

Computer Problems 4 / 5

2-D and 3-D nonlinear quasi-compressible flow

Description: 2-D/3-D evolution of density currents

Due: Pgm4/5 is due during Finals week. *Ignore/omit all v terms and y -derivatives for 2-D.*

A. Equations

There are 5 unknowns: horizontal flow components (u and v , m s^{-1}), vertical flow (w , m s^{-1}), potential temperature (θ , deg. K), and perturbation pressure (p' , Pa). The *base-state* time-invariant density ($\bar{\rho}$, g kg^{-1}) and temperature ($\bar{\theta}$) are functions of height *only*. The quasi-compressible set has “pseudo” sound waves traveling at speed c_s ; the pressure approaches an anelastic solution (Droegemeier and Wilhelmson 1987, *J. Atmos. Sci.*, p. 1187). The continuous form w/ advection, diffusion, pressure, and buoyancy:

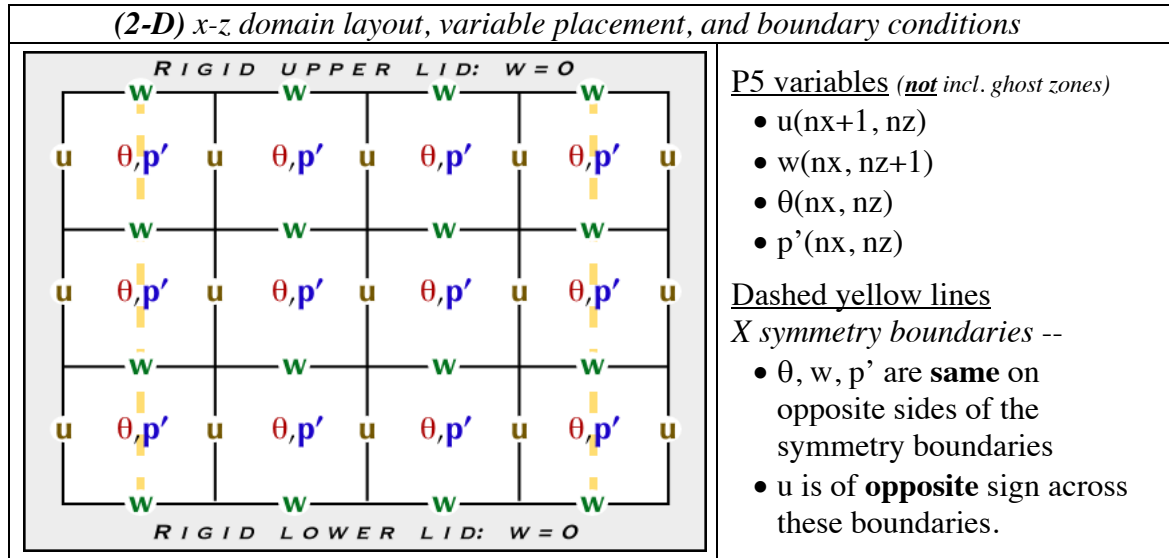
u -momentum:	$u_t = -uu_x - vu_y - wu_z - \frac{1}{\bar{\rho}} p'_x + K(u_{xx} + u_{yy} + u_{zz})$
v -momentum: (3-D, only)	$v_t = -uv_x - vv_y - wv_z - \frac{1}{\bar{\rho}} p'_y + K(v_{xx} + v_{yy} + v_{zz})$
w -momentum: ($\theta' = \theta - \bar{\theta}$)	$w_t = -uw_x - vw_y - ww_z - \frac{1}{\bar{\rho}} p'_z + g \frac{\theta'}{\bar{\theta}} + K(w_{xx} + w_{yy} + w_{zz})$
Perturbation pressure:	$p'_t = -c_s^2 \left(\bar{\rho} \frac{\partial u}{\partial x} + \bar{\rho} \frac{\partial v}{\partial y} + \frac{\partial}{\partial z} (\bar{\rho} w) \right)$
θ (pot. temperature):	$\theta_t = -(u\theta)_x - (v\theta)_y - (w\theta)_z + \theta(u_x + v_y + w_z) + K(\theta_{xx} + \theta_{yy} + \theta'_{zz})$

