Finite Difference and Pseudospectral Methods applied to the Shallow Water Equations in Spherical Coordinates

by

DAVID WARREN MERRILL

B.A., Bates College, 1992

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Masters of Arts

Department of Mathematics

1997

This thesis for the Masters of Arts degree by

David Warren Merrill

has been approved for the

Department of Mathematics

by

_____

Bengt Fornberg

_____

Richard Holley

_____

Arlan Ramsay

_____

Ruediger Jakob-Chien

Date _____

Merrill, David Warren (M.A., Mathematics)

Finite Difference and Pseudospectral Methods Applied to the Shallow Water Equations in Spherical Coordinates

Thesis directed by Professor Bengt Fornberg

The shallow water equations are a set of equations used to model many fluid flows. They are particularly well suited-and often used-to test numerical techniques for weather prediction. In this study we carry through two numerical test cases involving the shallow water equations. We do this using three different numerical methods for calculating derivatives; second and fourth order finite differences, and a pseudospectral method. We will show that, by using our pseudospectral method,

i. we get similar accuracy compared to a particular implementation of a spectral transform method based on spherical harmonic techniques and far higher accuracy than with finite differences (for the same two test cases),
ii. the operation count for each time step is much lower compared to the above mentioned spectral transform method,
iii. our longitude-latitude grid allows for a particularly easy formulation of the code and,
iv. pole singularities will not cause any difficulties.

Acknowledgments

I would like to express my sincere appreciation to Bengt Fornberg, my thesis advisor, for giving his time to me generously, always patiently explaining new and difficult concepts, and always being encouraging during the past year. I would also like to thank Ruediger Jakob-Chien for meeting with us every few weeks during the last six months. He gave us insights on how the test cases should work, helped find a bug in the Runge-Kutta code, and made many comments on various drafts of the thesis. I would like to thank my friends in the Applied Math Department, Rudy Horne, Christina Perez, Bernard Deconinck, and Chris Higginson, for letting me share my excitement and frustrations with them during the past year. Finally, I would like to thank my wife, Emma Holder, for putting up with the stress of preparing for the defense, thanks babe.

Table of Contents

TABLES

FIGURES

Finite Difference and Pseudospectral Methods applied to the Shallow Water Equations in Spherical Coordinates.

Chapter 1

Introduction

The shallow water equations are a set of equations used to model many fluid flows. They are particularly well suited-and often used-to test numerical techniques for weather prediction. In this thesis we carry through the first two test cases from Williamson [10]. We do this using three different numerical methods for calculating derivatives; second and fourth order finite differences, (FD2, FD4) and a Pseudospectral (PS) method. All methods are based on regular longitude-latitude grids. We will show that, by using our Pseudospectral method,

i. we get similar accuracy compared to a Williamson's [10] implementation of a spectral transform method based on spherical harmonic techniques and far higher accuracy compared to finite differences (for the same two test cases),
ii. the operation count for each time step is much lower compared to the above mentioned spectral transform method,
iii. our longitude-latitude grid allows for a particularly easy formulation of the code and,
iv. pole singularities will not cause any difficulties.

There have been various studies dealing with the shallow water equations and their numerical solution in a spherical geometry. In 1974, Merilees [6] tried to apply a pseudospectral method to the shallow water equations, but runs into some problems which we do not encounter. In

1995, Fornberg [2] completed a successful numerical analysis of a simpler set of fluid flow equations on a spherical geometry. This study further develops and tests ideas presented by Merilees and Fornberg.

In section 1.1, we describe a formulation of the shallow water equations in rectangular coordinates.

In section 1.2 we describe the numerical methods used to represent the sphere, calculate partial differential equations, smooth data, and complete an error analysis. We also describe how time stepping is done and provide an operation count (per time step) for the second test case (which involves the full set of shallow water equations).

In Chapter 2 we describe the first two test cases and results. For Case 1, a comparison of FD2, FD4 and PS methods is shown, and an error analysis is done and compared to Williamson's results. For Case 2, an error analysis is done.

Chapter 3 contains concluding remarks.

1.1 The Shallow Water Equations

Euler's Equations of Motion of an ideal fluid on a rectangular domain are as follows,

$$\frac{Du}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + fv \qquad (1.1)$$

$$\frac{Dv}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial y} - fu \qquad (1.2)$$

$$\frac{Dw}{Dt} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - g \qquad (1.3)$$

where $\frac{D}{Dt}(\ )$ is the *substantial* (or *total*) derivative which expands to

$$\frac{D}{Dt}(\ ) = \frac{\partial}{\partial t}(\ ) + u\frac{\partial}{\partial x}(\ ) + v\frac{\partial}{\partial y}(\ ) + w\frac{\partial}{\partial z}(\ ).$$

ρ is the density of the fluid, and p is the pressure. The Coriolis parameter f, is given by $f = 2\Omega\sin\theta$, (Ω is the angular velocity of the earth, θ is the latitude), g is the acceleration due to gravity and u and v are the speed of fluid in the x and y direction respectively.

The Hydrostatic Approximation

Using the hydrostatic approximation

$$\frac{\partial p}{\partial z} = -\rho g \qquad (1.4)$$

(note that this implies $\frac{Dw}{Dt} = 0$ in 1.3) we have then,

$$\frac{\partial}{\partial z}\left(\frac{\partial p}{\partial x}\right)=\frac{\partial}{\partial x}\left(\frac{\partial p}{\partial z}\right)=\frac{\partial}{\partial x}(-\rho g)=0$$
$$\frac{\partial}{\partial z}\left(\frac{\partial p}{\partial y}\right)=\frac{\partial}{\partial y}\left(\frac{\partial p}{\partial z}\right)=\frac{\partial}{\partial y}(-\rho g)=0$$

which implies that the pressure gradient force in the *x* and *y* directions are independent of height (or depth).

The Continuity Equation and Vertical motion

We assume the pressure, *p*, of our fluid is constant. This tells us that $\frac{\partial p}{\partial t}=0$ and

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0. \qquad (1.5)$$

We call equation (1.5) the continuity equation (or the incompressibility condition). By solving for $\frac{\partial w}{\partial z}$ and integrating with respect to *z*, we can come up with an expression for *w*.

$$\frac{\partial w}{\partial z}=-\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right) \qquad (1.6)$$

$$w=\int_0^h -\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)dz=-h\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right) \qquad (1.7)$$

The surface (of the fluid) boundary condition on *w* is that the fluid particles follow the surface. $\left(i.e.\ \frac{Dh}{Dt}=w\Big|_{\text{at the surface}}\right)$ Thus,

$$\frac{Dh}{Dt} = -h\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right). \qquad (1.8)$$

Expanding $\frac{Dh}{Dt}$ in (1.8) we get,

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} = -h\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right). \qquad (1.9)$$

Bringing $u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y}$ to the right hand side we get.

$$\frac{\partial h}{\partial t} = -\left(u\frac{\partial h}{\partial x} + h\frac{\partial u}{\partial x} + v\frac{\partial h}{\partial y} + h\frac{\partial v}{\partial y}\right). \qquad (1.10)$$

Which simplifies to

$$\frac{\partial h}{\partial t} = -\left[\frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y}\right]. \qquad (1.11)$$

The Pressure of the Fluid

To get an expression for the pressure in the fluid we integrate the hydrostatic equation (1.4) from $p = 0$ at the top, downward.

$$p(x,y,z) = \int_{h}^{z} -g\rho \, da = (h - z)\rho g \qquad (1.12)$$

If we then take the partial derivatives of p, (at the surface) with respect to x and y we will have

$$\frac{\partial p}{\partial x} = \frac{\partial}{\partial x}\big((h-z)\rho g\big) = \rho g \frac{\partial h}{\partial x} \;\Rightarrow\; -\frac{1}{\rho}\frac{\partial p}{\partial x} = -g\frac{\partial h}{\partial x}$$

and

$$\frac{\partial p}{\partial y} = \frac{\partial}{\partial y}\big((h-z)\rho g\big) = \rho g \frac{\partial h}{\partial y} \;\Rightarrow\; -\frac{1}{\rho}\frac{\partial p}{\partial y} = -g\frac{\partial h}{\partial y}.$$

Final formulation of the Shallow water equations

Taking everything into account we have then (in Cartesian coordinates):

$$\frac{Du}{Dt} = -g\frac{\partial h}{\partial x} + fv \qquad (1.13)$$

$$\frac{Dv}{Dt} = -g\frac{\partial h}{\partial y} - fu \qquad (1.14)$$

$$\frac{Dh}{Dt} = -h\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right] \qquad (1.15)$$

A full formulation of the Shallow Water Equations in spherical coordinates can be found in Holton [3]. The equations are as follows (as they are used in Williamson's [10] second case):

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

$$\frac{\partial}{\partial t}(v) + \frac{u}{a\cos\theta}\frac{\partial}{\partial \varphi}(v) + \frac{v}{a}\frac{\partial}{\partial \theta}(v) + \left(f + \frac{u\tan\theta}{a}\right)u + \frac{g}{a}\frac{\partial h}{\partial \theta} = 0$$

$$\frac{\partial}{\partial t}(h) + \frac{u}{a\cos\theta}\frac{\partial}{\partial \varphi}(h) + \frac{v}{a}\frac{\partial}{\partial \theta}(h) + \frac{h}{a\cos\theta}\left[\frac{\partial u}{\partial \varphi} + \frac{\partial(v\cos\theta)}{\partial \theta}\right] = 0$$

where f is the Coriolis parameter, g is the acceleration due to gravity, a is the mean radius of the sphere and u and v are the speed of fluid in the φ and θ direction respectively.

1.2 The Numerical Method

This study discusses Williamson's [10] first two test cases (which are described in detail in chapter 2). In those cases the functions that will be of importance are the height field h, and the wind velocities, u and v. This section describes the methods (FD2, FD4, and PS) by which derivatives of those variables were taken numerically. We use the same methods (subroutines) to take derivatives of each variable (with special consideration to whether the variable represents vector or scalar data). Because of this, throughout this section we use the dummy variable f to denote a function to be differentiated. Also, at various points, we need to describe our space and time steps. Throughout this section we will use h and k to denote the size of space and time steps respectively.

1.2.1 The Domain

We take the globe and map it to a two dimensional grid in (φ, θ) coordinates.

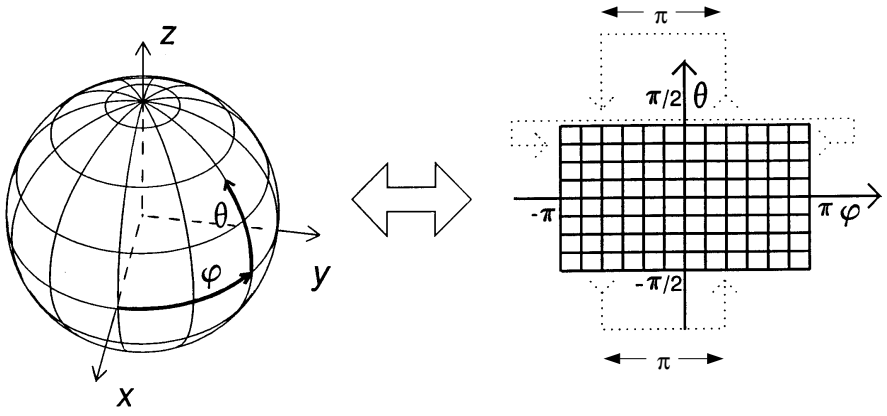


Figure 1.2.1-1. Unrolling the sphere onto a two dimensional grid.

For example, as one goes north over the pole from the first quadrant, one will come down south over the pole in the second quadrant. (the same idea applies as one goes over the south pole) In Fortran the grid is allocated as the following array.

```
HEIGHT(-2*M:2*M,1:2*M)
```

There are $4M+1$ grid points in the zonal (φ) direction and $2M$ grid points in the meridional (θ) direction.

1.2.2 Grid Point Equations

To describe φ and θ at each subscript (i, j) on the orthogonal grid we use the following notation (where h is the angular grid spacing).

$$h = \frac{\pi}{2M}$$

$$\varphi_i = ih,\ i = -2M,\ldots,2M \quad (\text{note } \varphi_{-2M} = \varphi_{2M})$$

$$\theta_j = \left[(j-1) - M + \frac{1}{2}\right]h,\ j = 1,\ldots 2M$$

Calculating φ and θ are straightforward. Table 1.2.2-1 shows sample output for values of θ for different values of j (for $M = 5$).

$\theta_1 \approx$ -1.4137	$\theta_6 \approx$ 0.1570
$\theta_2 \approx$ -1.0995	$\theta_7 \approx$ 0.4712
$\theta_3 \approx$ -0.7853	$\theta_8 \approx$ 0.7853
$\theta_4 \approx$ -0.4712	$\theta_9 \approx$ 1.0995
$\theta_5 \approx$ -0.1570	$\theta_{10} \approx$ 1.4137

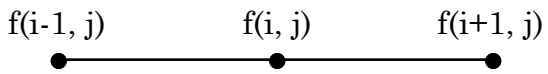
Table 1.2.2-1. Latitude at each grid row with $M = 5$.

This gives us $2M$ grid points across $-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$ without having grid points at the poles or the equator (M grid points between $\theta = 0$ and $\theta = \frac{\pi}{2}$ and M grid points between $\theta = 0$ and $\theta = -\frac{\pi}{2}$). This was done to help avoid singularities at the poles in the spherical coordinate version of the shallow water equations (where terms like $\frac{1}{\cos\theta}$ often crop up).

1.2.3 Spatial Second Order Finite Differences

ZonalDerivatives

To take φ derivatives we have the following stencil, and second order finite difference approximation,



$$\left.\frac{\partial f}{\partial \varphi}\right|_{(\varphi_i,\theta_j)} \approx \frac{f(i+1,j) - f(i-1,j)}{2h}$$

Figure 1.2.3-1. FD2 scheme for φ derivatives.

where $h = \dfrac{\pi}{2M}$, is the grid point spacing.

We need to use the following code, which handles the interior of the grid plus the left and right edges. Since the left and right edges represent the same data we only calculate the values at the right edge, and set the left edge equal to these values. (here M2 = 2*M)

```
      DO 24 J=1,M2
C--- Do right edge and then set left edge = to right edge.
        ANS( M2,J) = (F(-M2+1,J)-F(M2-1,J))*H2
        ANS(-M2,J) = ANS(M2,J)
```

```
C--- Do the interior and top and bottom of grid.

            DO 24 I=-M2+1,M2-1
   24          ANS(I,J) = (F(I+1,J)-F(I-1,J))*H2
```

Notice how the derivatives on the edge are calculated.

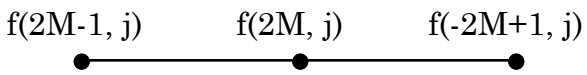
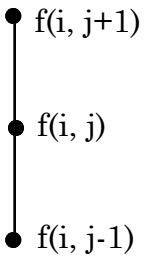


Figure 1.2.3-2. FD2 edge φ derivatives.

(-2M+1, J) is the grid point to the right of (2M, J) but it is on the other side of the grid.

MeridionalDerivatives

To take θ derivatives we have the following stencil, and second order finite difference approximation,



$$\left.\frac{\partial f}{\partial \theta}\right|_{(\varphi_i,\theta_j)} \approx \frac{f(i,j+1)-f(i,j-1)}{2h}$$

Figure 1.2.3-3. FD2 scheme for θ derivatives.

where h is defined as before.

We need to use the following code, which handles the interior of the grid plus the left and right edges :

```
         DO 22 I=-M2,M2
C--- Do interior, and left and right edges.
            DO 22 J=2,M2-1
               ANS(I,J)   = ( F(I,J+1) - F(I,J-1) )*H2
   22    CONTINUE
```

To calculate θ derivatives at the top and bottom of the grid we want to keep in mind that the grid point on the other side of the pole is π. radians ($2M$ grid points) away from the rest of the grid points in the stencil. For example imagine traveling northwards along the line of longitude where $\varphi = -\pi$. As we pass over the pole we would be traveling southward along the line of longitude where $\varphi = 0$. Another example would be if we were traveling northwards along the line of longitude where $\varphi = \frac{\pi}{4}$. As we pass over the pole we would be traveling southward along the line of longitude where $\varphi = -\frac{3\pi}{4}$. To code this, we write the following.

```
         DO 22 I=-M2,M2
C--- Calculate index on other side of pole.
            IF (I.LE.0) THEN
               IC = I+M2         ! M2 corresponds to PI.
            ELSE
               IC = I-M2         ! M2 corresponds to PI.
            ENDIF
C--- Do top and bottom of grid.
            ANS(I,M2) = ( VEC*F(IC,M2) - F(I,M2-1) )*H2
            ANS(I, 1) = ( F(I, 2)  - VEC*F(IC, 1) )*H2
   22    CONTINUE
```

In each calculation of ANS(I, M2) and ANS(I, 1) a VEC term being multiplied to F(IC, M2) and F(IC, 1) respectively. VEC has the value 1 or -1 depending on whether we are taking derivatives of scalar or vector quantities.

