SOFTWARE FOR THE STAGGERED AND UNSTAGGERED TURKEL-ZWAS SCHEMES FOR THE SHALLOW WATER EQUATIONS ON THE SPHERE

F. X. Giraldo
NRC Reasearch Associate
Department of Mathematics
Naval Postgraduate School
Code MA/Fg
Monterey, California 93943

B. Neta
Department of Mathematics
Naval Postgraduate School
Code MA/Nd
Monterey, California 93943

December 3, 1995

KEYWORDS: shallow water equations; finite differences; Turkel-Zwas scheme; spherical coordinates; staggering.

SUMMARY

A linear analysis of the shallow water equations in spherical coordinates for the Turkel-Zwas[1] explicit large time-step scheme was presented by Neta, Giraldo and Navon[2]. This report presents the software developed to test the staggered[2] as well as the unstaggered[1] Turkel-Zwas scheme for the solution of the shallow water equations on the sphere.

1. INTRODUCTION

In this paper we present the software developed for the solution of the shallow water equations in spherical coordinates. The unstaggered (original) Turkel-Zwas scheme[1] and the staggered[2] one are both given.

The shallow water equations in spherical coordinates are given by

$$\frac{\partial u}{\partial t} + \frac{1}{a\cos\theta}\left[u\frac{\partial u}{\partial \lambda} + v\cos\theta\frac{\partial u}{\partial \theta}\right] - (f + \frac{u}{a}\tan\theta)v + \frac{g}{a\cos\theta}\frac{\partial h}{\partial \lambda} = 0 \tag{1}$$

$$\frac{\partial v}{\partial t} + \frac{1}{a\cos\theta}\left[u\frac{\partial v}{\partial \lambda} + v\cos\theta\frac{\partial v}{\partial \theta}\right] + (f + \frac{u}{a}\tan\theta)u + \frac{g}{a}\frac{\partial h}{\partial \theta} = 0 \tag{2}$$

$$\frac{\partial h}{\partial t} + \frac{1}{a\cos\theta}\left[\frac{\partial}{\partial \lambda}(hu) + \frac{\partial}{\partial \theta}(hv\cos\theta)\right] = 0\ . \tag{3}$$

Here, f is the Coriolis parameter given by

$$f = 2\Omega\sin\theta \tag{4}$$

where Ω is the angular speed of the rotation of the earth, h is the height of the homogeneous atmosphere, u and v are the zonal and meridional wind components respectively, θ and λ are the latitudinal and longitudinal directions respectively, a is the radius of the earth, and g is the gravitational constant.

In section 2 we present the unstaggered scheme (modified as suggested by Neta[3]). In section 3 we present the staggered method as developed by Neta, Giraldo and Navon[2]. In section 4 we present the input file required including a logical parameter to choose between the staggered and unstaggered versions. In section 5 we present the code developed.

2. UNSTAGGERED TURKEL-ZWAS SCHEME

The Turkel-Zwas scheme for the nonlinear shallow water equations in spherical coordinates takes the following form:

$$
\begin{aligned}
u_{k,j}^{\ell+1} = u_{k,j}^{\ell-1} \quad & -\sigma\left[\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}(u_{k+1,j}^{\ell} - u_{k-1,j}^{\ell}) + v_{k,j}^{\ell}(u_{k,j+1}^{\ell} - u_{k,j-1}^{\ell}) \right. \\
& \left. + \tfrac{g}{p\cos\theta_j}(h_{k+p,j}^{\ell} - h_{k-p,j}^{\ell})\right] \\
& +2\Delta t\left[(1-\alpha)(f_j + \tfrac{u_{k,j}^{\ell}}{a}\tan\theta_j)v_{k,j}^{\ell} \right. \\
& + \tfrac{\alpha}{2}(f_j + \tfrac{u_{k+p,j}^{\ell}}{a}\tan\theta_j)v_{k+p,j}^{\ell} \\
& \left. + \tfrac{\alpha}{2}(f_j + \tfrac{u_{k-p,j}^{\ell}}{a}\tan\theta_j)v_{k-p,j}^{\ell}\right]
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
v_{k,j}^{\ell+1} = v_{k,j}^{\ell-1} \quad & -\sigma\left[\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}\left(v_{k+1,j}^{\ell} - v_{k-1,j}^{\ell}\right) + v_{k,j}^{\ell}\left(v_{k,j+1}^{\ell} - v_{k,j-1}^{\ell}\right) \right. \\
& \left. + \tfrac{g}{q}\left(h_{k,j+q}^{\ell} - h_{k,j-q}^{\ell}\right)\right] \\
& -2\Delta t\left[(1-\alpha)\left(f_j + \tfrac{u_{k,j}^{\ell}}{a}\tan\theta_j\right)u_{k,j}^{\ell} \right. \\
& + \tfrac{\alpha}{2}\left(f_{j+q} + \tfrac{u_{k,j+q}^{\ell}}{a}\tan\theta_{j+q}\right)u_{k,j+q}^{\ell} \\
& \left. + \tfrac{\alpha}{2}\left(f_{j-q} + \tfrac{u_{k,j-q}^{\ell}}{a}\tan\theta_{j-q}\right)u_{k,j-q}^{\ell}\right]
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
h_{k,j}^{\ell+1} = h_{k,j}^{\ell-1} \quad & -\sigma\left\{\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}\left(h_{k+1,j}^{\ell} - h_{k-1,j}^{\ell}\right) + v_{k,j}^{\ell}\left(h_{k,j+1}^{\ell} - h_{k,j-1}^{\ell}\right) \right. \\
& + \tfrac{h_{k,j}^{\ell}}{\cos\theta_j}\left[(1-\alpha)\left(u_{k+p,j}^{\ell} - u_{k-p,j}^{\ell}\right) \right. \\
& \left. + \tfrac{\alpha}{2}\left(u_{k+p,j+q}^{\ell} - u_{k-p,j+q}^{\ell} + u_{k+p,j-q}^{\ell} - u_{k-p,j-q}^{\ell}\right)\right]\tfrac{1}{p} \\
& + \tfrac{h_{k,j}^{\ell}}{\cos\theta_j}\left[(1-\alpha)\left(v_{k,j+q}^{\ell}\cos\theta_{j+q} - v_{k,j-q}^{\ell}\cos\theta_{j-q}\right) \right. \\
& + \tfrac{\alpha}{2}\left(v_{k+p,j+q}^{\ell}\cos\theta_{j+q} - v_{k+p,j-q}^{\ell}\cos\theta_{j-q} \right. \\
& \left.\left.\left. + v_{k-p,j+q}\cos\theta_{j+q} - v_{k-p,j-q}\cos\theta_{j-q}\right)\right]\tfrac{1}{q}\right\}
\end{aligned} \tag{7}
$$

where

$$\sigma = \frac{\Delta t}{a\Delta\lambda} = \frac{\Delta t}{a\Delta\theta}. \tag{8}$$

For $\alpha = \frac{1}{3}$ the geostrophic balance and the incompressibility condition are satisfied to a higher order in the Cartesian coordinate case (See Turkel and Zwas,[1] Navon and de Villiers[5]).

