ATM 562:
Numerical methods and modeling

Robert G. Fovell
Department of Atmospheric and Environmental Sciences
University at Albany, State University of New York
rfovell@albany.edu

November 4, 2021

"All models are false, but some are useful." – George E. P. Box

Preface

These notes support a course in numerical methods and modeling. The course involves both theory and hands-on practice, the latter leading to a two-dimensional (or better) "cloud model" capable of simulating some interesting phenomena. It was developed as a 10-week course offered to undergraduate seniors at UCLA, in part from a need to provide scientific programming experience, and also to provide opportunities and inspiration for individual research projects, and included six (now seven) "Model Tasks" that lead to a working model. The model you will create based on this guidance is not new, and nowhere near the state of the art with respect to design, formulation, or model physics, but will be new to *you*, and it will be enabling, and that's what matters.

Please feel free to contact me with suggestions and corrections.

Contents

Part I

Model theory and design

Chapter 1

Introduction and basic tools

1.1 Taylor series

1.1.1 Infinite and truncated series

Taylor expansions of infinite series will be used extensively in this course. Some examples of infinite series are:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

The general Taylor expansion form is given below. Note this is an *exact relationship*, not an approximate one, because the series has not (yet) been truncated.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''}{2!}(x - x_0)^2 + \frac{f'''}{3!}(x - x_0)^3 + \cdots$$

This is termed the "Taylor series expansion of the function $f(x)$ about the point $x = x_0$", and can be rewritten using summation notation:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

where $f^{(k)}$ is the k-th derivative of $f(x)$.

Example. Suppose $f(x) = 2x^2 - 5x + 1$. What is the Taylor series expansion about the point $x_0 = 2$? Using the definition above, one finds the answer to be $f(x) = -1+3(x-2)+2(x-2)^2$. This is exact because the derivatives of order three and above are exactly zero. Now evaluate the function and the Taylor expansion at the point $x = 5$. One should find they are both equal to 26.

Many series — like e^x and $\sin x$ — are truly infinite and this need to be truncated in practice. Truncating a series leads to *error*. Fortunately, many series are convergent, which means that each higher order term makes smaller and smaller contributions to the whole. This is the justification for truncating the series at some point.

1.1.2 The Mean Value Theorem and definition of the derivative

A special case of the Taylor series is the *Mean Value Theorem*, one of the fundamental theorems of calculus. The MVT is based on a Taylor series in which only the first derivative is retained. Start with:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0).$$

This is a truncated series in this instance (assuming at least one higher order derivative is nonzero), so it is only an approximation. We can replace the $\approx$ sign with strict equality if we evaluate the derivative at a different point somewhere between x and x_0. Let x^* be such a point. Then:

$$f(x) = f(x_0) + f'(x_*)(x - x_0).$$

This is the MVT. If we had kept still higher order terms, then it would have been the Extended MVT. No matter how many terms are kept, if the last term's derivative is evaluated at an intermediate point, then the relationship is still exact.

Note that the MVT is simply the definition of the derivative in disguise! To see this, rewrite it as:

$$f'(x^*) = \frac{f(x) - f(x_0)}{(x - x_0)}.$$

The derivative of $f(x)$ with respect to x is simply the difference between the function values at two locations, x and x_0, divided by the distance between those locations. The MVT says that there is a point x_* between x and x_0 where this is exactly true. At other points between x and x_0, possibly including x and x_0, it is an approximation.

1.1.3 The basis of forecasting

The approximated, truncated Taylor series provides the basis of *finite differencing*, the most commonly employed method for making forecasts. To see this a bit more easily, make f a

function of time instead:

$$f(t_2) \approx f(t_1) + f'(t_1)\Delta t.$$

Here, times t_1 and t_2 represent present and future times, respectively, separated by a time interval Δt. Note that the derivative on the right hand side is evaluated at the present time; we don't know where the intermediate time t^* should be.

Clearly, if we know $f(t_1)$ and $f(t_2)$, we can get at least an estimate for the time derivative of f by rearranging the above equation. For the purposes of forecasting, say instead we have an estimate for $f'(t_1)$ in addition to knowing $f(t_1)$. Then, we can use the above equation to get a forecast for $f(t_2)$, the function value at the future time. This involves not just knowledge of $f'(t_1)$, but also the assumption that $f'(t_1)$ remains constant over the time interval Δt. This is extrapolation, and the danger for error should be obvious.