Here are a couple of pictures to help visualize taking derivatives along the top and bottom of the grid. (in this case $M = 3$, thus $M2 = 2*M = 6$)

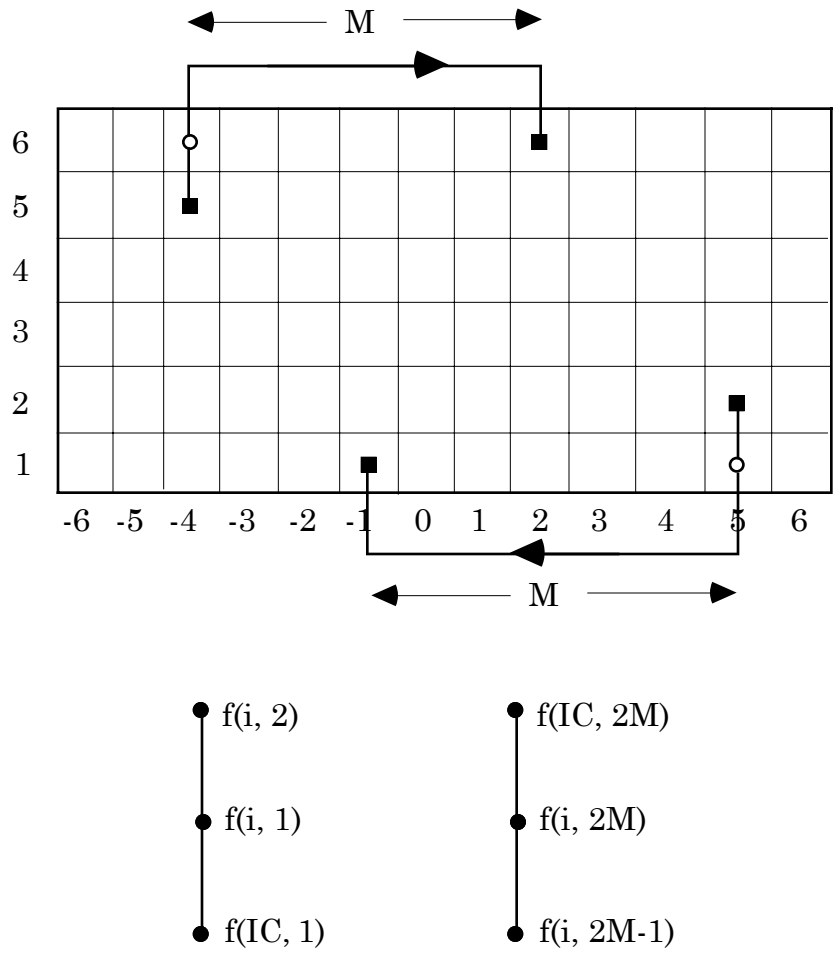


Figure 1.2.3-4. FD2 edge θ derivatives.

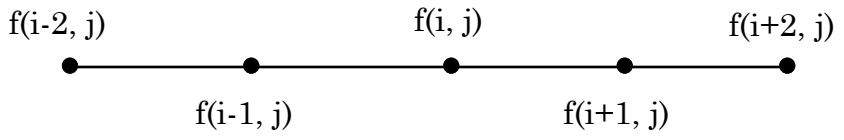
In the first test case the only derivatives being taken are of the height function h. In the second test case we take derivatives of h and the two velocity functions u and v. The problem with taking derivatives of u and v is that as one crosses the pole those values immediately change sign with respect to the other points in the stencil. For example if at the point (4, 6) in the grid (in Figure 1.2.3-4) the value of u is positive, then the value of u

at the grid point (-4, 6) (on the other side of the pole) is negative. Since we use the same subroutine to take θ derivatives of scalar and vector data we want to account for the sign change that occurs with vector data. Using the parameter VEC. (with values 1 for scalar data and -1 for vector data) accomplishes this accounting.

1.2.4 Spatial Fourth Order Finite Differences

ZonalDerivatives

To take φ derivatives we have the following stencil, and finite difference.



$$\left.\frac{\partial f}{\partial \varphi}\right|_{(\varphi_i,\theta_j)} \approx \frac{1}{12h}\left[f(i-2,j)-8f(i-1,j)+8f(i+1,j)-f(i+2,j)\right]$$

Figure 1.2.4-1. FD4 scheme for φ derivatives.

We need to use the following code which handles the interior of the grid plus the left and right edges. Since the left and right edge are identified we only calculate the value at the right edge and set the left edge equal to the same value. Also, since FD4 requires five grid points, we need to take care of next grid point in from right and left edge of the grid. We can then take care of the rest of the interior of the grid.

```
      DO 26 J=1,M2
```

```
C--- Do right edge and then set left edge = right edge.

      ANS(M2,J) = (F(M2-2,J)-F(-M2+2,J)+8.D0*(F(-M2+1,J)-F(M2-1,J)))*H12
      ANS(-M2,J) = ANS(M2,J)

C--- Do next grid point in from right and left side.

      ANS(-M2+1,J) = (F(M2-1,J)-F(-M2+3,J)+8.D0*(F(-M2+2,J)-F(-M2,J)))*H12
      ANS(M2-1,J) = (F(M2-3,J)-F(-M2+1,J)+8.D0*(F(M2,J)-F(M2-2,J)))*H12

C--- Do Interior and top and bottom of grid.

      DO 26 I=-M2+2,M2-2

   26    ANS(I,J) = (F(I-2,J)-F(I+2,J)+8.D0*(F(I+1,J)-F(I-1,J)))*H12
```

Notice how the derivatives on the right edge are calculated.

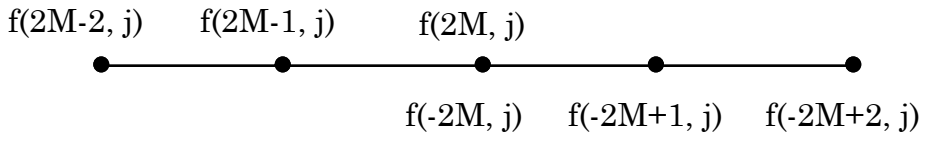


Figure 1.2.4-2. FD4 right edge φ derivatives.

Notice how the derivatives are calculated one grid point in from the right edge.

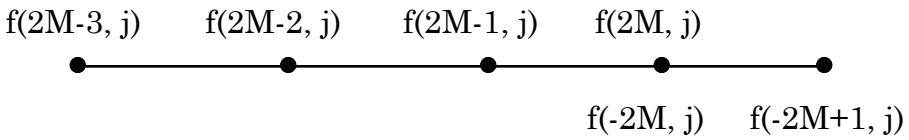


Figure 1.2.4-3. FD4 φ derivatives one grid point in from right edge.

Finally notice how the derivatives are calculated one grid point in from the left edge.

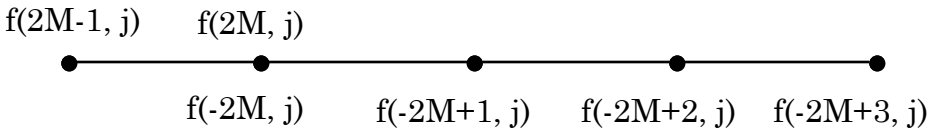
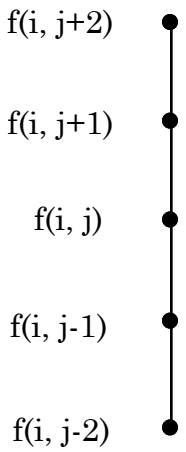


Figure 1.2.4-4. FD4 φ derivatives one grid point in from left edge.

MeridionalDerivatives

To take θ derivatives we have the following stencil, and finite difference.



$$\left.\frac{\partial f}{\partial \theta}\right|_{(\varphi_i,\theta_j)} \approx \frac{1}{12h}\left[f(i,j-2)-8f(i,j-1)+8f(i,j+1)-f(i,j+2)\right]$$

Figure 1.2.4-5. FD4 scheme for θ derivatives.

We need to use the following code, which handles going over the pole in the same way it was done in the FD2 scheme. Notice in this case not only to do we have to treat the top and bottom grid points in that same special way, but we also have to treat the grid points one down from the top and one up from the bottom of the grid in the same manner. This is because the FD4 stencil is 5 steps wide. Notice how the interior and the left and right edges of the grid are taken care of at the end of the I loop.

```
  DO 24 I=-M2,M2

C--- Calculate index on other side of pole.

   IF (I.LE.0) THEN
      IC = I+M2                ! M2 corresponds to PI.
   ELSE
      IC = I-M2                ! M2 corresponds to PI.
   ENDIF
```

```
C--- Do grid points that are one grid point away from the top and bottom.

   ANS(I,M2-1) = (F(I,M2-3)-VEC*F(IC,M2)+8.D0*(F(I,M2)-F(I,M2-2)))*H12
   ANS(I,2) = (VEC*F(IC,1)-F(I,4)+8.D0*(F(I,3)-F(I,1)))*H12

C--- Do top and bottom of grid.

   ANS(I,M2) = (F(I,M2-2)-VEC*F(IC,M2-1)+8.D0*(VEC*F(IC,M2)-F(I,M2-1)))*H12
   ANS(I,1) = (VEC*F(IC,2)-F(I,3)+8.D0*(F(I,2)-VEC*F(IC,1)))*H12

C--- Do interior, and left and right edges.

   DO 24 J=3,M2-2

      ANS(I,J) = (F(I,J-2)-F(I,J+2)+8.D0*(F(I,J+1)-F(I,J-1)))*H12

  24 CONTINUE
```

Figure 1.2.4-6 helps visualize taking derivatives along the top and bottom of the grid (with $M = 3$).

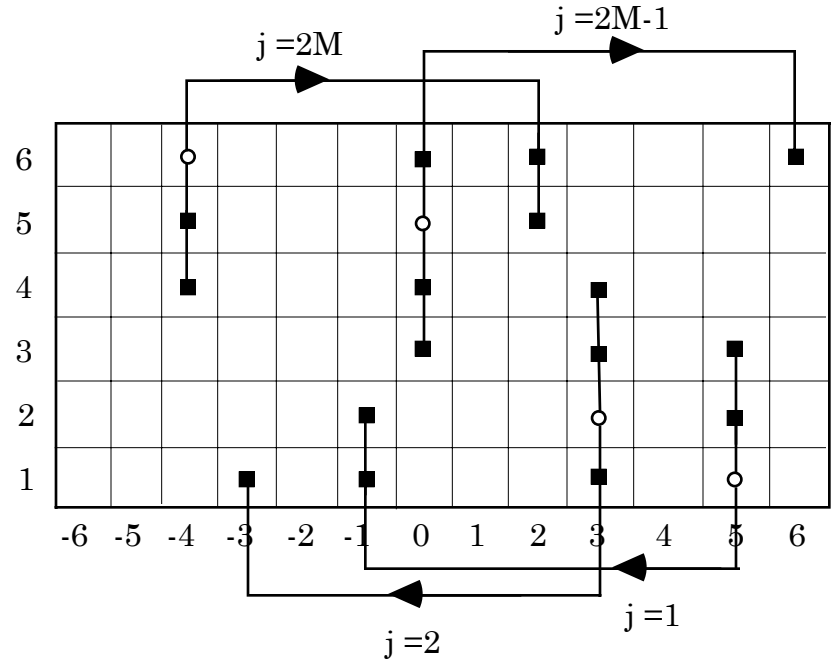


Figure 1.2.4-6. FD4 derivatives for θ along top and bottom edge.

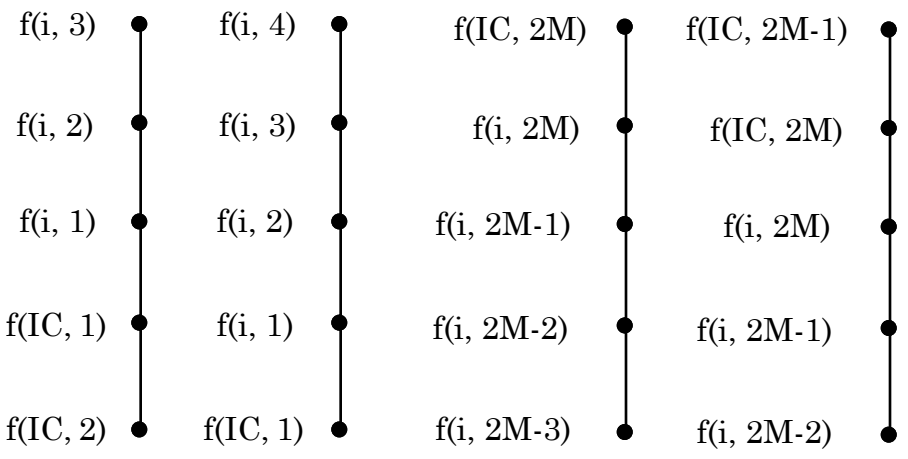


Figure 1.2.4-6 (cont.). FD4 derivatives for θ along top and bottom edge.

Again in each calculation of a derivative, whenever a term of the form F(IC, *) comes up, we multiply it by VEC which has the value 1 or -1 depending on whether we are taking derivatives of scalar or vector quantities.

1.2.5 The Pseudospectral Method

Fornberg [1] shows that the pseudospectral method can be viewed as a limiting case of high order centered finite difference methods. One can think of this as the following : On an equispaced grid covering one period of a one dimensional data set, the pseudospectral method amounts to interpolating with a trigonometric polynomial, and then taking the derivative analytically. The limiting centered finite difference method can then be applied to any grid point. The result becomes then exactly the same as one obtained by the trigonometric interpolation (Fourier pseudospectral method).

Numerically, the subroutine that invokes the pseudospectral method involves calls to a subroutine which in turn, calls a fast Fourier transform. Since the FFT subroutine requires two arrays of $4M$ elements, coding for φ and θ derivatives is a little different.

ZonalDerivatives

For the φ direction, we have $4M$ points east-west. This allows us to take two rows at a time when taking derivatives. The code starts off like this:

```
      DO 40 J=1,M2-1,2

         DO 41 I=0,M4-1
            S1(I) = F(I-M2, J)       ! Set up S1() and S2() for
41          S2(I) = F(I-M2, J+1)     ! the call to PS(). (take 2
                  ! rows at a time)
```

So we want to think of the following picture

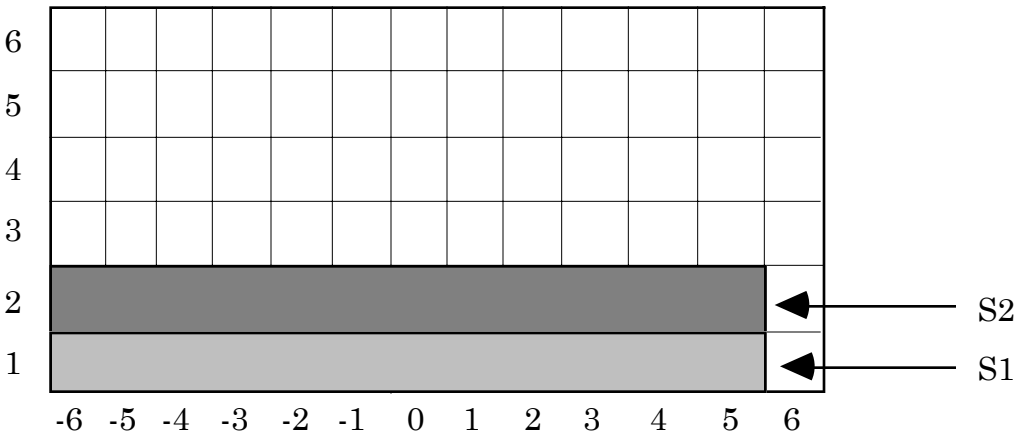


Figure 1.2.5-1. PS derivatives for φ.

When the I loop first starts, it grabs the first and second row. Then, as the I loop goes on, it grabs the next two rows, and so on.

Then we call the PS( ) subroutine which returns the derivatives in the original arrays S1( ) and S2( ) (we will describe how PS( ) gets the actual derivatives in the next section).

```
      CALL PS (S1,S2,M)          ! Find derivative with PS method.
```

Then we place those answers in the array ANS( ).

```
      DO 42 I=0,M4-1
         ANS(I-M2, J)   = S1(I)   ! Get the answers from S1()
42       ANS(I-M2, J+1) = S2(I)   ! and S2().
```

We also make sure to set the values on the right side of grid equal to the values on the left side of the grid.

```
      ANS(M2,J)   = ANS(-M2,J)    ! Set right edge = left edge.
      ANS(M2,J+1) = ANS(-M2,J+1)
40    CONTINUE
```

MeridionalDerivatives

For the θ direction we have $2M$ points north-south. This allows us to take four columns (see Figure 1.2.5-2) at a time when taking derivatives. Recall in the FD2 and FD4 cases we had to take of the special case of when the center of the stencil was at the top or bottom of the grid. To get $4M$ points (so the FFT can do its work) we grab $2M$ points from one column then the $2M$ points on the other side of the pole. The code starts off like this:

```
      DO 45 I=0,M2-2,2
         DO 46 J=1,M2
            S1( M2-J )   = F(I,J)
            S1( M2-1+J ) = VEC*F(I-M2,J)
            S2( M2-J )   = F(I+1,J)
46          S2( M2-1+J ) = VEC*F(I+1-M2,J)
```

In this case we want to think of the following picture

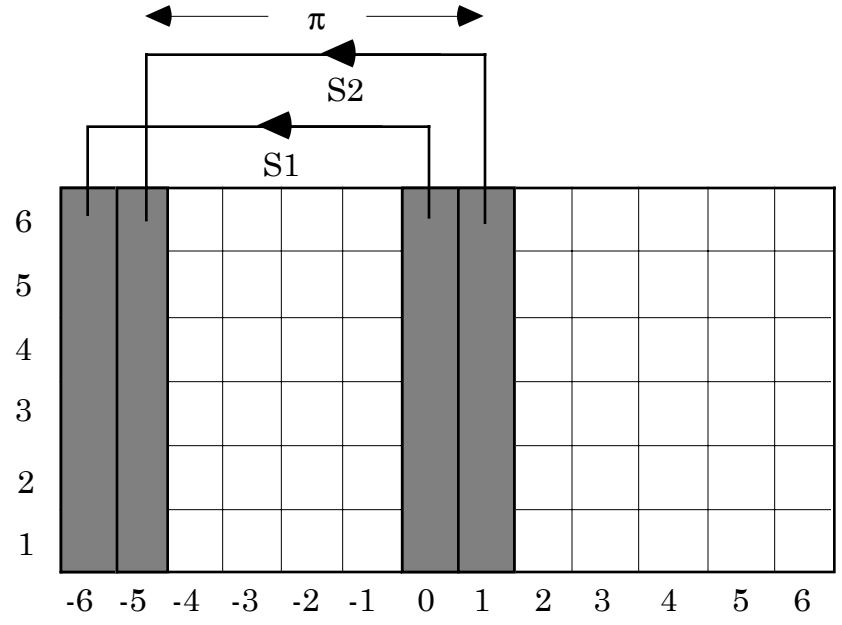


Figure 1.2.5-2. PS derivatives for θ.