Note that there is a typo in equation (11a) of Turkel-Zwas[1] which is our equation (5). We have also modified (to get a symmetric approximation as suggested by Neta[4] for a rectangular domain) the right hand side of (11c) in Turkel-Zwas[1] which is (7) here.

3. STAGGERED TURKEL-ZWAS SCHEME

The staggered version of the Turkel-Zwas scheme as proposed by Neta, Giraldo and Navon[2] takes the following form:

$$\begin{aligned}
u_{k,j}^{\ell+1} = u_{k,j}^{\ell-1} \quad & -\sigma\Big[\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}(u_{k+1,j}^{\ell}-u_{k-1,j}^{\ell})+v_{k,j}^{\ell}(u_{k,j+1}^{\ell}-u_{k,j-1}^{\ell}) \\
& +\tfrac{2g}{p\cos\theta_j}(h_{k+\frac{p}{2},j}^{\ell}-h_{k-\frac{p}{2},j}^{\ell})\Big] \\
& +2\Delta t\Big[(1-\alpha)(f_j+\tfrac{u_{k,j}^{\ell}}{a}\tan\theta_j)v_{k,j}^{\ell} \\
& +\tfrac{\alpha}{2}(f_j+\tfrac{u_{k+\frac{p}{2},j}^{\ell}}{a}\tan\theta_j)v_{k+\frac{p}{2},j}^{\ell} \\
& +\tfrac{\alpha}{2}(f_j+\tfrac{u_{k-\frac{p}{2},j}^{\ell}}{a}\tan\theta_j)v_{k-\frac{p}{2},j}^{\ell}\Big]
\end{aligned} \tag{9}$$

$$\begin{aligned}
v_{k,j}^{\ell+1} = v_{k,j}^{\ell-1} \quad & -\sigma\left[\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}\left(v_{k+1,j}^{\ell}-v_{k-1,j}^{\ell}\right)+v_{k,j}^{\ell}\left(v_{k,j+1}^{\ell}-v_{k,j-1}^{\ell}\right)\right. \\
& \left.+\tfrac{2g}{q}\left(h_{k,j+\frac{q}{2}}^{\ell}-h_{k,j-\frac{q}{2}}^{\ell}\right)\right] \\
& -2\Delta t\left[(1-\alpha)\left(f_j+\tfrac{u_{k,j}^{\ell}}{a}\tan\theta_j\right)u_{k,j}^{\ell}\right. \\
& +\tfrac{\alpha}{2}\left(f_{j+\frac{q}{2}}+\frac{u_{k,j+\frac{q}{2}}^{\ell}}{a}\tan\theta_{j+\frac{q}{2}}\right)u_{k,j+\frac{q}{2}}^{\ell} \\
& \left.+\tfrac{\alpha}{2}\left(f_{j-\frac{q}{2}}+\frac{u_{k,j-\frac{q}{2}}^{\ell}}{a}\tan\theta_{j-\frac{q}{2}}\right)u_{k,j-\frac{q}{2}}^{\ell}\right]
\end{aligned} \tag{10}$$

$$\begin{aligned}
h_{k,j}^{\ell+1} = h_{k,j}^{\ell-1} \quad & -\sigma\left\{\tfrac{u_{k,j}^{\ell}}{\cos\theta_j}\left(h_{k+1,j}^{\ell}-h_{k-1,j}^{\ell}\right)+v_{k,j}^{\ell}\left(h_{k,j+1}^{\ell}-h_{k,j-1}^{\ell}\right)\right. \\
& +\tfrac{2h_{k,j}^{\ell}}{\cos\theta_j}\left[(1-\alpha)\left(u_{k+\frac{p}{2},j}^{\ell}-u_{k-\frac{p}{2},j}^{\ell}\right)\right. \\
& \left.+\tfrac{\alpha}{2}\left(u_{k+\frac{p}{2},j+\frac{q}{2}}^{\ell}-u_{k-\frac{p}{2},j+\frac{q}{2}}^{\ell}+u_{k+\frac{p}{2},j-\frac{q}{2}}^{\ell}-u_{k-\frac{p}{2},j-\frac{q}{2}}^{\ell}\right)\right]\tfrac{1}{p} \\
& +\tfrac{2h_{k,j}^{\ell}}{\cos\theta_j}\left[(1-\alpha)\left(v_{k,j+\frac{q}{2}}^{\ell}\cos\theta_{j+\frac{q}{2}}-v_{k,j-\frac{q}{2}}^{\ell}\cos\theta_{j-\frac{q}{2}}\right)\right. \\
& +\tfrac{\alpha}{2}\left(v_{k+\frac{p}{2},j+\frac{q}{2}}^{\ell}\cos\theta_{j+\frac{q}{2}}-v_{k+\frac{p}{2},j-\frac{q}{2}}^{\ell}\cos\theta_{j-\frac{q}{2}}\right. \\
& \left.\left.\left.+v_{k-\frac{p}{2},j+\frac{q}{2}}\cos\theta_{j+\frac{q}{2}}-v_{k-\frac{p}{2},j-\frac{q}{2}}\cos\theta_{j-\frac{q}{2}}\right)\right]\tfrac{1}{q}\right\}
\end{aligned} \tag{11}$$

where σ is given by (8).

4. INPUT

The input file contains four lines. The first input line contains 2 integers:

nx = number of longitudinal points

ny = number of latitudinal points.

The second one contains 3 integers:

dt = time step in seconds

$time_{final}$ = final time in hours

iplot = number of iterations per plot

The third input line contains 2 integers and a real number:

p = stencil in longitudinal direction

q = stencil in latitudinal direction

alf=Pade-type differencing weighting factor

The last input line contains 2 logical variables:

pstag = staggering in p if .true.

qstag = staggering in q if .true.