The discrete equations use *forward* time differencing for θ , and *centered* for u , v , w , p :

u :	$\delta_{2t} u = -(\bar{u}^x \delta_x u)_{(n)}^x - (\bar{v}^x \delta_y u)_{(n)}^y - (\bar{w}^x \delta_z u)_{(n)}^z - \frac{1}{\bar{\rho}} \delta_x p'_{(n-1)} + K_m (\delta_{xx} u + \delta_{yy} u + \delta_{zz} u)_{(n-1)}$
v : (P5)	$\delta_{2t} v = -(\bar{u}^y \delta_x v)_{(n)}^x - (\bar{v}^y \delta_y v)_{(n)}^y - (\bar{w}^y \delta_z v)_{(n)}^z - \frac{1}{\bar{\rho}} \delta_y p'_{(n-1)} + K_m (\delta_{xx} v + \delta_{yy} v + \delta_{zz} v)_{(n-1)}$
w :	$\delta_{2t} w = -(\bar{u}^z \delta_x w)_{(n)}^x - (\bar{v}^z \delta_y w)_{(n)}^y - (\bar{w}^z \delta_z w)_{(n)}^z - \frac{1}{(\bar{\rho})^z} \delta_z p'_{(n-1)} + g \left(\frac{\theta'}{\bar{\theta}} \right)_{(n)}^z + K_m (\delta_{xx} w + \delta_{yy} w + \delta_{zz} w)_{(n-1)}$ (Note $\theta' \equiv \theta - \bar{\theta}$)
p' :	$\delta_{2t} p' = -c_s^2 \left[\bar{\rho} \delta_x u_{(n+1)} + \bar{\rho} \delta_y v_{(n+1)} + \delta_z \left\{ (\bar{\rho})^z w_{(n+1)} \right\} \right] \quad (c_s \text{ is the chosen sound speed})$
θ :	2D: PL advection, plus 2-D diffusion. 3D: Strang splitting + 3-D diffusion: $\left[F_x \left(\frac{\Delta t}{2} \right) \right] \left[F_y \left(\frac{\Delta t}{2} \right) \right] \left[F_z (\Delta t) \right] \left[F_y \left(\frac{\Delta t}{2} \right) \right] \left[F_x \left(\frac{\Delta t}{2} \right) \right] + K_\theta (\delta_{xx} \theta + \delta_{yy} \theta + \delta_{zz} \theta')_{(n)}$

u, v, w advection follow the (unsplit) “box method,” not to be confused with the implicit scheme of the same name. Pressure and diffusion terms are lagged (at time $n-1$). θ is advected with Lax-Wendroff or piecewise linear methods. *Note:* time levels, averaging!

B. Grid layout and boundary conditions



- Dimensions:
 - Use $\Delta x = \Delta z$; grid spacing, dimensions to be announced.
 - We will do *test cases* at coarse resolution, e.g. 200m or larger.
 - physical dimensions are **no longer** from $(-.5, -.5)$ to $(+.5, +.5)$
 - x coordinates (for θ, p') = $\Delta x/2 + \Delta x(i-1)$, $i=1 \dots nx$ (*Fortran*)
 - bottom-left corner θ, p' are at $(x=\Delta x/2, z=\Delta z/2)$
 - w (at $k=1$ in Fortran, $k=K1$ in C) is at $z=0$
 - u (at $i=1$ in Fortran, $i=I1$ in C) is at $x=0$
- Top, bottom boundaries:
 - free slip (no drag on u); rigid lids ($w=0$ at $k=1$ and $k=nz+1$ in *Fortran*)
 - 0-gradient for all variables; any variable $\xi(k=0) = \xi(k=1, \text{Fortran})$, etc.
- Lateral (x) boundaries: symmetry boundaries shown with dashed yellow lines
 - $u(1) = -u(2)$ $u(nx+1) = -u(nx)$ (*Fortran indices here*)
 - $\theta(0) = \theta(2)$ $\theta(nx+1) = \theta(nx-1)$ (same for w, p')
- Lateral (y) boundaries: (*3D, only*)
 - Y boundaries are *periodic*. Consider the periodic boundary to sit at the V wind locations for $j=1$ and $j=ny+1$.
 - You will only integrate V from $1:ny$ (*Fortran indices*); the value of V at $(ny+1)$ will always be set equal to V at $j=1$.
 - Other variables are periodic in Y as $\xi(ny+1) = \xi(1)$, etc.