So when the I loop first starts, it grabs the 0^{th} and first column, then as the I loop goes on it grabs the next two columns (to the right of the column marked 1) and so on.

To make S1( ) and S2( ) work properly with PS( ) we think of them like this (take S1( ) for an example)

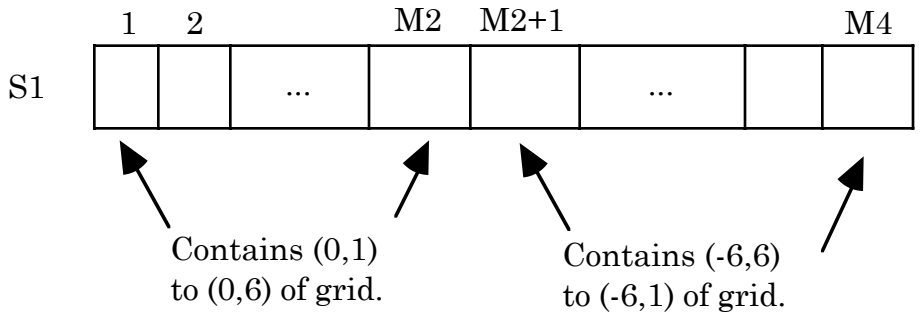


Figure 1.2.5-3. What S1 contains at the beginning of the I loop.

Note that we multiply the values of the second part (elements M2 to M4-1) of S1( ) and S2( ) by VEC in case we have taken derivatives of vector data.

Then we call the PS( ) subroutine and get the derivatives back out of S1( ) and S2( ). Notice that we use the negative of the values in the first part (elements 0 to M2-1) of S1( ) and S2( ). This is because the derivatives on one side of the globe should be the opposite sign of the derivatives on the opposite side. This is because we think of indexing J from M2 to 1 (to represent θ values from $\frac{\pi}{2}$ to $-\frac{\pi}{2}$) that we use the negative of the values in the first part of S1( ) and S2( ).

Also we multiply the values of the second part (elements M2 to M4-1) of S1( ) and S2( ) by VEC in case we have taken derivatives of vector data (as vector data on the other side of the pole will have the opposite sign).

```
         CALL PS (S1,S2,M)
      DO 47 J=1,M2

         ANS(I,J)      = -S1( M2-J )
         ANS(I-M2,J)   = VEC*S1( M2-1+J )
         ANS(I+1,J)    = -S2( M2-J )
47       ANS(I+1-M2,J) = VEC*S2( M2-1+J )
45       CONTINUE
```

We also make sure to set the values on the right side of grid equal to the values on the left side of the grid.

```
      DO 49 J=1,M2
49       ANS(M2,J) = ANS(-M2,J)
```

1.2.6 Pseudospectral Derivatives

The PS( ) subroutine takes two arrays of data A and B from the sphere.

```
SUBROUTINE PS (A,B,M)
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
DIMENSION A(0:4*M-1),B(0:4*M-1)

M2 = 2*M
M4 = 4*M
```

Then we call FFT( ) to take that data to Fourier space.

```
CALL FFT (A,B,M4,-1)            ! Transform to fourier space
```

Now A holds the real part of that data, while B holds the imaginary part. Right now A and B look like this (with Fourier modes indicated on the top).

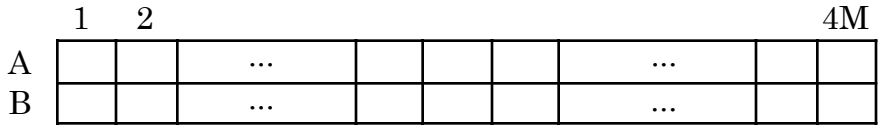


Figure 1.2.6-1. Modes in arrays A and B.

But we can instead think of A and B like this. (with new Fourier modes indicated on the bottom)

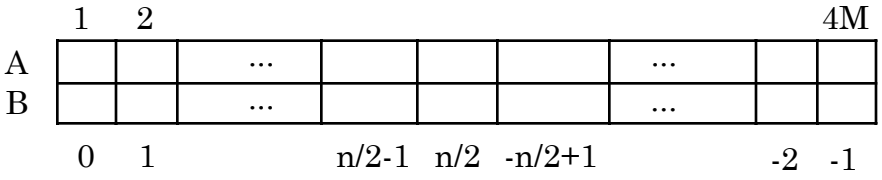


Figure 1.2.6-2. New arrangement of modes in arrays A and B.

The lowest order mode does not contribute anything to the derivative. (the lowest order mode is a constant and the derivative of a constant is zero). The highest order mode does not contribute anything to the derivative either. The highest order mode has maxima and minima at

successive modes, and thus, the first derivative is equal to zero at all the grid points.

```
      A(0) = 0.D0             ! 0th mode doesn't add to derivative.
      B(0) = 0.D0             ! 0th mode doesn't add to derivative.
      A(M2) = 0.D0            ! 1st derivative of M2 mode = 0.
      B(M2) = 0.D0            ! 1st derivative of M2 mode = 0.
```

To numerically take the derivative, we work on the real and imaginary parts separately. Notice that we take advantage of the second way to think of ordering the modes in the DO loop. A and B are indexed from 0 to M4 (4*M), but the DO loop runs from 1 to 2M - 1. What is happening here is that we are working from the left and right of A and B and going towards the middle.

```
C --- Do the Pseudospectral method ---
      DO 10 I=1,M2-1
         FC = 1.D0*I/M4                ! FC is what we mult the
         T    = B(I)                   ! modes by.
         B(I) = A(I)*FC
         A(I) = -T  *FC
         T       = B(M4-I)
         B(M4-I) = -A(M4-I)*FC
   10    A(M4-I) = T       *FC
```

First we calculate a multiplying factor FC = I/4M. FC has a factor of $1/4M$ because if we simply call the forward and backward FFT( ) all the elements in A and B will be multiplied by a factor $4M$.

The factor of I gets worked into FC since it represents the mode number. Suppose we had to take the derivative of $e^{i\omega t}$ with respect to t. Very simply we would have

$$\frac{\partial e^{i\omega t}}{\partial t} = i\omega \cdot e^{i\omega t}.$$

So to take the derivative we need to multiply each mode by $i\omega$. In the code I represents the mode number, ω. We discuss how to handle the imaginary number *i* below.

Now, if we multiply the real parts in A by $i\omega$ we need to move them to B (which is supposed to hold the imaginary part). In order to <u>not</u> overwrite what already exists in B, we first let T = B(I) and then set B(I) = A(I)*FC.

We also need to multiply the elements of B by $i\omega$. But we want to think of the elements in B already being multiplied by *i* (since it's the imaginary part). So if we multiply by $i\omega$ on top of that, we have simply multiplied it by -1, and now it's real, so we need to move it to A. That is why we have A(I) = -B(I)*FC.

Then we call the inverse FFT( ) to take the data back to real space (and we consequently get our derivatives).

```
CALL FFT (A,B,M4,+1)              ! Transform to real space
RETURN
END
```

1.2.7 <u>Smoothing</u>

After each time step, numerical smoothing (FFT-based) was applied in φ direction. (no smoothing was applied in the θ direction) This was done as little as possible to avoid affecting the accuracy. Smoothing involves taking data points from the grid, transforming them to Fourier modes, and then multiplying the modes (from lower to higher) by values from 1 to 0 in a smooth way.

For each line of latitude θ, the proportion $1-\cos\theta$ of modes were gradually smoothed out. So as $|\theta|$ approaches $\frac{\pi}{2}$, <u>more</u> of the higher modes

are gradually smoothed out. None were affected at the equator, and all but the constant mode affected at the poles)

Smoothing the modes leads to approximately the same resolution at all parts of the sphere and bypasses the otherwise restrictive polar CFL-stability condition.

To code this we do the following : First we declare an array, WKSP( ), to use as workspace.

```
      ALLOCATABLE WKSP(:,:)                ! Declare WKSP, an
                                           ! allocatable array
      ALLOCATE( WKSP(4*M,2) )              ! for workspace.
```

Then we go row by row and put data values from the grid into WKSP( ) so that we can move those data points to Fourier space.

```
      DO 10 J=M,1,-1                       ! Loop to smooth F values

         DO 20 I=1,M4                      ! Put 2 rows of F values in
                    ! workspace
            WKSP(J,1) = F( I,-M2+J)
20          WKSP(J,2) = F(-I,-M2+J)
```

Notice how we take two rows at a time, symmetrically from the equator. (i.e. the first row above and below the equator, then the second row above and below the equator, and so on)

Right now both WKSP(*,1) and WKSP(*,2) look like this (with fourier modes indicated on top) :

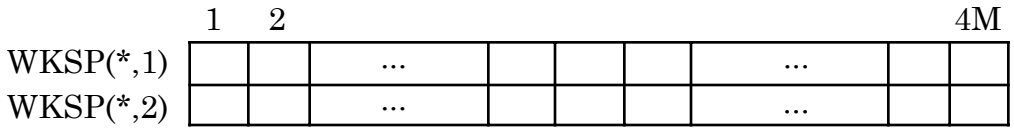


Figure 1.2.7-1. Modes in arrays WKSP(*,*).

Each array contains one whole row of data values. (WKSP(*,1) contains row I and WKSP(*,2) contains row -I) Now we call FFT( ).

```
CALL FFT (WKSP(1,1),WKSP(1,2),M4,-1)  ! Transform to Fourier space
```

Recall that we can think of reordering the modes in WKSP( ). Now we want to think of the modes as indicated at the bottom.

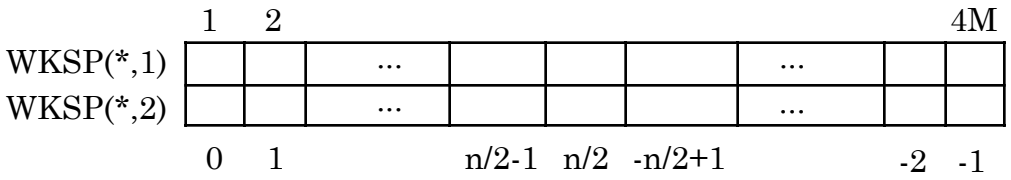


Figure 1.2.7-2. New arrangement of modes in WKSP(*,*).

We want to think of modes 0 through $n/2$ as representing $0 \le \theta \le \frac{\pi}{2}$, and modes -1 through -$n/2$+1 as representing $-\frac{\pi}{2} < \theta < 0$. Thus what would be the left and right side of the grid actually meet up in the middle of WKSP( ) after the call to FFT( ).

Now what we want to do is suppress more and more modes for bigger values of θ. (we start with suppressing higher modes and go towards suppressing lower modes) To do this we use a variable called IS. IS tells how far out from the highest mode (i.e. mode $\frac{n}{2}$, in the middle) of WKSP( ) do we go out to either side to suppress modes. IS is ruled by the term $1-\cos\theta$. As θ approaches $\frac{\pi}{2}$ (or $-\frac{\pi}{2}$) IS (the number of modes we'll suppress) will get bigger. Notice that the whole thing is multiplied by (M2-1). This accounts for the fact that the length of the row depends on M, so we want to scale IS accordingly.

```
TH = ( (J-1) - M + 0.5D0)*H
IS = BETA*(1.D0-COS(TH))*(M2-1)
```

So as we increment J, more and more modes will be altered. For example with M = 10 and $\beta = 1$ table 1.2.7-1 describes the number of modes altered per latitude.

Row above and below equator	Number of modes altered
1	0
2	0
3	2
4	3
5	5
6	7
7	10
8	13
9	16

Table 1.2.7-1. Number of modes altered per grid row.

Remember that we're taking two rows of data at a time. So this chart says, for example, the first two rows above and below the equator do not get any modes altered at all., the third row above and below the equator gets the two highest modes altered, and so on.

Various values of $\beta < 1$ were tried, but in the end $\beta = 1$ was used. Other values did not seem to make much difference.

So now we alter those modes. (notice how IS affects the DO loop)

```
DO 30 I=M2+1-IS,M2+1+IS          ! Gradually smooth coefficients
```

Now for each value of I, (i.e. depending what column we are in) we need to calculate a value of *t* (we will use *t* to help us calculate a scaling factor).

```
IF(IS.EQ.0) THEN
 T = 0
ELSE
 T = (I - (M2+1))*(PID2)/IS
ENDIF
```

In a previous version of the code, instead of scaling down the modes gradually, they were cut right down to zero. To make the code more robust, (i.e. to get rid of some of the highest modes yet still keep some of the data for the modes in-between) it was decided to scale down gradually on the order of $1-\alpha\cos^2 t$ over the interval determined by IS.

The α term has nothing to do with the α term in the two test cases (which described how fluid flowed around the sphere). We wanted to be able to manipulate how much we were smoothing down the modes. As with β, many values of α were used but we settled on $\alpha = 1$ because again, other values did not seem to make much difference.

```
            SCALE = 1.D0 - ALPHA*COS(T)**2     ! Calculate scaling factor.
            WKSP(I,1) = SCALE*WKSP(I,1)        ! Do the scaling.
            WKSP(I,2) = SCALE*WKSP(I,2)

30      CONTINUE
```

Then transform the data back to real space :

```
            CALL FFT (WKSP(1,1),WKSP(1,2),M4, 1)  ! Transform to real space
```

Then assign the new found values back to the grid. Notice that we need to scale down the WKSP( ) values on the way out. This accounts for the fact that a forward and backward call to FFT( ) will multiply all array values by $4M$.

```
            DO 40 J=1,M4                       ! Lay out new F values
               F( I,-M2+J) = WKSP(J,1)*DM4     ! Scale down FFT
40             F(-I,-M2+J) = WKSP(J,2)*DM4     ! Scale down FFT
```

We make sure that we take care of the edges (make the left side equal to the right).

```
      F( I,-M2) = F( I,M2)              ! Set edges equal
10    F(-I,-M2) = F(-I,M2)              ! Set edges equal
```

End the subroutine.

```
      DEALLOCATE( WKSP )                ! Release space used by WKSP
      RETURN
      END
```

1.2.8 Time Stepping

We do our time stepping with a fourth order Runge-Kutta scheme. Recall that we use the following,

$$y_{n+1} = y_n + \tfrac{1}{6}\left(d_1 + 2d_2 + 2d_3 + d_4\right),$$

to solve $\frac{\partial y}{\partial t} = f(t,y)$ where,

$$\begin{aligned} d_1 &= kf\left(t_n, y_n\right) \\ d_2 &= kf\left(t_n + \tfrac{k}{2}, y_n + \tfrac{1}{2}d_1\right) \\ d_3 &= kf\left(t_n + \tfrac{k}{2}, y_n + \tfrac{1}{2}d_2\right) \\ d_4 &= kf\left(t_n + k, y_n + d_3\right) \end{aligned}$$

and k is the size of the time step. In the code (for Case 1) it looks like the following:

```
M2 = 2*M
C--- Declare an allocatable array RKH
      ALLOCATABLE RKH(:,:,:)
C--- Allocate space for the array ---
      ALLOCATE( RKH(-2*M:2*M,1:2*M,0:4) )
C--- Get d(1) into RKH(I,J,1).
      DO 10 I=-M2,M2
         DO 10 J=1,M2
   10       RKH(I,J,0) = HEIGHT(I,J)
      CALL EVAL (RKH,U,V,RKH(-M2,1,1),M,METHOD)
C--- Get d(2) into RKH(I,J,2).
      DO 12 I=-M2,M2
         DO 12 J=1,M2
```

```
   12       RKH(I,J,0) = HEIGHT(I,J)+0.5D0*DT*RKH(I,J,1)
      CALL EVAL (RKH,U,V,RKH(-M2,1,2),M,METHOD)
C--- Get d(3) into RKH(I,J,3).
      DO 14 I=-M2,M2
         DO 14 J=1,M2
   14       RKH(I,J,0) = HEIGHT(I,J)+0.5D0*DT*RKH(I,J,2)
      CALL EVAL (RKH,U,V,RKH(-M2,1,3),M,METHOD)
C--- Get d(4) into RKH(I,J,4).
      DO 16 I=-M2,M2
         DO 16 J=1,M2
   16       RKH(I,J,0) = HEIGHT(I,J)+DT*RKH(I,J,3)
      CALL EVAL (RKH,U,V,RKH(-M2,1,4),M,METHOD)
C--- Put the different parts of the RK algorithm
C--- together to get one time step
      DO 18 I=-M2,M2
         DO 18 J=1,M2
          HEIGHT(I,J) = HEIGHT(I,J) +
  1        DT*(RKH(I,J,1)+2.D0*(RKH(I,J,2)+RKH(I,J,3))+RKH(I,J,4))/6.D0
   18 CONTINUE
```

RKH( ) is a three dimensional array. It holds five 'grids'. RKH(*, *, 0) is used as workspace to hold data before the space derivatives are taken. RKH(*, *, 1) through RKH(*, *, 4) hold d_1 through d_4 respectively. These are put back together in the DO loops.

EVAL( ) handles taking the space derivatives with the FD2, FD4, or the pseudospectral method depending on the value of METHOD.

