Numerical calculation of linear quasigeostrophic baroclinic instability

Shafer Smith
shafer@cims.nyu.edu

This document describes the numerical computation of the plane-wave solution to the linear quasigeostrophic equations, for arbitrary mean stratification and velocity profiles, and its implementation in the MATLAB codes `qggrz.m` and `pv_stretch_opz.m`. The code `pmodesz.m`, which computes the standard vertical modes and internal deformation wavenumbers, is also explained.

1 Linear quasigeostrophic equations

Assume a local, slowly-varying mean stratification and horizontal flow that depend only on z (see Pedlosky, 1984)

$$\begin{aligned} \boldsymbol{U} &= U(z)\hat{\boldsymbol{\imath}} + V(z)\hat{\boldsymbol{\jmath}} \\ N^2 &= N^2(z) = -\frac{g}{\rho_0}\frac{d\bar{\rho}}{dz} \\ Q_x &= Q_x(z) = \Gamma V \\ Q_y &= Q_y(z) = \beta - \Gamma U \end{aligned}$$

where the PV stretching operator is defined as

$$\Gamma\phi \equiv \frac{\partial}{\partial z}\left(\frac{f^2}{N^2}\frac{\partial \phi}{\partial z}\right).$$

Consistent with slowly-varying background assumption, assume horizontally periodic boundary conditions. The quasigeostrophic equations linearlized about this mean state are then

$$q_t + \boldsymbol{U}\cdot\nabla q + \boldsymbol{u}\cdot\nabla Q = 0, \quad q = \nabla^2\psi + \Gamma\psi$$

where $\boldsymbol{u} = -\psi_y\hat{\boldsymbol{\imath}} + \psi_x\hat{\boldsymbol{\jmath}}$ is the eddy velocity field expressed in terms of the horizontal streamfunction, $\psi = \psi(x, y, z, t)$.

Numerical solution of the linear (or nonlinear) problem proceeds by first discretizing in the vertical. We use a vertical finite-difference grid defined as in Figure 1. On this grid, the discrete stretching operator Γ_{ij} is

$$\Gamma_{ij}\psi_j = \frac{f^2}{g\delta_i}\left(\frac{\psi_{i-1}-\psi_i}{r_{i-1}} - \frac{\psi_i - \psi_{i+1}}{r_i}\right), \quad \text{where} \quad r_i = \frac{\bar{\rho}_{i+1}-\bar{\rho}_i}{\rho_0}, \tag{1}$$

with $\bar{\rho}_i$ the background potential density at level i and ρ_0 the average density (not density at $i = 0$).

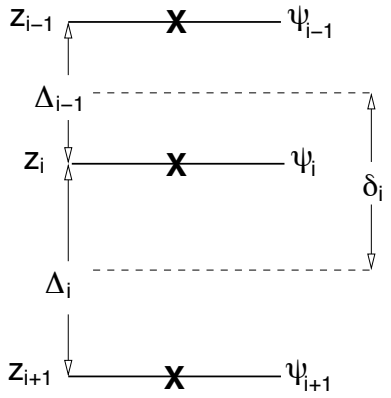

Figure 1: The grid used to represent vertical structure. Δ_i is the spacing between ψ_i and ψ_{i+1}, while δ_i is the distance between half spaces: $\delta_i = (\Delta_{i-1} + \Delta_i)/2$.

The vertical boundary conditions in quasigeostrophic flow — $w = 0$ at the upper and lower surfaces — are related to the time-dependent advection of eddy buoyancy at the surfaces. The linearized equations are

$$b_t^T + \boldsymbol{u}^T \cdot \nabla B^T + \boldsymbol{U}^T \cdot \nabla b^T = 0$$
$$b_t^B + \boldsymbol{u}^B \cdot \nabla B^B + \boldsymbol{U}^B \cdot \nabla b^B = 0$$

where

$$b = f\psi_z$$

and the superscripts T and B indicate evaluation at the upper and lower boundaries, $z = z^T$ and $z = z^B$, respectively.

On the grid defined in the figure, the buoyancy at any half level (indicated by dashed lines) is

$$b_i = f_0 \frac{\psi_{i-1} - \psi_i}{\Delta_{i-1}}.$$

We will incorporate the buoyancy boundary conditions by (1) writing the top-most PV and buoyancy values in terms of a streamfunction value ψ_0 at a ghost point z_0, then (2)

eliminating ψ_0 by subtracting the tendency equations for top-most PV and buoyancy values. Setting $\Delta_0 = \Delta_1 = \delta_1$ and $r_0 = r_1$, the top-most PV and buoyancy values are then

$$q_1 = \nabla^2\psi_1 + \frac{f^2}{g\delta_1 r_1}\left(\psi_0 - 2\psi_1 + \psi_2\right) \quad \text{and} \quad b_{1/2} = \frac{f}{\delta_1}(\psi_0 - \psi_1)$$

Defining a new variable

$$\tilde{q}_1 = q_1 - \frac{f}{gr_1}b_{1/2} = \nabla^2\psi_1 + \frac{f^2}{g\delta_1 r_1}(\psi_2 - \psi_1)$$

eliminates the streamfunction at the ghost point, ψ_0. To form the tendency equation for $\tilde{q}_1$, we form the sum

$$[\text{Tendency equation for } q_1] - \left(\frac{f}{gr_1}\right)[\text{Tendency equation for } b_1]$$

which gives at level 1,

$$\partial_t\tilde{q}_1 + \frac{f^2}{g\delta_1 r_1}\big[u_1(V_2 - V_1) - v_1(U_2 - U_1)\big] + \boldsymbol{U}_1 \cdot \nabla\tilde{q}_1 = 0,$$

where we have used thermal wind balance for the upper level mean buoyancy gradient

$$\nabla B_1 = \frac{f}{\delta_1}\big[(V_1 - V_2)\hat{\boldsymbol{\imath}} + (U_2 - U_1)\hat{\boldsymbol{\jmath}}\big].$$

We proceed similarly at the bottom boundary, but omit the details.

