u  ! zonal wind
    real*8,  dimension(1:n1,1:n) ::    v  ! meridional wind
    real*8,  dimension(1:n1,1:n) ::    wh ! geopotential height

    real*8,  dimension(1:n1,1:n) ::    wu ! u*sqrt(wh)
    real*8,  dimension(1:n1,1:n) ::    wv ! v*sqrt(wh)

end module module_array
```

3 cs.f90

cs.f90 是一个 subroutine 子程序，用来计算下面公式中要用到的参数值。比如说所有纬度的正弦值和余弦值，计算一些系数的值，比如 $c_{11}(j)=\frac{1}{2acos\theta_j\Delta\lambda}$, $c_{12}(j)=\frac{1}{2acos\theta_j\Delta\theta}$, $c_{13}(j)=\frac{1}{4acos\theta_j\Delta\lambda}$, $c_{14}(j)=\frac{1}{4acos\theta_j\Delta\theta}, j=1,2,\cdots,n$. 还有与科氏参数相关的量，比如 $f0=2\omega_0\sin\theta$, $f2=\frac{\tan\theta}{a}$, $f^*=f0+u*f2=2\omega_0\sin\theta+\frac{\tan\theta}{a}$. 详细的参数介绍请看下面的注释。

```
 subroutine cs
    use module_para      ! 调用module_para模块
    use module_array     ! 调用module_array模块

    implicit none        ! 变量必须显式声明，不允许隐式声明

    real*8  :: ai, aj    ! working variables
    integer :: j         ! working variable

    pi=datan(1d0)*4      ! 计算pi=4*arctan(1)
    deta1=4*pi/p*a       ! a是地球半径，由于p=720,故deta1=2*2*pi*a/p
                         ! 是纬向网格的间隔的两倍->2a*Delta_lambda
    deta=pi/q            ! 经向分辨率，网格相差的弧度
    deta2=deta*a*2       ! 是经向网格的间隔的两倍->2a*Delta_theta
    detb=deta*nq         ! detb=-89.5 degree

    do j=2,n-1           ! 计算各个纬度的cos和sin值(从-89.5到89.5 degree)
        ai    = j*deta-detb ! 计算纬度值
        c1(j) = dcos(ai)
        s1(j) = dsin(ai)
    end do

    ai=1.5*deta-detb     ! 近似计算南极点的cos值,注意不能取0
    c1(1)=dcos(ai)/4     ! cos(-90)~cos(-88.75)/4

    ai=(n-0.5)*deta-detb! 近似计算北极点的cos值,注意不能取0
    c1(n)=dcos(ai)/4

    s1(1)=-1.0d0         ! 南极点的sin值-1
    s1(n)=1.0d0          ! 北极点的sin值1

    do j=1,n             ! 将c1的值赋给c2
        c2(j)=c1(j)
    end do

    do j=1,n
        ai=s1(j)             ! 不同纬度正弦值
        aj=c1(j)*a
        f1(j)=2.0d0*omg0*ai ! 计算不同纬度的科氏参数f0=2*Omega*sin(theta)
        f2(j)=ai/aj          ! 与f*相关的量，f2=tan(theta)/a f*=f0+u*f2
    end do
```

```
    do j=1,n                    ! 计算用到的几个系数，相当于公式中的(如下):
        ai=deta1*c1(j)
        aj=deta2*c1(j)
        c11(j)=1/ai             ! c11_j = 1/[2a*cos(theta_j)*Delta_lambda]
        c12(j)=1/aj             ! c12_j = 1/[2a*cos(theta_j)*Delta_theta]
        c13(j)=c11(j)*0.5       ! c13_j = 1/[4a*cos(theta_j)*Delta_lambda]
        c14(j)=c12(j)*0.5       ! c14_j = 1/[4a*cos(theta_j)*Delta_theta]
    end do

