Weather and Climate Modelling course MTMW14:
Numerical modelling of the atmospheres and oceans
http://www.met.rdg.ac.uk/cag/courses

The practical assignment:

"Forecasting the weather numerically using the barotropic vorticity equation"

David B. Stephenson
E-mail: D.B.Stephenson@reading.ac.uk
January 8, 2004

Contents

1 Introduction

The earliest numerical weather prediction experiments were carried out with a model based on the barotropic vorticity equation, which treats the atmosphere as a single homogeneous layer of fluid (Charney et al. 1950) . Although very limited, this model was capable of simulating the zonal propagation of waves in the westerlies reasonably realistically. The aim of this practical is to reproduce these experiments on the departmental computers.

You are required to write a program to integrate the barotropic vorticity equation forward in time. This can be thought of as a two dimensional extension of practicals 5 and 6 from course MTMW12. Some initial data are supplied, and the aim is to use your model to produce a 24 and 48 hour "forecast" from this initial data. Before the last friday of this term, you should hand in the following items:

1. A brief write-up (approx. 5 pages) that includes
   - A description of your model, including the finite difference forms of the equations and boundary conditions which you used and your reasons for choosing those particular forms.
   - An explanation of how you tested and calibrated your model.
   - a description and discussion of your results.
   - any optional exercises you have attempted.
2. A printout of your program code.
3. Plots of your 24 and 48 hour forecasts.

In assessing your work, we shall be looking for a sensible choice of numerical method, clear and economical programming and accuracy of the final result. It is not necessary to repeat all the information given on the assignment sheet or information that can easily be read from the program listing. Only include a picture if you have something explicit to say about it in the text of your report. The assignment will be assessed out of 50 marks as follows:

1. Numerical method [15 marks]
   Description of the numerical method and your reasons for choosing it, including accuracy, stability, ease of coding and any other factors you may have considered.
2. Testing and calibration [15 marks]
   How the model solution and choice of parameters, including timestep

and diffusion coefficient, were verified. This verification should be based mainly on inspection of the model results, but will also include any sensitivity tests you may have performed.

3. **Results** [5 marks]
   How your 24 and 48 hour forecasts compare with the reference solution.

4. **FORTRAN code** [10 marks]
   Organisation and readability of you FORTRAN code. Marks may be lost if the program is grossly inefficient due to a lack of understanding of the algorithm or of the FORTRAN language.

5. **Additional exercises** [5 marks]
   Additional insight beyond what is required in the preceding sections. Doing the suggested optional exercises may contribute to this mark, but only if they contribute to a greater understanding of the model. Marks are not given simply for extra programming or plots.

NOTE: Full marks can be obtained by spending a total of 15 hours on this project !

2 Equations and boundary conditions

The barotropic vorticity equation with some eddy viscosity dissipation may be written:

$$\frac{\partial \xi}{\partial t} + \mathbf{v}.\nabla\xi + \beta v = \nu\nabla^2\xi \tag{1}$$

where ξ is the relative vorticity, $\mathbf{v} = (u, v)$is the horizontal velocity vector, and $\beta = \partial f/\partial y$ is the poleward gradient of the Coriolis parameter. In this assignment, use values appropriate for 45°N. The term on the right hand side is an "eddy viscosity" diffusion term which must generally be included to prevent excessive enstrophy cascade to small scales and avoid non-linear computational instability. The velocity is related to a geostrophic streamfunction ψ by:

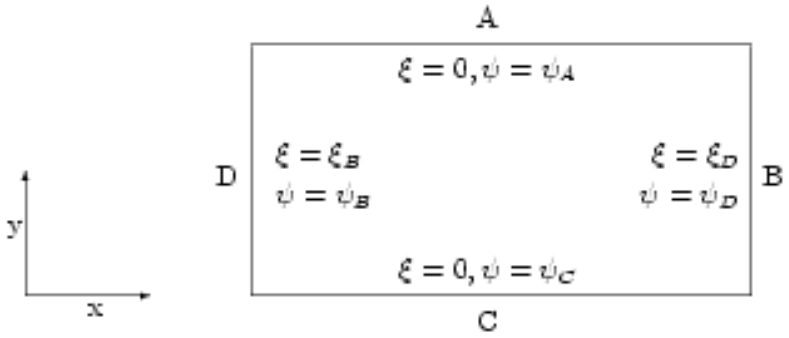
$$u = -\frac{\partial \psi}{\partial y} \tag{2}$$

$$v = \frac{\partial \psi}{\partial x} \tag{3}$$

and the streamfunction is calculated by solving the elliptic "Poisson equation"

$$\nabla^2\psi = \xi \tag{4}$$

that relates streamfunction to relative vorticity (see section 4). For this exercise, the domain of integration is a periodic channel in the x-direction that crudely models a latitude band centered on 45°N. A free-slip condition is applied at the north and south channel walls, that is $v = 0$, and $\partial u/\partial y = 0$ at the boundaries in y. This leads to the conditions on ξ and ψ illustrated below.