1.2.9 Operation Count

An operation count (additions, subtractions, multiplications, and divisions) was completed for one call to EVAL( ) in the second test case in order to compare to the operation count in Williamson [11]. Recall that we describe the resolution of our grid with the parameter M ($4M$+1 points east-west and $2M$ points north south). The number of operations depending on M is given by

$$72M(4M+1)+24M(2M-1)+240M^2\log_2 4M.$$

This table gives the grid resolution (number of grid points east-west versus the number of grid points north-south) and number of operations for different values of *M*.

M	Grid Resolution	Operation Count
16	64x32	0.455e+06
32	128x64	2.065e+06
64	256x128	9.243e+06

Table 1.2.9-1. Grid resolution and Operation count for various *M*.

In Williamson [11] the number of operations for their spherical harmonics code is given by

$$8 + 15.6n + 195n^2 + 550n^2 \log_2 n + 35.2n^3$$

where n describes the grid resolution ($2n$ points east-west and n points north-south). So an equivalent grid resolution corresponds to $M = n/2$. The following table gives the grid resolution and number of operations for different values of n. The notation Tn (which stands for triangulation truncation at wave number n) is standard notation for spectral transform models.

n	Grid Resolution	Operation Count
31	64x32	3.855e+06
42	84x42	8.184e+06
63	128x64	22.625e+06

Table 1.2.9-2. Grid resolution and Operation count for various *n*.

The tables 1.2.9-1-2 show that the present PS scheme has roughly one order of magnitude less steps than Williamson's implementation of a spectral transform method based on spherical harmonic techniques.

1.2.10 Error Analysis

Normalized global errors were calculated (for Case 1) as described by Williamson et al. [6]. The following norms were calculated.

$$l_1(h) = \frac{I\left[\left|h(\varphi,\theta) - h_T(\varphi,\theta)\right|\right]}{I\left[\left|h_T(\varphi,\theta)\right|\right]}$$

$$l_2(h) = \frac{\left\{I\left[\left(h(\varphi,\theta) - h_T(\varphi,\theta)\right)^2\right]\right\}^{\frac{1}{2}}}{\left\{I\left[h_T(\varphi,\theta)^2\right]\right\}^{\frac{1}{2}}}$$

$$l_\infty(h) = \frac{\max_{all\ \varphi,\theta}\left|h(\varphi,\theta) - h_T(\varphi,\theta)\right|}{\max_{all\ \varphi,\theta}\left|h_T(\varphi,\theta)\right|}$$

where $h_T(\varphi,\theta)$ is the true solution.

The function $I[\ ]$ that was used was an approximate surface integral given by,

$$I\left[h(\varphi,\theta)\right] \approx \sum_{i=-2M+1}^{2M}\sum_{j=1}^{2M} h\left(\varphi_i,\theta_j\right)\cos\theta_j$$

where φ_i and θ_j are defined as they were at the beginning of this section. $I\left[h(\varphi,\theta)\right]$ simply returns the sum of the product of the height and $\cos\theta$ at every grid point.

To calculate the analytic solution, $h_T(\varphi,\theta)$, at each time step, we rotate the center $\left(\varphi_c,\theta_c\right)$, of the cosine bell around the sphere and use the initial condition described in Section 2.1. To calculate the rotated coordinates of the center the following method was used.

First how much time t, has passed during the rotation is noted. Then if $\alpha = 0$, (which implies rotation in the direction of the equator) then we just increment φ_c by $u_0 t / a$, where u_0 is the base advecting wind speed, and a is the mean radius of the earth (described in Section 2.1). The center is then located at $(\varphi_c + u_0 t / a, \theta_c)$.

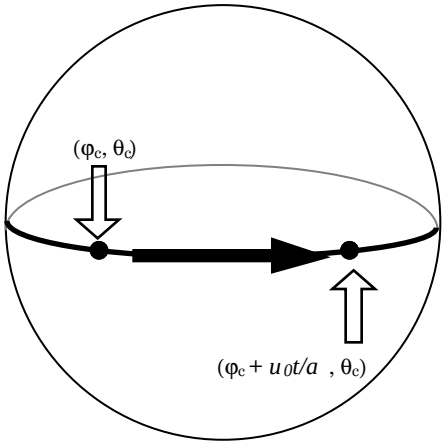


Figure 1.2.10-1. Rotation of cosine bell with $\alpha = 0$.

If $\alpha > 0$, a change of coordinates is necessary in order to determine the solution in (φ, θ) coordinates. We change coordinate systems so that in the new (φ', θ') system, the equator is tilted to the old equator by the angle α.

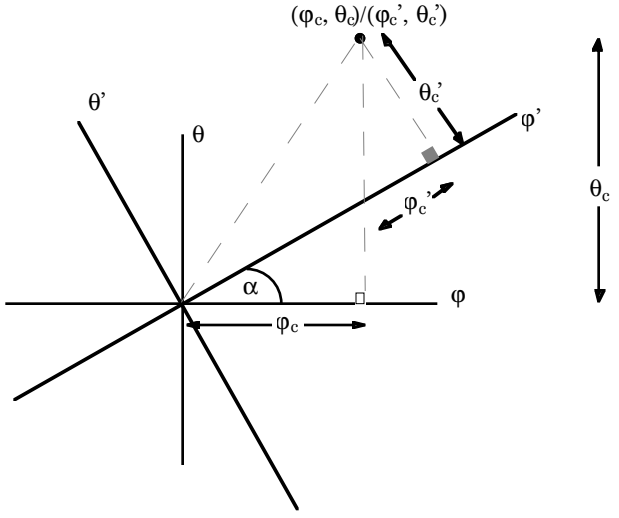


Figure 1.2.10-2. Transformation to new (φ', θ') coordinate system.

To change coordinates to the new system, the following equations were used.

$$\sin\theta' = -\sin\varphi\cos\theta\sin\alpha + \sin\theta\cos\alpha$$
$$\sin\varphi'\cos\theta' = \sin\varphi\cos\theta\cos\alpha + \sin\theta\sin\alpha$$

Once we have the center in the new (φ', θ') system, we increment φ_c' by $u_0 t / a$ to rotate the cosine bell around the sphere. We then transform coordinates back to the old (φ, θ) system with the following equations.

$$\sin\theta = \sin\varphi'\cos\theta'\sin\alpha + \sin\theta'\cos\alpha$$
$$\sin\varphi\cos\theta = \sin\varphi'\cos\theta'\cos\alpha - \sin\theta'\sin\alpha$$

Chapter 2

Two Test Cases

This chapter describes the first two test cases in Williamson [10]. In each case, we describe the equations used, and do an error analysis.

2.1 Williamson's First Case

Holton's first case tests the advective component of the shallow water equations,

$$\frac{\partial}{\partial t}(h)+\frac{u}{a\cos\theta}\frac{\partial}{\partial \varphi}(h)+\frac{v}{a}\frac{\partial}{\partial \theta}(h)+\frac{h}{a\cos\theta}\left[\frac{\partial u}{\partial \varphi}+\frac{\partial(v\cos\theta)}{\partial \theta}\right]=0.$$

The solid body rotation is given by

$$u = u_0\left(\cos\theta\cos\alpha+\sin\theta\cos\left(\varphi+\tfrac{3\pi}{2}\right)\sin\alpha\right)$$
$$v = -u_0\sin\left(\varphi+\tfrac{3\pi}{2}\right)\sin\alpha$$

where $u_0 = \dfrac{2\pi a}{(12\ \text{Days})} \approx 40$ m/s, $a = 6.37122x10^6 m$ (the mean radius of the earth) and α is the angle between the axis of solid body rotation and the polar axis of the spherical coordinate system (for example $\alpha = 0$ would give flow parallel to the equator). Note that for u and v, φ is incremented by $\frac{3\pi}{2}$. compared to Williamson's [6] definition of u and v. This was done to obtain the proper flow structure (illustrated in Figure 2.1-1) for the case of $\alpha = \frac{\pi}{2}$. Figure 2.1-1 is similar to Figure 5 in Fornberg [2].

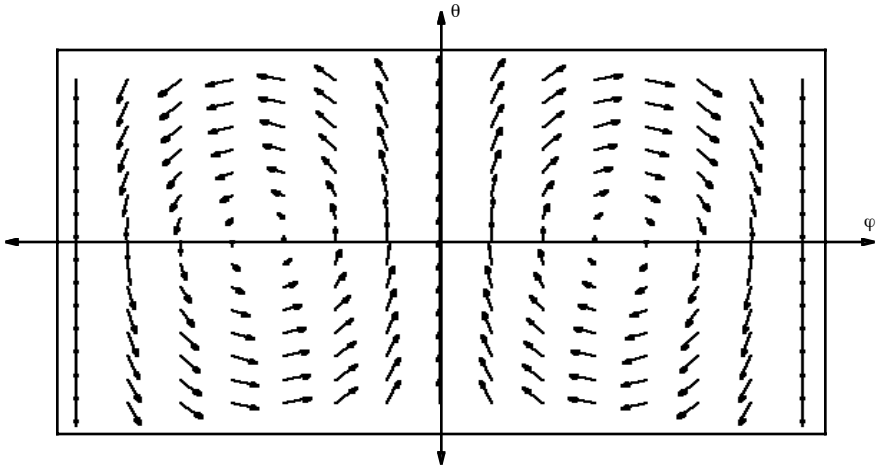


Figure 2.1-1. Flow structure on the sphere for $\alpha = \frac{\pi}{2}$.

Note that for this particular choice of u and v

$$\nabla \cdot (u, v) = \frac{1}{a \cos\theta}\left[\frac{\partial u}{\partial \varphi} + \frac{\partial (v \cos\theta)}{\partial \theta}\right] = 0.$$

The initial cosine bell test pattern to be advected is given by

$$h(\varphi, \theta) = \begin{cases} \frac{h_0}{2}\left(1 + \cos\left(\frac{\pi r}{R}\right)\right) & \text{if r } < \text{ R} \\ 0 & \text{if r } \geq \text{ R,} \end{cases}$$

where $h_0 = 1000m$, $R = \frac{a}{3}$, $r = a \text{ arc } \cos\left[\sin\theta_c \sin\theta + \cos\theta_c \cos\theta \cos(\varphi - \varphi_c)\right]$, which is the great circle distance between (φ, θ) and the center, $(\varphi_c, \theta_c) = (0,0)$.

Figure 2.1.1 illustrates what the cosine bell should look like on the grid with $M = 32$.

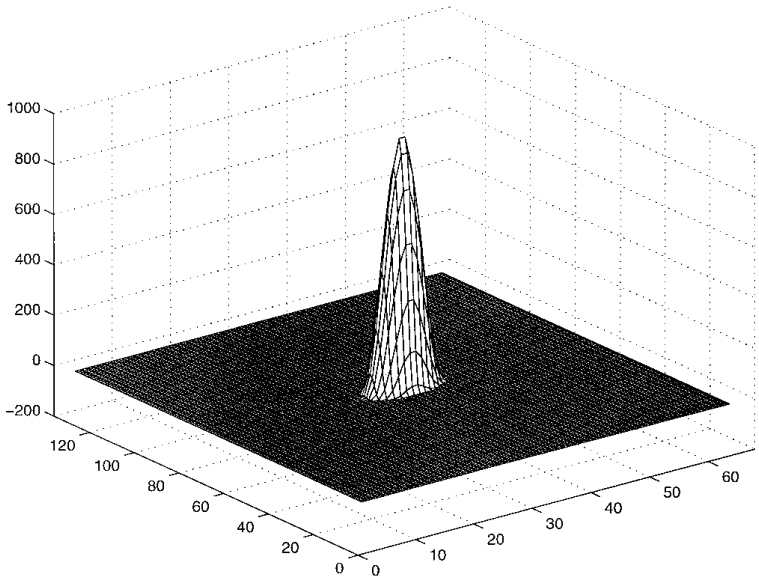


Figure 2.1-2. Initial cosine bell for Case 1 ($M = 32$).

2.1.1 A comparison of the different numerical methods

As one runs the first case, the cosine bell should simply be pushed around the sphere. Ideally, after one rotation, the cosine bell should end up back in the middle of the grid and look the same as the initial condition (the shape of the bell should be preserved). For Case 1 we show FD2, FD4, and PS results for $\alpha = \frac{\pi}{2} - 0.05$ (to verify the numerical stability of advecting the cosine bell near the pole), and $M = 32$.

Figure 2.1.1-1 shows the result after one revolution using the FD2 scheme. As with many finite difference schemes, there are trailing nonphysical oscillations behind the now once rotated cosine bell.

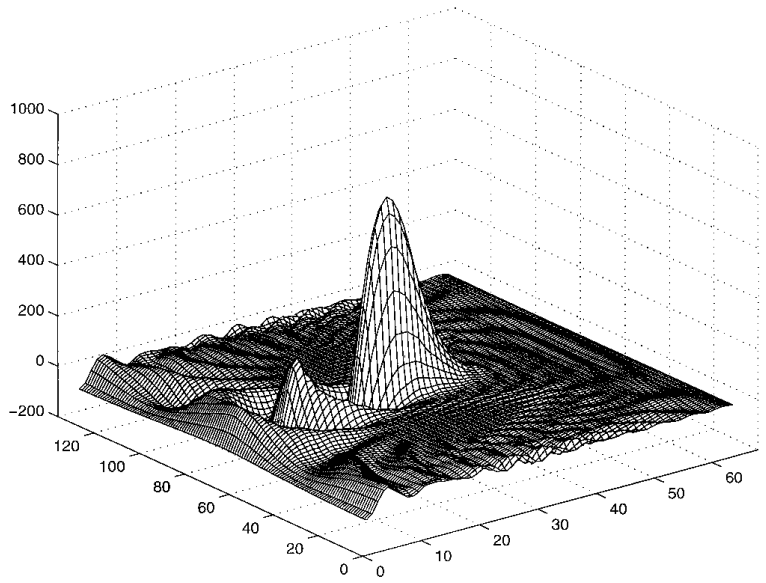


Figure 2.1.1-1. Cosine bell after one revolution with FD2 scheme ($M = 32$).

Figure 2.1.1-2 shows the result after one revolution using the FD4 scheme. Note that the nonphysical oscillations behind the cosine bell are much smaller.

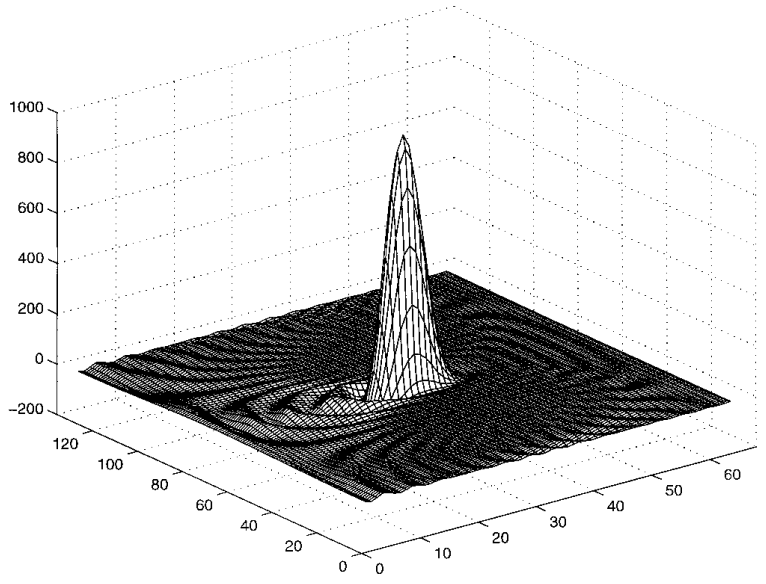


Figure 2.1.1-2. Cosine bell after one revolution with FD4 scheme ($M = 32$).

Figure 2.1.1-3 shows the result after one revolution using the PS scheme. Note that we have visibly achieved our goal of retaining the initial shape the cosine bell, and the nonphysical oscillations are unnoticeable in this figure (an error analysis of the PS case follows).

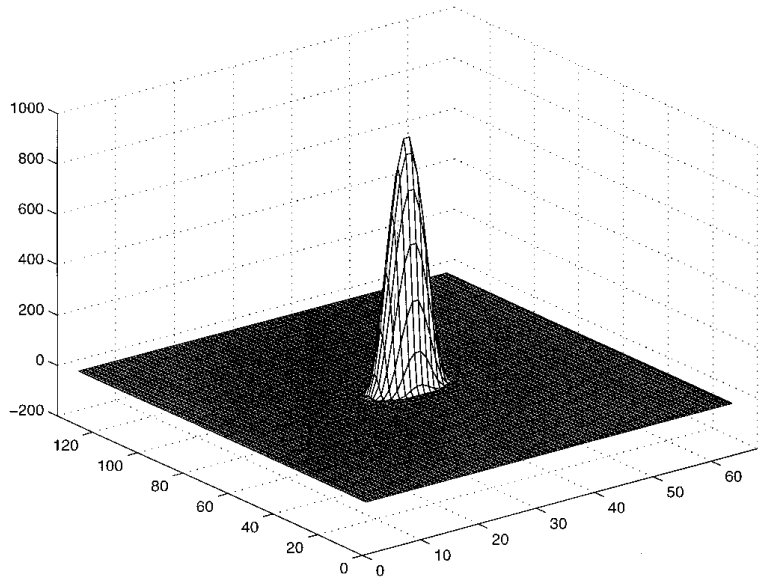


Figure 2.1.1-3. Cosine bell after one revolution with PS scheme ($M = 32$).

We also show Case 1 FD2, FD4 and PS results for $\alpha = \frac{\pi}{2} - 0.05$ with $M = 16$. One should notice that while the FD2 and FD4 methods are much worse with a coarser resolution, the PS scheme still retains the shape of the cosine bell after one revolution.