How do you get $f'(t_1)$? We need equations that take knowledge of the function *now* (and perhaps at past times as well) and uses this to estimate how the function is changing with time *now*. The functions we are dealing with include the temperaure, pressure, density and wind components. The equations for the time rate of change of these functions or variables are derived below, starting in Section 1.2.

1.1.4 Multivariate Taylor expansions

The Taylor series examples above assumed expansion took place using just one variable. The multivariate Taylor expansion of $f(a + \Delta a, b + \Delta b)$ about the point $f(a, b)$ can be written (in extended and compact notation) as:

$$f(a + \Delta a, b + \Delta b) = f(a, b) + \Delta a \frac{\partial f}{\partial a} + \Delta b \frac{\partial f}{\partial b} + \frac{(\Delta a)^2}{2} \frac{\partial^2 f}{\partial a^2} + \frac{(\Delta b)^2}{2} \frac{\partial^2 f}{\partial b^2} + \Delta a \Delta b \frac{\partial^2 f}{\partial a \partial b} + \cdots$$

$$f(a + \Delta a, b + \Delta b) = \sum_{k=0}^{\infty} \frac{1}{k!} \left[\Delta a \frac{\partial}{\partial a} + \Delta b \frac{\partial}{\partial b} \right]^k f.$$

The notational changes above remind you of the variety of notations used for such common objects as variables, derivatives and the like.

1.2 Eulerian and Lagrangian viewpoints

Suppose we identify an air parcel that has a fixed amount of mass (a *control mass*, or CM) but is free to move in space. If we track the parcel's temperature as a function of time, we can write its temperature tendency as $\frac{dT}{dt}$. This is the Lagrangian viewpoint. Suppose instead we identify a fixed volume in space (a *control volume*, or CV), through which air

parcels may flow freely. We can write the tendency of temperature within this fixed space as $\frac{\partial T}{\partial t}$. This is the Eulerian viewpoint. Now, finally suppose that temperature varies through the three spatial dimensions as well as with time. We can relate the two viewpoints through the chain rule for partial derivatives:

$$\frac{dT}{dt} = \frac{\partial T}{\partial t}\frac{dt}{dt} + \frac{\partial T}{\partial x}\frac{dx}{dt} + \frac{\partial T}{\partial y}\frac{dy}{dt} + \frac{\partial T}{\partial z}\frac{dz}{dt}$$

Of course, $\frac{dt}{dt} = 1$, and the other time derivatives on the right hand side are the velocity components we conventionally designate u, v and w, respectively. Letting the vector velocity be designated as $\vec{V} = u\hat{i} + v\hat{j} + w\hat{k}$ (where $\hat{i}$, $\hat{j}$, and $\hat{k}$ are unit vectors in the x, y and z directions, respectively) then:

$$\frac{dT}{dt} = \frac{\partial T}{\partial t} + \vec{V} \cdot \nabla T$$

where $\nabla T = \frac{\partial T}{\partial x}\hat{i} + \frac{\partial T}{\partial y}\hat{j} + \frac{\partial T}{\partial z}\hat{k}$.

The quantity $\vec{V} \cdot \nabla T$ is called temperature advection. In practice, we wish to compute the temperature tendency at a fixed point, so we rewrite the above in terms of the Eulerian derivative:

$$\frac{\partial T}{\partial t} = \frac{dT}{dt} - \vec{V} \cdot \nabla T$$

This is interpreted as follows: The rate of temperature change at this location (the Eulerian derivative on the left-hand side) is determined by the temperature of the air that is flowing into this location (cold or warm advection) plus how that air's temperature is changing (by radiation or diabatic heating, for example) as it is being advected (the Lagrangian derivative).

1.3 Ideal gas law and virtual temperature

The ideal gas law (IGL) relates pressure p, temperature T and density ρ:

$$p = \rho R T$$

where R is the "gas constant", which varies among the different gases. Air is, of course, a mixture of gases, each with a different R. The good news is that since the constituents of dry air vary little in time or space we can specify a gas constant for the dry air mixture as a weighted average of the consitutent constants. This will be called R_d, and taken to be 287 J kg^{-1} K^{-1}. The bad news is that water vapor varies tremendously in both space and time, and so we need to further adjust R_d according to the amount of vapor in the air.