For example:

```
64 32
100     24  100000
1 1 0.0
.false. .false.
```

5. CODE

```
*-----------------------------------------------------------------------*
*These lines of code contain the parameter statements for the global
*definitions of many important parameters.
*-----------------------------------------------------------------------*
      implicit real*8(a-h,o-z)
      parameter ( imax=128, jmax=64 )
      parameter ( mx=imax*jmax, mxpoi=mx, mxele=mx, mxbou=mx/5, nd=4 )
      parameter ( tol=1.0e-6, g=10.0, rk=0.1 )


*-----------------------------------------------------------------------*
*-----------------------------------------------------------------------*
*This program solves the Shallow Water Equations
*on a sphere with Periodic B.C.'s in the latitudinal direction (theta)
*and longitudinal direction (lambda) using a
*Staggered Turkel-Zwas Scheme as suggested by B. Neta.
*Derivatives are obtained via 2nd order differencing with some matching
*conditions developed by F.X. Giraldo to satisfy continuous derivatives
*across the poles.
*Written by F.X. Giraldo on 10/95
*           NRC Fellow
*           Department of Mathematics
*           Naval Postgraduate School
*           Monterey, CA 93940
*-----------------------------------------------------------------------*
*-----------------------------------------------------------------------*
      program nturkel
      include 'param.h'
                            !global matrices
      real taray(2)
```

```
      dimension f(mxpoi)
      dimension coord(mxpoi,2)
      integer node(imax,jmax), p, q
      logical pstag, qstag
                        !***primitive variables arrays***
                            !u velocity arrays
      dimension um(mxpoi),     u0(mxpoi),    up(mxpoi), ui(mxpoi)
                            !v velocity arrays
      dimension vm(mxpoi),     v0(mxpoi),    vp(mxpoi), vi(mxpoi)
                            !phi arrays
      dimension phim(mxpoi),   phi0(mxpoi),  phip(mxpoi), phii(mxpoi)
                            !Read the Input Variables and create the Grid
      call init(phi0,u0,v0,phii,ui,vi,node,coord,f,
     $          npoin,xmin,xmax,ymin,ymax,comega,nx,ny,dx,dy,dt,
     $          ntime,rade,iplot,omega,alpha,velmax,cfl,p,q,alf,
     $          pstag,qstag)
                           !Calculate Total Available Potential Energy
      call energy(ae,ae0,phi0,u0,v0,npoin,time)
      write(*,'("      Energy = ",e12.4)')ae
      ae0=ae

      time=0.0
      pi=4.0*atan(1.0)
      open(1,file='phi.out')
      open(2,file='u.out')
      open(3,file='v.out')
      if (mod(ntime,iplot).eq.0) then
         isets=ntime/iplot + 1
      else
         isets=ntime/iplot + 2
      endif
      write(1,*)isets
```

```
 write(2,*)isets
 write(3,*)isets
 call output(phi0,u0,v0,npoin,time,nx,ny,phi_mean)
                 !***TIME MARCH
 time1=dtime(taray)
                      !Do itime=1 Eulerian steps
 do itime=1,ntime
    time=time + dt
    ttime=time/(3600.0)
    write(*,'(" timestep time (hours) = ",i5,2x,e12.4)')itime,ttime
    if (itime.eq.1) then
       call matsuno(phim,phi0,phip,um,u0,up,vm,v0,vp,
$                   coord,f,npoin,dt,dx,dy,node,nx,ny,rade,comega,
$                   alpha,p,q,alf)
    else
       call tzstag(phim,phi0,phip,um,u0,up,vm,v0,vp,coord,
$                    f,npoin,dt,dx,dy,node,nx,ny,rade,comega,
$                    alpha,p,q,alf,pstag,qstag)
    endif
    call sfilter(phip,up,vp,node,nx,ny,dx,dy)
    call time_filter(phim,phi0,phip,um,u0,up,vm,v0,vp,npoin,
$                       itime)
    call update(phim,phi0,phip,um,u0,up,vm,v0,vp,npoin)
    if (mod(itime,iplot).eq.0)
$      call output(phi0,u0,v0,npoin,time,nx,ny,phi_mean)
    call energy(ae,ae0,phi0,u0,v0,npoin,time)
    write(*,'("      Energy = ",e12.4)')ae
 end do

 time2=etime(taray)
 tclock=(taray(1)+taray(2))
 write(*,'(" Total CPU time in seconds = ",e12.4)')tclock
```

```
                          !Check time for printing output
      if (mod(ntime,iplot).ne.0)
     $    call output(phi0,u0,v0,npoin,time,nx,ny,phi_mean)
      close(1)
                           !Compute the L2 Error Norm
      call norm(phi0,u0,v0,phii,ui,vi,node,coord,dx,dy,nx,ny,
     $          phi_norm,u_norm)
      print*,' L2 NORM = ',phi_norm,u_norm
      print*,' dt dx dy velmax = ',dt,dx,dy,velmax
      print*,' ** CFL = ',cfl
      stop
      end
*----------------------------------------------------------------------*
*This subroutine calculates the Available Energy of the 2D Shallow Water
*Equations in spherical coordinates
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine energy(ae,ae0,phi,u,v,npoin,time)
      include 'param.h'
                !global arrays
      dimension phi(mxpoi), u(mxpoi), v(mxpoi)

      ae=0.0
                !loop thru the elements
      do ip=1,npoin
         vel2=u(ip)**2 + v(ip)**2
         ae=ae + (phi(ip))*vel2 + phi(ip)**2
      end do
      ae=ae/(2.0*g)
      if (time.gt.0.0) then
         if (ae.gt.1.1*ae0.or.ae.lt.0.9*ae0) then
```

```
            write(*,'("      *Fatal Error* 10% Init Energy Exceeded!")')
            write(*,'("      Current_Energy Initial_Energy= "
     $             ,2(1x,e12.4))')ae,ae0
         endif
      endif
      return
      end
*----------------------------------------------------------------------*
*This subroutine reads in the input file.
*The info read is:  the number of grid points in x and y (nx,ny),
*                    time step, final time, and time steps per plotting,
*                    p, q, alpha
*                    pstag, qstag.
*where .true. means that it is staggered and .false. means it is unstaggered.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine init(phi0,u0,v0,phii,ui,vi,node,coord,f,
