NUMERICAL WEATHER PREDICTION USING THE BAROTROPIC VORTICITY EQUATION

Dr David B. Stephenson

d.b.stephenson@reading.ac.uk

http://www.met.rdg.ac.uk/cag/courses/MTMW14

1 January 2005

The first successful numerical weather prediction experiments were made using a model based on the barotropic vorticity equation (Charney, J. G., Fjørtoft, R. and Von Neumann, J. (1950) Numerical integration of the barotropic vorticity equation. Tellus, 2, pp. 237–254). Although very limited, this model was able to qualitatively capture the eastward propagation of synoptic-scale waves over the United States in the mid-latitude westerly flow. It was pivotal in demonstrating that numerical weather prediction was possible and it led the way to all future numerical forecasting developments in atmospheric and oceanic science.

The aim of this assignment is for you to get hands-on experience in numerical weather forecasting by developing your own numerical model and then performing experiments with it. The main aim of this assignment is to produce 24 and 48 hour ahead forecasts of mid-latitude 500hPa geopotential height by numerical time integration of the following dissipative unforced barotropic vorticity equation:

$$\frac{\partial \xi}{\partial t} + u\frac{\partial \xi}{\partial x} + v\frac{\partial \xi}{\partial y} + \beta v = \kappa\left(\frac{\partial^2 \xi}{\partial x^2} + \frac{\partial^2 \xi}{\partial y^2}\right) \qquad (1)$$

where ξ is the relative vorticity, (u,v) is the horizontal velocity vector, and $\beta = \partial f / \partial y$ is the meridional gradient of the Coriolis parameter (i.e. the background vorticity gradient due to differential rotation of the Earth). The term on the right hand side is an "eddy viscosity" diffusion term with eddy diffusivity κ, which has to be included to prevent excessive generation of small-scale noise caused by the non-linear advection term. With no diffusion, the non-linear cascade to small spatial scales can lead to non-linear computational instability. An initial 500hPa geopotential height field is supplied. The program code should be written in Fortran90 (on UNIX machines) and the visualisation of results is best performed using MATLAB.

Before **Friday 18 March 2005**, you should hand in to the Department of Meteorology office (not me!) a concisely written scientific report (not more than 15 pages including figures!) that includes:

- A signed declaration that this is all your own work (plagiarism will be severely penalised!)
- METHOD: 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. A clear explanation of how you carefully tested and tuned your model;
- RESULTS: a scientific discussion of your results and maps showing your 24 and 48 hour forecasts and interesting results from your numerical experiments;
- CODE: a printout of your Fortran90 program code.

It is not necessary to repeat all the information given here or coding 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:

- **Numerical method [10 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.

- **Testing and tuning [10 marks]**
  How the model solution was tested and how you chose the model parameters (the time step and eddy diffusion coefficient). This validation should be based on careful inspection of the model results, and should include systematic tests of subroutines and sensitivity of results to changes in the model parameters;

- **Accuracy of forecasts [10 marks]**
  Whether or not your 24 and 48 hour forecasts appear physically reasonable. You should include a careful interpretation of how the weather systems evolve throughout the forecast period.

- **Numerical experiments [15 marks]**
  Additional scientific insight gained from your numerical experiments performed in week 4;

- **Well-written FORTRAN code [5 marks]**
  Structure and readability of you FORTRAN code. Marks may be lost if the program is inefficient due to a lack of understanding of the algorithm or of the FORTRAN language.

You should aim to complete the modelling and experimenting work in the five weeks of the course. In addition to the 2 hours of supervised practical class per week, you should also expect to spend a couple of hours per week working on your own in order to avoid slipping behind with the practicals. No more than 2 days should be required to write up your final report. The demonstrators and I will provide you with advice and support during the practical classes. It is important to organise your work on the project each week so that sufficient time is available to finish all aspects of the assignment, and to make best use of the contact time.

Week 1: Initial conditions and design of numerical scheme

1. Open an **xterm** window on a UNIX machine and set up and move into a new project directory for your assignment work (e.g. **mkdir baro; cd baro**). To open an xterm window on the University of Reading's UNIX machine main from an ITS PC, double click on the icon labelled *Communications*, then on *Access Unix*, and then on *X-Windows to main*. Type in your username and password and then when a blue menu panel appears click on *xterm*.

2. Use an internet browser (such as Internet Explorer or Netscape) to download the 500hPa geopotential height initial conditions. Start the internet browser and then save the file **msc11.dat** from the course web site **http://www.met.rdg.ac.uk/cag/courses/** into your project directory by doing a right mouse click and choosing **save link as**.