    return                      ! 子程序返回
end subroutine cs
```

4 dif.f90

dif.f90 是计算部分差分结果的子程序，在里面主要有三个子程序，分别是 difuh、difv 和 advct。其中 advct 是计算差分格式中的平流项差分结果的程序。

在调用 adcvt(u,v,f,df) 时，如果调用格式为 call advct(u,vv,wu,du), 则与 advct 中参数的对应关系为 $u=u, v=vv, f=wu, df=du$. 根据 advct 子程序的内容可以看出，此时，程序中的变量与公式中的对应关系如表1所示.

表 1: difuh 与 adcvt 中变量对应关系

advct 程序中	调用时	Manual 公式中
u	u	u
v	vv	$v^*=v\cos\theta$
f	wu	$U=hu=\sqrt{\varphi}u$
dx		$u_{i,j}(U_{i+1,j}-U_{i-1,j})+(u_{i+1,j}U_{i+1,j}-u_{i-1,j}U_{i-1,j})$
dy		$v_{i,j}(U_{i,j+1}-U_{i,j-1})\cos\theta_j+(v_{i+1,j}U_{i+1,j}\cos\theta_{j+1}-v_{i-1,j}U_{i-1,j}\cos\theta_{j-1})$
df	du	$\frac{1}{4a\cos\theta_j\Delta\lambda}dx+\frac{1}{4a\cos\theta_j\Delta\theta}dy$
su		uU
sv		$v^*U=vU\cos\theta$

在调用 adcvt(u,v,f,df) 时，如果调用格式为 call advct(u,vv,wv,dv), 则与 advct 中参数的对应关系为 $u=u, v=vv, f=wv, df=dv$. 根据 advct 子程序的内容可以看出，此时，程序中的变量与公式中的对应关系如表2所示。

在 difuh.f90 中，我们用到的变量除了在表1列出的一些外，还有表3中所示的变量。需要说明的是，在调用 advct 子程序返回 du 之后, 程序接下来又对 du 做了进一步处理，使 du 由返回的

$$\begin{aligned}du_{i,j}=&\frac{1}{4a\cos\theta_j\Delta\lambda}[u_{i,j}(U_{i+1,j}-U_{i-1,j})+(u_{i+1,j}U_{i+1,j}-u_{i-1,j}U_{i-1,j})]\\&+\frac{1}{4a\cos\theta_j\Delta\theta}[v_{i,j}(U_{i,j+1}-U_{i,j-1})\cos\theta_j+(v_{i+1,j}U_{i+1,j}\cos\theta_{j+1}\\&-v_{i-1,j}U_{i-1,j}\cos\theta_{j-1})]\end{aligned}$$

表 2: difv 与 adcvt 中变量对应关系

advct 程序中	调用时	Manual 公式中
u	u	u
v	vv	$v^* = v\cos\theta$
f	wv	$V = hv = \sqrt{\varphi}v$
dx		$u_{i,j}(V_{i+1,j} - V_{i-1,j}) + (u_{i+1,j}V_{i+1,j} - u_{i-1,j}V_{i-1,j})$
dy		$v_{i,j}(V_{i,j+1} - V_{i,j-1})\cos\theta_j + (v_{i+1,j}V_{i+1,j}\cos\theta_{j+1} - v_{i-1,j}V_{i-1,j}\cos\theta_{j-1})$
df	dv	$\frac{1}{4a\cos\theta_j\Delta\lambda}dx + \frac{1}{4a\cos\theta_j\Delta\theta}dy$
su		uV
sv		$v^*V = vV\cos\theta$

变为

$$\begin{aligned} du_{i,j} =& \frac{1}{4a\cos\theta_j\Delta\lambda}\left[u_{i,j}(U_{i+1,j} - U_{i-1,j}) + (u_{i+1,j}U_{i+1,j} - u_{i-1,j}U_{i-1,j})\right] \\ &+ \frac{1}{4a\cos\theta_j\Delta\theta}[v_{i,j}(U_{i,j+1} - U_{i,j-1})\cos\theta_j + (v_{i+1,j}U_{i+1,j}\cos\theta_{j+1} \\ &- v_{i-1,j}U_{i-1,j}\cos\theta_{j-1})] - f^*V_{i,j} \end{aligned}$$

返回的 dh 为:

$$dh_{i,j} = \frac{1}{4a\cos\theta_j\Delta t}(h_{i,j+1}V_{i,j+1}\cos\theta_{j+1} - h_{i,j-1}V_{i,j-1}\cos\theta_{j-1})$$

表 3: difuh 与 Manual 中变量对应关系

difuh 程序中	Manual 公式中
ff	$f^* = 2\omega_0\sin\theta + \frac{u}{a}\tan\theta$
$f1$	$2\omega_0\sin\theta$
$f2$	$\frac{\tan\theta}{a}$
hy	$Vh\cos\theta$

在 difv.f90 中，在返回 dv 之后，进行了与上面 du 不同的处理，主要是将返回的 dv 由

$$\begin{aligned} dv_{i,j} =& \frac{1}{4a\cos\theta_j\Delta\lambda}[u_{i,j}(V_{i+1,j} - V_{i-1,j}) + (u_{i+1,j}V_{i+1,j} - u_{i-1,j}V_{i-1,j})] \\ &+ \frac{1}{4a\cos\theta_j\Delta\theta}[v_{i,j}(V_{i,j+1} - V_{i,j-1})\cos\theta_j + (v_{i+1,j}V_{i+1,j}\cos\theta_{j+1} \\ &- v_{i-1,j}V_{i-1,j}\cos\theta_{j-1})] \end{aligned}$$

变为了

$$
\begin{aligned}
dv_{i,j} =& \frac{1}{4a\cos\theta_j\Delta\lambda}\left[u_{i,j}(V_{i+1,j}-V_{i-1,j})+(u_{i+1,j}V_{i+1,j}-u_{i-1,j}V_{i-1,j})\right] \\
&+\frac{1}{4a\cos\theta_j\Delta\theta}[v_{i,j}(V_{i,j+1}-V_{i,j-1})\cos\theta_j+(v_{i+1,j}V_{i+1,j}\cos\theta_{j+1} \\
&-v_{i-1,j}V_{i-1,j}\cos\theta_{j-1})]+\frac{h_{i,j}(\varphi_{i,j+1}-\varphi_{i,j-1})}{2a\Delta\theta}+f^*U_{i,j}
\end{aligned}
$$

对于这种不同的原因，我会在 euler.f90 的分析中进一步阐述。

```
subroutine difuh(wu,wv,du,dh,h)
   use module_para                  ! 调用module_para模块
   implicit none                    ! 变量必须显式声明，不允许隐式声明
                                    ! 定义用到的二维数组和其他变量
   real*8,dimension(1:n1,1:n)   :: dh,h,hy
   real*8,dimension(1:n1,1:n)   :: wu,du,u
   real*8,dimension(1:n1,1:n)   :: wv,v,vv
   real*8                       :: hyn,hys
   real*8                       :: ff
   integer                      :: i,j

   do j=2,n-1                             ! 循环时注意顺序
      do i=2,np
         u (i,j)=wu(i,j)/h(i,j)           ! 纬向风u，wu 相当于公式中的U
         v (i,j)=wv(i,j)/h(i,j)           ! 经向分v，wv 相当于公式中的V
         vv(i,j)=v (i,j)*c1(j)            ! 公式中的v*=v cos(theta)
      end do
   end do

   ! 调用计算平流项查分的函数，du 下面要重点利用
   ! 返回的du相当于Manual公式中的平流项的查分格式，即：
   ! du_ij= 1/[4a*cos(theta)*Delta_lambda]*[ u_ij*(U_i+1,j-U_i-1,j)
   !         + (u_i+1,j*U_i+1,j - u_i-1,j*U_i-1,j)] +
   !  1/[4a*cos(theta)*Delta_theta]*[v_ij*cos(theta_j)*(U_i,j+1-U_i,j-1)+
   !  (v_i,j+1*U_i,j+1*cos(theta_j+1) - (v_i,j-1*U_i,j-1*cos(theta_j-1)]
   call advct(u,vv,wu,du)

   do j=2,n-1
      do i=2,np
         ff=f1(j)+u(i,j)*f2(j)            ! 公式中的f*，原因是f*=(2*Omega*
                                          !  sin(theta)) + u*tan(theta)/a
         du(i,j)=du(i,j)-ff*wv(i,j)       ! =-dU/dt-h/a/cos(theta)dphi/dlambda
         hy(i,j)=wv(i,j)*h(i,j)*c1(j)     ! = V*h*cos(theta)
                                          ! 用来计算phi在y方向的差分
      end do
   end do