The rectangular domain can be divided into a rectangular grid of points (x_i, y_j) indexed by the integers $i = 1, 2, \ldots, N_x$ and $j = 1, 2, \ldots, N_y$:

$$(x_i, y_j) = (x_1 + (i-1)\Delta, y_1 + (j-1)\Delta) \tag{5}$$

where the grid spacing Δ is the same in both the $x-$ and $y-$ directions. Derivatives of ξ and ψ should be approximated by finite difference formulae. You will wish to consider of the accuracy of the formulae you use, as well as their stability properties. One way the periodic boundary conditions in x can be implemented is by setting:

$$\begin{aligned} \xi_{1,j} &\Leftarrow \xi_{N_x-1,j} \\ \xi_{N_x,j} &\Leftarrow \xi_{2,j} \end{aligned} \tag{6}$$

Where $\Leftarrow$ mean the value is assigned (not equals)

and similarly for ψ. $\xi_{i,j} = \xi(x_i, y_j)$ is the value of relative vorticity at grid point (i, j) (and likewise for ψ).

Once your code is working, experiment with the time step and the eddy viscosity diffusivity ν and note the changes to your forecast as you vary them. What is the longest time step you can safely take? Does it agree with the CFL time step?

The eddy viscosity coefficient, ν should be as small as you can get away with! You will wish to experiment with suitable values. As a rough guide, start by trying values for ν in the range $10^{5-7} m^2 s^{-1}$.

3 Initial conditions

The initial conditions are a gridded field of 50 kPa geopotential height $\Phi_{i,j}$. The grid point values are stored in an array of 34×17 points, with a grid spacing of 312.5km. The model domain is actually 32x16 grid intervals in size – the extra points are for the boundary conditions. The data is stored in the file `msc11.dat`, available from
http://www.met.rdg.ac.uk/cag/courses). Type *www* in your xterm window to load up the web, and save the file into your home directory by doing a right click on mouse and *save link as*).

An example Fortran script which reads in the intial file and performs some manipulations can be found on the website and is called *modelbase.f90*. To compile this script type:

```
f90 -o modelb.out modelbase.f90
```

The file can also be read in using the following code:

```
!
!
      INTEGER IX,IY
      INTEGER NXC,NYC
      INTEGER NX,NY
      PARAMETER(NX=34,NY=17)
!
      REAL HT(NX,NY)
      REAL DX
       .
       .
       .
! FIRST THE OPEN STATEMENT ASSOCIATES THE DATA FILE
! WITH YOUR FORTRAN PROGRAM.
      OPEN(10,FILE='MSC11.DAT')
       .
       .
       .
      READ (10,100) NXC,NYC,DX
100   FORMAT(2I4,E15.7)
!
! NXC, NYC PROVIDE A CHECK - THEY SHOULD AGREE
! WITH THE VALUES OF NX, NY DEFINED IN YOUR PROGRAM.
!
      DO IY=1,NY
         READ (10,101) (HT(IX,IY),IX=1,NX)
      END DO
 101  FORMAT(5E15.7)
!
      CLOSE(10)
       .
       .
       .
```

The streamfunction ψ can then be calculated from the geopotential height by using the expression

$$\psi = \frac{g}{f_0}\Phi \tag{7}$$

The initial relative vorticity can then be calculated from the streamfunction by using a finite difference approximation of $\nabla^2\psi = \xi$. Note: take care on the boundaries A and C with the finite differences and use subroutines to perform these operations.

4 How to solve the Poisson equation

To make life easier for you, we have provided you with a subroutine that can solve the Poisson equation for this domain. Given a field of ξ, it returns a field of ψ. The subroutine is called PSOLVE and is in the file psolve.f90, available from
http://www.met.rdg.ac.uk/cag/courses.

