Computer Problems 5 and 6
2-D and 3-D nonlinear quasi-compressible flow

Description: Program 5 ("P5") is 2-D. Program 6 ("P6") is fully 3-D.
Equations for program 6 (3-D) follow. For program 5, ignore/omit all ***v*** *terms and* ***y****-derivatives.*

A. Equations

Program 6 has 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 ($\overline{\rho}$, g kg^{-1}) and temperature ($\overline{\theta}$) 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 with advection, diffusion, pressure, & buoyancy:

u-momentum:	$u_t = \; -uu_x \; -vu_y \; -wu_z - \dfrac{1}{\overline{\rho}} p'_x + K\left(u_{xx} + u_{yy} + u_{zz}\right)$
v-momentum: (program 6 only)	$v_t = \; -uv_x \; -vv_y \; -wv_z - \dfrac{1}{\overline{\rho}} p'_y + K\left(v_{xx} + v_{yy} + v_{zz}\right)$
w-momentum: $\left(\theta' = \theta - \overline{\theta}\right)$	$w_t = -uw_x - vw_y - ww_z - \dfrac{1}{\overline{\rho}} p'_z + g\dfrac{\theta'}{\overline{\theta}} + K\left(w_{xx} + w_{yy} + w_{zz}\right)$
Perturbation pressure:	$p'_t = -c_s^2\left(\overline{\rho}\dfrac{\partial u}{\partial x} + \overline{\rho}\dfrac{\partial v}{\partial y} + \dfrac{\partial}{\partial z}\left(\overline{\rho} w\right)\right)$
θ *(pot. temperature):*	$\theta_t = -\left(u\theta\right)_x - \left(v\theta\right)_y - \left(w\theta\right)_z + \theta\left(u_x + v_y + w_z\right) + K\left(\theta_{xx} + \theta_{yy} + \theta'_{zz}\right)$

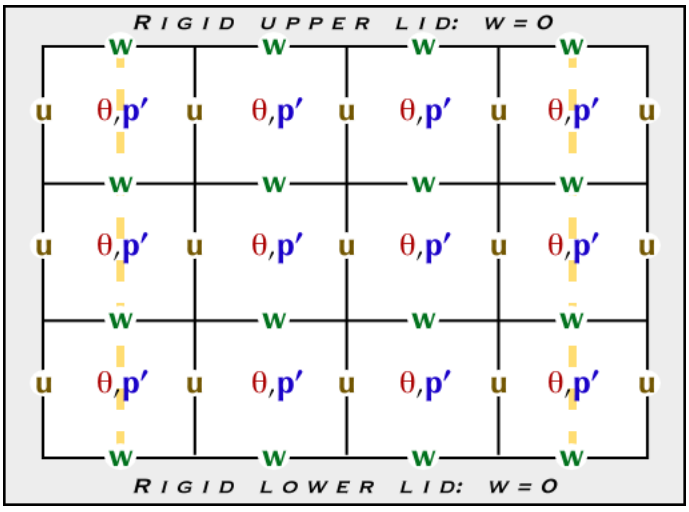
The discrete equations use *forward* time differencing for θ, and *centered* for u, v, w, p:

u:	$\delta_{2t}u = -\overline{\left(\overline{u}^x \delta_x u\right)}^x_{(n)} - \overline{\left(\overline{v}^x \delta_y u\right)}^y_{(n)} - \overline{\left(\overline{w}^x \delta_z u\right)}^z_{(n)} \quad - \dfrac{1}{\overline{\rho}}\delta_x p'_{(n-1)} + K_m\left(\delta_{xx}u + \delta_{yy}u + \delta_{zz}u\right)_{(n-1)}$
v: *(P6)*	$\delta_{2t}v = -\overline{\left(\overline{u}^y \delta_x v\right)}^x_{(n)} - \overline{\left(\overline{v}^y \delta_y v\right)}^y_{(n)} - \overline{\left(\overline{w}^y \delta_z v\right)}^z_{(n)} \quad - \dfrac{1}{\overline{\rho}}\delta_y p'_{(n-1)} + K_m\left(\delta_{xx}v + \delta_{yy}v + \delta_{zz}v\right)_{(n-1)}$
w:	$\delta_{2t}w = -\overline{\left(\overline{u}^z \delta_x w\right)}^x_{(n)} - \overline{\left(\overline{v}^z \delta_y w\right)}^y_{(n)} - \overline{\left(\overline{w}^z \delta_z w\right)}^z_{(n)} - \dfrac{1}{\overline{\left(\overline{\rho}\right)}^z}\delta_z p'_{(n-1)} + g\overline{\left(\dfrac{\theta'}{\overline{\theta}}\right)}^z_{(n)}$ (*Note* $\theta' \equiv \theta - \overline{\theta}$) $\quad + K_m\left(\delta_{xx}w + \delta_{yy}w + \delta_{zz}w\right)_{(n-1)}$
p'*:*	$\delta_{2t}p' = -c_s^2\left[\overline{\rho}\delta_x u_{(n+1)} + \overline{\rho}\delta_y v_{(n+1)} + \delta_z\left\{\overline{\left(\overline{\rho}\right)}^z w_{(n+1)}\right\}\right]$ (c_s is the sound speed)
θ*:*	**P5**: PL advection, plus 2-D diffusion. **P6**: *Strang splitting* + 3-D diffusion: $\left[F_x\left(\dfrac{\Delta t}{2}\right)\right]\left[F_y\left(\dfrac{\Delta t}{2}\right)\right]\left[F_z\left(\Delta t\right)\right]\left[F_y\left(\dfrac{\Delta t}{2}\right)\right]\left[F_x\left(\dfrac{\Delta t}{2}\right)\right] + K_\theta\left(\delta_{xx}\theta + \delta_{yy}\theta + \delta_{zz}\theta'\right)_{(n)}$

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

(2-D) x-z domain layout, variable placement, and boundary conditions



P5 variables *(**not** incl. ghost zones)*
- u(nx+1, nz)
- w(nx, nz+1)
- θ(nx, nz)
- p'(nx, nz)

Dashed yellow lines
X symmetry boundaries --
- θ, w, p' are **same** on opposite sides of the symmetry boundaries
- u is of **opposite** sign across these boundaries.

- Dimensions:
  - Use Δx=Δz; 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') = Δx/2+Δx(i-1), i=1…nx *(Fortran)*
  - bottom-left corner *θ, p'* are at (x=*Δx/2*, z=*Δz/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 Ξ(*k=0*) = Ξ(k=1, *Fortran*), etc.
- Lateral (x) boundaries: symmetry boundaries shown with dashed yellow lines
  - u(1) = -u(2) u(nx+1) = -u(nx) *(Fortran indices here)*
  - θ(0) = θ(2) θ(nx+1) = θ(nx-1) (same for w,p')
- Lateral (y) boundaries: *(program 6, 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 Ξ(ny+1) = Ξ(1), etc.

C. Initial conditions (base state)

- First you must define the *base state vertical profiles* for density $\overline{\rho}$ and base-state potential temperature $\overline{\theta}$. You only save $\overline{\rho}$ for later use; other variables (z, P, T) are used only to calculate $\overline{\rho}$. 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.
- In the expressions below, z refers to the height at a θ and p' level. The notation given is for Fortran.

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

Check your initial state with this data for Δz=100m, at *(Fortran)* **k=11**, z=1050m:

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

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

Program 5: 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:

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

so θ(i,k) at time t=0 equals the base state (constant) $\overline{\theta}$ *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 is for program 6, in 3-D; *simplify appropriately* for program 5:

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

Fortran	***C*** *requires <math.h>*

`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  (+3 more enddo's)`
	`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 = dx/2.0 + dx*(float)(j-J1);`
	`  z = dx/2.0 + dx*(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 */`

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" .

<u>*Program 6, only*</u>*:* In P6, 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:

$$\theta_{i,j,k} = \bar{\theta} + \sum_{m=1}^{2}\left[\Delta\theta'_m \frac{\cos(r_m\pi)+1}{2} \text{ if } r_m \le 1, \text{ else } 0\right]$$

$$v_{i,j,k} = \sum_{m=1}^{2}\left[\Delta v_m \frac{\cos(r_m\pi)+1}{2} \text{ if } r_m \le 1, \text{ else } 0\right]$$

$$r_m = \sqrt{\left(\frac{x_i - x_0(m)}{xradius_m}\right)^2 + \left(\frac{y_j - y_0(m)}{yradius_m}\right)^2 + \left(\frac{z_k - z_0(m)}{zradius}\right)^2}$$

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).