   do i=2,np                              ! 将边界值设为0
      du(i,1)=0.0                         ! 将du的第一列设为0
      du(i,n)=0.0                         ! 将du的最后一列设为0

      hy(i,1)=0.0                         ! 将dy的第一列设为0
      hy(i,n)=0.0                         ! 将dy的最后一列设为0
   end do

   do j=2,n-1
      do i=2,np
         dh(i,j)=(hy(i,j+1)-hy(i,j-1))*c12(j)
         ! = -d_phi/d_t - 1/a/cos(theta) d(hU)/d_lambda
      end do
   end do

   hys=0.0
   hyn=0.0
   do i=2,np
```

```
        hys=hys+hy(i,2)      ! hy=V*h*cos(theta)第二列的和
        hyn=hyn-hy(i,n-1)    ! hy 第n-1列的和的相反数
    end do
    hys=hys*c12(1)/(np-1)
    hyn=hyn*c12(n)/(np-1)

    do i=2,np                ! dh在上下边界的值
        dh(i,1)=hys
        dh(i,n)=hyn
    end do

    return                   ! 返回
end subroutine difuh         ! difuh子程序结束

!****************** subroutine difv ******************!
subroutine difv(wu,wv,wh,dv,h)
    use module_para          ! 调用module_para模块
    implicit none            ! 变量必须显式声明，不允许隐式声明

    real*8,dimension(1:n1,1:n)  :: wh,h,hh
    real*8,dimension(1:n1,1:n)  :: wu,u
    real*8,dimension(1:n1,1:n)  :: wv,dv,v,vv
    real*8                      :: ff
    integer                     :: i,j

    do j=2,n-1
        do i=2,np
            hh(i,j)=h(i,j)*c1(j)
            u(i,j)=wu(i,j)/h(i,j)       ! 纬向风u，wu 相当于公式中的U
            v(i,j)=wv(i,j)/h(i,j)       ! 经向分v，wv 相当于公式中的V
            vv(i,j)=v(i,j)*c1(j)        ! 相当于公式中的v*=v cos(theta)
        end do
    end do

    call advct(u,vv,wv,dv)

    do j=2,n-1
        do i=2,np
            ff=f1(j)+u(i,j)*f2(j)        !公式中的f*,原因是f*=(2*Omega*
                                         !  sin(theta)) + u*tan(theta)/a
            dv(i,j)=dv(i,j)+hh(i,j)*(wh(i,j+1)-wh(i,j-1))*c12(j)+ff*wu(i,j)
        end do
    end do

    do i=2,np
        dv(i,1)=0.0
        dv(i,n)=0.0
    end do

    return
end subroutine difv

!****************** subroutine advct ******************!
subroutine advct(u,v,f,df)
    use module_para                 ! 调用module_para模块
    implicit none                   ! 变量必须显式声明，不允许隐式声明

    real*8,dimension(1:n1,1:n) :: f,df ! 定义一些数组
    real*8,dimension(1:n1,1:n) :: u,v
    real*8,dimension(1:n1,1:n) :: su,sv
    real*8                     :: dx,dy
    integer                    :: i,j
    !-------- 如果调用形式为--call advct(u,vv,wu,du)-------------!
    ! f=wu=U(Manual公式中) --> su=uU(公式中)
    ! f=wu=U(Manual公式中),v=vv=v*(公式中) --> sv=v*U(公式中)
    do j=2,n-1
        do i=2,np
            su(i,j)=f(i,j)*u(i,j) ! 相当于Manual公式中的u_ij * U_ij
            sv(i,j)=f(i,j)*v(i,j) !相当于Manual公式中的v_ij*U_ij*cos(theta_j)
```

```
        end do
        su(1,j)=su(np,j)              ! 利用周期条件得到su和f在边界的值
        su(n1,j)=su(2,j)

        f(1,j)=f(np,j)
        f(n1,j)=f(2,j)
    end do

    do i=2,np                         ! 设定sv和df在上下边界的值为0
        sv(i,1)=0
        sv(i,n)=0
        df(i,1)=0.0
        df(i,n)=0.0
    end do