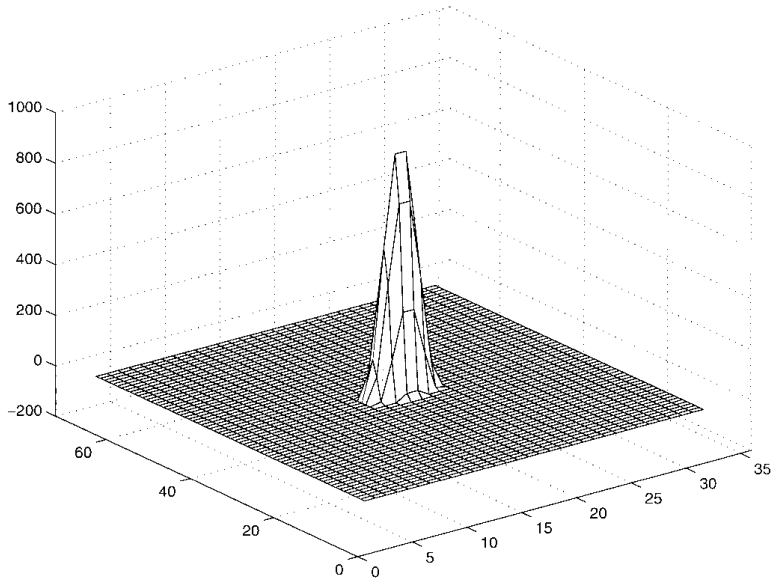


Figure 2.1.1-4. Initial cosine bell for Case 1 ($M = 16$).

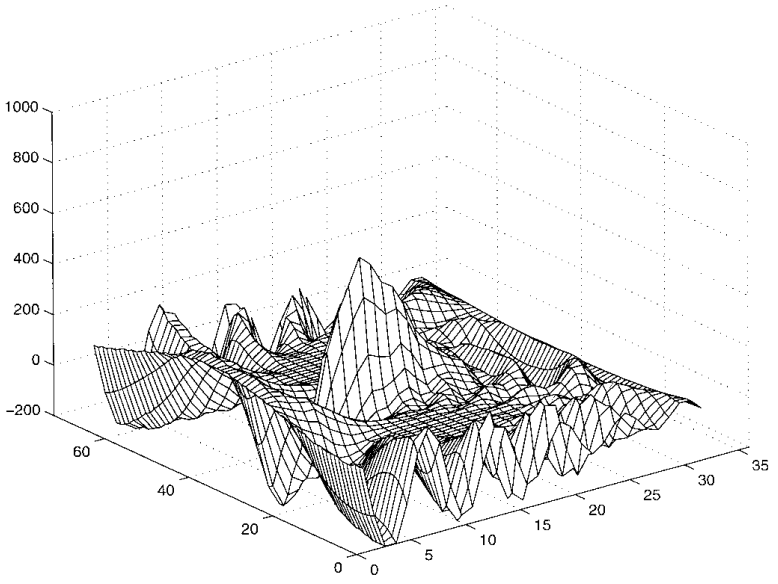


Figure 2.1.1-5. Cosine bell after one revolution with FD2 scheme ($M = 16$).

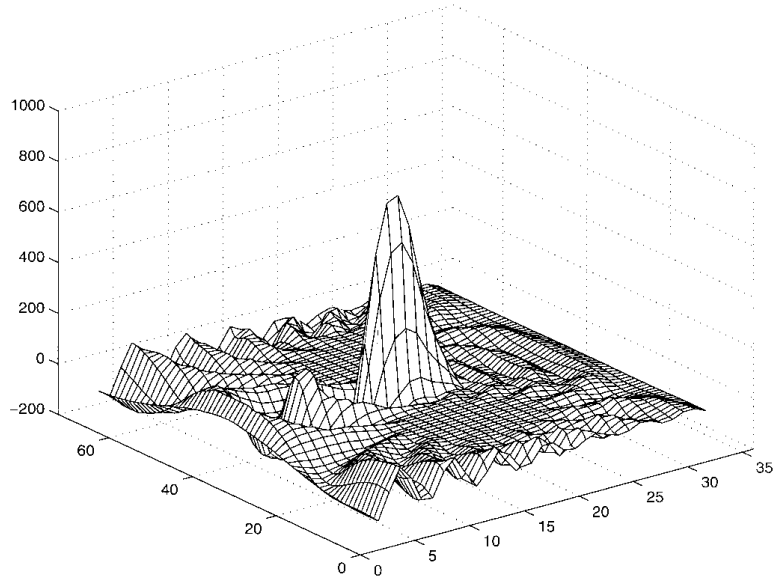


Figure 2.1.1-6. Cosine bell after one revolution with FD4 scheme ($M = 16$).

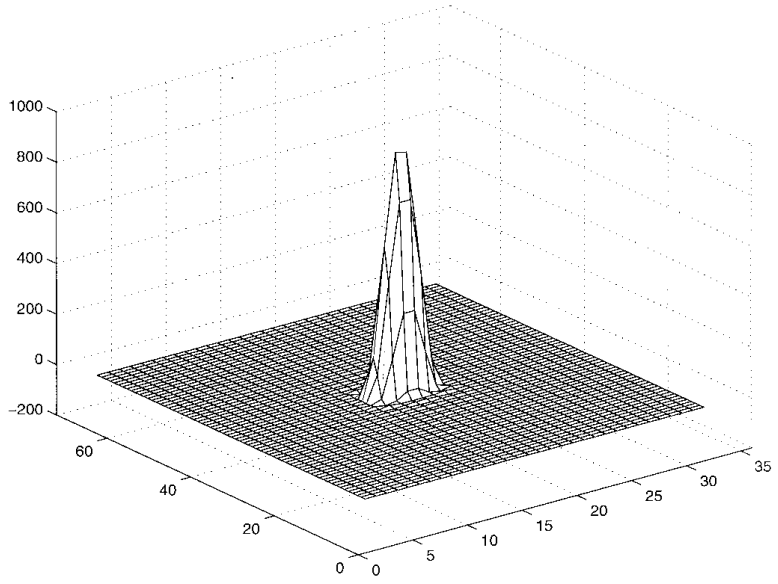


Figure 2.1.1-7. Cosine bell after one revolution with PS scheme ($M = 16$).

2.1.2 Error Analysis

An error analysis was done for the PS method results for $\alpha = \frac{\pi}{2} - 0.05$, and $M = 16$ (to compare to Williamson's results for a similar grid resolution, T31). Figure 2.1.2-1 shows our errors and Figure 2.1.2-2 shows Williamson's errors.

Our errors where sampled 1280 times during one rotation (every 810 seconds in model-time). Williamson's were sampled 120 times (every hour in model-time). This accounts for the different appearance in the two figures. The oscillations in the graphs happen when the peak of the cosine bell falls between two grid points.

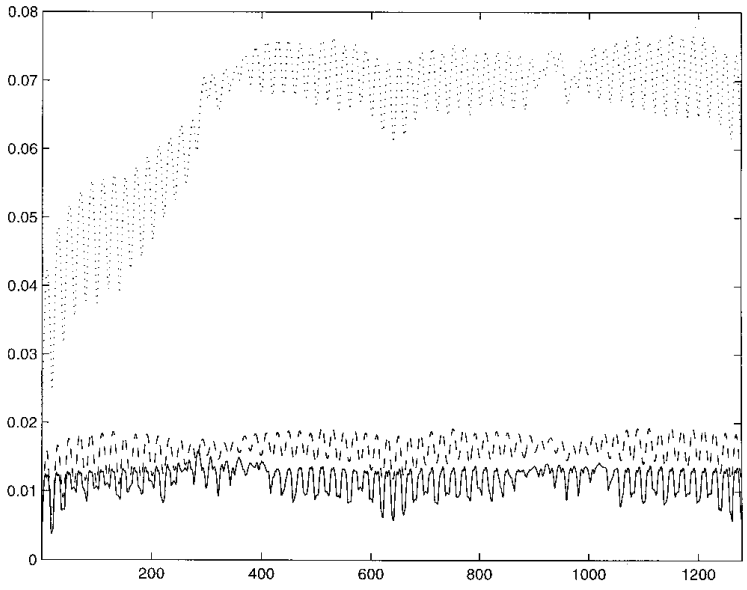


Figure 2.1.2-1. l_1 (dotted), l_2 (dashed), and l_∞ (solid) norms for PS scheme ($M = 16$).

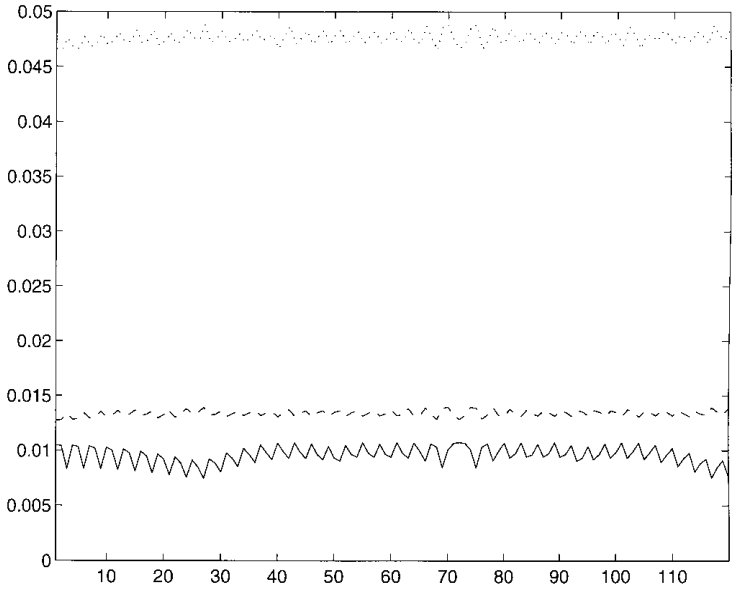


Figure 2.1.2-2. l_1 (dotted), l_2 (dashed), and l_∞ (solid) norms for Williamson (T31).

So we see that for the same resolution, the two numerical methods (our PS and Williamson's implementation of a spectral transform scheme based on spherical harmonics) perform comparably in terms of accuracy.

An error analysis was also done for the PS scheme results for $\alpha = 0$, and $M = 32$ to show how the error does not grow over time when the cosine bell revolves around the sphere. Figure 2.1.2-3 shows the l_∞ norm.

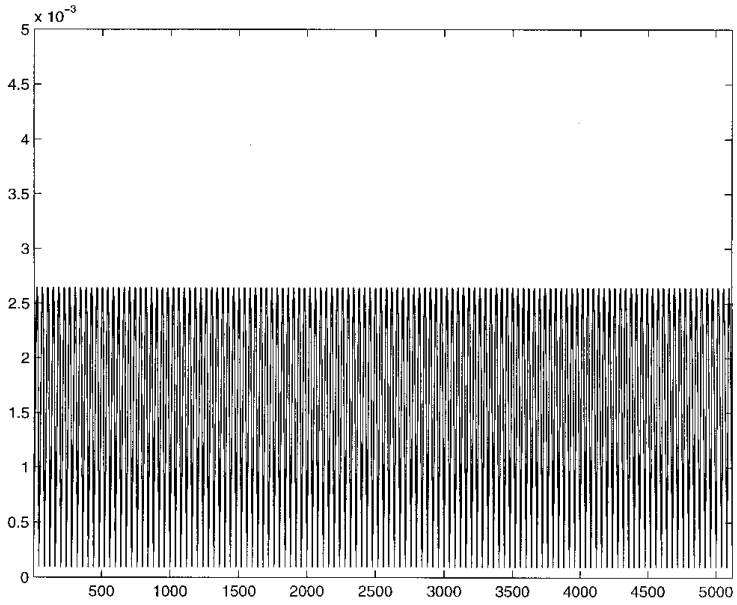


Figure 2.1.2-3. l_∞ norm for $\alpha = 0$, $M = 32$, PS scheme.

The l_1 and l_2 norms are similar in appearance and size. This $(\alpha = 0)$ is the most favorable case in terms of error , whereas, $\alpha = \frac{\pi}{2} - 0.05$ is the most unfavorable case (with the peak passing close to the poles). The $\alpha = \frac{\pi}{2} - 0.05$ case serves as an upper estimate for the error for all choices of α.

2.2 Williamson's Second Case

Holton's second case tests the steady state solution to the shallow water equations,

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

The solid body rotation is given as in Case 1. The Coriolis f is given by

$$f=2\Omega\left(-\cos\varphi\cos\theta\sin\alpha+\sin\theta\cos\alpha\right).$$

where α is given as in Case 1.

Initially $h(\varphi,\theta)$ is described by

$$gh(\varphi,\theta)=gh_0-\left(a\Omega u_0+\frac{{u_0}^2}{2}\right)\left(-\cos\varphi\cos\theta\sin\alpha+\sin\theta\cos\alpha\right)^2$$

or

$$h(\varphi,\theta)=h_0-\frac{1}{g}\left(a\Omega u_0+\frac{{u_0}^2}{2}\right)\left(\frac{f}{2\Omega}\right)^2$$

where $gh_0=2.94\times10^4 m^2/s^2$, g is the gravitational constant, and u_0 and a are as they were in Case 1.

2.2.1 Error Analysis

As one runs the second case, the test pattern should retain the same shape on the sphere throughout the rotation. l_1, l_2, and l_∞ norms were calculated for a PS run with $\alpha = \frac{\pi}{2} - 0.05$ and $M = 32$. These norms are displayed below in Figure 2.2.1-1.

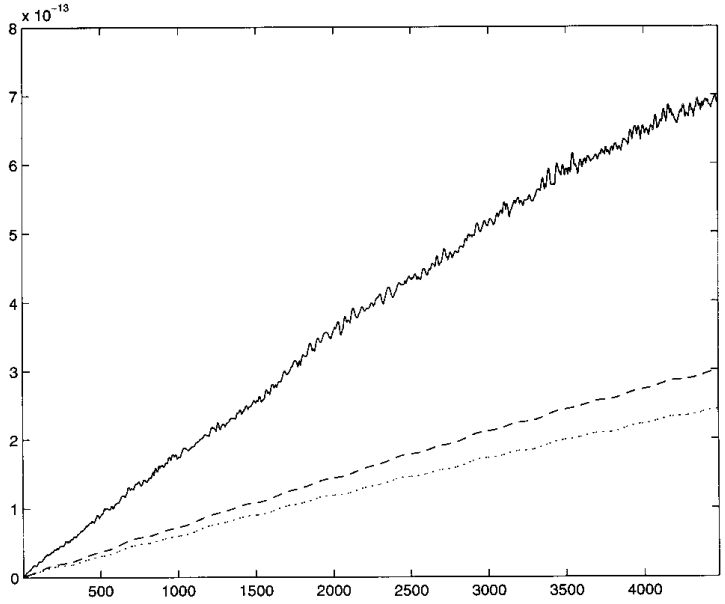


Figure 2.2.1-1. l_1 (dotted), l_2 (dashed), and l_∞ (solid) norms for Case 2 PS run ($M = 32$).

One can see that the norms are on the order of 10^{-13}. This is because the initial conditions for u, v and h contain no high frequencies and the solution is stationary in time. What we see in Figure 2.2.1-1 is the growth of rounding and not tuncation errors.

Chapter 3

Conclusion

We have compared a latitude-longitude based Pseudospectral scheme to finite difference schemes and a Spherical Harmonic based scheme for the shallow water equations on a sphere.

In terms of accuracy , the second and fourth order finite difference schemes are not nearly as good as our Pseudospectral scheme, while Williamson's spectral transform scheme based on spherical harmonics scheme is comparable.

In terms of operation counts for each time step, our Pseudospectral scheme has roughly one order of magnitude less operations than Williamson's implementation for a similar grid resolution.

Williamson's implementation allows for various grid resolutions (T31, T42, T63, etc.). To best utilize FFT's our PS method is confined to $2M$ points north-south and 4Mpoints east-west where M is a power of two. The present PS method is particularly easy to implement since it is associated with an orthogonal grid, and all spatial derivatives are obtained by one dimensional approximations along grid lines.

We tested the PS scheme for test case one and two and obtained good accuracy and performance for all values of a (the direction of convection relative to the coordinate system). 'Traditional' pole problems did not arise. All cases, in particular the traditionally difficult ones of $\alpha = \frac{\pi}{2}$ and $\alpha = \frac{\pi}{2} - 0.05$ (where the cosine bell passes over or near the poles respectively) were completed without numerical instabilities.

BIBLIOGRAPHY

1. B. Fornberg, *A Practical Guide to Pseudospectral Methods*, Cambridge University Press, (1996).

2. B. Fornberg, A Pseudospectral Approach for Polar and Spherical Geometries, *SIAM Journal of Scientific Computing*, Vol. 16, No. 5, pp. 1071-1081, Sept. 1995.

3. J. R. Holton, *An Introduction to Dynamic Meteorology*, Academic Press, New York, London, (1972).

4. R. Jakob-Chien, Spectral Transform Solutions to the Shallow Water Test Set, *Journal of Computational Physics*, Vol.119, pp. 164-187, (1995).

5. B. Machenhauer, in *Numerical Methods Used in Atmospheric Models*, GARP Pub. Ser. No. 17 (JOC, WMO, Geneva, 1979)

6. P. E. Merilees, The Pseudospectral Approximation Applied to the Shallow Water Equations on a Sphere, *Atmosphere*, Vol. 11, No. 1, pp. 13-20, (1973).

7. P. E. Merilees, Numerical Experiments with the Pseudospectral Method in Spherical Coordinates, *Atmosphere*, Vol. 12, No. 3, pp. 77-96, (1974).

8. S. A. Orzag, "Fourier series on spheres", *Monthly Weather Review* 102:56-75 (1974).

9 W. M.,Washington, *An Introduction to Three-Dimensional Climate Modeling*, Oxford University Press, 1986.

10. D. L. Williamson et al., A Standard Test Set for Numerical Approximations to the Shallow Water Equations in Spherical Geometry, *Journal of Computational Physics*, Vol. 102, No. 1 Sept. 1992.