     $                npoin,xmin,xmax,ymin,ymax,comega,nx,ny,dx,dy,dt,
     $                ntime,rade,iplot,omega,alpha,velmax,cfl,p,q,alf,
     $                pstag,qstag)
      include 'param.h'
      dimension coord(mxpoi,2)
      dimension phi0(mxpoi), u0(mxpoi), v0(mxpoi)
      dimension phii(mxpoi), ui(mxpoi), vi(mxpoi), f(mxpoi)
      integer node(imax,jmax), p, q
      logical pstag,qstag
                                  !Read Input File
      read(*,*)nx,ny
      read(*,*)dt,time_final,iplot
      read(*,*)p,q,alf
      read(*,*)pstag,qstag
                                  !check bounds
```

```
if (nx.gt.imax.or.ny.gt.jmax) then
   write(*,'(" Error! - Need to Enlarge IMAX and JMAX")')
   write(*,'(" nx ny imax jmax = ",4(i3,1x))')nx,ny,imax,jmax
   stop
endif
                              !Set some constants
pi=4.0*atan(1.0)
rade=6.37e+06
time_final=time_final*3600.0
ntime=nint(time_final/dt)
xmin=0.0
xmax=2.0*pi
ymin=-pi/2.0
ymax=pi/2.0
xl=xmax-xmin
yl=ymax-ymin
dx=xl/(nx)
dy=yl/(ny)
phi_mean=5.768e4
omega=20.0
comega=7.292e-05
velmax=-1e5
alpha_fcor=0.0
alpha=0.0
                            !set the Initial Conditions
ip=0
do j=1,ny
   olat=ymin + real(j-0.5)*dy
   do i=1,nx
      olon=xmin + real(i-0.5)*dx
      ip=ip+1
      node(i,j)=ip
```

```
            coord(ip,1)=olon
            coord(ip,2)=olat
            f(ip)=2.0*comega*( -cos(olon)*cos(olat)*sin(alpha_fcor) +
     $                          sin(olat)*cos(alpha_fcor) )
            u0(ip)=omega*sin(olon)*(sin(olat)**3 -
     $                              3*sin(olat)*cos(olat)**2)
            v0(ip)=omega*sin(olat)**2*cos(olon)
            phi0(ip)=phi_mean +
     $          2*comega*rade*omega*sin(olat)**3*cos(olat)*sin(olon)
            phii(ip)=phi0(ip)
            ui(ip)=u0(ip)
            vi(ip)=v0(ip)
            vel1=abs(u0(ip)) + abs(v0(ip)) + sqrt(2*phi0(ip))
            velmax=max(velmax,vel1)
         end do
      end do
      dl=sqrt(dx**2 + dy**2)
      cfl=dt*velmax/(dl*rade)
      print*,' dt dx dy velmax = ',dt,dx,dy,velmax
      print*,' ** CFL = ',cfl
      npoin=nx*ny

      return
      end
*----------------------------------------------------------------------*
*This subroutine solves the 2D Shallow Water Equations in Spherical
*Coordinates using a Staggered Turkel-Zwas Scheme.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine matsuno(phim,phi0,phip,um,u0,up,vm,v0,vp,
     $                   coord,f,npoin,dt,dx,dy,node,nx,ny,rade,comega,
     $                   alpha,p,q,alf)
```

```
include 'param.h'
dimension phim(mxpoi), phi0(mxpoi), phip(mxpoi)
dimension um(mxpoi), u0(mxpoi), up(mxpoi)
dimension vm(mxpoi), v0(mxpoi), vp(mxpoi)
dimension coord(mxpoi,2), f(mxpoi)
integer node(imax,jmax), p, q, ph, qh
          !Loop through the points and integrate using Forward Time
          !and Centered Space...
                     !Predictor Stage (forward Euler)
ph=p
qh=q
nxh=nx/2
do i=1,nx            !Loop through Longitudinal Nodes
   i1=i-1
   i2=i+1
   i3=i-p
   i4=i+p
   i3h=i-ph
   i4h=i+ph
                     !Longitudinal Periodicity
   if (i1.lt.1) i1=i1 + nx
   if (i2.gt.nx) i2=i2 - nx
                     !Longitudinal Periodicity -P's and +P's
   if (i3.lt.1) i3=i3 + nx
   if (i4.gt.nx) i4=i4 - nx
                     !Longitudinal Periodicity -P/2's and +P/2's
   if (i3h.lt.1) i3h=i3h + nx
   if (i4h.gt.nx) i4h=i4h - nx
                     !Loop through Latitudinal Nodes
   do j=1,ny
      j1=j-1
      j2=j+1
```

```
j3=j-q
j4=j+q
j3h=j-qh
j4h=j+qh
j1sign=1
j2sign=1
j3sign=1
j4sign=1
j3hsign=1
j4hsign=1
                  !South Pole Periodicity
ij1=i
if (j1.lt.1) then
   j1=1
   j1sign=-1
   ij1=ij1 + nxh
   if (ij1.gt.nx) ij1=ij1 - nx
endif
                  !North Pole Periodicity
ij2=i
if (j2.gt.ny) then
   j2=ny
   j2sign=-1
   ij2=ij2 + nxh
   if (ij2.gt.nx) ij2=ij2 - nx
endif
                  !South Pole Periodicity -Q's
ij3=i
ippj3=i4
impj3=i3
if (j3.lt.1) then
   j3=1 - j + q
```

```
    j3sign=-1
    ij3=ij3 + nxh
    ippj3=ippj3 + nxh
    impj3=impj3 + nxh
    if (ij3.gt.nx) ij3=ij3 - nx
    if (ippj3.gt.nx) ippj3=ippj3 - nx
    if (impj3.gt.nx) impj3=impj3 - nx
endif
                  !North Pole Periodicity +Q's
 ij4=i
 ippj4=i4
 impj4=i3
 if (j4.gt.ny) then
    j4=2*ny + 1 - j - q
    j4sign=-1
    ij4=ij4 + nxh
    ippj4=ippj4 + nxh
    impj4=impj4 + nxh
    if (ij4.gt.nx) ij4=ij4 - nx
    if (ippj4.gt.nx) ippj4=ippj4 - nx
    if (impj4.gt.nx) impj4=impj4 - nx
endif
                  !South Pole Periodicity -Q/2's
 ij3h=i
 ippj3h=i4h
 impj3h=i3h
 if (j3h.lt.1) then
    j3h=1 - j + qh
    j3hsign=-1
    ij3h=ij3h + nxh
    ippj3h=ippj3h + nxh
    impj3h=impj3h + nxh
```

```
        if (ij3h.gt.nx) ij3h=ij3h - nx
        if (ippj3h.gt.nx) ippj3h=ippj3h - nx
        if (impj3h.gt.nx) impj3h=impj3h - nx
    endif
                          !North Pole Periodicity +Q/2's
     ij4h=i
     ippj4h=i4h
     impj4h=i3h
     if (j4h.gt.ny) then
        j4h=2*ny + 1 - j - qh
        j4hsign=-1
        ij4h=ij4h + nxh
        ippj4h=ippj4h + nxh
        impj4h=impj4h + nxh
        if (ij4h.gt.nx) ij4h=ij4h - nx
        if (ippj4h.gt.nx) ippj4h=ippj4h - nx
        if (impj4h.gt.nx) impj4h=impj4h - nx
    endif
                           !Set up the Node Pointers
                                 !Centered Diff Grid Points
     ip=node(i,j)
     ip1=node(i1,j)
     ip2=node(i2,j)
     jp1=node(ij1,j1)
     jp2=node(ij2,j2)
                                 !Turkel-Zwas Grid Points
     ip3=node(i3,j)
     ip4=node(i4,j)
     jp3=node(ij3,j3)
     jp4=node(ij4,j4)
     ip3jp3=node(impj3,j3)
     ip4jp3=node(ippj3,j3)
```

```