To use the routine include the following lines in your program:

```
REAL Z(NX,NY),PSI(NX,NY),WORK(NWK)
   .
   .
   .
CALL PSOLVE(Z,PSI,WORK,NX,NY,DX)
```

where NWK$\geq 2*(NX-2)*(NY-1)$. Note that the method requires that the dimensions of the arrays, NX and NY should satisfy:

$$NX = 2^X + 2, NY = 2^Y + 1, X \text{ and } Y \text{ being integers.}$$

Hence the choice of NX $= 34$ and NY $= 17$ for the initial data.

- On entry, array Z contains the vorticity or "source function" (right hand side of equation 4). It is unaltered on exit.
- Array PSI contains the solution on exit. On entry, it must contain any field which satisfies the boundary conditions on boundaries A and C. Boundary conditions on PSI on boundaries B and D are set automatically by the subroutine. However, initialising the array with all zeros can cause problems, so avoid this.
- WORK is a work array used within the subroutine.
- DX is the gridlength, assumed the same in x and y.

PSOLVE was written by Jason Lander and Sheila Bryant, based on Fast Fourier Transform techniques. It is probably as fast as it is possible to make such a FORTRAN subroutine. It must be linked with your program by "including" it by adding the following statement after the last END statement in your program:

```
INCLUDE 'psolve.f90'
```

If you wish, you can look at the code by opening it with emacs, or looking at it on the web.

Important Note: For PSOLVE to work correctly the upper and lower edges of PSI must be assigned with the intial field: (PSI(I,1) and PSI(I,NY) must have the intial values in them for i=1,2,...nx. This is because inversion of the 2nd order Laplacian requires two boundary conditions in order to give a unique solution.

Appendices

A Some tips on structured programming

To get your code working quickly and efficiently you may want to apply some of the following ideas:

1. Do not rush into writing FORTRAN. Plan your program by writing out the finite difference equations carefully, and by preparing a flow chart for your program. Think before you start writing the code.

2. Structure the program into a number of smaller, self contained tasks – *ie*, setting up initial data, taking a time step, resetting boundary conditions, solving the Poisson equation, plotting out results, etc. The secret of getting the program to work quickly is to carry out each of these separate sub-tasks in separate self-contained subroutines which can be written and tested independently.

3. When you first run a subroutine, put in plenty of write statements to check that variables are containing the values you expect. As the program begins to work, you can remove them. If you do encounter an error, put in extra write statements to locate exactly where the problem first occurs, and if it is not obvious, how it is manifested (*i.e.*, which pass through a loop gives the problem, which subroutine are you in, etc). Take particular care to match the parameters in a subroutine call with those expected by the subroutine.

4. When you do locate a programming error, resist the temptation to correct it and run the program again immediately. Satisfy yourself that you can see why the error produced the results you obtained, and that the correction will change things in the right way. The principle is to understand exactly what you program is doing and why at each stage.

5. Make your code as easily followed and neat as possible. If the code is easy to follow, you make fewer mistakes and it is much easier for someone else to help you. For example, indent the contents of each DO loop

or IF block - this makes no difference to the computer but helps you enormously to identify the ordering of nested logical structures.

6. Add "meaningful" comments as you build up the code. You will be surprised how difficult it can be to follow your own code after a week or two away from it, and it can be impossible for some-one else to understand it without comments. All major loops should be commented, especially when long, and if you must use statement labels, the purpose of each should be explained with a comment.

7. Use sensible and meaningful names for variables, subroutines, etc. You will save negligible time and resource by using single letter names instead of longer, meaningful names, but will waste a great deal of your own time when you cannot follow your program.

8. More guidelines can be found in in "Numerical Recipes" by Press et al (Cambridge University Press, 1989), pages 4-15, which is recommended for further reading once you are happy with the basic FORTRAN language:

B Additional exercises

If you are able to complete the initial forecast quickly, you might wish to try some experiments with your model to learn more about its properties, and about the barotropic vorticity equation. These experiments are suggestions only, and you might like to think of some others:

- **Enhanced diagnostics** - As well as plotting the streamfunction or geopotential height, you should plot the vorticity and see how that evolves during the integration. More interestingly, the continuous equations, in the absence of diffusion, should conserve a quantity called the "potential vorticity" p, *ie*,

$$\frac{Dp}{Dt} = 0 \text{ where } p = \xi + \beta y \tag{8}$$

  Try plotting this to see how well your numerical scheme conserves p.

- **Longer range runs** - When you are quite convinced that your program is working correctly, try carrying out a much longer run of your model. Observe particularly how the vorticity field evolves and changes its character after several days. This is a classical example of what is sometimes called "geostrophic turbulence". You will find the long time evolution of the vorticity field is sensitive to the value of β you prescribe.