C. Initial conditions (base state)

- First you must define the *base state vertical profiles* for density $\bar{\rho}$ and base-state potential temperature $\bar{\theta}$. We only need $\bar{\rho}$ for later use; other variables (z, P, T) are used only to calculate $\bar{\rho}$ (there is no need to save $T(z)$ and $P(z)$).
- The first vertical velocity level, w at *Fortran* $k=1$, is at $z=0$ consistent with our *C-grid* staggering. Variables u, p, q start at $z=\Delta z/2$.
- In the expressions below, z refers to the height at a θ and p' level. The notation given is for Fortran.

$$\left. \begin{aligned} z(k) &= \frac{\Delta z}{2} + \Delta z(k-1) \\ \bar{T}(z) &= 300.0 - \frac{g}{c_p} z \\ \bar{P}(z) &= P_0 \left(\frac{\bar{T}}{\bar{\theta}} \right)^{c_p/R_d} \\ \bar{\rho}(z) &= \frac{\bar{P}}{R_d \bar{T}} \end{aligned} \right\} \text{ where } \left\{ \begin{aligned} z &= \text{height (m) of } \theta, u, p' \text{ levels} \\ \bar{\theta}(z) &= 300 \text{ K} = (\text{constant}) \text{ potential temperature} \\ g &= 9.81 \text{ ms}^{-2} = \text{gravity} \\ c_p &= 1004 \text{ J kg}^{-1} \text{ K}^{-1} = \text{specific heat at constant pressure} \\ R_d &= 287 \text{ J kg}^{-1} \text{ K}^{-1} = \text{dry air gas constant} \\ P_0 &= 10^5 \text{ Pa} = \text{standard pressure at sea level} \\ \bar{\rho} &= \text{density (kg m}^{-3}) \text{ at } \theta, u, p' \text{ levels} \end{aligned} \right.$$

Check your initial state with this data for $\Delta z=100\text{m}$, at (*Fortran*) $k=11$, $z=1050\text{m}$:

- $P=88540 \text{ Pa}$
- $T=289.74 \text{ K}$
- $\rho_{u \text{ level}} = 1.065 \text{ g kg}^{-1}$, $\rho_{w \text{ level}} = 1.069 \text{ g kg}^{-1}$.
- Note you compute $\rho_{u \text{ level}}$ as above, and average in height to get $\rho_{w \text{ level}}$; this is why $\rho_{w \text{ level}}$ is written as $(\bar{\rho})^z$ on page 1.
- $\rho_{w \text{ level}}$ at $k=1$ can have any value; it is only used where multiplied by w , and $w_{\text{ground}} = 0$.

D. Initial conditions (perturbation potential temperature and u, w)

2D: The solution evolves from an initial state with *zero* mean flow $U(z)$ and *constant* potential temperature (θ). We begin with temperature perturbations: where θ' is warm (cool) the air will rise (sink). The initial u, w , and p' are zero. For θ , use:

$$\theta_{i,k} = \bar{\theta} + \sum_{m=1}^2 \left[\Delta \theta'_m \frac{\cos(r_m \pi) + 1}{2} \text{ if } r_m \leq 1, \text{ else } 0 \right], \quad r_m = \sqrt{\left(\frac{x_i - x_0(m)}{x \text{ radius}} \right)^2 + \left(\frac{z_k - z_0(m)}{z \text{ radius}} \right)^2}$$

so $\theta(i,k)$ at time $t=0$ equals the base state (constant) $\bar{\theta}$ plus any perturbation $\Delta \theta'(m)$, for up to **two** initial temperature perturbations $m=1,2$.