3. Download the sample Fortran code **modelbase.f90** that reads in the initial data. Read through the code carefully using a good editor such as **emacs** (type xemacs in the xterm window) to make sure you fully understand it. Then compile and run the code and write the input data out in a format that can be read by MATLAB. To compile the code type the command **f90 –o exe modelbase.f90** and then run the compiled executable by typing **exe**. If you haven't used the f90 command before on the ITS unix system, you must first enter the command *configure –s –c sw* in the xterm window and then exit before restarting a new xterm window.

4. Visualise this output using MATLAB using the commands given in Appendix C of these notes. MATLAB provides you with a powerful diagnostic tool for visualising what comes out of your model. What synoptic features do you see in the initial data? Can you identify low pressure troughs and high pressure ridges?

Continued on next page …

5. Design a numerical scheme for integrating the barotropic vorticity Eqn. (1):

    a. Read carefully the information provide in Appendices A and B of these notes!

    b. Write down clearly all the differential equations you need to solve this prognostic equation for relative vorticity and think carefully about what each term in the equations does.

    c. Choose a sufficiently accurate finite difference scheme for the horizontal derivatives – think about potential problems you might encounter with boundary conditions.

    d. Which time difference schemes are stable for Eqn. 1? Gain understanding by doing Von Neumann stability analysis for the components separately: i) the diffusion equation and ii) the advection equation. What do you learn from this?

    e. Write down a complete set of finite difference equations for the problem that you think will give stable accurate solutions.

    f. Consider other important issues such as the amount of storage needed (how many time levels), the computational speed, and the possible existence of any computational modes.

    g. How will you store the different variables and which subroutines will you need to write to solve these equations?

    h. Sketch a flow chart of the algorithm you will use to solve the equations.

Week 2: Coding and testing of subroutines

1. Start coding up the subroutines you have identified in the design of your program. Take care with what you need to pass them in the header – what needs to go in and what needs to come out of the subroutine?

2. Check through each subroutine to make sure array indices are in the correct range, that variables are all defined (using IMPLICIT NONE forces you to do this!), that there are no stupid typographical mistakes, etc.

3. Test each subroutine in all the ways you can think of. For example, wind velocity vectors should be parallel to streamfunction contour lines (the barotropic flow is non-divergent). Check that the numbers coming out for geopotential heights, streamfunction, wind velocities, and relative vorticity are reasonable orders of magnitude (hint: use dimensional analysis to develop an idea of what values you should expect for each of these quantities based on synoptic storm systems with typical velocities of 10m/s and horizontal length scales of 1000km).

4. Once you are fully happy that all your subroutines are working then go ahead and put them together to solve the barotropic vorticity equation. Don't add all the terms at once – first of all try only the diffusion term with no advection and no beta terms and then try just advection without diffusion, and then finally if these work then (and only then) put everything together.

5. Debug your program if it crashes. If it crashes before doing any time stepping then you have made some silly mistake that needs to be found by going carefully by eye through each line of code (dry running). If it crashes after a number of time steps then you have an instability problem. Use the animation facility in MATLAB to see what happens with the fields before it crashes – this may give you some clues. Go back to stability analysis and think hard about why your scheme might be unstable. Pay particular attention to boundaries.

6. If your scheme seems stable for runs up to 48 hours then proof test it by running it for 5-10 days to see if it still remains stable. If it doesn't then think carefully about what aspects of your numerical scheme might have led to the instability.

7. Develop some diagnostic procedures that can help you to see what your program is doing. For example, a simple subroutine that gives you the min, max, and mean of any field and MATLAB scripts that can produce informative maps and even animations of your variables.

Week 3: Parameter tuning and validation of forecasts

1. Run your model for 24 hours and 48 hours and look at the resulting forecasts. Interpret the development in synoptic terms – what has happened to the ridges and troughs and why? You might find it helpful to store fields at several intermediate steps and then look at these using the animation facility in MATLAB (see the course web site for details of how to do this).

2. Your model has some tuneable parameters. What are they and how do you think you should tune them to be the best values? Do you think there is a best value?

3. Do you expect your model to go unstable for any values of the parameters based on what you know from CFL and Von Neumann stability analysis of your scheme? If yes, then does the model go unstable for values of parameter that you expect from your stability analysis? If not, why not?