    ! 当df=du时,有dx dy df 等价于公式中的:
    ! dx = [u_ij * (U_i+1,j - U_i-1,j) + (u_i+1,j*U_i+1,j - u_i-1,j*U_i-1,j)]
    ! dy = [v_ij*cos(theta_j) * (U_i,j+1 - U_i,j-1) +
    !  (v_i,j+1 * U_i,j+1 * cos(theta_j+1) - (v_i,j-1*U_i,j-1*cos(theta_j-1)]
    ! df_ij=du_ij= 1/ [4a*cos(theta)*Delta_lambda] * dx
    !            + 1/ [4a*cos(theta)*Delta_theta ] * dy
    do j=2,n-1
        do i=2,np
            dx=u(i,j)*(f(i+1,j)-f(i-1,j))+su(i+1,j)-su(i-1,j)
            dy=v(i,j)*(f(i,j+1)-f(i,j-1))+sv(i,j+1)-sv(i,j-1)
            df(i,j)=dx*c13(j)+dy*c14(j)
        end do
    end do
    return                            ! 返回
end subroutine advct
```

5 euler.f90

euler.f90 是所有代码的核心，这里是真正进行时间积分的地方。在 manual 的第三部分，我们可以看到求解 φ 的一个方程，即

$$a_{i-1,j}c_{i-1,j}\varphi^{k+1}_{i-2,j}+(1-a_{i-1,j}c_{i-1,j}+a_{i+1,j}b_{i+1,j})\varphi^{k+1}_{i,j}-a_{i+1,j}b_{i+1,j}\varphi^{k+1}_{i+2,j}=-b_{i+1,j}e_{i+1,j}-c_{i-1,j}e_{i-1,j}+f_{i,j}$$

对于方程中的各个参数，当我们对比前面的公式时，可以得到：

$$a_{i,j}=\frac{\Delta t\ h_{i,j}}{4a\cos\theta_j\Delta\lambda}\quad b_{i+1,j}=\frac{\Delta t\ h_{i+1,j}}{4a\cos\theta_j\Delta\lambda}\quad c_{i-1,j}=-\frac{\Delta t\ h_{i-1,j}}{4a\cos\theta_j\Delta\lambda}$$

$$\begin{aligned}e_{i,j}=U_{i,j}&-\frac{1}{8a\cos\theta_j\Delta\lambda}\left[u_{i,j}(U_{i+1,j}-U_{i-1,j})+(u_{i+1,j}U_{i+1,j}-u_{i-1,j}U_{i-1,j})\right]\\&+\frac{1}{8a\cos\theta_j\Delta\theta}[v_{i,j}(U_{i,j+1}-U_{i,j-1})\cos\theta_j+(v_{i+1,j}U_{i+1,j}\cos\theta_{j+1}\\&-v_{i-1,j}U_{i-1,j}\cos\theta_{j-1})]+\frac{\Delta t}{2}f^*V_{i,j}\end{aligned}$$

$$f_{i,j}=\varphi_{i,j}-\frac{1}{4a\cos\theta_j\Delta\theta}[h_{i,j+1}V_{i,j+1}\cos\theta_{j+1}-h_{i,j-1}V_{i,j-1}\cos\theta_{j-1}]$$

这些变量与程序中各个变量的对应关系如表4所示。我们注意到 a,b,c 只是下标不同 (c 多了一个负号)，因此我们可以令 $a_{i,j}=aj$(euler 程序中的变量), $aj*aj=a2$(euler 程序中的变量), 所以有

$$a_{i-1,j}c_{i-1,j}=a2_{i-1},\quad a_{i+1,j}b_{i+1,j}=a2_{i+1},$$

$$b_{i+1,j}e_{i+1,j} = aj_{i+1}e_{i+1,j} = ru_{i+1}, \quad b_{i-1,j}e_{i-1,j} = aj_{i-1}e_{i-1,j} = ru_{i-1}$$

原方程变为

$$-a2_{i-1}\varphi^{k+1}_{i-2,j} + (1 + a2_{i-1} + a2_{i+1})\varphi^{k+1}_{i,j} - a2_{i+1}\varphi^{k+1}_{i+2,j} = f_{i,j} - ru_{i-1} - ru_{i+1}$$

在这里需要特别指出的是，U 方向的迭代过程和 V 方向的迭代过程是不同的，不同之

表 4: euler.f90 与 Manual 中变量对应关系

euler 程序中	Manual 公式中
$tu_{i,j}$	$U_{i,j} - \frac{\Delta t}{2} du_{i,j} = e_{i,j}(du$ 见表1)
$th_{i,j}$	1. $\varphi_{i,j} - \frac{\Delta t}{2} dh_{i,j} = f_{i,j}(dh$ 见表1)
	2. $\varphi_{i,j}$(调用 LU 分解程序之后)
ai	$\frac{\Delta t}{4a\cos\theta_j \Delta\lambda}$
aj	$\frac{\Delta t h_{i,j}}{4a\cos\theta_j \Delta\lambda}$
ru_j	$a_{i,j}e_{i,j}$
rh_j	$f_{i,j} - ru_{i-1} - ru_{i+1}$
$fm, f0, fp, rf$	i 下标为偶数的 $\varphi_{i-2,j}, \varphi_{i,j}, \varphi_{i+2,j}$ 的系数，方程右端的常数项
$gm, g0, gp, rg$	i 下标为奇数的 $\varphi_{i-2,j}, \varphi_{i,j}, \varphi_{i+2,j}$ 的系数，方程右端的常数项

处在于迭代时所用的 φ 是不同的。在 U 的差分迭代中，φ 用的是第 $k+1$ 次迭代的信息，而在 V 的迭代中用的就是第 k 次的信息。因此就造成了 U 方向迭代过程的复杂性。在求解第 k+1 次迭代的 U 时，我们需要用到第 k+1 次迭代的 φ, 因此，我们需要先求解 φ. 求解 φ 我们采用了 LU 分解的方法，根据方程的特点，即方程是关于 $\varphi_{i-2,j}$, $\varphi_{i,j}$ 和 $\varphi_{i+2,j}$ 的，因此对于不同的 i 来说，$\varphi_{i-2,j}$，$\varphi_{i,j}$ 和 $\varphi_{i+2,j}$ 的下标 i 奇偶性是相同的，因此我们分奇偶分别求解一个线性方程组，这样即可求得所需的 φ.

在 lu0(a,b,c,r,n) 这个子程序中，相当于在解下面的循环三对角方程组

$$\mathbf{Ax} = \mathbf{r}$$

其中

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & a_1 \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & & & a_n & b_n \end{bmatrix}, \mathbf{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n-1} \\ r_n \end{bmatrix}$$

我们对 A 做 LU 分解，可以得到如下的形式：

$$L=\begin{bmatrix} 1 & & & & & \\ l_2 & 1 & & & & \\ & \ddots & \ddots & & & \\ & & & l_{n-1} & 1 & \\ t_1 & t_2 & \cdots & t_{n-2} & t_{n-1}+l_n & 1 \end{bmatrix}, U=\begin{bmatrix} u_1 & c_1 & & & & s_1 \\ & u_2 & c_2 & & & s_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \ddots & \ddots & s_{n-2} \\ & & & & u_{n-1} & c_{n-1}+s_{n-1} \\ & & & & & u_n \end{bmatrix}$$

根据矩阵乘法可以得到

$$\begin{cases} u_1=b_1, t_1=a_1, s_1=\frac{c_n}{u_1} \\ l_i=\frac{a_i}{u_{i-1}}, u_i=b_i-l_ic_{i-1}, s_i=-l_is_{i-1}, t_i=-\frac{t_{i-1}c_{i-1}}{u_{i-1}}, i=2,3,\cdots,n-1. \\ l_n=\frac{a_n}{u_{n-1}}, u_n=b_n-(t_{n-1}+l_n)(c_{n-1}+s_{n-1})-\sum_{i=1}^{n-2}t_is_i. \end{cases}$$

求出 L 和 U 即可按照 $Ly=d$ 和 $Ux=y$ 求解 x 和 y.

$$\begin{cases} y_1=r_1, \\ y_i=r_i-l_iy_{i-1}, i=2,3,\cdots,n-1. \\ y_n=d_n-\sum_{i=1}^{n-2}t_iy_i-(l_i+t_{n-1})y_{n-1}. \end{cases}$$

$$\begin{cases} x_n=\frac{y_n}{u_n} \\ x_i=\frac{1}{u_i}(y_i-c_ix_{i+1}-s_ix_n), i=n-1,n-2,\cdots,1. \end{cases}$$

实际在进行 LU 分解时，程序中并没有显式的给出 L 和 U，而是将 L 的计算结果存储到了临时变量 ai 中，将 U 的结果存储到了 b 中，同时，也没用显式的给出程序的返回值 x，而是将返回值存到了常数项 r 中。