Structure your IC code for setting up θ like that given below. The example code below is for 3-D; *simplify appropriately* for 2D:

distance / radius calculations for initial condition of programs 5/6

Fortran	C requires <math.h>
<pre> do k = 1,nz do j = 1,ny do i = 1,nx x = dx/2+dx*real(i-1) y = dy/2+dy*real(j-1) z = dz/2+dz*real(k-1) do m = 1,2 xd = (x-x0(m)) yd = (y-y0(m)) zd = (z-z0(m)) rad = sqrt((xd/xrad(m))**2 & + (yd/yrad(m))**2 & + (zd/zrad(m))**2) if (rad.lt.1.0) then ! ...your θ code here... endif enddo enddo enddo enddo (+3 more enddo's) </pre>	<pre> for (i=I1; i<=I2; i++) { for (j=J1; j<=J2; j++) { for (k=K1; k<=K2; k++) { x = dx/2.0 + dx*(float)(i-I1); y = dy/2.0 + dy*(float)(j-J1); z = dz/2.0 + dz*(float)(k-K1); for (m=0; m<2; m++) { rm = sqrt(pow((x-x0[m])/xradius[m],2.0) +pow((y-y0[m])/yradius[m],2.0) +pow((z-z0[m])/zradius[m],2.0)); if (rm <= 1.0) { /* your θ code here */ } /* rm */ } /* m */ } /* k */ } /* j */ } /* i */ </pre>

These *two* thermal perturbations $\Delta\theta'$ have different center (x,z) coordinates. The x- and z-“radius” *may vary* between perturbations, so you must store two sets of “radii”.

3D, only: Perturbations have 3-D center positions (x,y,z). You will also create perturbations to the **v** flow component, *using the same code* as for θ :

$\theta_{i,j,k} = \bar{\theta} + \sum_{m=1}^2 \left[\Delta\theta'_m \frac{\cos(r_m\pi) + 1}{2} \text{ if } r_m \leq 1, \text{ else } 0 \right]$ $v_{i,j,k} = \sum_{m=1}^2 \left[\Delta v'_m \frac{\cos(r_m\pi) + 1}{2} \text{ if } r_m \leq 1, \text{ else } 0 \right]$	$r_m = \sqrt{\left(\frac{x_i - x_0(m)}{xradius_m}\right)^2 + \left(\frac{y_j - y_0(m)}{yradius_m}\right)^2 + \left(\frac{z_k - z_0(m)}{zradius}\right)^2}$
---	--

Calculate r_m for each point (i,j,k), and use it in the computation of *both* perturbation θ and v-wind (*ignore* staggered grid positions in doing so; use the *same* r_m value, code).

For 3-D, we also utilize random initial **u** values, up to $\pm(\text{upertur}/2)$. Use the default Intel Fortran/C random number generator. Here is sample code:

Fortran	C requires <math.h>
<pre> real upertur,rand call srand(0.0) do k = 1,nz do j = 1,ny do i = 1,nx+1 u1(i,j,k) = & (rand(0)-0.5)*upertur enddo enddo enddo </pre>	<pre> float upertur; srand(0.0); /* seed */ for (i=I1+1; i<=I2; i++) { for (j=J1; j<=J2; j++) { for (k=K1; k<=K2; k++) { u1[i][j][k] = upertur * ((float)rand() / (RAND_MAX + 1.0)) - upertur*0.5; } } } </pre>

E. Code layout

The code layout guidelines include those from past programs *plus the following*:

- Do not put your integration (advection, diffusion, pressure gradient, buoyancy, initialization...) steps in your main program; put **each** in a separate subroutine. You must also use (to build your program) and submit (for grading) a *makefile*.
- **Read in** from the keyboard or a file, or use via a Fortran namelist:
 - times to plot (or, a plotting interval) • temperature perturbations and their center locations (x,z or x,y,z) • diffusion coefficients K_m and K_{θ} .
- Use ghost zones as before, as needed for the numerical schemes being applied.
- Set up the initial conditions (1-D for density, and 2-D or 3-D fields) entirely in one subroutine. Plot the initial potential temperature perturbation $(\theta - \bar{\theta})$.
- You must put common processes *for different variables* in the same subroutines: advection (u,w, θ) (with 1-D advection still handled by a separate 1D routine); diffusion (u,w, θ), and pressure gradient force/buoyancy (u,w,p').
- Your main program must **ONLY** read input data, print out information as desired and call subroutines. All other code must be in subroutines for full credit.
- Remember $w=0$ at the top and bottom levels ($k=1$ and $k=nz+1$ in Fortran). So you do z mixing for w only from $k=2:nz$ (in Fortran).
- Don't evolve u outside of the symmetry boundaries; compute $u(2 \dots nx)$ and then determine $u(1)$ and $u(nx+1)$ using the (a)symmetric boundary conditions.
- For pressure, first compute new values for u and w at time level $(n+1)$. Then update the pressure from $(n-1)$ to $(n+1)$ using $u^{(n+1)}$ and $w^{(n+1)}$ to get $p^{(n+1)}$.
- The order of computation is: advection (u,w, θ); diffusion, and pressure terms.
- Use a forward time step to start the integration (there is a short cut we'll discuss).
- 3D: For full credit, you must make a reasonable attempt at parallelizing your code, and part of your grade also requires visualization.

F. Plotting

2D: plot contours as usual. We will not use surface plotting.

3D: You *will not* be calling plot routines directly from your 3D program. Doing so is slow and wasteful considering how long your programs will (at full resolution) take to rerun. Instead, you will call a C or Fortran routine *putfield* (provided to you) to write your output to disk. Required plots will be listed on the class web site.

G. Hints

- Do initial testing at *reduced resolution*, e.g. $\Delta x = \Delta z = 200\text{m}$, $\Delta t = 0.5\text{s}$.
- In testing (for Fortran), do early tests compiling with subscript checking:
-g -check all -traceback.
- Beware! **$NX \neq NZ$ here**. Think where you have assumed $NX = NZ$ in programs 1-3.
- For min/max stats and plotting, average u and w to θ/p locations; and plot $(\theta - \bar{\theta})$
- We are using forward time differencing for θ advection, and centered time for everything else. So, $\delta_{2t}u$ means $u_3 = u_1 + 2\Delta t^*(...)$; θ advection is forward in time, so only two arrays are needed [handled as in programs 1-3].

H. Checking your code

There are various checks you could carry out to test parts of your code. Some tests you could perform include:

1. Linear advection: observe movement of θ' field with constant u and/or w fields while disabling diffusion, buoyancy and pressure gradient terms.
2. 1-D: reduce two-dimensional initial condition to 1-D (e.g. let θ , u , or w vary as $\sin(x)$) for advection tests.
3. Diffusion only: disable advection, buoyancy and pressure gradient terms, and damp only θ or some pre-determined function of u or w .
4. Pressure gradient and buoyancy terms, only: disable advection and diffusion, and integrate using the pressure gradient terms (influences u , w), buoyancy term (influences w), and the pressure field update itself (from gradients in u , w). In this test, the θ field stays constant with time, and a circulation develops in the u and w fields. This is a particularly useful test. The sequence of evolution to look for is:
 - a. The temperature perturbation θ' leads to vertical acceleration, changing w
 - b. The new, nonzero w field creates pressure gradients (from $\partial w / \partial z$)
 - c. The pressure gradients lead to horizontal acceleration, changing u
5. Look for symmetry in your solutions. For example, an initial temperature perturbation placed at the very center of the domain will lead to minima and maxima of opposite sign in u ; this should remain true as your solution evolves.
 - a. But: the symmetry is in x ; comparable symmetry will not occur in z due to the density variation with height.

I. Visualization (3D only)

Required plots will be discussed in class. Beyond this, part of the program grade (see below) involves creating a few 3-D plots with the visualization tool **VisIt**. See the class web site for details.

Following is a broad description of how my program is coded.