In program 6, we also utilize random initial **u** values, up to +/-(*upertur/2)*. Use the default Intel Fortran/C random number generator. Here is sample code:

Fortran	***C*** *requires <math.h>*

`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`
	`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;`
	`  }`
	`} }`

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_{theta}.
- 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 $(\theta - \overline{\theta})$.
- 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…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).
- Program 6 only: For full credit, you must make a reasonable attempt at parallelizing your code, and part of your grade also requires visualization.

F. Plotting

Program 5: plot contours as usual. We will not use surface plotting.

Program 6: You *will not* be calling plot routines directly from your program #6. Doing so is slow and wasteful considering how long your programs will (at full resolution) take to rerun. Instead, you will call C or Fortran routine *putfield* (provided to you) to write your output to disk in a unformatted binary file. Use program *plot3d* to read this file and to make any number of plots (X-Y, Y-Z, X-Z slices, or 3-D). See the class web page for details. These routines, program *plot3d* and demonstration programs will be available on stampede at *~tg457444/502/Pgm6*. Required plots will be listed on the class web site.

G. Hints

- Do initial testing at *reduced resolution*, e.g. $\Delta x=\Delta z=200$m, $\Delta t=0.5$s.
- In testing (for Fortran), do early tests compiling with subscript checking:
  -g –check all –traceback.
- Beware! **NX≠NZ here**. Think where you have used NX=NZ in programs 2-4.
- For min/max stats and plotting, average *u* and *w* to θ/p locations; and plot $\left(\theta - \bar{\theta}\right)$
- We are using forward time differencing for θ advection, and centered time for everything else. So, δ_{2t}u means u3 = u1 + 2Δt*(…); θ advection is forward in time, so only two arrays are needed [handled as in programs 2-4].

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 ∂w/∂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.

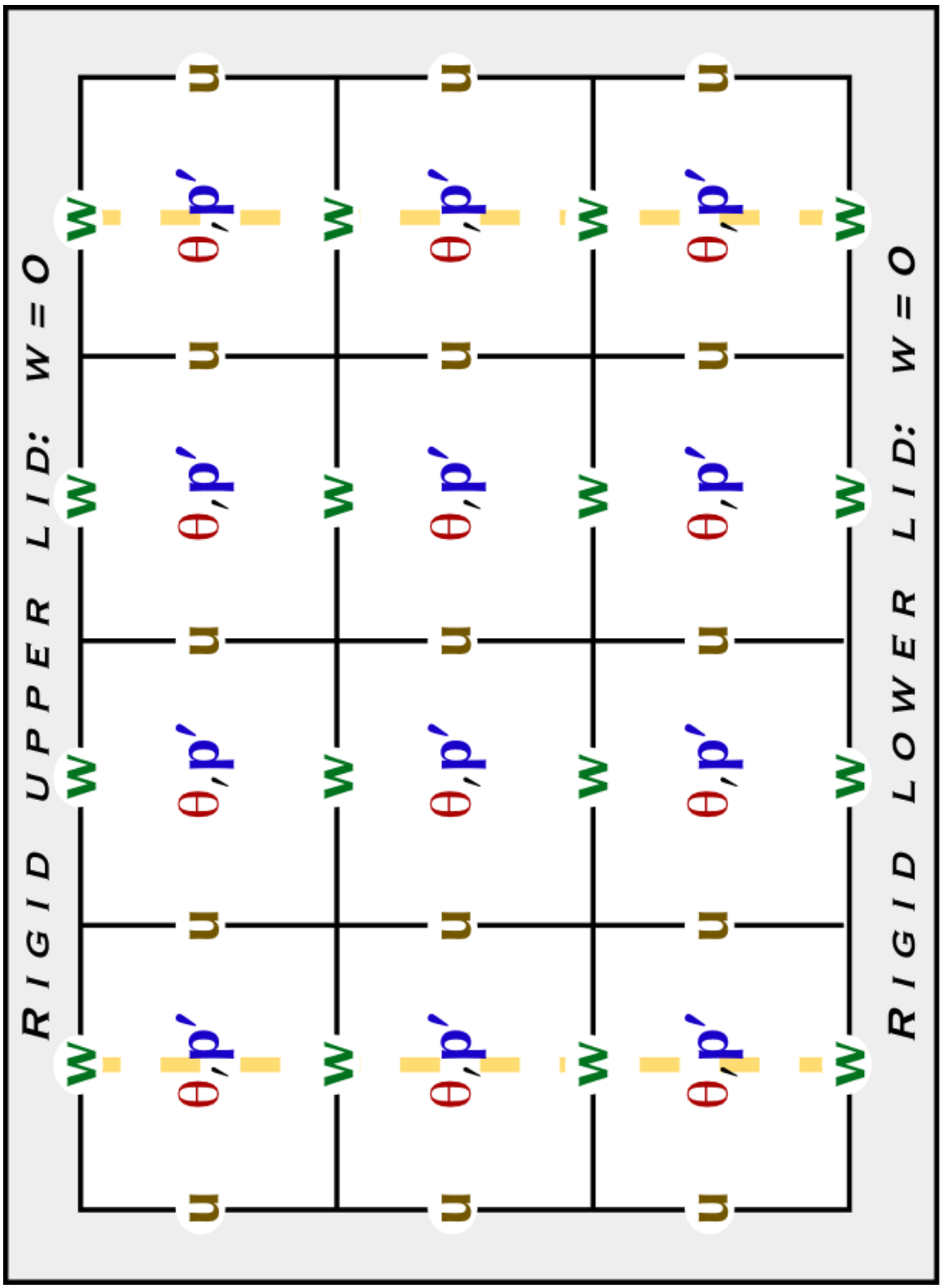
   a. But: the symmetry is in x; comparable symmetry will not occur in z due to the density variation with height.