```
subroutine euler(dt,iter)
   use module_para          ! 调用module_para模块
   use module_array         ! 调用module_array模块
   implicit none            ! 变量必须显式声明，不允许隐式声明

   integer i,i1,i2,j,k,iter

   real*8,dimension(1:n1,1:n)  :: tu,tv,th,h
   real*8,dimension(1:n1,1:n)  :: du,dv,dh
   real*8,dimension(1:n1)      :: a1,a2,rh,ru
   real*8,dimension(1:kn)      :: fm,fp,f0,gm,gp,g0,rf,rg
   real*8                      :: ai,aj,dt,dt2,en,en0,den
   real*8,external             :: inner    ! 调用外部的子程序

   dt2=dt*0.5d0                 ! 时间步长的一半Delta_t/2

   do j=1,n                     ! 将数组wu wv wh的值复制到tu tv th
      do i=2,np
         tu(i,j)=wu(i,j)        ! tu = sqrt(phi)*u
         tv(i,j)=wv(i,j)        ! tv = sqrt(phi)*v
         th(i,j)=wh(i,j)        ! th = phi --> geopotential height
      end do
   end do
```

```
en0=inner(wu,wv,wh,wu,wv,wh) ! 调用inner计算初始能量值

do k=1,1000                  ! 迭代最大次数为1000

    do j=1,n
        do i=2,np            ! 相当于公式中的h=sqrt(phi)
            h(i,j)=dsqrt(th(i,j))
        end do
    end do

    call difuh(tu,tv,du,dh,h)! 调用difuh计算U和h的差分

    do j=1,n                 !
        do i=2,np
            tu(i,j)=wu(i,j)-dt2*du(i,j) ! tu_ij ---> e_ij
            th(i,j)=wh(i,j)-dt2*dh(i,j) ! th_ij ---> f_ij
        end do
    end do

    do j=2,n-1
        ai=dt2*c11(j) !相当于公式中的Delta_t/(4a*cos(theta)Delta_lambda)

        do i=2,np
            aj=ai*h(i,j) ! aj相当于公式中的
                         ! h_ij*Delta_t/(4a*cos(theta)Delta_lambda)
            a1(i)=aj
            ru(i)=tu(i,j)*aj ! ru_ij ---> b*e_ij
            a2(i)=aj*aj
        end do

        ru(1)=ru(np)
        ru(n1)=ru(2)

        do i=2,np        !相当于要求解的方程的右端常数项
            rh(i)=th(i,j)-ru(i+1)+ru(i-1)
        end do
        ! 由Manual最后的迭代公式，我们可以知道phi可以按下标分为奇偶来求。
        do i=1,kn                    ! kn=p/2,p=720
            i1=i*2                   ! 偶数项下标
            i2=i1+1                  ! 奇数项下标

            fp(i)=-a2(i2)            ! 偶数项对应的方程中phi_i+2,j的系数
            rf(i)=rh(i1)             ! 偶数项对应的方程右端常数项

            gm(i)=-a2(i1)            ! 奇数项对应的方程中phi_i-2,j的系数
            rg(i)=rh(i2)             ! 奇数项对应的方程右端常数项
        end do

        do i=2,kn
            fm(i)=fp(i-1)            ! 偶数项对应的方程中phi_i-2,j的系数
        end do
        fm(1)=fp(kn)                 ! 周期条件

        do i=1,kn-1
            gp(i)=gm(i+1)            ! 奇数项对应的方程中phi_i+2,j的系数
        end do
        gp(kn)=gm(1)                 ! 周期条件

        do i=1,kn
            f0(i)=1.0-fm(i)-fp(i)    ! 关于phi的偶数项方程组中phi_ij的系数
            g0(i)=1.0-gm(i)-gp(i)    ! 关于phi的奇数项方程组中phi_ij的系数
        end do

        call lu0(fm,f0,fp,rf,kn)     ! 调用LU分解计算i为偶数的phi_ij
                                     ! rf返回计算结果
        call lu0(gm,g0,gp,rg,kn)     ! 调用LU分解计算i为奇数的phi_ij
                                     ! rg返回计算结果
```

```
            do i=1,kn
                i1=i*2                      ! 偶数项下标
                i2=i1+1                     ! 奇数项下标

                th(i1,j)=rf(i)              ! 将奇偶项的结果重新付给th,
                th(i2,j)=rg(i)              ! 即公式中的phi
            end do

            th(1,j)=th(np,j)                ! 周期条件
            th(n1,j)=th(2,j)

            do i=2,np                       ! 计算第k+1步的U的值
                tu(i,j)=tu(i,j)-a1(i)*(th(i+1,j)-th(i-1,j))
            end do
        end do

        call difv(tu,tv,th,dv,h)            ! 调用函数计算v方向的差分

        do j=1,n
            do i=2,np
                tv(i,j)=wv(i,j)-dt2*dv(i,j) ! 公式中v的迭代格式
            end do
        end do

        en=inner(tu,tv,th,tu,tv,th)         ! 计算本次循环之后的能量

        den=dabs(en-en0)*2.0/(en+en0)       ! 定义一个衡量迭代前后能量差的标志
        en0=en
        if (den.lt.1.0d-15) goto 10         ! 如果den小于10**-15,结束循环
                                            ! 前后两次的能量差距非常小
    end do

