The Barotropic Vorticity Equation Solver

Chris Chapman

February 24, 2018

1 Introduction

The barotropic vorticity equation is basically the simplest model of the mid-latitude atmosphere or oceans that retains important features such as the generation of Rossby-waves (look them up) and non-linearity. It's two dimensional, being dependant only on the x (east-west direction) and y (north-south direction), so you can think of it as a model of the atmospheric/oceanic flow in a vertically averaged sense. The basic equations are the conservation of vorticity:

$$\frac{\partial \zeta}{\partial t} + u\frac{\partial \zeta}{\partial x} + v\frac{\partial \zeta}{\partial x} = \text{Forcing} - \text{Dissipation} \tag{1}$$

where ζ is the total vorticity, which includes the relative rotation of fluid *and* the rotation of the Earth, the later measured by the Corriolis parameter $f = f(y)$, so the vorticity is:

$$\zeta = \zeta_{\text{rel}} + f = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} + f. \tag{2}$$

u and v are the components of the velocity vector in the x and y directions. We further assume that the atmosphere and/or ocean are incompressible, so that:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{3}$$

which means that we can write the velocity in terms of a *streamfunction* ψ:

$$u = \frac{\partial \psi}{\partial y};\ v = \frac{\partial \psi}{\partial x}. \tag{4}$$

Plugging into the definition of vorticity gives us:

$$\zeta = \nabla^2 \psi + f. \tag{5}$$

In simple geometry (i.e. not on the spherical Earth) we approximate f as a linear function of y:

$$f = f_0 + \beta y \tag{6}$$

wheref_0 and β are constants. This approximation is called the β plane and models the fact that the Earth's angular velocity is higher at the equator than the poles (because the Earth is fatter there). As a short hand, we often write the non-linear terms in the streamfunction by replacing $u\frac{\partial \zeta}{\partial x} + v\frac{\partial \zeta}{\partial x}$ with $\frac{\partial \psi}{\partial x}\frac{\partial \zeta}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \zeta}{\partial x}$ (verify by direct substitution for u and v if you must), which is referred to as the *Jacobian* term, as it's the determinant of the Jacobian matrix for ψ and ζ.

Putting it all together gives us a set of two equations to solve the system. First we advance the hyperbolic equation in time step:

$$\frac{\partial \zeta_{\text{rel}}}{\partial t} = -\left[\frac{\partial \psi}{\partial x}\frac{\partial \zeta_{\text{rel}}}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \zeta_{\text{rel}}}{\partial x}\right] + \beta\frac{\partial \psi}{\partial x} + \text{Forcing} - \text{dissipation} \tag{7}$$

and then solve the Elliptic equation to get a new estimate of the streamfunction:

$$\nabla^2 \psi = \zeta_{\text{rel}}. \tag{8}$$

together with appropriate initial and boundary conditions.

2 Numerical solution

To solve equations 7 and 8 numerically, I've used a very simple finite difference scheme. We write variables at time step n and grid points $x_i = i\Delta x$ and $y_j = j\Delta y$ as ζ_{ij}^n.

Time stepping is handled by a finite difference leaf frog scheme:

$$\frac{\partial \zeta}{\partial t} = \frac{\zeta_{i,j}^{n+1} - \zeta_{i,j}^{n-1}}{2\Delta t} \tag{9}$$

so that

$$\zeta_{i,j}^{n+1} = \zeta_{i,j}^{n-1} + 2\Delta t \mathrm{RHS}_{i,j}^n \tag{10}$$

where the RHS is the right-hand side of equation 7, which consists of two terms (assuming no forcing or dissipation... we can add those in later), the β term and the non linear advection term (often called the Jacobian). The β term is linear and simple to estimate with centered finite differences:

$$\beta \frac{\partial \psi}{\partial x} = \frac{\psi_{i+1,j}^n - \psi_{i-1,j}^n}{2\Delta x}. \tag{11}$$

The Jacobian term is more difficult, because a naïve implementation using centered differences doesn't conserve energy or enstrophy (defined as ζ^2, something that should be conserved). To estimate it, we use something called the Arakawa Jacobian:

$$J_1 = \frac{\partial \psi}{\partial x}\frac{\partial \zeta}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \zeta}{\partial x} = \left(\frac{\psi_{i+1,j}^n - \psi_{i-1,j}^n}{2\Delta x}\frac{\zeta_{i,j+1}^n - \zeta_{i,j-1}^n}{2\Delta y}\right) - \left(\frac{\psi_{i,j+1}^n - \psi_{i,j-1}^n}{2\Delta y}\frac{\zeta_{i+1,j}^n - \zeta_{i-1,j}^n}{2\Delta x}\right) \tag{12}$$

$$J_2 = \frac{\partial}{\partial x}\left(\psi\frac{\partial \zeta}{\partial y}\right) - \frac{\partial}{\partial y}\left(\psi\frac{\partial \zeta}{\partial x}\right) = \text{a bunch of difference calculus} \tag{13}$$

$$J_3 = \frac{\partial}{\partial y}\left(\zeta\frac{\partial \psi}{\partial x}\right) - \frac{\partial}{\partial x}\left(\zeta\frac{\partial \psi}{\partial y}\right) = \text{more difference calculus} \tag{14}$$

$$J = \frac{1}{3}\left(J_1 + J_2 + J_3\right). \tag{15}$$

The details of the implementation are tedious but the stencil widths are all the same (between $i+1, i-1$ and $j+1$ and $j-1$. Go have a look in the code if you are unsure. Having time-stepped equation 10, we now have an updated estimate of ζ, $\zeta_{i,j}^{n+1}$. We now need to invert the elliptic Poisson equation:

$$\nabla^2 \psi = \zeta \tag{16}$$

to get an updated estimate of ψ, $\psi_{i,j}^{n+1}$. To do this, I use an interative successive over-relaxation method, discretising the laplace operator as:

$$\nabla^2 \psi_{i,j}^n = \frac{\psi_{i+1,j}^n + \psi_{i-1,j}^n - 2\psi_{i,j}^n}{\Delta x^2} + \frac{\psi_{i,j+1}^n + \psi_{i,j-1}^n - 2\psi_{i,j}^n}{\Delta y^2}. \tag{17}$$

which you can find in Numerical Recipes or about 1000 places on the Internet. Boundary conditions are $\psi = 0$ on $y = 0$ and $y = L_y$ and periodic in x.

3 Code Structure

The code is written in C++ for reasons that made sense to me at the time. The gist of the structure is:

- Initialise the streamfunction $\psi_{i,j}^0$;
- Get initial vorticity distribution from $\zeta_{i,j}^0 = \nabla^2 \psi_{i,j}^0$;
- Take time step to update vorticity;
- invert the Poisson equation to get the new value of streamfunction.

The model has two classes: doubleArray, which basically handles indexing of a single dimensional array, and modelGrid, which looks after grid parameters. The code is fairly modular and has a few subfunction calls that should be self explanatory.

4 To Do

A list of small and not-so-small improvements to make:

- Adams-Bashforth 3rd order time-stepping or 4th order Runge-Kutta;
- NetCDF I/O;
- Improve the model by going to 2 or N-layers using the *quasi-geostrophic* equations instead of the Barotropic Equations (changes the elliptic equation to solve from a simple Poisson equation to a more complicated Helmholtz equation, and there are now N equations to time-step instead of 1, but the rest stays the same);
- higher order advection scheme.