1. <u>MAIN PROGRAM</u>	<u>NOTES</u>
<ul style="list-style-type: none"> a. read in parameters; call IC b. plot initial condition c. call MAXMIN d. call BC e. <i>set tstep</i> = Δt f. TIME LOOP: <i>n</i>=1,max_steps <ul style="list-style-type: none"> • set $u_3=u_1$, $w_3=w_1$, $t_3=t_2$ • call ADVECT • call DIFFUSION • call PGF • array update • if (<i>n</i>=1) set <i>tstep</i> = $2\Delta t$ • call BC • call MAXMIN • if desired time: PLOT g. END OF TIME LOOP h. plot time traces 	<p>Always plot θ', not total θ Print min, max of all fields [optional] Set ghost points for first time step Because your first step is a <u>forward</u> one</p> <p>Array copy helps start this time step. Advection of θ, u, and w. Mix: u, w, and θ (<i>note: in general $K_m \neq K_{theta}$</i>) Obtain u_3, w_3; get new p_3 This is the usual array switch between old, new time levels. There are <u>three</u> time levels for u, w, p, and θ. But: if first step, <u>don't</u> update u_1, w_1, or p_1.</p> <p>Switch from forward to centered time for u, w, p.</p> <p>Get BCs ready for next time step. ... also store max/min info for later use. Call contour routine for u, w, θ', and p</p> <p>.. using min/max $u/w/\theta$ I have already stored</p>
<p>2. <u>IC ROUTINE</u></p> <ul style="list-style-type: none"> a. compute 1D arrays b. set $p', u, w = 0$ (in the 3D case, we set v using perturbations, and set u to random numbers; $u_{i=0}$ is nonzero in 3D) c. set θ' based on handout. 	<p>Compute constants and 1D arrays here. includes density(z) at θ and w levels Do this for (n) and ($n-1$) variables; this is part of preparing for the first, forward time step (hence <i>tstep</i> is first set to Δt, and later to $2\Delta t$) Remember you <i>read in</i> the temperature perturbations and their locations</p>
<p>3. <u>BC ROUTINE</u></p> <ul style="list-style-type: none"> a. 0-gradient top, bottom b. w, θ, and p' are same on either side of symmetry boundary c. Anti-symmetry for u 	<p>So $u(i, nz+1) = u(i, nz)$ So $p(0, k) = p(2, k)$, $p(nx+1, k) = p(nx-1, k)$ So $u(1, k) = -u(2, k)$, $u(0, k) = -u(3, k)$</p>
<p>4. <u>ADVECT ROUTINE</u></p> <ul style="list-style-type: none"> a. u: $u_3 = u_3 + tstep*(\text{box terms})$ b. w: $w_3 = w_3 + tstep*(\text{box terms})$ c. θ advection as usual (but have 3 arrays) 	<p>Recall u, w, p have centered time derivatives. For 3D, do v advection here, too.</p> <p>Piecewise linear method (initially, use Lax-W)</p>
<p>5. <u>DIFFUSION ROUTINE</u></p> <ul style="list-style-type: none"> a. $u_3 = u_3 + tstep*(x, z \text{ mixing terms})$ b. $w_3 = w_3 + tstep*(x, z \text{ mixing terms})$ c. Mix θ ... 	<p>For 3D, do v diffusion, too. W is always zero at $k=1$ and at $k=nz+1$</p>
<p>6. <u>PGF SUBROUTINE</u></p> <ul style="list-style-type: none"> a. $u_3 = u_3 - tstep*(\text{pgf terms})$ b. $w_3 = w_3 - tstep*(\text{pgf terms})$ + $tstep*(\text{buoyancy terms})$ c. set u, w BCs (could call BC, or set here) d. $p_3 = p_1 - (\text{pgf terms})$ 	<p>Pressure gradient / buoyancy routine. Adding to the u_3 array. Adding to the w_3 array. Remember $w=0$ at $k=1$ and $k=nz+1$</p> <p>Get ready for derivatives in p equation pgf terms use new u, w at time ($n+1$)</p>