    10   continue                           ! 继续向下执行
    iter=k                                  ! 实际迭代步数

    do j=1,n                                    ! 将计算的第n+1/2位置的值
        do i=2,np                               ! 转化为n+1位置的值?
            wu(i,j)=tu(i,j)*2.0d0-wu(i,j)       ! U^(n+1) = 2*U^(n+1/2)-U^(n)
            wv(i,j)=tv(i,j)*2.0d0-wv(i,j)       ! V^(n+1) = 2*V^(n+1/2)-V^(n)
            wh(i,j)=th(i,j)*2.0d0-wh(i,j)       ! phi^(n+1)=2*phi^(n+1/2)-phi^(n)
        end do
    end do

    return
end subroutine euler                            ! euler.f90 结束

subroutine lu0(a,b,c,r,n)                       ! LU分解求线性方程组的解
    implicit none                               ! 必须显式声明

    integer                :: i,n
    real*8                 :: ai,sn,rn
    real*8,dimension(1:n)  :: a,b,c,r
    real*8,dimension(1:n)  :: s,t

    s(1)=a(1)
    t(1)=c(n)

    sn=0.0
    rn=0.0
    ! 这里分解时并没有显式的给出l和u, l就存在了临时变量ai中,
    do i=2,n-1                          ! Ax=r LUx=Ly=r Ux=y
        ai=a(i)/b(i-1)                  ! l_i
        b(i)=b(i)-ai*c(i-1)             ! u_i = b_i - l_i * c_i-1
        r(i)=r(i)-ai*r(i-1)             ! y_i = r_i - l_i * y_i-1
        s(i)=-ai*s(i-1)                 ! s_i = - l_i * s_i-1

        ai=t(i-1)/b(i-1)                ! 临时变量 t_i-1 / u_i-1
                                        ! i=2时, ai = c_n/u_1= t_1, b_1=u_1
```

```
        t(i)=-ai*c(i-1)              ! t_i = - t_i-1 * c_i-1 / u_i-1
        sn  =sn-ai*s(i-1)            ! 累积求和项
        rn  =rn-ai*r(i-1)            ! 累积求和项
    enddo

    a(n)=a(n)+t(n-1)                 ! l_n + t_n-1
    b(n)=b(n)+sn                     ! u_n = b_n - Sum(t_i*s_i),i=2...n-2
    c(n-1)=c(n-1)+s(n-1)             ! c_n-1+s_n-1
    r(n)=r(n)+rn                     ! y_n = y_n -Sum(t_i*r_i),i=2...n-2

    ai=a(n)/b(n-1)                   ! l_n=a_n/u_n-1
    b(n)=b(n)-ai*c(n-1)              ! u_n =  b_n - Sum(t_i*s_i) ,i=2...n-2
    r(n)=r(n)-ai*r(n-1)              ! y_n

    ! 开始逆向求解方程组的解x，之后的r中存储的是返回值
    r(n)=r(n)/b(n)                   ! x_n = y_n / u_n
    r(n-1)=(r(n-1)-c(n-1)*r(n))/b(n-1)
    do i=n-2,1,-1
        ai=r(i)-s(i)*r(n)-c(i)*r(i+1) ! x_i=(y_i-s_i*x_n-c_i*x_i+1)/u_i
        r(i)=ai/b(i)
    enddo

    return                           ! 程序返回
end subroutine
```

6 haurwitz.f90

haurwitz.f90 是一个子程序，在 n0 这个参数等于 0 的情况下才会被调用，是将波数为 4 的 Rossby-Haurwitz 波作为整个模拟的初始条件。在这个子程序中，主要是设置了 u, v, wh 等的初值和边界条件。

```
!-------------------------------------------------------------------------------
!            This subroutine is to provide initial condition using
!                        four-wave Rossby-Haurwitz waves
!-------------------------------------------------------------------------------

subroutine haurwitz

    use module_para     ! 调用module_para模块
    use module_array    ! 调用module_array模块

    implicit none       ! 变量必须显式声明，不允许隐式声明

    real*8, parameter :: omg = 3.924d-6! angular velocity of rh wave
    real*8, parameter :: fi0 = 78400d0 ! minimun potential height
    real*8, parameter :: r   = 4d0     ! wave number of rh wave
    !
    real*8            :: af,ai,aj,ak,al    ! working variable
    real*8            :: bf,bi,bj,fi,r1,r2 ! working variable
    real*8            :: cf,detar,u0,v0,u1 ! working variable
    !
    integer           :: i,j               ! working variable
    !
    detar=2*pi/p*r
    r1=r+1
    r2=r*r

    do j=2,n-1
        do i=2,np
            aj=c1(j)
            ai=s1(j)
            !---------------------- u(x,y,0) -------------------------
            ! 计算t0 时刻的纬向风场
```

```
            ak=aj**r
            al=aj*aj
            bi=i*detar-detar
            bj=dcos(bi)
            u1=aj+ak/aj*ai*ai*bj*r-ak*aj*bj
            u0=u1*a*omg
            !---------------------- v(x,y,0) ------------------------
            ! 计算t0 时刻的经向风场
            bj=dsin(bi)
            v0=-a*r*omg*ak/aj*ai*bj
            !---------------------- h(x,y,0) ------------------------
            ! 计算t0 时刻的位势高度场
            bj=dcos(bi*2)
            bi=dcos(bi)
            af=r1*al+2*r2-r-2-2*r2/al
            af=af*ak*ak
            af=af*omg*omg/4+omg*(omg0*2+omg)*al/2
            bf=r2+2*r+2-r1*r1*al
            bf=bf*ak*2*omg*(omg+omg0)
            bf=bf/r1/(r+2)
            cf=r1*al-r-2
            cf=cf*omg*omg*ak*ak/4
            fi=af+bf*bi+cf*bj
            fi=fi0+fi*a*a
            !-------------------------------------------------------------
            wh(i,j)=fi
            u(i,j)=u0
            v(i,j)=v0
        enddo
    enddo

    do j = 2, n                 ! 利用周期边界条件，数组首尾相等
        wh(1,j)=wh(np,j)        ! 设置的经纬网格左右的边界条件
        wh(np+1,j)=wh(2,j)
        !
        u(1,j)= u(np,j)
        u(np+1,j)= u(2,j)
        !
        v(1,j)= v(np,j)
        v(np+1,j)= v(2,j)
    end do
    !
    do i=1,np+1                 ! 将极点的风场设为0，位势高度
        fi = fi0                ! 设置经纬网格上下的边界条件
        wh(i,1)=fi
        wh(i,n)=fi
        u(i,1)=0
        u(i,n)=0
        v(i,1)=0
        v(i,n)=0
    enddo

    return
end subroutine haurwitz
```

7 main.f90

这是主程序，主要的功能是:

1. 根据相关参数的设置，确定初始条件的形式，是由 Rossby-Haurwitz 波来提供，还是由文件读入。
2. 调用 cs.f90 程序进行必要的参数设置

3. 调用 euler.f90 程序进行时间积分和迭代

4. 计算积分过程的质量和能量，并在屏幕上输出必要的提示信息

5. 将程序输出的结果存入相应的文件中

详细的说明请看下面程序代码中的注释。