      ip3jp4=node(impj4,j4)
      ip4jp4=node(ippj4,j4)
                        !Staggered Grid Points
      ip3h=node(i3h,j)
      ip4h=node(i4h,j)
      jp3h=node(ij3h,j3h)
      jp4h=node(ij4h,j4h)
      ip3hjp3h=node(impj3h,j3h)
      ip4hjp3h=node(ippj3h,j3h)
      ip3hjp4h=node(impj4h,j4h)
      ip4hjp4h=node(ippj4h,j4h)
                  !Longitudes and Latitudes
      olon=coord(ip,1)
      olat=coord(ip,2)
      olonpp=olon + p*dx
      olonmp=olon - p*dx
      olonpq=olon
      if (j4sign.eq.-1) olonpq=olonpq + pi
      olatpq=olat + q*dy
      olonmq=olon
      if (j3sign.eq.-1) olonmq=olonmq + pi
      olatmq=olat - q*dy
                 !Staggered Longitudes and Latitudes
      olonpqh=olon
      if (j4hsign.eq.-1) olonpqh=olonpqh + pi
      olatpqh=olat + qh*dy
      olonmqh=olon
      if (j3hsign.eq.-1) olonmqh=olonmqh + pi
      olatmqh=olat - qh*dy
                 !Coriolis Force
      fip=2*comega*( -cos(olon)*cos(olat)*sin(alpha) +
     $                sin(olat)*cos(alpha) )
```

```
      fip4=2*comega*( -cos(olonpp)*cos(olat)*sin(alpha) +
     $                   sin(olat)*cos(alpha) )
      fip3=2*comega*( -cos(olonmp)*cos(olat)*sin(alpha) +
     $                   sin(olat)*cos(alpha) )
      fjp4=2*comega*( -cos(olonpq)*cos(olatpq)*sin(alpha) +
     $                   sin(olatpq)*cos(alpha) )
      fjp3=2*comega*( -cos(olonmq)*cos(olatmq)*sin(alpha) +
     $                   sin(olatmq)*cos(alpha) )
      fip=f(ip)
      fip4=f(ip4)
      fip3=f(ip3)
      fjp4=f(jp4)
      fjp3=f(jp3)
                        !integrate PHI
      phip(ip)=phi0(ip)
     $      - dt*u0(ip)/(rade*cos(olat))*(phi0(ip2)-phi0(ip1))/(2*dx)
     $      - dt*v0(ip)/(rade)*(phi0(jp2)-phi0(jp1))/(2*dy)
     $      - dt*phi0(ip)/(rade*cos(olat))*(
     $      (1.0-alf)*( (u0(ip4h)-u0(ip3h))/(2*ph*dx) +
     $                  (j4hsign*v0(jp4h)*cos(olatpqh) -
     $                   j3hsign*v0(jp3h)*cos(olatmqh))/(2*qh*dy) ) +
     $alf/2*( (j4hsign*u0(ip4hjp4h)-j4hsign*u0(ip3hjp4h))/(2*ph*dx) +
     $        (j3hsign*u0(ip4hjp3h)-j3hsign*u0(ip3hjp3h))/(2*ph*dx) +
     $        (j4hsign*v0(ip4hjp4h)*cos(olatpqh) -
     $         j3hsign*v0(ip4hjp3h)*cos(olatmqh))/(2*qh*dy) +
     $        (j4hsign*v0(ip3hjp4h)*cos(olatpqh) -
     $         j3hsign*v0(ip3hjp3h)*cos(olatmqh))/(2*qh*dy) ) )

c             phip(ip)=phi0(ip)
                           !integrate U
      up(ip)=u0(ip)
     $        - dt*u0(ip)/(rade*cos(olat))*(u0(ip2)-u0(ip1))/(2*dx)
```

```
     $        - dt*v0(ip)/rade*(j2sign*u0(jp2)-j1sign*u0(jp1))/(2*dy)
     $        - dt/(rade*cos(olat))*(phi0(ip4h)-phi0(ip3h))/(2*ph*dx)
     $        + dt*(
     $             (1.0-alf)*(fip + u0(ip)/rade*tan(olat))*v0(ip) +
     $             alf/2*(fip4 + u0(ip4)/rade*tan(olat))*v0(ip4) +
     $             alf/2*(fip3 + u0(ip3)/rade*tan(olat))*v0(ip3) )

c            up(ip)=u0(ip)
                                 !integrate V
            vp(ip)=v0(ip)
     $        - dt*u0(ip)/(rade*cos(olat))*(v0(ip2)-v0(ip1))/(2*dx)
     $        - dt*v0(ip)/rade*(j2sign*v0(jp2)-j1sign*v0(jp1))/(2*dy)
     $        - dt/rade*( phi0(jp4h)-phi0(jp3h) )/(2*qh*dy)
     $        - dt*(
     $       (1.0-alf)*(fip + u0(ip)/rade*tan(olat))*u0(ip) +
     $       alf/2*(fjp4 +
     $              j4sign*u0(jp4)/rade*tan(olatpq))*j4sign*u0(jp4) +
     $       alf/2*(fjp3 +
     $              j3sign*u0(jp3)/rade*tan(olatmq))*j3sign*u0(jp3) )

c            vp(ip)=v0(ip)
         end do
      end do
      return
      end
*----------------------------------------------------------------------*
*This subroutine computes the L2 Norm
*for the Geopotential and Velocity using
*a Trapezoid Rule Integration.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine norm(phi0,u0,v0,phii,ui,vi,node,coord,dx,dy,nx,ny,
```

```
$                phi_norm,u_norm)
 include 'param.h'
 dimension phi0(mxpoi), u0(mxpoi), v0(mxpoi)
 dimension phii(mxpoi), ui(mxpoi), vi(mxpoi)
 dimension coord(mxpoi,2), phih(128,64), uh(128,64), vh(128,64)
 integer node(imax,jmax)

 pi=4.0*atan(1.0)
 open(10,file='phih.out')
 open(20,file='uh.out')
 open(30,file='vh.out')
 do j=1,64
    do i=1,128
       read(10,*)phih(i,j)
       read(20,*)uh(i,j)
       read(30,*)vh(i,j)
    end do
 end do
 close(10)
 close(20)
 close(30)
 do j=1,ny
    do i=1,nx
       ip=node(i,j)
       ui(ip)=uh(2*i-1,2*j-1)
       vi(ip)=vh(2*i-1,2*j-1)
       phii(ip)=phih(2*i-1,2*j-1)
    end do
 end do

 phi_top=0.0
 phi_bot=0.0
```

```
u_top=0.0
u_bot=0.0
do j=1,ny-1
   do i=1,nx-1
      i1=node(i,j)
      i2=node(i+1,j)
      i3=node(i+1,j+1)
      i4=node(i,j+1)
      olat1=coord(i1,2)
      olat2=coord(i2,2)
      olat3=coord(i3,2)
      olat4=coord(i4,2)
      phi1=(phi0(i1) - phii(i1))*cos(olat1)
      u1=(u0(i1) - ui(i1))*cos(olat1)
      v1=(v0(i1) - vi(i1))*cos(olat1)
      phi2=(phi0(i2) - phii(i2))*cos(olat2)
      u2=(u0(i2) - ui(i2))*cos(olat2)
      v2=(v0(i2) - vi(i2))*cos(olat2)
      phi3=(phi0(i3) - phii(i3))*cos(olat3)
      u3=(u0(i3) - ui(i3))*cos(olat3)
      v3=(v0(i3) - vi(i3))*cos(olat3)
      phi4=(phi0(i4) - phii(i4))*cos(olat4)
      u4=(u0(i4) - ui(i4))*cos(olat4)
      v4=(v0(i4) - vi(i4))*cos(olat4)
      phi=dx*dy*(phi1 + phi2 + phi3 + phi4)/4
      phie=dx*dy*(phii(i1) + phii(i2) + phii(i3) + phii(i4))/4
      u=dx*dy*(u1 + u2 + u3 + u4)/4
      ue=dx*dy*(ui(i1) + ui(i2) + ui(i3) + ui(i4))/4
      v=dx*dy*(v1 + v2 + v3 + v4)/4
      ve=dx*dy*(vi(i1) + vi(i2) + vi(i3) + vi(i4))/4

      phi_top=phi_top + ( phi )**2
```

```
    phi_bot=phi_bot + ( phie )**2
            u_top=u_top + ( u )**2 + ( v )**2
    u_bot=u_bot + ( ue )**2 + ( ve )**2
 end do
      end do
      phi_norm=1.0/(4.0*pi)*sqrt(phi_top/phi_bot)