This is an irritant, since the gas "constant" is no longer a constant. We'll handle this by taking $R = R_d$ always, and fold the influence of water vapor into another term in the equation — temperature. (The advantage of this will be mentioned briefly below.) Let the

water vapor mixing ratio, expressed as grams of water vapor per gram of dry air, be q_v. Then, we can define a *virtual temperature* as:

$$T_v = T(1 + 0.61q_v).$$

By folding the water vapor information into temperature, we can employ the dry air constant R_d in the ideal gas law, and we write this as:

$$p = \rho R_d T_v.$$

The virtual temperature is interpreted as follows. Since the vapor gas constant, $R_v = 461$ J kg^{-1} K^{-1}, is greater than the dry air mixture constant, an air parcel composed solely of vapor is less dense at the same temperature and pressure than a parcel made up of dry air. This is obvious from the IGL itself. Thus, a moist parcel is less dense than a dry one (at the same T, p).

There are two ways of decreasing a parcel's density without changing the pressure. One way is to increase its temperature, since for fixed pressure ρ must decrease as T rises. The other way is to replace some dry air with water vapor at constant T and p, which effectively forces the mixture R to increase and thus decreases ρ. The virtual temperature can be thought of as representing the temperature a perfectly dry air parcel must have to have the same density as a moist air parcel at the same pressure. However, a better way of employing it is as a density-weighted measure that is useful in judging the buoyancy of air. Less dense air parcels want to rise relative to more dense parcels, and parcel density reflects not only temperature but also vapor content.

1.4 The hydrostatic equation

Hydrostatic balance represents the stalemate between the vertical pressure gradient and gravity forces. Pressure is force per unit area, and thus $\frac{dp}{dz}$ is force per unit volume. Newton's second law for a mass subjected to gravity tells us the gravity force is object mass (m) times the acceleration of gravity (g), so the gravity force per unit volume (V) is $\frac{mg}{V}$ or ρg. The balance between these two, then, is:

$$\frac{dp}{dz} = -\rho g$$

where the minus sign is needed because p decreases with increasing height z.

1.5 Potential temperature and nondimensional pressure

The potential temperature, θ, is conserved for isolated, insulated air parcels undergoing strictly "dry" (subsaturated) adiabatic processes. It is defined as:

$$\theta = T \left[\frac{p_0}{p} \right]^{\frac{R_d}{c_{pd}}},$$

where $p_0 \equiv 1000$ mb or 100000 N m^{-2}. (There is a very small error involved in applying this equation to moist, but subsaturated, air parcels.) If we define the nondimensional pressure, π, as

$$\pi = \left[\frac{p}{p_0} \right]^{\frac{R_d}{c_{pd}}}$$

then temperature and potential temperature are related as $T = \theta\pi$.

The advantage of using nondimensional pressure instead of pressure in numerical modeling will become obvious later. At this point, we need to rework the hydrostatic equation in terms of π. Apply logarithms to the definition of π and then differentiate it:

$$\begin{aligned} \ln \pi &= \frac{R_d}{c_{pd}} (\ln p - \ln p_0) \\ \frac{d \ln \pi}{dz} &= \frac{R_d}{c_{pd}} \frac{d \ln p}{dz} \\ \frac{1}{\pi} \frac{d\pi}{dz} &= \frac{R_d}{c_{pd}} \frac{1}{p} \frac{dp}{dz} \end{aligned}$$

At this point, use both the hydrostatic equation and the IGL to yield:

$$\begin{aligned} \frac{d\pi}{dz} &= -\frac{\pi g}{c_{pd} T_v} \\ &= -\frac{g}{c_{pd} \theta_v}. \end{aligned}$$

In the last step, the virtual potential temperature, $\theta_v = \theta(1. + 0.61 q_v)$, was employed.