4. How do your solutions depend on the horizontal diffusion scheme? Try different values of eddy diffusivity in the range $10^4 - 10^8 m^2 s^{-1}$ and look at how your solutions depend on the amount of diffusivity. Based on this and sensitivity analysis of any other parameters decide upon optimal parameters for your model.

Week 4: Numerical experiments

Now that you have got a reasonable forecast model, you can use it to perform some interesting numerical experiments. Here are some suggested ideas that you can try (individually at first but then perhaps some together) and you might also like to think of some of your own experiments and also include these:

- **Potential vorticity** – As well as plotting maps of the prognostic variable (relative vorticity), the geopotential height, and the streamfunction and velocity field, you might also find it interesting to produce maps and animations of the quantity $\zeta = \xi + \beta y$. This is the *potential vorticity* of a constant thickness slab of fluid on a beta plane. Using the barotropic vorticity equation, calculate the total time derivative:

$$\frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} + v\frac{\partial \zeta}{\partial y} = ?$$

  What does this tell you about potential vorticity for a packet of air moving with the flow when the eddy viscosity is small? Now look carefully at your maps of PV and see if you can see this happening and compare it to what you could see from your maps of relative vorticity. Which of these two fields is more informative about the dynamics?

- **Geostrophic turbulence** - When you are quite convinced that your program is working correctly, try carrying out a much longer run (i.e. several days) of your model with a low value of eddy viscosity. Look at the evolution of the relative vorticity (or better still the potential vorticity) and describe what you see happening to the cyclone and anti-cyclone systems. Do you see any signs of chaotic behaviour (turbulent flow) or does the evolution look regular and predictable to you (laminar flow)? Investigate the sensitivity of this behaviour to different values of β (i.e. try different latitudes other than 45N) – does smaller beta make the flow more or less turbulent?

- **The butterfly effect** – weather forecasts can be very sensitive to their initial conditions. Test how sensitive your forecasts are by adding a small local perturbation onto your initial condition and then repeat the forecast. Make maps of the difference between this forecast and your original unperturbed forecast to see how the perturbation evolves in time. Describe what you see. Try putting the initial perturbation in places in the initial flow where you would expect it to have the largest impact on the forecast. Does it?

- **Effect of orography** – Vorticity can be generated by flow passing over orography. To represent such effects an extra term can be added to the right hand side of equation (1):

$$-\frac{f}{H}\left(u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial x}\right)$$

  where H is the mean constant depth of the uniform fluid layer, f is the Coriolis parameter (assumed constant on a beta plane) and $h(x, y)$ is the height of the orography. Try adding such a term to your equations choosing a simple isolated mountain such as a Gaussian hill $h(x, y) = h_0 e^{-d^2/2r^2}$ where $d^2 = (x - x_0)^2 + (y - y_0)^2$ is the squared distance from some point in the middle of the domain. Look carefully at the difference between your forecasts of vorticity with and without the hill and describe what effects the hill has had on cyclone development.

Week 5: Completion of work

1. Finish up your work and make good versions of the plots you would like to include in your assignment report.

2. Think carefully about how to write up your work to show how thoughtful you have been in developing your model and how well you have interpreted the results. Don't be afraid of reporting results that didn't turn out the way you expected them. Ed Lorenz had trouble believing at first that his simple equations gave chaotic solutions – he even suspected that his results might have been caused by a computer problem (finite numerical precision) but he thankfully still published his work!

3. Write up your assignment in a concise and precise scientific style. Figures should be carefully labelled with meaningful captions and your statements should be as factual and precise as possible. For guidance on writing up scientific reports look at lecture 3 in the scientific communication skills course on http://www.met.rdg.ac.uk/cag/courses.

Appendix A: Useful information

The spatial domain represents a latitudinal band (constant beta plane) wrapping around the extra-tropics centred on 45°N (see Figure 1 below). The 500hPa geopotential height initial conditions are defined on a regular grid with horizontal grid spacing $\Delta = 312.5km$ in both the latitude and longitude directions. The positions of the grid points can be written as $(x_i, y_j) = (i\Delta, j\Delta)$ where $i = 1,2,\ldots,N_x$ and $j = 1,2,\ldots,N_y$ with $N_x = 34$ and $N_y = 17$ .The 500hPa geopotential height initial conditions at these grid points can be denoted by $\Phi_{i,j}^{(0)}$ where the suffix (0) signifies the initial time t=0. These can be stored in a computer in a (34×17) two-dimensional array.