      u_norm=1.0/(4.0*pi)*sqrt(u_top/u_bot)

      return
      end
*----------------------------------------------------------------------*
*This subroutine writes the output.  It is currently set only to
*print the geopotential and wind velocities at each node point.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine output(phi,u,v,npoin,time,nx,ny,phi_mean)
      include 'param.h'
      dimension phi(mxpoi), u(mxpoi), v(mxpoi)

      pi=4.0*atan(1.0)
      dtime=time/3600.0
      write(1,'(2(i6,1x),e16.8)')nx,ny,dtime
      write(1,'(e12.4)')(phi(ip), ip=1,npoin)
      write(2,'(2(i6,1x),e16.8)')nx,ny,dtime
      write(2,'(e12.4)')(u(ip), ip=1,npoin)
      write(3,'(2(i6,1x),e16.8)')nx,ny,dtime
      write(3,'(e12.4)')(v(ip), ip=1,npoin)
      return
      end
*----------------------------------------------------------------------*
*This subroutine performs the Robert time filtering using a
*Laplacian type time-diffusion term that smoothens the values spatially.
```

```
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine sfilter(phi0,u0,v0,node,nx,ny,dx,dy)
      include 'param.h'
      dimension phi0(mxpoi), phip(mxpoi)
      dimension u0(mxpoi), up(mxpoi)
      dimension v0(mxpoi), vp(mxpoi)
      integer node(imax,jmax)

      do i=1,nx
         i1=i-1
         i2=i+1
         if (i1.lt.1) i1=nx
         if (i2.gt.nx) i2=1
         do j=1,ny
            if (j.gt.2.or.j.lt.ny-1) goto 100
            j1=j-1
            j2=j+1

                                 !Set up the Node Pointers
            ip=node(i,j)
            ip1=node(i1,j)
            ip2=node(i2,j)
            jp1=node(i,j1)
            jp2=node(i,j2)

            phi0_xx=( phi0(ip2) - 2*phi0(ip) + phi0(ip1) )/(dx*dx)
            u0_xx=( u0(ip2) - 2*u0(ip) + u0(ip1) )/(dx*dx)
            v0_xx=( v0(ip2) - 2*v0(ip) + v0(ip1) )/(dx*dx)
            phi0_yy=( phi0(jp2) - 2*phi0(ip) + phi0(jp1) )/(dy*dy)
            u0_yy=( u0(jp2) - 2*u0(ip) + u0(jp1) )/(dy*dy)
            v0_yy=( v0(jp2) - 2*v0(ip) + v0(jp1) )/(dy*dy)
```

```
                    !South Pole Periodicity
          if (j1.lt.1) then
             j1=1
             ij1=i + nx/2
             if (ij1.gt.nx) ij1=ij1 - nx
             jp1=node(ij1,j1)
             phi0_yy=( phi0(jp2) - 2*phi0(ip) + phi0(jp1) )/(dy*dy)
             u0_yy=( u0(jp2) - 2*u0(ip) + u0(jp1) )/(dy*dy)
             v0_yy=( v0(jp2) - 2*v0(ip) + v0(jp1) )/(dy*dy)
          endif
                    !North Pole Periodicity
          if (j2.gt.ny) then
             j2=ny
             ij2=i + nx/2
             if (ij2.gt.nx) ij2=ij2 - nx
             jp2=node(ij2,j2)
             phi0_yy=( phi0(jp2) - 2*phi0(ip) + phi0(jp1) )/(dy*dy)
             u0_yy=( u0(jp2) - 2*u0(ip) + u0(jp1) )/(dy*dy)
             v0_yy=( v0(jp2) - 2*v0(ip) + v0(jp1) )/(dy*dy)
          endif

          phip(ip)=phi0(ip) + rk*( phi0_xx + phi0_yy )
          up(ip)=u0(ip) + rk*( u0_xx + u0_yy )
          vp(ip)=v0(ip) + rk*( v0_xx + v0_yy )
100       continue
       end do
    end do

    do i=1,nx
       do j=1,ny
          if (j.gt.2.or.j.lt.ny-1) goto 200
```

```
            ip=node(i,j)
            phi0(ip)=phip(ip)
            u0(ip)=up(ip)
            v0(ip)=vp(ip)
 200     continue
         end do
      end do

      return
      end
*----------------------------------------------------------------------*
*This subroutine performs the Robert time filtering using a
*Laplacian type time-diffusion term that smoothens the values temporally.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
      subroutine time_filter(phim,phi0,phip,um,u0,up,vm,v0,vp,npoin,
     $                       itime)
      include 'param.h'
      dimension phim(mxpoi), phi0(mxpoi), phip(mxpoi)
      dimension um(mxpoi), u0(mxpoi), up(mxpoi)
      dimension vm(mxpoi), v0(mxpoi), vp(mxpoi)

      if (itime.eq.2) then
         do ip=1,npoin
            phi0(ip)=phi0(ip) + rk*( phip(ip) - phi0(ip) )
            u0(ip)=u0(ip) + rk*( up(ip) - u0(ip) )
            v0(ip)=v0(ip) + rk*( vp(ip) - v0(ip) )
         end do
      else if (itime.gt.2) then
         do ip=1,npoin
            phi0(ip)=phi0(ip) + rk*( phip(ip) - 2*phi0(ip) + phim(ip) )
            u0(ip)=u0(ip) + rk*( up(ip) - 2*u0(ip) + um(ip) )
```

```
         v0(ip)=v0(ip) + rk*( vp(ip) - 2*v0(ip) + vm(ip) )
      end do
    endif
    return
    end
*----------------------------------------------------------------------*
*This subroutine solves the 2D Shallow Water Equations in Spherical
*Coordinates using a Staggered Turkel-Zwas Scheme.
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
    subroutine tzstag(phim,phi0,phip,um,u0,up,vm,v0,vp,coord,
   $                 f,npoin,dt,dx,dy,node,nx,ny,rade,comega,
   $                 alpha,p,q,alf,pstag,qstag)
    include 'param.h'
    dimension phim(mxpoi), phi0(mxpoi), phip(mxpoi)
    dimension um(mxpoi), u0(mxpoi), up(mxpoi)
    dimension vm(mxpoi), v0(mxpoi), vp(mxpoi)
    dimension coord(mxpoi,2), f(mxpoi)
    integer node(imax,jmax), p, q, ph, qh
    logical pstag, qstag
              !Loop through the points and integrate using Forward Time
              !and Centered Space...
                         !Predictor Stage (forward Euler)
    if (pstag) then
       ph=p/2
    else
       ph=p
    endif
    if (qstag) then
       qh=q/2
    else
       qh=q
```

```
endif
alfh=0
alfu=0
alfv=0
nxh=nx/2
do i=1,nx              !Loop through Longitudinal Nodes
   i1=i-1
   i2=i+1
   i3=i-p
   i4=i+p
   i3h=i-ph
   i4h=i+ph
                       !Longitudinal Periodicity
   if (i1.lt.1) i1=i1 + nx
   if (i2.gt.nx) i2=i2 - nx
                       !Longitudinal Periodicity -P's and +P's
   if (i3.lt.1) i3=i3 + nx
   if (i4.gt.nx) i4=i4 - nx