- **The butterfly effect** - Add a small local perturbation onto your initial condition and then repeat the forecast. Compare this forecast with the original to see how fast the difference grows in time.

- **Effect of orography** - An extension to the model is to add in a term representing the generation of vorticity by orography:

$$\frac{\partial \xi}{\partial t} + \mathbf{v}.\nabla\xi + \beta v = -\frac{f}{H}\,\mathbf{v}.\nabla h + \nu\nabla^2\xi \qquad (9)$$

  where H is the mean depth of the uniform fluid layer, f is the Coriolis parameter and $h = h(x,y)$ is the height of the orography. If you add this term in, I recommend you define a simple isolated mountain using some analytic function such as $\cos^2[\pi r/2R]$) centred in the middle of the channel. You will run into problems unless $h(x,y) = 0$ on the boundaries of the channel.

C Contour plotting using MATLAB

MATLAB is a useful tool for visualisation of data. It is particularly good at displaying two dimensional fields (such as ψ and ξ in this practical). You should be familiar with using MATLAB from last term's practicals. To get MATLAB started, type `matlab` in your xterm window. To finish MATLAB, enter `quit` at the MATLAB prompt (it is important that you do this when not using MATLAB as the university only has 21 licenses for the UNIX system).

So that MATLAB can read the data produced by your FORTRAN code, you can output it in the following form in your FORTRAN program:

```
   .
   .
OPEN(5,FILE='ht.dat')
DO IX=1,NX
   DO IY=1,NY
      WRITE(5,*) HT(IX,IY)
   ENDDO
ENDDO
CLOSE(5)
   .
   .
```

To read this data into MATLAB, at the MATLAB prompt, type

```
>> ht=textread('ht.dat','%n')
>> ht=reshape(ht,17,34)
```

These commands read the data as a string of numbers and then format the data into a matrix which can be used for plotting. The *%n* command is a formatting statement that tells the command to read numbers.

Some useful plotting commands in Matlab are.

```
contour(ht,n)   :Produces a contour plot of ht with n contour levels
contourf(ht,n)  :Produces a filled contour plot with n contour levels
surf(ht)        :Produces a surface plot of ht
quiver(u,v)     :Produces a plot of a vector field given components

axis([xmin xmax ymin ymax]) :Sets the axis limits
axis equal     :Makes the scaling of the axes equal
title('My Basic Plot')  :Adds a title to the plot
view(angle1,angle2)  :Changes the viewpoint for viewing 3D plots
```

A script file can be written so that Matlab performs many operations at once. To do this open a text file called *ploter.m* and input the Matlab commands as you would do for interactive plotting. When in Matlab type *ploter* at the prompt and the script should run. An example script which plots the diagnostics output by modelbase.f90 is included on the webpage.

If you don't want a command to dump its output to screen then put a semi-colon after the command.

To add more than one figure to a page use the *subplot(n,m,i)* command, where the number of plots per page is n x m and the plot is at position i.

To launch a new window use *figure*.

To plot one field over another (ie wind contours over a height field) use the command *hold on* after the first plot and *hold off* after the second.

You can specify further options to the *contour* command, which are explained in the online MATLAB help. This can be accessed by typing *helpdesk* at the MATLAB prompt. You may also wish to experiment with other methods of visualising your data, using commands such as *surf*.

To print the contour plot, click on 'file', and then 'print', and then choose the printer name 3167psc.

To save the contour plot, click on 'file', and then 'print', and then choose to

save it to a file in your home directory. It is saved in postscript format, which can be imported into a word or latex document.

D Suggested Work Plan

It is important to organise your work on the project from week to week so that sufficient time is left to finish all aspects of the problem, and to make best use of the contact time available. Below is a suggested work plan which shows which aspects of the practical you should aim to finish by the end of each weeks practical session.

Week 1

- Open input data file and read data into code.
- Print field out to a file in a format which can be read by MATLAB.
- Plot this field using Matlab.

Week 2

- Write down clearly the algorithm you are going to use to solve the equations and think carefully about the numerical stability of each term.
- Start to code this into FORTRAN.
- Test each subroutine systematically.

Week 3

- Finish coding the model, including adding the poisson solver.
- Start to investigate the model sensitivity to different parameters.

Week 4

- Finish investigating the basic model.
- Code additional exercises (if you have time).
- Perform experiments for additional sections.

Week 5

- Finish all experiments.
- Select all figures.
- Write up experiment.