```
!************************************************************************!
!                   The Barotropic Model on Arakawa A-grid                !
!                                by Bin Wang                              !
!                   with the implicit scheme of energy conservation       !
!************************************************************************!
program main
    use module_para                 ! 调用module_para模块
    use module_array                ! 调用module_array模块

    implicit none                   ! 变量必须显式声明，不允许隐式声明

    real*8          :: tener,tener0 ! total energy at tn and t0, respectively
    real*8          :: tmass,tmass0 ! total mass at tn and t0, respectively
    real*4, dimension(1:n1,1:n) :: pu,pv,ph     ! for grads saving
    real*8          :: ai,dt        ! working variables
    integer         :: tlp,iter,irecu,irecv,irech     ! working variables
    integer         :: i,j,iws,nt,nw,iwr,tt           ! working variables

    real*8, external:: inner ! a external function to calculate inner product

    tt = t0                 ! 定义初始时间
    iwr=(t1-tt)/tlo         ! 时间方向的运行步数
    tlp=t1-tt-tlo*iwr       ! 剩余的时间

    if (tlp.gt.0) then      ! 如果剩余的时间大于0
        iwr=iwr+1           ! 令时间方向iwr运行步数加1
    else                    ! 如果剩余的时间等于0
        tlp=tlo             ! 令剩余的时间为时间步长
    end if

    ! 打印时间相关信息，包括开始时间，结束时间和时间步长
    print *,'initial time is',tt
    print *,'final time is',t1
    print *,'time stepsize is',tlo

    ! 根据n0 标志位的信息打印相关说明
    ! n0=0 使用 rh waves 作为初始条件; n0=1,从文件读入初始场
    if (n0.eq.0) then
        print *,'this is a rh-wvae experiment,'
        print *,'i.e., the ic is rh-wave'
    else
        print *,'this is self-defined experiment,'
        print *,'i.e., the ic is read from files'
    end if

    ! 根据标志位nyn 来输出相关信息
    ! nyn==0  输出每一步的能量信息; nyn.ne.0 thalf=12h时间输出一次能量信息
    if (nyn.ne.0) then
        print *,'the energy... will be shown once 12 hours'
    else
        print *,'the energy... will be shown at each time step'
    end if

    ! parameter setting
    call cs                 ! 调用cs子程序计算相关参数值

    ! initial condition
    if (n0.eq.0) then
        ! haurwitz.f90 是一个子程序，在n0 这个参数等于0的情况下才会被调用，
```

```
    ! 是将波数为4的Rossby-Haurwitz波作为整个模拟的初始条件。
    ! 在这个子程序中，主要是设置了u, v, wh等的初值和边界条件。
    ! using rossby-haurwitz waves as initial condition
    call haurwitz
else
    ! read initial conditon from your files
    open(10,file=fu)     ! 打开初始场的纬向速度数据文件
    open(11,file=fv)     ! 打开初始场的经向速度数据文件
    open(12,file=fh)     ! 打开初始场的位势高度数据文件

    read(10,130)(( u(i,j),j=1,n),i=1,n1) ! 读纬向速度到u数组
    read(11,130)(( v(i,j),j=1,n),i=1,n1) ! 读经向速度到v数组
    read(12,130)((wh(i,j),j=1,n),i=1,n1) ! 读位势高度到h数组

    close(10)            ! 关闭文件
    close(11)
    close(12)
end if

call opf(np,n,'uu.dat',12) ! 分别打开文件uu.dat vv.dat hh.dat
call opf(np,n,'vv.dat',13)
call opf(np,n,'hh.dat',14)

irecu=0                  ! 初始的纪录数为0
irecv=0
irech=0

do j=1,n
    do i=2,np
        pu(i,j)=u(i,j)       ! 保存u,v, wh的初始值
        pv(i,j)=v(i,j)
        ph(i,j)=wh(i,j)
        ai=dsqrt(wh(i,j))    ! 进行坐标变换，计算公式中的h=sqrt(phi)
        wu(i,j)=u(i,j)*ai    ! 进行坐标变换，计算公式中的U=hu=u*sqrt(phi)
        wv(i,j)=v(i,j)*ai    ! 进行坐标变换，计算公式中的V=hv=v*sqrt(phi)
    end do
    pu(1,j)=pu(np,j)         ! 根据周期边界条件，赋值
    pv(1,j)=pv(np,j)
    ph(1,j)=ph(np,j)

    wu(1,j)=wu(np,j)
    wu(n1,j)=wu(2,j)

    wv(1,j)=wv(np,j)
    wv(n1,j)=wv(2,j)

    wh(1,j)=wh(np,j)
    wh(n1,j)=wh(2,j)
end do

call wr(pu,np,n,12,irecu)
call wr(pv,np,n,13,irecv)
call wr(ph,np,n,14,irech)

tener0=inner(wu,wv,wh,wu,wv,wh) ! 计算t0 时的能量，通过内积计算

tmass0 = 0                      ! 计算初始时的质量，初值设为0
do j=1,n                        ! 通过循环计算初始质量
    do i=2,np
        tmass0=tmass0+wh(i,j)*c2(j)
    end do
end do

print *,'the total energy is ',tener0 ! 打印初始能量和质量的信息
print *,'the total mass is ',tmass0