11. D. L. Williamson et al., A Standard Test Set for Numerical Approximations to the Shallow Water Equations in Spherical Geometry with Example Solutions, *Computer Hardware, Advanced Mathematics and Model Physics Pilot Project Final Report*, U.S. Department of Energy, p. 8 May. 1992.

Appendix

A. Software and Graphics Description

The code was written in standard Fortran-90. It was compiled and executed on a Sun workstation with SunOS 5.5. All data for plots were formatted for Matlab and the plots themselves were made with Matlab V5.0 (with the exception of Figure 2.1-1 which was made with Mathematica V2.2).

B. Code Listing

Each individual file that contributed to the numerical analysis will be listed with a short description preceding the code.

B1. Case 1

This code ran Williamson's first case. It set up the initial height and wind fields, did the time stepping and smoothing and called subroutines to do the error analysis for various methods (FD2, FD4, and PS), values of α (the direction of convection relative to the coordinate system) and M (grid resolution).

```
     PROGRAM CASE_1
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     PARAMETER (M = 16)
 DIMENSION HEIGHT(-2*M:2*M,1:2*M),U(-2*M:2*M,1:2*M),V(-2*M:2*M,1:2*M),
   1       HEIGHT_INIT(-2*M:2*M,1:2*M), HEIGHT_ANALYTIC(-2*M:2*M,1:2*M)

C--- Declare a few variables that we'll need in the program

     WRITE *, 'M = ', M
```

```
      M2 = 2*M
      PI = 4.D0*ATAN(1.D0)          ! Get machine precision for Pi
      HALF_PI = 0.5D0*PI
      FOURTH_PI = 0.25D0*PI
      EIGHTH_PI = 0.125D0*PI

      H  = HALF_PI/M

C--- METHOD Method for space discretization:
C           1   FD2
C           2   FD4
C           3   PS
C    NRAT   Ratio space/time steps
C--- NTPR   Number of time steps per revolution

      METHOD = 1                    ! Choose which method.

C--- Set NRAT to the proper value. (determined empirically)

      NRAT = 20

      IF(M.EQ.32) THEN
        NRAT = 40
      ENDIF

      WRITE *, 'NRAT = ', NRAT

      NTPR   = 4*NRAT*M                    ! Calculate # time steps/revolution
      WRITE *, 'METHOD = ', METHOD
      WRITE *, 'NTPR = ', NTPR

C
C--- ALPHA is the angle between the axis of solid body rotation and the
C    polar axis of the shpherical coordinate system. ALPHA = 0 implies
C    rotation aligned with the equator. (see assignments to U() and V() )
C

      ALPHA = HALF_PI - 0.05D0
      WRITE *, 'ALPHA = ', ALPHA

C--- Put the initial wind data in U() and V(). ---
C
C--- A is the mean radius of the earth.
C--- U_NOT is the advecting wind velocity 2*PI*A/(12 days) in m/s.

      A = 6.37122D0*10**6
    U_NOT = 2.0D0*PI*A/(12.0D0*24.0D0*60.0D0*60.0D0)

      DO 14 I=-M2,M2

        FI = I*H

        DO 12 J=1,M2

          TH = ( (J-1) - M + 0.5D0 )*H

          TPI = 3.D0*HALF_PI
        U(I,J) = U_NOT*( COS(TH)*COS(ALPHA) + SIN(TH)*COS(FI+TPI)*SIN(ALPHA) )
         V(I,J) = -U_NOT*( SIN(FI+TPI)*SIN(ALPHA) )

   12    CONTINUE

   14 CONTINUE
```

```
C--- Put the initial Height field in HEIGHT and HEIGHT_INIT.

    CALL ANALYTIC(HEIGHT,0.D0,0.D0,M)
   CALL ANALYTIC(HEIGHT_INIT,0.D0,0.D0,M)

C--- Display initial data if we want.  ---
C     CALL SHOW_U(U,M,ALPHA)
C     CALL SHOW_V(V,M,ALPHA)
   CALL SHOW_MATLAB (HEIGHT,1000.D0,M,0,ALPHA,METHOD,NT,NRAT)

C--- Set up the size of the time steps and the number of time steps. ---

     DT = 2.D0*PI*A/(U_NOT*NTPR)          ! NTPR steps for full turn.
      NT = NTPR                                  ! Take a 'full' turn.

      WRITE *, 'DT = ', DT
      WRITE *, 'NT = ', NT

C--- Loop to step forward in time  ---
      DO 16 L=1,NT

       CALL RK4 (HEIGHT,U,V,M,DT,METHOD)  ! Do one time step.

       CALL DSMOOTH (HEIGHT,M,1.D0,1.D0)  ! Smooth after that one step.

     CALL ANALYTIC(HEIGHT_ANALYTIC,1.D0*L*DT,ALPHA,M) ! Calc analytic sol.

     CALL CALC_NORMS(HEIGHT,HEIGHT_ANALYTIC,L,NT,M)    ! Calc Error Norms.

        WRITE *, 'TIME STEP ',L,' (of ', NT, ') DONE'

   16 CONTINUE

C--- Show the final data. ---
   CALL SHOW_MATLAB (HEIGHT,1000.D0,M,1,ALPHA,METHOD,NT,NRAT)

      END

C-----------------------------------------------------------------------
    SUBROUTINE RK4 (HEIGHT,U,V,M,DT,METHOD)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
  DIMENSION HEIGHT(-2*M:2*M,1:2*M),U(-2*M:2*M,1:2*M),V(-2*M:2*M,1:2*M)

C--- Declare an allocatable array RKH

C--- This works with f90 ---
     ALLOCATABLE RKH(:,:,:)

C--- Allocate space for the array ---
    ALLOCATE( RKH(-2*M:2*M,1:2*M,0:4) )

      M2 = 2*M

C --- Calculate 1 time step with Runge Kutta Order 4, recall          ---
C --- Y(n+1) = y(n) = (1/6)*[ d(1) + 2d(2) + 2d(3) + d(4)]            ---
C --- RK(I,J,0) is just workspace,that's why RK is dimensioned (:,:,0:4) ---

C--- Get d(1) into RKH(I,J,1).
      DO 10 I=-M2,M2
         DO 10 J=1,M2
   10      RKH(I,J,0) = HEIGHT(I,J)
    CALL EVAL (RKH,U,V,RKH(-M2,1,1),M,METHOD)

C--- Get d(2) into RKH(I,J,2).
```

```
      DO 12 I=-M2,M2
         DO 12 J=1,M2
   12     RKH(I,J,0) = ( HEIGHT(I,J)+0.5D0*DT*RKH(I,J,1) )
     CALL EVAL (RKH,U,V,RKH(-M2,1,2),M,METHOD)

C--- Get d(3) into RKH(I,J,3).
      DO 14 I=-M2,M2
         DO 14 J=1,M2
   14     RKH(I,J,0) = ( HEIGHT(I,J)+0.5D0*DT*RKH(I,J,2) )
     CALL EVAL (RKH,U,V,RKH(-M2,1,3),M,METHOD)

C--- Get d(4) into RKH(I,J,4).
      DO 16 I=-M2,M2
         DO 16 J=1,M2
   16     RKH(I,J,0) = ( HEIGHT(I,J)+DT*RKH(I,J,3) )
     CALL EVAL (RKH,U,V,RKH(-M2,1,4),M,METHOD)

C--- Put the different parts of the RK algorithm
C--- together to get one time step
      DO 18 I=-M2,M2
         DO 18 J=1,M2
         HEIGHT(I,J) = HEIGHT(I,J) +
     1          DT*(RKH(I,J,1)+2.D0*(RKH(I,J,2)+RKH(I,J,3))+RKH(I,J,4))/6.D0

   18 CONTINUE

      DEALLOCATE( RKH )                          ! Release space used by RKH

      RETURN
      END

C-------------------------------------------------------------------------
     SUBROUTINE EVAL (RKH,U,V,DRKH_DT,M,METHOD)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
   DIMENSION RKH(-2*M:2*M,1:2*M),U(-2*M:2*M,1:2*M),V(-2*M:2*M,1:2*M),
     1         DRKH_DT(-2*M:2*M,1:2*M)

C--- Declare some allocatable arrays to hold various derivatives.

      ALLOCATABLE DH_DFI(:,:)
      ALLOCATABLE DH_DTH(:,:)
      ALLOCATABLE DU_DFI(:,:)
      ALLOCATABLE DV_DTH(:,:)

C--- Allocate space for those arrays.

    ALLOCATE( DH_DFI(-2*M:2*M,1:2*M), DH_DTH(-2*M:2*M,1:2*M) )
    ALLOCATE( DU_DFI(-2*M:2*M,1:2*M), DV_DTH(-2*M:2*M,1:2*M) )

      PI = 4.D0*ATAN(1.D0)
      H  = 0.5D0*PI/M
      M2 = 2*M

C--- Calculate DH_DFI, DH_DTH, DU_DFI and DV_DTH.

     CALL D_FI(RKH,DH_DFI,METHOD,M)
     CALL D_TH(RKH,DH_DTH,METHOD,M,1.D0)
      CALL D_FI(U,DU_DFI,METHOD,M)
     CALL D_TH(V,DV_DTH,METHOD,M,-1.D0)

      A = 6.37122D0*10**6
      A_INV = 1.D0/A

C--- Put everything together.
```

```
      DO 20 I=-M2,M2

       DO 20 J=1,M2

         TH = ( (J-1) - M + 0.5D0 )*H

      DRKH_DT(I,J) = -A_INV*(U(I,J)*DH_DFI(I,J)/COS(TH) + V(I,J)*DH_DTH(I,J))
     1               - RKH(I,J)*A_INV*( DU_DFI(I,J)/COS(TH)
     2                                  + DV_DTH(I,J) - V(I,J)*TAN(TH) )

   20  CONTINUE

     DEALLOCATE( DH_DFI, DH_DTH, DU_DFI, DV_DTH )       ! Release array space.

C---
      RETURN
      END
```

B2. Case 2

This code ran Williamson's second case. It set up the initial height and wind fields, did the time stepping and smoothing and called subroutines to do the error analysis for various methods (FD2, FD4, and PS), values of α (the direction of convection relative to the coordinate system) and M (grid resolution).

```
      PROGRAM CASE_2
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (M = 16)
  DIMENSION HEIGHT(-2*M:2*M,1:2*M),U(-2*M:2*M,1:2*M),V(-2*M:2*M,1:2*M),
     1         HEIGHT_INIT(-2*M:2*M,1:2*M)

C--- Declare a few variables that we'll need in the program

      WRITE *, 'M = ', M

      M2 = 2*M
     PI = 4.D0*ATAN(1.D0)          ! Get machine precision for Pi
     HALF_PI = 0.5D0*PI
     FOURTH_PI = 0.25D0*PI
     EIGHTH_PI = 0.125D0*PI

      H  = HALF_PI/M

C--- METHOD Method for space discretization:
C           1   FD2
C           2   FD4
C           3   PS
C    NRAT   Ratio space/time steps
C--- NTPR   Number of time steps per revolution
```

```
      METHOD = 3

     IF(METHOD.EQ.1) THEN
       NRAT = 20
      ELSE
       NRAT = 70
      ENDIF

     WRITE *, 'NRAT = ', NRAT

      NTPR   = 4*NRAT*M                    ! Calculate # time steps/revolution
     WRITE *, 'METHOD = ', METHOD
     WRITE *, 'NTPR = ', NTPR

C--- A is the mean radius of the earth.
C--- OMEGA is the rate of rotation of the earth.
C--- G is the acceleration due to gravity.
C
C--- ALPHA is the angle between the axis of solid body rotation and the
C   polar axis of the shpherical coordinate system. ALPHA = 0 implies
C   rotation aligned with the equator. (see assignments to U() and V() )
C
C--- U_NOT is the advecting wind velocity 2*PI*A/(12 days) in m/s.
C--- H_NOT is the height parameter given in the Williamson paper for Case 2.

     A = 6.37122D0*10**6
    OMEGA = 7.292D0/(1.D0*10**5)
     G = 9.80616D0
     G_INV = 1.D0/G

     WRITE *, 'A = ', A
     WRITE *, 'OMEGA = ', OMEGA
     WRITE *, 'G = ', G
     WRITE *, 'G_INV = ', G_INV

     ALPHA = HALF_PI - 0.05D0
   U_NOT = 2.0D0*PI*A/(12.0D0*24.0D0*60.0D0*60.0D0)
    H_NOT = (2.94D0*10**4)*G_INV

     WRITE *, 'ALPHA = ', ALPHA
     WRITE *, 'U_NOT = ', U_NOT
     WRITE *, 'H_NOT = ', H_NOT
     WRITE *, 'G*H_NOT = ', G*H_NOT

C--- Put the initial data in HEIGHT() and U() V(). ---
     DO 14 I=-M2,M2

        FI = I*H

        DO 12 J=1,M2

           TH = ( (J-1) - M + 0.5D0 )*H

         HEIGHT(I,J) = H_NOT - G_INV*( A*OMEGA*U_NOT + (U_NOT**2)/2.D0 )*
    1                (-COS(FI)*COS(TH)*SIN(ALPHA) + SIN(TH)*COS(ALPHA))**2

          HEIGHT_INIT(I,J) = HEIGHT(I,J)

           TPI = 3.D0*HALF_PI
         U(I,J) = U_NOT*( COS(TH)*COS(ALPHA) + SIN(TH)*COS(FI+TPI)*SIN(ALPHA) )
          V(I,J) = -U_NOT*( SIN(FI+TPI)*SIN(ALPHA) )

   12    CONTINUE
```

```
  14 CONTINUE

C--- Display initial data if we want.  ---
C     CALL SHOW_U(U,M,ALPHA)
C     CALL SHOW_V(V,M,ALPHA)
   CALL SHOW_MATLAB (HEIGHT,H_NOT,M,0,ALPHA,METHOD,NT,NRAT)

C--- Set up the size of the time steps and the number of time steps. ---

     DT = 2.D0*PI*A/(U_NOT*NTPR)          ! NTPR steps for full turn.
      NT = NTPR                                  ! Take a 'full' turn.
      WRITE *, 'DT = ', DT
      WRITE *, 'NT = ', NT

      SUM = 0.D0
    CALL APPROX_SURFACE_INTEGRAL (HEIGHT,M,SUM)

C--- Loop to step forward in time  ---
      DO 16 L=1,NT

       CALL RK4 (HEIGHT,U,V,M,DT,METHOD,ALPHA,L)       ! Do one time step.

       CALL DSMOOTH (HEIGHT,M,1.D0,1.D0)        ! Smooth data.
       CALL DSMOOTH (U,M,1.D0,1.D0)
       CALL DSMOOTH (V,M,1.D0,1.D0)

      CALL CALC_NORMS(HEIGHT,HEIGHT_INIT,L,NT,M)

        WRITE *, 'TIME STEP ',L,' (of ', NT, ') DONE'

  16 CONTINUE

C--- Show the final data.  ---
   CALL SHOW_MATLAB (HEIGHT,H_NOT,M,1,ALPHA,METHOD,NT,NRAT)

C--- Do some one line error analysis for ourselves.
      FSUM = 0.D0
    CALL APPROX_SURFACE_INTEGRAL (HEIGHT,M,FSUM)
     WRITE *, 'Before Time Stepping finishes, SUM = ', SUM
     WRITE *, 'After Time Stepping finishes,  SUM = ', FSUM
     WRITE *, 'Differnce = ', SUM - FSUM

      END

C-----------------------------------------------------------------------
    SUBROUTINE RK4 (HEIGHT,U,V,M,DT,METHOD,ALPHA,K)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
  DIMENSION HEIGHT(-2*M:2*M,1:2*M),U(-2*M:2*M,1:2*M),V(-2*M:2*M,1:2*M)

C--- Declare an allocatable array RK

C--- This works with f90 ---
     ALLOCATABLE RKH(:,:,:)
     ALLOCATABLE RKU(:,:,:)
     ALLOCATABLE RKV(:,:,:)

C--- Allocate space for the array ---
    ALLOCATE( RKH(-2*M:2*M,1:2*M,0:4) )
    ALLOCATE( RKU(-2*M:2*M,1:2*M,0:4) )
    ALLOCATE( RKV(-2*M:2*M,1:2*M,0:4) )

      M2 = 2*M

C --- Calculate 1 time step with Runge Kutta Order 4, recall          ---
```

```
C --- Y(n+1) = y(n) = (1/6)*[ d(1) + 2d(2) + 2d(3) + d(4)]            ---
C --- RK(I,J,0) is just workspace,that's why RK is dimensioned (:,:,0:4) ---

C--- Get d(1) into RK(I,J,1).
     DO 10 I=-M2,M2
        DO 10 J=1,M2
          RKH(I,J,0) = HEIGHT(I,J)
          RKU(I,J,0) = U(I,J)
  10      RKV(I,J,0) = V(I,J)
   CALL EVAL (RKH(-M2,1,0),RKU(-M2,1,0),RKV(-M2,1,0),
    1        RKH(-M2,1,1),RKU(-M2,1,1),RKV(-M2,1,1),M,METHOD,ALPHA)

C--- Get d(2) into RK(I,J,2).
     DO 12 I=-M2,M2
        DO 12 J=1,M2
         RKH(I,J,0) = HEIGHT(I,J) + 0.5D0*DT*RKH(I,J,1)
          RKU(I,J,0) = U(I,J)     + 0.5D0*DT*RKU(I,J,1)
  12      RKV(I,J,0) = V(I,J)     + 0.5D0*DT*RKV(I,J,1)
   CALL EVAL (RKH(-M2,1,0),RKU(-M2,1,0),RKV(-M2,1,0),
    1        RKH(-M2,1,2),RKU(-M2,1,2),RKV(-M2,1,2),M,METHOD,ALPHA)

C--- Get d(3) into RK(I,J,3).
     DO 14 I=-M2,M2
        DO 14 J=1,M2
         RKH(I,J,0) = HEIGHT(I,J) + 0.5D0*DT*RKH(I,J,2)
          RKU(I,J,0) = U(I,J)     + 0.5D0*DT*RKU(I,J,2)
  14      RKV(I,J,0) = V(I,J)     + 0.5D0*DT*RKV(I,J,2)
   CALL EVAL (RKH(-M2,1,0),RKU(-M2,1,0),RKV(-M2,1,0),
    1        RKH(-M2,1,3),RKU(-M2,1,3),RKV(-M2,1,3),M,METHOD,ALPHA)

C--- Get d(4) into RK(I,J,4).
     DO 16 I=-M2,M2
        DO 16 J=1,M2
          RKH(I,J,0) = HEIGHT(I,J) + DT*RKH(I,J,3)
          RKU(I,J,0) = U(I,J)     + DT*RKU(I,J,3)
  16      RKV(I,J,0) = V(I,J)     + DT*RKV(I,J,3)
   CALL EVAL (RKH(-M2,1,0),RKU(-M2,1,0),RKV(-M2,1,0),
    1        RKH(-M2,1,4),RKU(-M2,1,4),RKV(-M2,1,4),M,METHOD,ALPHA)

C--- Put the different parts of the RK algorithm
C--- together to get one time step
     DO 18 I=-M2,M2
        DO 18 J=1,M2
          HEIGHT(I,J) = HEIGHT(I,J) +
    1        DT*(RKH(I,J,1)+2.D0*(RKH(I,J,2)+RKH(I,J,3))+RKH(I,J,4))/6.D0
          U(I,J) = U(I,J) +
    2        DT*(RKU(I,J,1)+2.D0*(RKU(I,J,2)+RKU(I,J,3))+RKU(I,J,4))/6.D0
          V(I,J) = V(I,J) +
    3        DT*(RKV(I,J,1)+2.D0*(RKV(I,J,2)+RKV(I,J,3))+RKV(I,J,4))/6.D0

  18 CONTINUE

    DEALLOCATE( RKH, RKU, RKV )          ! Release space used by arrays.

     RETURN
     END

C-------------------------------------------------------------------------
  SUBROUTINE EVAL (RKH,RKU,RKV,DRKH_DT,DRKU_DT,DRKV_DT,M,METHOD,ALPHA)
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)
 DIMENSION RKH(-2*M:2*M,1:2*M),RKU(-2*M:2*M,1:2*M),RKV(-2*M:2*M,1:2*M),
    1       DRKH_DT(-2*M:2*M,1:2*M),DRKU_DT(-2*M:2*M,1:2*M),
    2         DRKV_DT(-2*M:2*M,1:2*M)
```

```
C--- Declare some allocatable arrays to hold various derivatives.

     ALLOCATABLE DH_DFI(:,:)                              ! This works with f90.
     ALLOCATABLE DH_DTH(:,:)
     ALLOCATABLE DU_DFI(:,:)
     ALLOCATABLE DU_DTH(:,:)
     ALLOCATABLE DV_DFI(:,:)
     ALLOCATABLE DV_DTH(:,:)

C--- Allocate space for the arrays.

   ALLOCATE( DH_DFI(-2*M:2*M,1:2*M), DH_DTH(-2*M:2*M,1:2*M) )
   ALLOCATE( DU_DFI(-2*M:2*M,1:2*M), DU_DTH(-2*M:2*M,1:2*M) )
   ALLOCATE( DV_DFI(-2*M:2*M,1:2*M), DV_DTH(-2*M:2*M,1:2*M) )

     PI = 4.D0*ATAN(1.D0)
     H  = 0.5D0*PI/M
     M2 = 2*M

C--- Calculate DH_DFI, DH_DTH, DU_DFI, DU_DTH, and DV_DFI.

    CALL D_FI(RKH,DH_DFI,METHOD,M)
    CALL D_TH(RKH,DH_DTH,METHOD,M, 1.D0)
    CALL D_FI(RKU,DU_DFI,METHOD,M)
   CALL D_TH(RKU,DU_DTH,METHOD,M,-1.D0)
    CALL D_FI(RKV,DV_DFI,METHOD,M)
   CALL D_TH(RKV,DV_DTH,METHOD,M,-1.D0)

     A = 6.37122D0*10**6
     A_INV = 1.D0/A
    OMEGA = 7.292D0/(1.D0*10**5)
     G = 9.80616D0

C--- Put everything together.

     DO 20 I=-M2,M2

      FI = I*H

      DO 20 J=1,M2

        TH = ( (J-1) - M + 0.5D0 )*H

      F = 2.D0*OMEGA*( -COS(FI)*COS(TH)*SIN(ALPHA) + SIN(TH)*COS(ALPHA) )

      DRKH_DT(I,J) = -A_INV*(RKU(I,J)*DH_DFI(I,J)/COS(TH)
     1                                  + RKV(I,J)*DH_DTH(I,J))
     2               - RKH(I,J)*A_INV*( DU_DFI(I,J)/COS(TH)
     3                                 + DV_DTH(I,J) - RKV(I,J)*TAN(TH) )

      DRKU_DT(I,J) = -A_INV*( (RKU(I,J)*DU_DFI(I,J) + G*DH_DFI(I,J))/COS(TH)
     1                        + RKV(I,J)*DU_DTH(I,J) )
     2               + ( F + A_INV*RKU(I,J)*TAN(TH) )*RKV(I,J)

      DRKV_DT(I,J) = -A_INV*( RKU(I,J)*DV_DFI(I,J)/COS(TH) + G*DH_DTH(I,J)
     1                        + RKV(I,J)*DV_DTH(I,J) )
     2               - ( F + A_INV*RKU(I,J)*TAN(TH) )*RKU(I,J)

   20  CONTINUE

    DEALLOCATE( DH_DFI, DH_DTH, DU_DFI, DU_DTH, DV_DFI, DV_DTH )

C---
      RETURN
```

```
      END
```