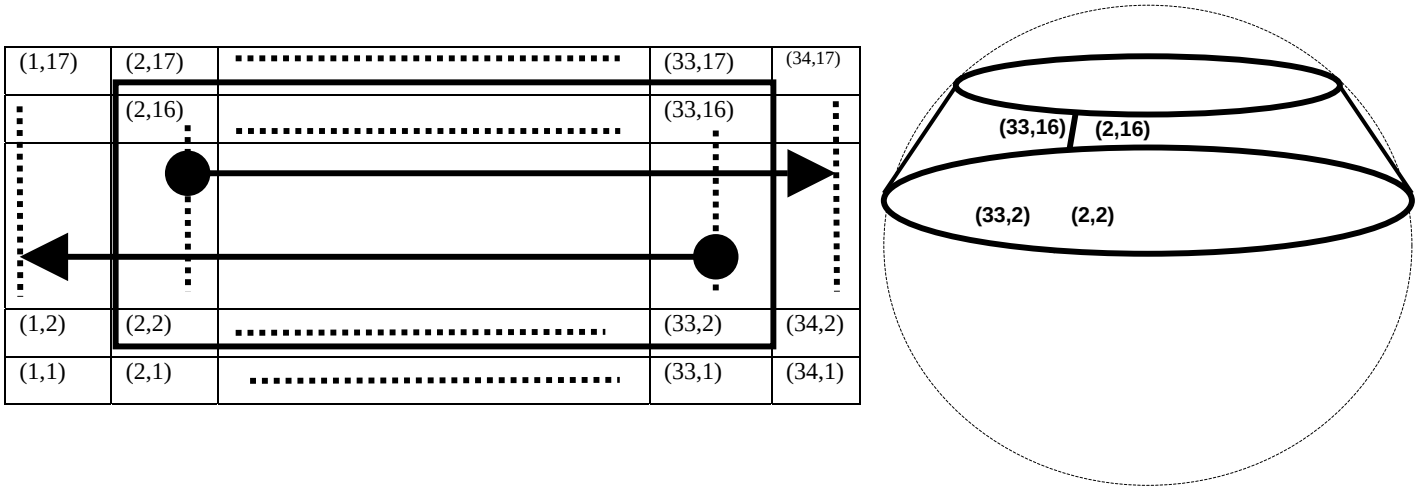

Figure 1. Schematic showing the spatial domain of integration with array indices denoted by (i,j). The domain represents a latitudinal band (constant beta plane) centred on 45N.

The domain represents a latitude band that wraps all the way around the extra-tropics and so east-west boundary conditions on the array are periodic. Such conditions are most efficiently programmed by adding buffer columns 1 and 34 that are updated before each new updating iteration by copying in the whole contents of column 2 and 33, respectively. Note that the prognostic variable, relative vorticity, only needs to be updated in the inner domain shown by the bold rectangle in Figure 1.

Free-slip boundary conditions can be assumed on the north and south channel walls: $v = 0$ (no meridional flow through boundaries) and $\partial u / \partial y = 0$ (no meridional shear of zonal wind). This leads to constant streamfunction and zero relative vorticity along the north (row 17) and south (row 1) boundaries (simple unmarked exercise: try to show this mathematically!). Take particular care on the southern and northern boundaries with finite differences and use carefully tested subroutines to perform these operations. Bugs love boundaries!

The zonal and meridional wind components are given in terms of the streamfunction by

$$(u, v) = \left(-\frac{\partial \psi}{\partial y}, \frac{\partial \psi}{\partial x} \right) \qquad (2)$$

Therefore wind velocities can be found by taking finite differences of the streamfunction.

The streamfunction can be found from the geopotential height by using the geostrophic relationship

$$\psi = \frac{g}{f}\Phi \qquad (5)$$

where g is the acceleration due to gravity and f is the Coriolis parameter (assume constant values typical of 45^{o}N). The relative vorticity is given by

$$\xi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \nabla^2 \psi \qquad (3)$$

and so can be found by taking the second order differences of the streamfunction.

During the time-stepping, the streamfunction can be found from the updated relative vorticity by solving the set of linear equations known as the Poisson equation:

$$\nabla^2 \psi = \xi \qquad (4)$$

To make life easier for you, we have provided you with a quick and accurate Poisson solver written by Jason Lander and Sheila Bryant, based on Fast Fourier Transform techniques, that can solve the Poisson equation for this domain. The Poisson solver is a subroutine called PSOLVE in the file **psolve.f90** downloadable from the course web site. To use the routine include the following lines in your program:

```
PARAMETER(NWK=4*(NX-2)*(NY-1))
REAL VOR(NX,NY),PSI(NX,NY),WORK(NWK)
        .
        .
CALL PSOLVE(VOR,PSI,WORK,NX,NY,DX)
        .
        .
END
INCLUDE 'psolve.f90'
```

So that the compiler knows that you want to compile PSOLVE, the source file must be linked with your program by adding the INCLUDE statement after the last END statement in your main program. On entry, array VOR contains the relative vorticity "source function" (right hand side of the Poisson equation). VOR is unaltered on exit. Array PSI contains the solution on exit. WORK is a temporary work array used within the subroutine. DX is the grid spacing of 312500 metres.