! 控制输出信息的变量，当nt=23时输出"-------------"和标题信息
! 根据下面的程序，实际上每次提示后输出11组信息
```

```
nt=23
dt=tlo                              ! 时间步长
nw=0                                ! 不知道干什么用的
!
print *,'the main part of this program has started'
! iwr 是时间方向的运行总步数
print *,'number of integration steps is',iwr

do iws=1,iwr                        ! iwr 为时间方向运行步数
    if (iws.eq.iwr) then
        dt=tlp                      ! 最后一步的时间步长？
        tt=tt+tlp                   ! 总时间累计tt 起始为t0
    else
        tt=tt+tlo
    end if

    nw=nw+1

    if (nt.eq.23) then
        print *,'
            ------------------------------------------------------------------------
            '
        print *,'       the energy           the total-mass        the
            iteration number'
        nt=1                        ! 打印提示信息，同时将nt重新置1
    end if
    !-------------------------------------------------
    !       the time integration
    !-------------------------------------------------
    call euler(dt,iter)

    do j=1,n                        ! 将通过euler子程序计算得到的wu,wv转化为u
        ,v
        do i=2,np
            ai=dsqrt(wh(i,j))       ! 计算公式中的sqrt(phi)=h
            u(i,j)=wu(i,j)/ai       ! 计算公式中的u=U/h
            v(i,j)=wv(i,j)/ai       ! 计算公式中的v=V/h
        end do
    end do
    !
    if ((tt-thalf.ge.0).and.(tt-thalf.lt.dt)) then
                                    ! 如果积分时间到了thalf
        do j=1,n                    ! 将u,v,wh的信息存入pu,pv,ph
            do i=2,np
                ai=dsqrt(wh(i,j))
                pu(i,j)=u(i,j)
                pv(i,j)=v(i,j)
                ph(i,j)=wh(i,j)
            end do                  ! 利用周期条件，首尾相等
            pu(1,j)=pu(np,j)
            pv(1,j)=pv(np,j)
            ph(1,j)=ph(np,j)
        end do
        !                           ! 调用wr子程序将pu pv ph 的信息写入文件
        call wr(pu,np,n,12,irecu)
        call wr(pv,np,n,12,irecu)
        call wr(ph,np,n,14,irech)
        !
    end if
    ! module_para 中nyn = 0, 即使如此，也是12小时输出一次
    ! 43200=12*60*60 如果if 条件成立，则tt是12h的倍数
    if ((nyn.eq.1).or.(int(tt/43200)*43200.eq.tt)) then
        tener=inner(wu,wv,wh,wu,wv,wh)      ! 调用inner计算能量值
        tmass = 0
        do j=1,n                            ! 利用循环计算质量的值
            do i=2,np
                tmass=tmass+wh(i,j)*c2(j)
            end do
```

```
            end do
            !
            if (nyn.eq.0) then  ! 如果nyn .eq. 0, 则打印积分时间信息
                print *,'       (the integral time is      ',tt,')'
                nt=nt+1         ! nt + 1
            endif
            !
            print *,tener,tmass,iter ! 打印当前时刻的总能量，
                总质量和迭代次数信息
            nt=nt+1
        end if
    end do
    !------------------------ main part of the program end-----------
    print *,'the main part of this program has ended'
    !------------------------ out put the u,v,h ---------------------
    print *,'now,the working is to output result'

    open(1,file=ffu)      ! 打开文件ui.dat, 准备写入u 的结果
    open(2,file=ffv)      ! 打开文件vi.dat, 准备写入v 的结果
    open(3,file=ffh)      ! 打开文件hi.dat, 准备写入wh(=phi) 的结果
    !
    130 format(200f16.8) ! 200个浮点数(其中，总共占16位，小数部分占8位)
    ! 分别将计算的最终结果u v wh 写入打开的文件
    write(1,130) (( u(i,j),j=1,n),i=1,n1)
    write(2,130) (( v(i,j),j=1,n),i=1,n1)
    write(3,130) ((wh(i,j),j=1,n),i=1,n1)

    close(1)              ! 关闭三个文件
    close(2)
    close(3)
    !
    do j=1,n              ! 将u v wh等信息保存到pu pv ph
        do i=2,np
            pu(i,j)=u(i,j)
            pv(i,j)=v(i,j)
            ph(i,j)=wh(i,j)
        end do
        pu(1,j)=pu(np,j)
        pv(1,j)=pv(np,j)
        ph(1,j)=ph(np,j)
    end do
    !
    call wr(pu,np,n,12,irecu) ! 将pu pv ph 写入文件
    call wr(pv,np,n,12,irecu)
    call wr(ph,np,n,14,irech)
    !
    close(12)                 ! 关闭文件
    close(14)
    !
    stop
end

subroutine opf(nx,ny,ffn,nffn)              ! 子程序,打开文件
    character*6 :: ffn
    open(unit=nffn,file=ffn,form='unformatted',access='direct',recl=nx*ny*4)
    return
end subroutine opf

subroutine wr(rdata,nx,ny,nffn,irec)        ! 子程序，将数据写入文件
    dimension rdata(nx+1,ny)
    irec = irec + 1
    write(unit=nffn,rec=irec)((rdata(i,j),i=1,nx),j=1,ny)
    return
end subroutine wr

function inner(u1,v1,h1,u2,v2,h2)           ! 自定义求内积的函数
    use module_para                         ! 调用module_para
    implicit none                           ! 不允许隐式声明变量
```

```
    real*8, dimension(1:n1,1:n) :: u1,v1,h1
    real*8, dimension(1:n1,1:n) :: u2,v2,h2
    real*8                      :: inner     ! 返回值
    integer                     :: i,j

    inner = 0.0d0               ! inner初值为0,相加使用
    do j=1,n                    ! 根据fortran存储特点，最外层按列循环
        do i=2,np               ! 内层按行循环
            inner=inner+(u1(i,j)*u2(i,j)+v1(i,j)*v2(i,j)+h1(i,j)*h2(i,j))*c2(
                j)
            ! 为什么要乘以cos(theta)?
        end do
    end do
    return
end function inner
```

8 总结

从这个程序中我们可以学习到一些写程序的好习惯，比如说尽量采用模块化的思路来写，可以将整个程序都要用到的变量的定义放到 module 中，比如在这里将一些要用的参数和数组分别放到了 module_para.f90 和 module_array.f90 中；可以写一些子程序和自定义函数来实现部分功能，比如说这个程序通过 cs.f90 来设置参数，通过 euler.f90 来进行时间积分过程的计算，在 euler.f90 子程序的内部，又调用 dif.f90 中的各种差分程序来计算纬向和经向两个方向的差分，调用 lu0 子程序来用 LU 分解的方法求解线性代数方程组。

读这个程序最大的感受是变量名的定义很重要，因为我费了很大的劲在找程序中变量名字与代码手册中各个变量的对应关系。变量名字长一点的话，可能会比较容易看懂。