B3. Calculating the analytic solution

This subroutine calculated the analytic solution for Williamson's first case for any time and any α. It was this subroutine that calculated the analytic solutions to which numerical solutions were compared to. Setting TIME and ALPHA both equal to zero give the initial condition for the first case.

```
    SUBROUTINE ANALYTIC(F,TIME,ALPHA,M)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     DIMENSION F(-2*M:2*M,1:2*M)

      M2 = 2*M

     PI = 4.D0*ATAN(1.D0)                          ! Get machine precision
     HALF_PI = 0.5D0*PI                            ! for PI.

     H = HALF_PI/M

     FI_C = 0.D0                                   ! Original Center
     TH_C = 0.D0                                   ! of feature.

     FI_TR = 0.D0                                  !Initialize...
     TH_TR = 0.D0                                  !...transformation...
     FI_NEW = 0.D0                                 !...variables.
     TH_NEW = 0.D0

C
C--- Calculate how many radians the center has moved
C--- based on TIME
C
C--- A is the mean radius of the earth.
C--- U_NOT is the advecting wind velocity 2*PI*A/(12 days) in m/s.
C--- H_NOT is the height parameter given in the Williamson paper.
C

      A = 6.37122D0*10**6
   U_NOT = 2.0D0*PI*A/(12.0D0*24.0D0*60.0D0*60.0D0)

     DELTA_RAD = U_NOT*TIME/A                                ! Calulate how far...
                                                             ! ...center has moved.
C
C--- Calculate where the center of the feature has rotated to
C--- based on DELTA_RAD.
C


     IF(ALPHA.EQ.0) THEN                                     ! Alpha = 0 is easy.

       FI_C = FI_C + DELTA_RAD
```

```
      ELSE                                                  ! Otherwise tranform.

     CALL TRANSFORM(FI_C,TH_C,FI_TR,TH_TR,ALPHA, 1)         ! Forward Trans.

      IF(ABS(FI_C).GE.HALF_PI) FI_TR = PI - FI_TR           ! Correct fi.
      IF(FI_TR.GE.PI) FI_TR = FI_TR - 2.D0*PI

       FI_TR = FI_TR + DELTA_RAD                            ! Do Rotation.
      IF(FI_TR.GT.PI) FI_TR = FI_TR - 2.D0*PI               ! Keep FI_TR < Pi.

    CALL TRANSFORM(FI_TR,TH_TR,FI_NEW,TH_NEW,ALPHA,-1)      ! Inv. Trans.

      IF(ABS(FI_TR).GE.HALF_PI) FI_NEW = PI - FI_NEW        ! Correct fi.
      IF(FI_NEW.GE.PI)  FI_NEW = FI_NEW - 2.D0*PI

       FI_C = FI_NEW                                        ! Let FI_C and TH_C...
       TH_C = TH_NEW                                        ! hold center coords.

      ENDIF

C
C--- Set up feature based on the new coordinates of the center.
C
C--- R is used in setting up the initial height data.
C--- H_NOT is the height parameter given in the Williamson paper.
C

      R = A/3.D0
      H_NOT = 1000.D0

      DO 10 J=1,M2

         TH = ( (J-1) - M + 0.5D0 )*H                       ! Calculate TH.

         DO 10 I=-M2,M2

            FI = I*H                                        ! Calculate FI.

         RSM = A*ACOS(SIN(TH_C)*SIN(TH)+COS(TH_C)*COS(TH)*COS(FI-FI_C))

            IF(RSM.LT.R) THEN
            F(I,J) = (H_NOT/2.D0)*( 1.D0 + COS( PI*(RSM/R)  )  )
             ELSE
             F(I,J) = 0.D0
             ENDIF

   10    CONTINUE

        END
```

B4. Transforming coordinates

This subroutine transformed coordinates from the (φ, θ) to the (φ', θ') system (and back again) as described in Section 1.2.10.

```
  SUBROUTINE TRANSFORM(RLON,RLAT,ROTLON,ROTLAT,ALPHA,INV)
    IMPLICIT DOUBLE PRECISION (A-H,O-Z)

     PI = 4.D0*ATAN(1.D0)                        ! Get machine precision for Pi
     HALF_PI = 0.5D0*PI

     EPS = 1.D0/10**10                           ! Epislon pertebation term.

C
C--- ROTATED RLATITUDE
C

     IF(INV.EQ.1) THEN                    ! Transform to new coordinate system.

     TEST = -SIN(RLON)*COS(RLAT)*SIN(ALPHA) + SIN(RLAT)*COS(ALPHA)

     ELSEIF(INV.EQ.-1) THEN               ! Transform to old coordinate system.

       TEST=SIN(RLON)*COS(RLAT)*SIN(ALPHA)+SIN(RLAT)*COS(ALPHA)

      ENDIF

     ROTLAT = ASIN(TEST)

     IF(ROTLAT.LT.0.D0) THEN              ! Modify ROTLAT by EPS (epsilon).
       ROTLAT = ROTLAT + EPS
      ELSE
       ROTLAT = ROTLAT - EPS
      ENDIF

C
C--- ROTATED RLONGITUDE
C

     CRL = COS(ROTLAT)

     IF(INV.EQ.1) THEN                    ! Transform to new coordinate system.

     TEST = ( SIN(RLON)*COS(RLAT)*COS(ALPHA) + SIN(RLAT)*SIN(ALPHA) )/CRL

     ELSEIF(INV.EQ.-1) THEN               ! Transform to old coordinate system.

     TEST = ( SIN(RLON)*COS(RLAT)*COS(ALPHA) - SIN(RLAT)*SIN(ALPHA) )/CRL

      ENDIF

     IF(TEST.GE.1.0) THEN                 ! Make sure were not out of bounds
       TEST = HALF_PI                     ! for Arcsin().
     ELSEIF(TEST.LE.-1.0) THEN
       TEST = -HALF_PI
      ELSE
       ROTLON = ASIN(TEST)
      ENDIF
```

```
      RETURN
      END
```

B5.Calculatingderivatives

These two subroutines calculated FD2, FD4 or PS derivatives depending on the value of METHOD.

```
     SUBROUTINE D_FI (F,ANS,METHOD,M)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
    DIMENSION F(-2*M:2*M,1:2*M), ANS(-2*M:2*M,1:2*M)

C--- Declare some allocatable arrays we may need if we do the PS method.

C--- This works with f90 ---
     ALLOCATABLE S1(:)
     ALLOCATABLE S2(:)

      M2 = 2*M
      M3 = 3*M
      M4 = 4*M
     PI = 4.D0*ATAN(1.D0)
      H  = 0.5D0*PI/M
      H2 = 0.5D0/H
     H12 = 1.D0/(12.D0*H)

C-------------------------------------------------
C---  FD2 - SECOND ORDER FINITE DIFFERENCES  ---
C-------------------------------------------------
     IF (METHOD.EQ.1) THEN

        DO 20 J=1,M2

C--- Do right edge and then set left edge = to right edge.

          ANS( M2,J) = (F(-M2+1,J)-F(M2-1,J))*H2
           ANS(-M2,J) = ANS(M2,J)

C--- Do the interior and top and bottom of grid.

           DO 20 I=-M2+1,M2-1

             ANS(I,J) = (F(I+1,J)-F(I-1,J))*H2

   20    CONTINUE

      ENDIF

C-------------------------------------------------
C---  FD4 - FOURTH ORDER FINITE DIFFERENCES  ---
C-------------------------------------------------
     IF (METHOD.EQ.2) THEN

        DO 22 J=1,M2
```

```
C--- Do right edge and then set left edge = right edge.

       ANS(M2,J) = (F(M2-2,J)-F(-M2+2,J)+8.D0*(F(-M2+1,J)-F(M2-1,J)))*H12
          ANS(-M2,J) = ANS(M2,J)

C--- Do next grid point in from right and left side.

       ANS(M2-1,J) = (F(M2-3,J)-F(-M2+1,J)+8.D0*(F(M2,J)-F(M2-2,J)))*H12
       ANS(-M2+1,J) = (F(M2-1,J)-F(-M2+3,J)+8.D0*(F(-M2+2,J)-F(-M2,J)))*H12

C--- Do interior and top and bottom of grid.

          DO 22 I=-M2+2,M2-2

          ANS(I,J) = (F(I-2,J)-F(I+2,J)+8.D0*(F(I+1,J)-F(I-1,J)))*H12

   22    CONTINUE

     ENDIF

C-----------------------------------
C---   PS - PERIODIC PS METHOD   ---
C-----------------------------------
     IF (METHOD.EQ.3) THEN

C--- Allocate space for the arrays S1, S2.

       ALLOCATE( S1(0:4*M-1), S2(0:4*M-1) )

        DO 40 J=1,M2-1,2

           DO 41 I=0,M4-1
              S1(I) = F(I-M2, J)   ! Set up S1() and S2() for the call to
   41         S2(I) = F(I-M2, J+1) ! PS(). (take 2 rows at a time)

           CALL PS (S1,S2,M)              ! Find derivative with PS method.

           DO 42 I=0,M4-1
              ANS(I-M2, J)   = S1(I)      ! Get the answers from S1()
   42         ANS(I-M2, J+1) = S2(I)      ! and S2().

          ANS(M2,J)   = ANS(-M2,J)        ! Set right edge = left edge.
         ANS(M2,J+1) = ANS(-M2,J+1)

   40    CONTINUE

         DEALLOCATE( S1, S2 )             ! Release space for S1 and S2.

     ENDIF

C---
     RETURN
     END
C-------------------------------------------------------------------------
    SUBROUTINE D_TH (F,ANS,METHOD,M,VEC)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
   DIMENSION F(-2*M:2*M,1:2*M), ANS(-2*M:2*M,1:2*M)

C--- Declare some allocatable arrays we may need if we do the PS method.

C--- This works with f90 ---
     ALLOCATABLE S1(:)
     ALLOCATABLE S2(:)
```

```
      M2 = 2*M
      M3 = 3*M
      M4 = 4*M
     PI = 4.D0*ATAN(1.D0)
      H  = 0.5D0*PI/M
      H2 = 0.5D0/H
     H12 = 1.D0/(12.D0*H)

C-------------------------------------------------
C---  FD2 - SECOND ORDER FINITE DIFFERENCES  ---
C-------------------------------------------------
     IF (METHOD.EQ.1) THEN

        DO 24 I=-M2,M2

C--- Calculate correction for going over pole.

           IF (I.LE.0) THEN
              IC = I+M2
           ELSE
              IC = I-M2
           ENDIF

C--- Do top and bottom of grid.

          ANS(I,M2) = ( VEC*F(IC,M2) - F(I,M2-1) )*H2
           ANS(I, 1) = ( F(I, 2)  - VEC*F(IC, 1) )*H2

C--- Do interior, and left and right edges.

           DO 24 J=2,M2-1

             ANS(I,J) = ( F(I,J+1) - F(I,J-1) )*H2

   24    CONTINUE

      ENDIF

C-------------------------------------------------
C---  FD4 - FOURTH ORDER FINITE DIFFERENCES  ---
C-------------------------------------------------
     IF (METHOD.EQ.2) THEN

       DO 26 I=-M2,M2

C--- Calculate correction for going over pole.

        IF (I.LE.0) THEN
          IC = I+M2
        ELSE
          IC = I-M2
        ENDIF

C--- Do grid points that are one grid point away from the top and bottom.

    ANS(I,M2-1) = (F(I,M2-3)-VEC*F(IC,M2)+8.D0*(F(I,M2)-F(I,M2-2)))*H12
     ANS(I,2) = (VEC*F(IC,1)-F(I,4)+8.D0*(F(I,3)-F(I,1)))*H12

C--- Do top and bottom of grid.

    ANS(I,M2) = (F(I,M2-2)-VEC*F(IC,M2-1)+8.D0*(VEC*F(IC,M2)-F(I,M2-1)))*H12
     ANS(I,1) = (VEC*F(IC,2)-F(I,3)+8.D0*(F(I,2)-VEC*F(IC,1)))*H12

C--- Do interior, and left and right edges.
```

```
       DO 26 J=3,M2-2

       ANS(I,J) = (F(I,J-2)-F(I,J+2)+8.D0*(F(I,J+1)-F(I,J-1)))*H12

  26  CONTINUE

      ENDIF

C-----------------------------------
C---  PS - PERIODIC PS METHOD  ---
C-----------------------------------
     IF (METHOD.EQ.3) THEN

C--- Allocate space for the arrays S1, S2.

       ALLOCATE( S1(0:4*M-1), S2(0:4*M-1) )

        DO 45 I=0,M2-2,2

           DO 46 J=1,M2
              S1( M2-J )   = F(I,J)
             S1( M2-1+J ) = VEC*F(I-M2,J)
              S2( M2-J )   = F(I+1,J)
  46          S2( M2-1+J ) = VEC*F(I+1-M2,J)

           CALL PS (S1,S2,M)

        DO 47 J=1,M2

           ANS(I,J)      = -S1( M2-J )
          ANS(I-M2,J)   = VEC*S1( M2-1+J )
           ANS(I+1,J)    = -S2( M2-J )
  47      ANS(I+1-M2,J) = VEC*S2( M2-1+J )

  45       CONTINUE

        DO 49 J=1,M2
  49      ANS(M2,J) = ANS(-M2,J)

        DEALLOCATE( S1, S2 )          ! Release space used by S1, S2.

      ENDIF

C---
      RETURN
      END
C-------------------------------------------------------------------------
     SUBROUTINE PS (A,B,M)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     DIMENSION A(0:4*M-1),B(0:4*M-1)

      M2 = 2*M
      M4 = 4*M

      CALL FFT (A,B,M4,-1)            ! Transform to fourier space

      A(0) = 0.D0                    ! 0th mode doesn't add to derivative.
      B(0) = 0.D0                    ! 0th mode doesn't add to derivative.
      A(M2) = 0.D0                    ! 1st derivative of M2 mode = 0.
      B(M2) = 0.D0                    ! 1st derivative of M2 mode = 0.

C --- Do the Pseudospectral method ---
      DO 10 I=1,M2-1
```

```
         FC = 1.D0*I/M4              ! FC is what we mult the modes by.
         T    = B(I)
         B(I) = A(I)*FC
         A(I) = -T  *FC
         T       = B(M4-I)
        B(M4-I) = -A(M4-I)*FC
   10    A(M4-I) = T       *FC

       CALL FFT (A,B,M4,+1)           ! Transform to real space

       RETURN
       END
```