1.6 The continuity equation

The continuity equation is a prediction equation for density, but specifically expresses conservation of mass. In the Eulerian CV viewpoint, volume is fixed. To increase the density in the fixed volume, it is necessary to pack more mass into that volume. Thus, an airflow

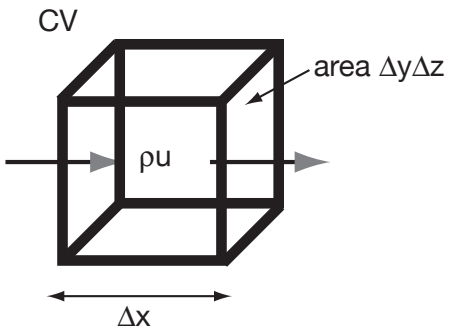

Figure 1.1: Flow in and out of a control volume.

that results in a net influx of mass is required. In the Lagrangian CM viewpoint, the mass is fixed and density can be increased by contracting the volume. Thus, if the parcel finds itself in a converging airflow, one that acts to squeeze the volume, density can increase.

Figure 1 shows a control volume, with volume $\Delta x \Delta y \Delta z$. For the x direction, the velocity component is u and mass flow per unit area in the x-direction at the center of the CV is ρu (units kg s^{-1} m^{-2}). For there to be net mass influx (or outflux), there needs to be a mass flux gradient, $\frac{\partial \rho u}{\partial x}$, across the box. The side walls normal to the flow, with area $\Delta y \Delta z$, are located at distances of $\frac{\Delta x}{2}$ from the CV's center.

We can use Taylor series, truncated to include only the first derivative, to specify the flow at the volume sides in terms of the flow at the CV's center and the flow gradient across the box. The flow in the left side of the box (through area $\Delta y \Delta z$) is

$$\left[\rho u - \frac{\partial \rho u}{\partial x}\frac{\Delta x}{2}\right] \Delta y \Delta z$$

while the flow out the right side is

$$\left[\rho u + \frac{\partial \rho u}{\partial x}\frac{\Delta x}{2}\right] \Delta y \Delta z.$$

Thus, the *net flow* (in minus out) is

$$-\frac{\partial \rho u}{\partial x} \Delta x \Delta y \Delta z.$$

For all three directions combined, the net flow is

$$-\left[\frac{\partial \rho u}{\partial x} + \frac{\partial \rho u}{\partial y} + \frac{\partial \rho u}{\partial z}\right] \Delta x \Delta y \Delta z.$$

Finally, divide by volume to get the net per unit volume. What remains can be written as $-\nabla \cdot \rho \vec{V}$. Since this net flow is acting to locally increase the density, we can write the result as:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot \rho \vec{V}.$$

Alternatively, rearrange the above expression as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{V} = 0.$$

This is the Eulerian form of the continuity equation. Interpret this equation as follows: in a CV, you can increase density (mass divided by volume) by packing more mass in the fixed volume. To do this, mass inflow has to exceed mass outflow. Conservation of mass means that the net mass inflow goes to change the local density, with nothing missing or left over.

One way to get the Lagrangian form of the continuity equation is by applying a vector identity to the Eulerian form. First, though, recall that the total and local time tendencies are related though

$$\frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \vec{V} \cdot \nabla \rho.$$

The vector identity is simply the chain rule applied to the vector dot product:

$$\nabla \cdot \rho \vec{V} = \rho \nabla \cdot \vec{V} + \vec{V} \cdot \nabla \rho.$$

Starting with the Eulerian continuity equation and applying both yields:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \vec{V} &= 0 \\
\left[\frac{\partial \rho}{\partial t} + \vec{V} \cdot \nabla \rho \right] + \rho \nabla \cdot \vec{V} &= 0 \\
\frac{d\rho}{dt} + \rho \nabla \cdot \vec{V} &= 0 \\
\frac{1}{\rho} \frac{d\rho}{dt} + \nabla \cdot \vec{V} &= 0.
\end{aligned}$$

That is the Lagrangian continuity equation. Interpret this equation as follows: for a control mass, you can increase the density by contracting the volume. Do this by placing the CM into a converging airflow, one that deforms the CM's shape (and thus volume).