I. Visualization *(program 6 only)*

Use program *plot3d*, provided to you, to produce the necessary contour plots. Beyond this, part of the program grade (see below) involves creating a few 3-D plots with the visualization tools ***vis5d*** or ***VisIt***. *plot3d* can convert your simulation output to the necessary format. See the class web site for details.

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

1. MAIN PROGRAM a. read in parameters; call ***IC*** b. plot initial condition c. call ***MAXMIN*** d. call ***BC*** e. *set tstep* = Δt f. TIME LOOP: *n=1,max_steps* • set u3=u1, w3=w1, t2=t1 • call ***ADVECT*** • call ***DIFFUSION*** • call ***PGF*** • array update • if (n=1) set tstep = $2\Delta t$ • call ***BC*** • call ***MAXMIN*** • if desired time: PLOT g. END OF TIME LOOP h. plot time traces	***NOTES*** Always plot θ', not total θ Find min, max of all fields Set ghost points for first time step Because your first step is a forward one Array copy helps start this time step. Advection of θ, u, and w. Mix: u, w, and θ *(note: in general $K_m \neq K_{theta}$)* Obtain u3,w3; get new *p3* This is the usual array switch between old, new time levels. There are three time levels for u,w,p, and two for θ. **But**: if first step, don't update u1, w1, or p1. Switch from forward to centered time for u,w,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 .. using min/max u/w/θ I have already stored
2. IC ROUTINE a. compute 1D arrays b. set p',u,w = 0 (in program 6, we set ***v*** using perturbations, and set ***u*** to random numbers; u is nonzero in P6!) c. set θ' based on handout.	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 tstep is first set to Δt, and later to $2\Delta t$) Remember you *read in* the temperature perturbations and their locations
3. BC ROUTINE a. 0-gradient top, bottom b. w, θ, and p' are same on either side of symmetry boundary c. Anti-symmetry for u	 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)
4. ADVECT ROUTINE a. ***u***: u3 = u3 + tstep*(box terms) b. ***w***: w3 = w3 + tstep* (box terms) c. θ advection as usual *(old "integrate" code)*	 Recall u,w,p have centered time derivatives. For program 6, do ***v*** advection here, too. Piecewise linear advection.
5. DIFFUSION ROUTINE a. u3 = u3 + tstep*(x, z mixing terms) b. w3 = w3 + tstep*(x, z mixing terms) c. Mix θ ...	 For program 6, do ***v*** diffusion, too. W is always zero at k=1 and at k=nz+1
6. PGF SUBROUTINE a. u3 = u3 - tstep*(pgf terms) b. w3 = w3 - tstep*(pgf terms) + tstep*(buoyancy terms) c. set u, w BCs (could call *BC,* or set here) d. p3 = p1 - (pgf terms)	Pressure gradient / buoyancy routine. Adding to the u3 array. Adding to the w3 array. Remember w=0 at k=1 and k=nz+1 Get ready for derivatives in p equation pgf terms use new u, w at time (n+1)


RIGID UPPER LID: W = O
RIGID LOWER LID: W = O
u
w
θ,p′