Since the Poisson equation is a second order differential equation, it requires two boundary conditions on ψ in order to find a unique solution for ψ in the interior of the domain. To find ψ in the interior domain, the subroutine PSOLVE uses the source field of ξ in the interior domain (the rectangle in Fig. 1) and the northern and southern rows of ψ. It is therefore very important that the PSI array used in the call to PSOLVE has the correct northern and southern rows obtained from the initial geopotential height conditions.

Appendix B: Brief guide to good programming style

Sloppy seat-of-the-pants programming will waste you large amounts of time and will be highly frustrating, whereas well-written code will have less bugs in it and will be easier to understand and modify in the future. It is therefore important that you develop a good programming style by following these tips:

1. **Design.** Do not rush into writing code. First design your program by carefully writing out the finite difference equations and a flowchart for the algorithm. Think carefully about all potential problems before rushing into implementation (writing the actual code). Computers are extremely logical and so you need to be methodical to program them!

2. **Structured code.** Structure the program into a number of smaller, self contained modular tasks. For example, identify tasks to set up initial data, take a time step, reset boundary conditions, solve the Poisson equation, write out results, etc. The secret of getting the program to work quickly without bugs is to code each of these smaller tasks in separate self-contained subroutines which can be written and tested independently. Subroutines should be the bricks you use to build your program. To test a subroutine, put in plenty of print statements to check that variables are containing the values you expect – these should be removed once you are satisfied the subroutine is working correctly.

3. **Debugging.** If you do encounter an error, put in extra print and stop statements to locate exactly where the problem first occurs e.g. which pass through a loop gives the problem, etc. Take particular care to match the parameters in a subroutine call with those expected by the subroutine header (for example, check that all array sizes for a particular array are consistent throughout the program). Avoid using common blocks to pass variables globally around the program – pass variables locally via subroutine headers. 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.

4. **Document clearly.** 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 some-one 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. Add concise "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. Use sensible and meaningful names for variables, subroutines, etc. Use longer more meaningful names rather than single letter names.

5. More guidelines on good programming style can be found on the internet, e.g.:
    - http://www.eg.bucknell.edu/~xmeng/Course/CS2330/Handout/StyleKP.html
    - http://sunsite.informatik.rwth-aachen.de/fortran/ch1-8.html (common mistakes)

Appendix C: Visualisation using MATLAB

MATLAB is a powerful tool for visualisation and analysis of data. It is particularly good at displaying maps of two dimensional fields and has good online documentation (use the *helpdesk* command in MATLAB to launch the online help or *help <command>* on the command line).

On the ITS system, please use MATLAB on the PC to look at data written to your N: drive by the Fortran code running on the UNIX system. To launch MATLAB on the PC, click on the *Start* button on the Desktop, then select *Departmental Sofware* followed by *Mathematics* followed by *MATLAB6.5*. Write out your data in the following form from your Fortran program so that MATLAB can then easily read it in:

```
.
.
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 ht which can be plotted. The %n is a formatting statement that tells the command to read numbers. Here are some useful MATLAB plotting commands:

- `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
- `subplot(n,m,i)` : Plots panel i on a figure with n rows and m cols
- `figure` : Launches a fresh graphics window
- `clf` : Clears the present figure
- `hold on` : Use these commands to overlay different plots
- `hold off` (put them before/after plot commands)

To print a plot, click on *file*, and then *print*, and then choose the appropriate printer name. To save a plot, click on *file*, and then *export* it to a file in your home directory. Postscript format is good for use in LaTeX documents whereas either JPEG or PNG is good for Word documents.

A MATLAB script file can be written to perform several operations at once – this will save you a lot of repetive typing! To do this use an editor to open a text file called say **plotex.m** and enter the MATLAB commands as you would do for interactive plotting. When in Matlab type **plotex** at the prompt and the script should then run automatically. An example script which plots the diagnostics output by modelbase.f90 is included on the course web site.