Chapter 2

Model equations and waves

2.1 Basic equations

Consider first a simple, two-dimensional (2D) model of a completely dry, adiabatic atmosphere. At a minimum, the model has four equations for four variables that need to be predicted: horizontal velocity u, vertical velocity w, potential temperature θ and density ρ. Note that because of the ideal gas law, we do not need prediction equations for both ρ and pressure p. Although we will solve them in the Eulerian reference frame, the equations are presented in the Lagrangian form for now. Diffusion, Coriolis and friction terms are neglected. The equations are:

$$\frac{du}{dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x} \tag{2.1}$$

$$\frac{dw}{dt} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - g \tag{2.2}$$

$$\frac{d\theta}{dt} = 0 \tag{2.3}$$

$$\frac{d\rho}{dt} = -\rho\left[\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}\right] \tag{2.4}$$

We will need to modify these equations substantially before they will be ready to put into the model.

First, however, we need to see what kind of atmospheric waves these equations can support. The equations involve two distinctly different kinds of phenomena: having advective and wave-like characteristics. Advective phenomena move with the flow, and always move downstream. Waves, on the other hand, can move independently of the flow, and can propagate downstream and/or upstream. Two very different kinds of waves are supported in these equations: sound (acoustic) waves and gravity waves. We shall reveal these waves and examine their characteristics by employing the perturbation method.

2.2 Perturbation method

The perturbation method represents a way of simplifying the equations. Each variable in the model is usually a function of all spatial dimensions, plus time; i.e., $p = p(x, z, t)$ in 2D. A given variable field can be apportioned into two parts, representing a mean value ($\bar{p}$) and a deviation from that mean (the perturbation, p'). Usually, the mean value is taken to be either constant or a function of height alone. Further, the perturbation is usually assumed to have but a small magnitude, especially compared to the mean. As an example, consider surface pressure. Average sea level pressure is ≈ 1000 mb (the mean). Variations about this average value are usually ± 50 mb or so, which is small compared to the mean value.

Because perturbations are usually small compared to the mean, we can reduce the degree of complexity in the equations by "linearizing" them (i.e., neglecting *products* of perturbation values). A useful trick to remember is that, for x small,

$$\ln(1 + x) \approx x.$$

[This was obtained by truncating the Taylor series for $\ln(1 + x)$]. So, if $\rho' << \bar{\rho}$, then:

$$\ln(\bar{\rho} + \rho') = \ln\left[\bar{\rho}(1 + \frac{\rho'}{\bar{\rho}})\right] \approx \ln\bar{\rho} + \frac{\rho'}{\bar{\rho}} \tag{2.5}$$

Another trick is embodied in the binomial expansion theorem:

$$\begin{aligned} \frac{1}{\bar{\rho} + \rho'} &= \frac{1}{\bar{\rho}}\left[1 + \frac{\rho'}{\bar{\rho}}\right]^{-1} \\ &\approx \frac{1}{\bar{\rho}}\left[1 - \frac{\rho'}{\bar{\rho}}\right] \end{aligned} \tag{2.6}$$

2.3 Acoustic (sound) waves

Sound waves exist owing to the compressibility (squeezability) of air. You know that air is compressible because both pressure and density decrease very rapidly with height. Compressibility also can be thought of as "slackness". Picture an air parcel that undergoes adiabatic expansion. What does the given parcel's expansion do to a neighboring parcel? The neighboring parcel can either get squeezed (compressed) or it can move out of the given parcel's way. In a compressible medium like air, both will happen — in sequence. First, the neighboring parcel will be compressed, and then it will rebound and move. The medium's degree of compressibility determines how rapidly the action of getting pushed will result in motion. In a nearly incompressible fluid, very little of the push results in compression, and so the neighboring parcel starts getting out of the way very quickly. In a very compressible medium, the neighboring parcel may not move at all.

Sound waves represent the adjustment process involved in parcel rebounding after compression. Picture a set of concentric rings, all centered on a common point. The innermost ring expands, causing the next ring out to compress. That ring rebounds and expands, causing the next ring outward to compress, and so on. Note the rings themselves do not experience any net motion; they merely push one way and then the other. However, the wave that results propagates outward. That is a sound wave. Further note that the back-and-forth motion within the rings is parallel to the wave's propagation direction. Such a wave is termed "longitudinal".