B6. Fast Fourier Transform

This subroutine performs the Fast Fourier Transform necessary to obtain Pseudospectral derivatives.

```
      SUBROUTINE FFT (A,B,N,IS)
+C----------------------------------------------------------------------+
 C-- | A CALL TO  FFT  REPLACES THE COMPLEX DATA VALUES  A(J) + i B(J),  |
 C-- | J=0,1,..,N-1 WHITH THEIR TRANSFORM                               |
 C-- |                                                                  |
 C-- |                              2 i IS PI K J / N                   |
 C-- |    SUM     (A(K) + i B(K)) e                 ,  J=0,1,..,N-1    |
 C-- |  K=0..N-1                                                        |
 C-- |                                                                  |
 C-- | N      NUMBER OF DATA VALUES, MUST BE A POWER OF TWO             |
 C-- | IS     USE  +1 OR -1  FOR DIRECT AND INVERSE TRANSFORM RESP.     |
 C-- |                                                                  |
+C----------------------------------------------------------------------+
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(*),B(*)
       J = 1
 C---  APPLY PERMUTATION MATRIX  ---------------
      DO 20 I=1,N-1
         IF (I.LT.J) THEN
            TR   = A(J)
            A(J) = A(I)
            A(I) = TR
            TI   = B(J)
            B(J) = B(I)
            B(I) = TI
         ENDIF
         K = N/2
   10    IF (K.LT.J) THEN
            J = J-K
            K = K/2
            GOTO 10
         ENDIF
   20    J = J+K
 C---  PERFORM THE LOG2 N MATRIX-VECTOR MULT. ---
      S =  0.0D0
      C = -1.0D0
```

```
   L = 1
30    LH = L
      L  = L+L
      UR = 1.0D0
      UI = 0.0D0
      DO 50 J=1,LH
         DO 40 I=J,N,L
            IP = I+LH
           TR = A(IP)*UR-B(IP)*UI
           TI = A(IP)*UI+B(IP)*UR
            A(IP) = A(I)-TR
            B(IP) = B(I)-TI
            A(I) = A(I)+TR
40          B(I) = B(I)+TI
        TI = UR*S+UI*C
        UR = UR*C-UI*S
50       UI = TI
       S = SQRT (0.5D0*(1.0D0-C))*IS
       C = SQRT (0.5D0*(1.0D0+C))
        IF (L.LT.N) GOTO 30
      RETURN
      END
```

B7.Smoothing

This subroutine handles smoothing the height (and also the wind fields in Williamson's second case) after each time step (as described in Section 1.2.7).

```
 SUBROUTINE DSMOOTH (F,M,ALPHA,BETA)
 IMPLICIT DOUBLE PRECISION (A-H,O-Z)
 DIMENSION F(-2*M:2*M,1:2*M)

  ALLOCATABLE WKSP(:,:)                          ! Declare workspace.

  ALLOCATE( WKSP(4*M,2) )                        ! Allocate memory.

   M2 = 2*M
   M4 = 4*M
   DM4 = 1.D0/M4
  PI = 4.D0*ATAN(1.D0)                           ! Get machine precision for Pi.
   PID2 = PI/2.D0
   HALF_PI = 0.5D0*PI
   H = HALF_PI/M

  DO 10 J=M,1,-1

     DO 20 I=1,M4

       WKSP(I,1) = F(-M2+I,J)                    ! Grab rows symmetric with
20     WKSP(I,2) = F(-M2+I,-J+M2+1)              ! respect to the equator.

    CALL FFT (WKSP(1,1),WKSP(1,2),M4,-1)         ! Transform to fourier space

     TH = ( (J-1) - M + 0.5D0 )*H                ! Calculate TH depending on J.
```

```
      IS = BETA*(1.D0-COS(TH))*(M2-1)          ! Find transition point.

      DO 30 I=M2+1-IS,M2+1+IS

         IF(IS.EQ.0) THEN                      ! T tells how much to scale
           T = 0                               ! down individual points
         ELSE                                  ! along a particular row.
         T = (I - (M2+1))*(PID2)/IS
         ENDIF

        SCALE = 1.D0 - ALPHA*COS(T)**2         ! Calculate scaling factor.

        WKSP(I,1) = SCALE*WKSP(I,1)            ! Do the scaling.
        WKSP(I,2) = SCALE*WKSP(I,2)

 30    CONTINUE

      CALL FFT (WKSP(1,1),WKSP(1,2),M4, 1)     ! Transform to real space

       DO 40 I=1,M4                            ! Lay out new F values

         F(-M2+I,J) = WKSP(I,1)*DM4
 40      F(-M2+I,-J+M2+1) = WKSP(I,2)*DM4

      F(-M2,J) = F(M2,J)                       ! Set left edge of each new row
     F(-M2,-J+M2+1) = F(M2,-J+M2+1)            ! = to the right edge.

 10    CONTINUE

    DEALLOCATE( WKSP )                         ! Release space used by WKSP

    RETURN
    END
```

B8. Calculating norms

This subroutine was used to calculate the l_1, l_2, and l_∞ norms after each time step. Output is formatted for Matlab.

```
C-----------------------------------------------------------------------
C
C--- CALC_NORMS() subroutine calculates the L-1, L-2 and L-Infinity
C--- norms of the error in Case 1.
C
C-----------------------------------------------------------------------

  SUBROUTINE CALC_NORMS(F,F_TRUE,I_CUR_TIME_STEP,I_TOTAL_TIME_STEPS,M)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
   DIMENSION F(-2*M:2*M,1:2*M),F_TRUE(-2*M:2*M,1:2*M)

     ALLOCATABLE DIFF(:,:)                     ! Declare difference array.
     ALLOCATABLE WKSP(:,:)                     ! Declare workspace.

    ALLOCATE( DIFF(-2*M:2*M,1:2*M) )           ! Allocate memory.
    ALLOCATE( WKSP(-2*M:2*M,1:2*M) )
```

```
      M2 = 2*M
C
C--- Open the files that we'll need. (and format for matlab)
C

     IF(I_CUR_TIME_STEP.EQ.1) THEN
      OPEN(9,FILE='l1_norm.m')
      OPEN(10,FILE='l2_norm.m')
      OPEN(11,FILE='l_inf_norm.m')
       WRITE (9,*) 'Y = [ '
       WRITE (10,*) 'Y = [ '
       WRITE (11,*) 'Y = [ '
      ENDIF

C
C--- Calculate L-1 Norm.
C

      DO 10 I=-M2+1,M2                          ! Calculate absolute value of
                                                ! difference of numerical and
         DO 10 J=1,M2                           ! and true solution.

           DIFF(I,J) = ABS( F(I,J) - F_TRUE(I,J) )
           WKSP(I,J) = ABS( F_TRUE(I,J) )       ! Calculate absolute
                                                ! value of the
                                                ! true solution.
   10    CONTINUE

       DIFF_INT = 0.D0
     CALL APPROX_SURFACE_INTEGRAL (DIFF,M,DIFF_INT)
       ABS_F_TRUE_INT = 0.D0
     CALL APPROX_SURFACE_INTEGRAL (WKSP,M,ABS_F_TRUE_INT)

      ONE_NORM = DIFF_INT/ABS_F_TRUE_INT

C
C--- Calculate L-2 Norm.
C

      DO 20 I=-M2+1,M2                          ! Calculate square of
                                                ! difference of numerical and
         DO 20 J=1,M2                           ! and true solution.

           DIFF(I,J) = ( F(I,J) - F_TRUE(I,J) )**2
           WKSP(I,J) = F_TRUE(I,J)**2           ! Calculate square of
                                                ! true solution.

   20    CONTINUE

       DIFF_INT = 0.D0
     CALL APPROX_SURFACE_INTEGRAL (DIFF,M,DIFF_INT)
       F_TRUE_SQRD_INT = 0.D0
     CALL APPROX_SURFACE_INTEGRAL (WKSP,M,F_TRUE_SQRD_INT)

      TWO_NORM = SQRT( DIFF_INT/F_TRUE_SQRD_INT )

C
C--- Calculate L-Infinity Norm.
C

      ABS_DIFF_MAX = 0.D0                           !Initialize.
      ABS_TRUE_MAX = 0.D0
```

```
     DO 30 I=-M2+1,M2                              !Calculate maximum difference
                                                   ! in magnitude of numerical and
        DO 30 J=1,M2                               ! and true solution.

          DIFF(I,J) = ABS( F(I,J) - F_TRUE(I,J) )
         IF(DIFF(I,J).GT.ABS_DIFF_MAX) THEN
            ABS_DIFF_MAX = DIFF(I,J)
           ENDIF

          WKSP(I,J) = ABS( F_TRUE(I,J) )           ! Calculate maximum of
         IF(WKSP(I,J).GT.ABS_TRUE_MAX) THEN        ! true solution.
            ABS_TRUE_MAX = WKSP(I,J)
           ENDIF

   30    CONTINUE

    THE_INF_NORM = ABS_DIFF_MAX/ABS_TRUE_MAX

C
C--- Report Norms to their respective files.
C

     WRITE (9,*)  ONE_NORM
     WRITE (10,*) TWO_NORM
     WRITE (11,*) THE_INF_NORM

     DEALLOCATE( DIFF, WKSP )                      ! Release space.

C
C--- If that's the last time step wrap things up.
C

   IF(I_CUR_TIME_STEP.EQ.I_TOTAL_TIME_STEPS) THEN
      WRITE (9,*)  '];'                            ! Format for matlab.
      WRITE (9,*)  'plot(Y);'
      WRITE (10,*) '];'
      WRITE (10,*)  'plot(Y);'
      WRITE (11,*) '];'
      WRITE (11,*)  'plot(Y);'
      CLOSE(UNIT=8)                                ! Close the files.
      CLOSE(UNIT=9)
      CLOSE(UNIT=10)
      CLOSE(UNIT=11)
     ENDIF

     RETURN
     END
```

B9. The surface integral

This subroutine calculates an approximate surface integral $I[\ ]$ that was discussed in section 1.2.10.

```
C-------------------------------------------------------------------------
C
C---APPROX_SURFACE_INTEGRAL()subroutineapproximatestheintegral
C--- of F which is defined on the sphere.
C
C-------------------------------------------------------------------------

    SUBROUTINE APPROX_SURFACE_INTEGRAL (F,M,SUM)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     DIMENSION F(-2*M:2*M,1:2*M)

      M2 = 2*M

     PI = 4.D0*ATAN(1.D0)                          ! Get machine precision for Pi.
     HALF_PI = 0.5D0*PI

     H = HALF_PI/M                                 ! Calculate step size H.

     SUM = 0.D0                                    !Initialize.

     DO 10 I=-M2+1,M2

        DO 10 J=1,M2

           TH = ( (J-1) - M + 0.5D0 )*H

          SUM = SUM + F(I,J)*COS(TH)

   10    CONTINUE

      RETURN
      END
```

B10. Formatting output

This subroutine prints out the height field in a format that Matlab can use to make the plots that were shown in this thesis.

```
C-------------------------------------------------------------------------
C
C--- My SHOW_MATLAB() subroutine which formats the data for matlab.
C--- The parameter K can be either 0 or 1 for output of intial and
C--- final data respectively.
C
C-------------------------------------------------------------------------

   SUBROUTINE SHOW_MATLAB (F,H_NOT,M,K,ALPHA,METHOD,NT,NRAT)
     IMPLICIT DOUBLE PRECISION (A-H,O-Z)
     DIMENSION F(-2*M:2*M,1:2*M)

      M2 = 2*M

     IF(K.EQ.0) THEN                               ! Name the file.
      OPEN(7,FILE='id.m')
      ELSE
      OPEN(7,FILE='fd.m')
```

```
      ENDIF

   12 FORMAT (66E11.2)                              ! For ALPHA = HALF_PI
C   12 FORMAT (132F8.2)                             ! For ALPHA = 0

      IF(M.EQ.8) THEN                               ! Print one format for M=8

       WRITE (7,*) 'Z = [ '
       DO 14 I=M2,-M2,-1
   14    WRITE (7,12) (F(I,J),J=1,M2)
       WRITE (7,*) ' ];'

      ELSEIF(M.EQ.16) THEN                          ! ...and another for M=16.

       WRITE (7,*) 'A = [ '
       DO 16 I=M2,0,-1
   16    WRITE (7,12) (F(I,J),J=1,M)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'B = [ '
       DO 18 I=M2,0,-1
   18    WRITE (7,12) (F(I,J),J=M+1,M2)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'C = [ '
       DO 20 I=-1,-M2,-1
   20    WRITE (7,12) (F(I,J),J=1,M)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'D = [ '
       DO 22 I=-1,-M2,-1
   22    WRITE (7,12) (F(I,J),J=M+1,M2)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'Z = [A B; C D];'

      ELSE                                          ! ...and another for M=32

       WRITE (7,*) 'A = [ '
       DO 24 I=M2,0,-1
   24    WRITE (7,12) (F(I,J),J=1,INT(M/2))
       WRITE (7,*) ' ];'

       WRITE (7,*) 'B = [ '
       DO 26 I=M2,0,-1
   26    WRITE (7,12) (F(I,J),J=INT(M/2)+1,M)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'C = [ '
       DO 28 I=M2,0,-1
   28    WRITE (7,12) (F(I,J),J=M+1,M+INT(M/2))
       WRITE (7,*) ' ];'

       WRITE (7,*) 'D = [ '
       DO 30 I=M2,0,-1
   30    WRITE (7,12) (F(I,J),J=M+INT(M/2)+1,M2)
       WRITE (7,*) ' ];'

       WRITE (7,*) 'E = [ '
       DO 32 I=-1,-M2,-1
   32    WRITE (7,12) (F(I,J),J=1,INT(M/2))
       WRITE (7,*) ' ];'

       WRITE (7,*) 'F = [ '
```

```
      DO 34 I=-1,-M2,-1
  34    WRITE (7,12) (F(I,J),J=INT(M/2)+1,M)
      WRITE (7,*) ' ];'

      WRITE (7,*) 'G = [ '
      DO 36 I=-1,-M2,-1
  36    WRITE (7,12) (F(I,J),J=M+1,M+INT(M/2))
      WRITE (7,*) ' ];'

      WRITE (7,*) 'H = [ '
      DO 38 I=-1,-M2,-1
  38    WRITE (7,12) (F(I,J),J=M+INT(M/2)+1,M2)
      WRITE (7,*) ' ];'

      WRITE (7,*) 'Z = [A B C D; E F G H];'

      ENDIF

     WRITE (7,*) 'mesh(Z);'
      WRITE (7,*) ' '
      IF(K.EQ.0) THEN
     WRITE (7,*) 'title(''Initial Height (METHOD = ',METHOD,', (ALPHA =
',ALPHA,')'');'
      ELSE
     WRITE (7,*) 'title(''Final Height (METHOD = ',METHOD,', ALPHA = ',ALPHA,',
NT = ',NT,', NRAT = ',NRAT,')'');'
      ENDIF
     WRITE (7,*) 'axis([0 ', 2*M+4, ' 0 ', 4*M+8, ' -200 ',INT(H_NOT), ' ]);'
     CLOSE(UNIT=7)
      RETURN

      END
```