                       !Longitudinal Periodicity -P/2's and +P/2's
   if (i3h.lt.1) i3h=i3h + nx
   if (i4h.gt.nx) i4h=i4h - nx
                       !Loop through Latitudinal Nodes
   do j=1,ny
      j1=j-1
      j2=j+1
      j3=j-q
      j4=j+q
      j3h=j-qh
      j4h=j+qh
      j1sign=1
      j2sign=1
      j3sign=1
```

```
j4sign=1
j3hsign=1
j4hsign=1
                 !South Pole Periodicity
ij1=i
if (j1.lt.1) then
   j1=1
   j1sign=-1
   ij1=ij1 + nxh
   if (ij1.gt.nx) ij1=ij1 - nx
endif
                 !North Pole Periodicity
ij2=i
if (j2.gt.ny) then
   j2=ny
   j2sign=-1
   ij2=ij2 + nxh
   if (ij2.gt.nx) ij2=ij2 - nx
endif
                 !South Pole Periodicity -Q's
ij3=i
ippj3=i4
impj3=i3
if (j3.lt.1) then
   j3=1 - j + q
   j3sign=-1
   ij3=ij3 + nxh
   ippj3=ippj3 + nxh
   impj3=impj3 + nxh
   if (ij3.gt.nx) ij3=ij3 - nx
   if (ippj3.gt.nx) ippj3=ippj3 - nx
   if (impj3.gt.nx) impj3=impj3 - nx
```

```
endif
                  !North Pole Periodicity +Q's
 ij4=i
 ippj4=i4
 impj4=i3
 if (j4.gt.ny) then
    j4=2*ny + 1 - j - q
    j4sign=-1
    ij4=ij4 + nxh
    ippj4=ippj4 + nxh
    impj4=impj4 + nxh
    if (ij4.gt.nx) ij4=ij4 - nx
    if (ippj4.gt.nx) ippj4=ippj4 - nx
    if (impj4.gt.nx) impj4=impj4 - nx
endif
                  !South Pole Periodicity -Q/2's
 ij3h=i
 ippj3h=i4h
 impj3h=i3h
 if (j3h.lt.1) then
    j3h=1 - j + qh
    j3hsign=-1
    ij3h=ij3h + nxh
    ippj3h=ippj3h + nxh
    impj3h=impj3h + nxh
    if (ij3h.gt.nx) ij3h=ij3h - nx
    if (ippj3h.gt.nx) ippj3h=ippj3h - nx
    if (impj3h.gt.nx) impj3h=impj3h - nx
endif
                  !North Pole Periodicity +Q/2's
 ij4h=i
 ippj4h=i4h
```

```
 impj4h=i3h
 if (j4h.gt.ny) then
    j4h=2*ny + 1 - j - qh
    j4hsign=-1
    ij4h=ij4h + nxh
    ippj4h=ippj4h + nxh
    impj4h=impj4h + nxh
    if (ij4h.gt.nx) ij4h=ij4h - nx
    if (ippj4h.gt.nx) ippj4h=ippj4h - nx
    if (impj4h.gt.nx) impj4h=impj4h - nx
endif
                   !Set up the Node Pointers
                         !Centered Diff Grid Points
 ip=node(i,j)
 ip1=node(i1,j)
 ip2=node(i2,j)
 jp1=node(ij1,j1)
 jp2=node(ij2,j2)
                         !Turkel-Zwas Grid Points
 ip3=node(i3,j)
 ip4=node(i4,j)
 jp3=node(ij3,j3)
 jp4=node(ij4,j4)
 ip3jp3=node(impj3,j3)
 ip4jp3=node(ippj3,j3)
 ip3jp4=node(impj4,j4)
 ip4jp4=node(ippj4,j4)
                         !Staggered Grid Points
 ip3h=node(i3h,j)
 ip4h=node(i4h,j)
 jp3h=node(ij3h,j3h)
 jp4h=node(ij4h,j4h)
```

```
      ip3hjp3h=node(impj3h,j3h)
      ip4hjp3h=node(ippj3h,j3h)
      ip3hjp4h=node(impj4h,j4h)
      ip4hjp4h=node(ippj4h,j4h)
                         !Longitudes and Latitudes
      olon=coord(ip,1)
      olat=coord(ip,2)
      olatpq=olat + q*dy
      olatmq=olat - q*dy
                        !Staggered Longitudes and Latitudes
      olatpqh=olat + qh*dy
      olatmqh=olat - qh*dy
                        !Coriolis Force
      fip=f(ip)
      fip4=f(ip4)
      fip3=f(ip3)
      fjp4=f(jp4)
      fjp3=f(jp3)
                     !integrate PHI
      phip(ip)=phim(ip)
$         - dt*u0(ip)/(rade*cos(olat))*(phi0(ip2)-phi0(ip1))/dx
$         - dt*v0(ip)/(rade)*(phi0(jp2)-phi0(jp1))/dy
$         - dt*phi0(ip)/(rade*cos(olat))*(
$         (1.0-alf)*( (u0(ip4h)-u0(ip3h))/(ph*dx) +
$                     (j4hsign*v0(jp4h)*cos(olatpqh) -
$                      j3hsign*v0(jp3h)*cos(olatmqh))/(qh*dy) ) +
$    alf/2*( (j4hsign*u0(ip4hjp4h)-j4hsign*u0(ip3hjp4h))/(ph*dx) +
$           (j3hsign*u0(ip4hjp3h)-j3hsign*u0(ip3hjp3h))/(ph*dx) +
$           (j4hsign*v0(ip4hjp4h)*cos(olatpqh) -
$            j3hsign*v0(ip4hjp3h)*cos(olatmqh))/(qh*dy) +
$           (j4hsign*v0(ip3hjp4h)*cos(olatpqh) -
$            j3hsign*v0(ip3hjp3h)*cos(olatmqh))/(qh*dy) ) )
```

```
c             phip(ip)=phi0(ip)
                                  !integrate U
            up(ip)=um(ip)
     $        - dt*u0(ip)/(rade*cos(olat))*(u0(ip2)-u0(ip1))/dx
     $        - dt*v0(ip)/rade*(j2sign*u0(jp2)-j1sign*u0(jp1))/dy
     $        - dt/(rade*cos(olat))*(phi0(ip4h)-phi0(ip3h))/(ph*dx)
     $        + 2*dt*(
     $             (1.0-alf)*(fip + u0(ip)/rade*tan(olat))*v0(ip) +
     $             alf/2*(fip4 + u0(ip4)/rade*tan(olat))*v0(ip4) +
     $             alf/2*(fip3 + u0(ip3)/rade*tan(olat))*v0(ip3) )
c             up(ip)=u0(ip)
                                  !integrate V
            vp(ip)=vm(ip)
     $        - dt*u0(ip)/(rade*cos(olat))*(v0(ip2)-v0(ip1))/dx
     $        - dt*v0(ip)/rade*(j2sign*v0(jp2)-j1sign*v0(jp1))/dy
     $        - dt/rade*( phi0(jp4h)-phi0(jp3h) )/(qh*dy)
     $        - 2*dt*(
     $       (1.0-alfv)*(fip
     $                  + u0(ip)/rade*tan(olat))*u0(ip)
     $       + alfv/2*(fjp4
     $                + j4sign*u0(jp4)/rade*tan(olatpq))*j4sign*u0(jp4)
     $       + alfv/2*(fjp3
     $                + j3sign*u0(jp3)/rade*tan(olatmq))*j3sign*u0(jp3) )
c             vp(ip)=v0(ip)
         end do
      end do
      return
      end
*----------------------------------------------------------------------*
*This subroutine updates the arrays PHIM,UM,VM,PHI0,U0,V0,
*Written by F.X. Giraldo on 10/95
*----------------------------------------------------------------------*
```

```
subroutine update(phim,phi0,phip,um,u0,up,vm,v0,vp,npoin)
include 'param.h'
dimension phim(mxpoi), phi0(mxpoi), phip(mxpoi)
dimension um(mxpoi),   u0(mxpoi),   up(mxpoi)
dimension vm(mxpoi),   v0(mxpoi),   vp(mxpoi)
                    !Loop through all the nodes and update
do ip=1,npoin
                    !Update F(x-2*alpha,t-dt)=F(x-alpha,t)
   phim(ip)=phi0(ip)
   um(ip)=u0(ip)
   vm(ip)=v0(ip)
                    !Update F(x-alpha,t)=F(x,t+dt)
   phi0(ip)=phip(ip)
   u0(ip)=up(ip)
   v0(ip)=vp(ip)
end do

return
end
```