To examine sound waves in a simple manner, we take our basic equations but neglect the w equation and the vertical direction. This is because a horizontally propagating sound wave does not induce vertical motion (the wave is longitudinal). So we have (revising the ρ and θ equations a little for convenience):

$$\frac{du}{dt} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0 \tag{2.7}$$

$$\frac{d\ln\rho}{dt} + \frac{\partial u}{\partial x} = 0 \tag{2.8}$$

$$\frac{d\ln\theta}{dt} = 0 \tag{2.9}$$

where (because of our simplifications)

$$\frac{d}{dt} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial x}.$$

This derivation closely follows that presented in Holton's text.

Potential temperature can be eliminated from the equations in the following fashion: Start with the definition of θ:

$$\theta = T\left[\frac{p_0}{p}\right]^{\frac{R_d}{c_{pd}}},$$

and use the ideal gas law to replace T:

$$\theta = \frac{p}{\rho R_d}\left[\frac{p_0}{p}\right]^{\frac{R_d}{c_{pd}}}.$$

Then, take the log of both sides, yielding:

$$\ln\theta = \frac{c_{vd}}{c_{pd}}\ln p - \ln\rho + constants. \tag{2.10}$$

Differentiate with time, and set equal to zero [owing to (2.9)]:

$$\frac{1}{\gamma}\frac{d\ln p}{dt} - \frac{d\ln\rho}{dt} = 0, \tag{2.11}$$

where $\gamma = \frac{c_{pd}}{c_{vd}}$. Use (2.8) to eliminate the time tendency of ρ, resulting in:

$$\frac{1}{\gamma}\frac{d\ln p}{dt} + \frac{\partial u}{\partial x} = 0. \tag{2.12}$$

This finally leaves us with two equations, which may be written as:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial p}{\partial x} = 0 \tag{2.13}$$

$$\frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x} + p\gamma\frac{\partial u}{\partial x} = 0. \tag{2.14}$$

Now expand u, p and ρ into mean and perturbation parts, where the mean parts are simply constants (as there is no vertical direction to worry about here):

$$u(x,t) = \bar{u} + u'(x,t),$$

$$p(x,t) = \bar{p} + p'(x,t),$$

$$\rho(x,t) = \bar{\rho} + \rho'(x,t).$$

Substitute these expressions into (2.13) and (2.14), yielding:

$$\frac{\partial}{\partial t}(\bar{u}+u') + (\bar{u}+u')\frac{\partial}{\partial x}(\bar{u}+u') + \frac{1}{\bar{\rho}+\rho'}\frac{\partial}{\partial x}(\bar{p}+p') = 0 \tag{2.15}$$

$$\frac{\partial}{\partial t}(\bar{p}+p') + (\bar{u}+u')\frac{\partial}{\partial x}(\bar{p}+p') + \gamma(\bar{p}+p')\frac{\partial}{\partial x}(\bar{u}+u') = 0 \tag{2.16}$$

Apply (2.6) to (2.15), expand the terms and neglect products of perturbations (like $u'u'$ and $u'p'$). This results in:

$$\left[\frac{\partial}{\partial t} + \bar{u}\frac{\partial}{\partial x}\right]u' + \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial x} = 0 \tag{2.17}$$

$$\left[\frac{\partial}{\partial t} + \bar{u}\frac{\partial}{\partial x}\right]p' + \gamma\bar{p}\frac{\partial u'}{\partial x} = 0 \tag{2.18}$$

Finally, (2.17) and (2.18) may be combined into a single equation in p':

$$\left[\frac{\partial}{\partial t} + \bar{u}\frac{\partial}{\partial x}\right]^2 p' - \frac{\gamma\bar{p}}{\bar{\rho}}\frac{\partial^2 p'}{\partial x^2} = 0. \tag{2.19}$$

At this point, we assume we will find wave-like solutions in p' in this equation. We further assume the wave has constant amplitude ("A") with wavelength L_x in the x-direction and it propagates with time with speed c. Let the horizontal wavenumber k be related to the horizontal wavelength by

$$k = \frac{2\pi}{L_x}.$$

Then, we can express the wave-like character as:

$$\begin{aligned} p' &= Ae^{ik(x-ct)} \\ &= AE \end{aligned} \tag{2.20}$$

where E is merely a convenient shorthand. This expression employs *Euler's relation*:

$$e^{i\phi} = \cos\phi + i\sin\phi$$

so clearly the expression involves wave-like undulations in both space and time. Notice that the expression is complex; only the real part has any physical significance.