The vertical boundary conditions are thus incorporated into the problem entirely through the construction of the stretching matrix, Γ. Specifically, we augment the definition (1) at the top ($i = 1$) and bottom ($i = N$) as follows,

$$\Gamma_{1j}\psi_j = \frac{f^2}{g\delta_1 r_1}(\psi_2 - \psi_1), \tag{2}$$

$$\Gamma_{Nj}\psi_j = \frac{f^2}{g\delta_N r_{N-1}}(\psi_{N-1} - \psi_N). \tag{3}$$

In MATLAB, the stretching operator matrix Γ_{ij} is constructed in the function `pv_stretch_opz`, which requires as its inputs one-dimensional arrays `z` containing the coordinates of the grid, and `rho` containing the background potential density values at these coordinates. The input parameter `F = f^2*rho0/g` (or its nondimensional equivalent — see the header to the function for more details). The output is the matrix `G` that corresponds to Γ_{ij}.

2 The plane-wave solution and eigenvalue problem

Substitution of a plane-wave solution of the form

$$\psi = \hat{\psi}(z)\ e^{i(k+\ell y-\omega t)}$$

where $\hat{\psi}$ is the complex amplitude, into the equations of motion gives

$$\omega\ (\Gamma - K^2)\ \hat{\psi} = \left[kQ_y - \ell Q_x + (kU + \ell V)(\Gamma - K^2)\right]\ \hat{\psi}$$

where $K = k^2 + \ell^2$. Letting $\hat{\psi}_i$ be the discrete amplitude vector, the generalized eignevalue problem may then be written

$$\omega\ B_{ij}\ \hat{\psi}_j = A_{ij}\ \hat{\psi}_j \tag{4}$$

where

$$\begin{aligned} B_{ij} &= \Gamma_{ij} - K^2\delta_{ij}, \\ A_{ij} &= (kQ_{y,m} - \ell Q_{x,m})\ \delta_{ijm} + (kU_m + \ell V_m)\ \delta_{inm}\ B_{nj}, \end{aligned}$$

and the δs are Kronecker tensors, equal to unity when all indeces are equal, and zero otherwise. The tensor products with δ in the second line generate diagnoal matrices with the vectors $kQ_{y,i} - \ell Q_{x,i}$ and $kU_i + \ell V_i$ on the diagonals, respectively.

In the discrete problem with N levels, there will be N eigenvectors $\hat{\psi}$ and eigenvalues ω. In order to avoid unneccesary complexification, I neglected the index for these above.

The generalized eigenvalue problem (4) is computed in `qggrz`, the header of which contains all the details for the inputs. Note that `pv_stretch_opz` is called directly by `qggrz`. A snippet from `qggrz` is shown below, where `U` and `V` are the background state zonal and meridional velocity vectors, `Q_x` and `Q_y` are the background PV gradients, (`k`,`l`) is the wavenumber and `G` is the stretching operator matrix:

```
Q_y    = beta - G*U(:);
Q_x    = G*V(:);
K2 = k^2+l^2;
KdotU = k*U(:) + l*V(:);
KxdelQ = k*Q_y(:) - l*Q_x(:);
B = G - K2*eye(nz);
A = diag(KxdelQ) + diag(KdotU) * B;
[evec,D] = eig(A,B);
w = diag(D);
```

Since there are `nz` eigenvectors $\hat{\psi}$ and eigenvalues ω, `evec` has size `nz` $\times$ `nz`, and the MATLAB function `eig` returns the eigenvaules on the diagonal of `D`. One must next choose the eigenvalue with the largest imaginary part (and its corresponding eignevector) to find the effective growth rate at this wavenumber. In the MATLAB code I do this as follows:

```
if max(abs(imag(w))) > eps
  [wi_max(kc,lc),ind] = max(imag(w));
  wr_max(kc,lc) = real(w(ind));
else
  [wr_max(kc,lc),ind] = max(real(w));
  wi_max(kc,lc) = imag(w(ind));
end
psiv(kc,lc,:) = evec(:,ind);
```

where `kc` and `lc` are the loop indeces for the current wavenumber. Thus if a given wavenumber is unstable, I choose the real eigenvalue and eigenvector corresponding with the largest growth rate; if the wave at that wavenumber is stable, I instead choose the mode corresponding to the largest real frequency (change this however you see fit).

3 Vertical modes and internal deformation radii

The matrix eigenvalue problem

$$\Gamma\Phi = -\Phi\Lambda^2$$

defines the vertical modes $\boldsymbol{\phi}_j$ (the columns of Φ), and the internal deformation wavenumbers, λ_j (the diagonal elements of the diagonal matrix Λ). Clearly, given Γ (or `G` in MATLAB), all one needs to do is call the MATLAB function `eig`. The function `pmodesz` does this, and returns as output a matrix `pm`, equivalent to Φ, and vector `kz`, containing λ_j, where the columns of `pm` are normalized such that `sum(dz.*pm(:,i))/sum(dz)` is 0 for any valid `i`, and `sum(dz.*pm(:,i).*pm(:,j))/sum(dz)` is 1 if $i = j$ and 0 (to machine accuracy) otherwise.