REFERENCES

1. E. Turkel and G. Zwas, Explicit large time-step schemes for the shallow water equations, in *Advances in Computer Methods for Partial Differential Equations*, R. Vichnevetsky and R.S. Stepleman (eds), IMACS, Lehigh University, 65–69 (1979).

2. B. Neta, F. X. Giraldo, and I. M. Navon, Analysis of the Turkel-Zwas scheme for the two-dimensional shallow water equations in spherical coordinates, submitted for publication (1995).

3. B. Neta and I. M. Navon, Analysis for the Turkel-Zwas scheme for the shallow water equations, *J. Comp. Phys.*, **81**, 277–299 (1989).

4. B. Neta, Analysis of the Turkel-Zwas scheme for the 2-D shallow water equations, IMACS *Transactions on Scientific Computing 1988, Vols. 1.1 and 1.2: Numerical and Applied Mathematics*, W. F. Ames and C. Brezinski (eds.) 1989.

5. I. M. Navon and R. deVilliers, The application of the T-Z explicit large time step scheme to a hemispheric barotropic model with constraint restoration, *Mon. Wea. Rev.*, **115**, 1036–1051 (1987).

Distribution List

	No. of copies
Director Defense Technology Information Center Cameron Station Alexandria, VA 22314	2
Dean of Research Code 09 Naval Postgraduate School Monterey, CA 93943	1
Library Code 52 Naval Postgraduate School Monterey, CA 93943	2
Department of Mathematics Code MA Naval Postgraduate School Monterey, CA 93943	1
Professor F. X. Giraldo Code MA/Fg Naval Postgraduate School Monterey, CA 93943	10

Professor Beny Neta 10
Code MA/Nd
Naval Postgraduate School
Monterey, CA 93943

Professor I. Michael Navon 1
Florida State University
SuperComputer Computation Research Institute
Tallahassee, FL 32306

Professor R. T. Williams 1
Code MR/Wu
Naval Postgraduate School
Monterey, CA 93943

Professor Melinda Peng 1
Code MR/Pg
Naval Postgraduate School
Monterey, CA 93943

Professor C. P. Katti 1
SC&SS
Jawaharlal Nehru University
New Delhi, 110067
INDIA

Lt. Chris Sagovac, USN 1
819 South Charles St.
Baltimore, MD 21230

Professor Zahari Zlatev 1
Department of Emissions and Air Pollution
National Environmental Res. Inst.
Frederiksborgvej 399
P. O. Box 358
DK-4000 Roskilde
DENMARK