Given (2.20), we thus produce:

$$\frac{\partial p'}{\partial t} = -Aikc E,$$

$$\frac{\partial p'}{\partial x} = Aik E,$$

and

$$\frac{\partial^2 p'}{\partial x^2} = -Ak^2 E,$$

where it is recalled that $i^2 = -1$. Put these in (2.19), and one winds up with:

$$AE(-ikc + ik\bar{u})^2 + AE\frac{\gamma\bar{p}}{\bar{\rho}}(k)^2 = 0, \tag{2.21}$$

a quadratic equation in the sound wave speed c. Canceling out the AE term (because the right hand side is zero) and solving for c yields:

$$\begin{aligned} c &= \bar{u} \pm \left[\frac{\gamma\bar{p}}{\bar{\rho}}\right]^{\frac{1}{2}} \\ &= \bar{u} \pm (\gamma R_d \bar{T})^{\frac{1}{2}}. \end{aligned} \tag{2.22}$$

The last expression employed the ideal gas law.

This derivation has shown that the "adiabatic speed of sound" in a calm atmosphere ($\bar{u} = 0$) is given by $c_s \equiv c = \sqrt{\gamma R_d T}$. For $T = 273$ K, $c_s \approx 331$ m s^{-1}. Note that (2.22) has two signs. This is because sound waves will propagate in both directions away from their point of origin (we're only considering a single coordinate axis here). The quantity $\sqrt{\gamma R_d T}$ is called the *intrinsic phase speed*, or the sound wave speed relative to the flow. To get the sound speed relative to the *ground*, we need to factor in the flow speed relative to the ground.

Finally, it is noted that wave frequency (σ) and phase speed are related through

$$\sigma = kc.$$

For a sound wave, the wave frequency determines the pitch of the sound we here. It is seen that when the flow is not calm, and the sound wave is propagating downstream relative to the flow, c_s is effectively increased. This causes the frequency to increase (raising the pitch). An upstream propagating sound wave, however, has its speed reduced and thus results in a smaller frequency and lower pitch. This is why the pitch of a car horn is higher when it is approaching you than when it is moving away, a phenomenon known as "Doppler shifting".

2.4 Gravity waves (buoyancy oscillations)

The model equations also support "gravity waves", a rather poor term describing stable buoyancy oscillations. Picture an insulated, dry air parcel moving vertically in a stable environment. (The important complication of moisture is ignored for now.) A rising parcel moves towards lower pressure. The decreasing pressure allows the parcel to expand, and expanding air cools. A sinking parcel moves towards higher pressure, and heats up as it compresses. The parcel's temperature changes according to the dry adiabatic lapse rate Γ_d, given by:

$$\Gamma_d = \frac{g}{c_{pd}} \approx 9.8 \text{ K km}^{-1}.$$

A stable environment is one in which the environmental lapse rate of temperature is less than the dry adiabatic rate. We can define a property that is conserved during such a displacement, and we call it the potential temperature θ. Potential temperature increases with height in a stable environment.

In the derivation of gravity waves, we usually presume the parcel remains in mechanical equilibrium with its environment, meaning that the *pressures inside and outside the parcel are the same*. In other words, $p' = 0$. This is not just a derivational convenience, but in reality, we would like to avoid doing it if at all possible. The implications and motivation for this assumption will be seen presently.

In a stable environment, a rising parcel soon finds itself cooler than its surroundings. If the parcel and environmental pressures are the same, being cooler also means being more dense (owing to the ideal gas law), and it is because of its density excess that the parcel wishes to sink. Similarly, a sinking parcel finds itself warmer — and thus less dense at the same pressure — than its surroundings, and as a result wishes to rise. In short, however it is displaced, the parcel seeks to return to its original location.

However, once it achieves its goal of returning to its original location, the displaced parcel cannot stop at that location because it still has an acceleration. Think of a pendulum bob suspended by a string. Its original location is straight down — gravitationally, its most stable point. If you push the bob either direction, and let it go, it seeks to return to that original point. Yet the bob keeps missing that point because it still has an acceleration that

carries it onward. Once initiated, the pendulum bob continues to oscillate about its original, stable point, until its energy is dissipated in some fashion. Not only is the stably displaced air parcel similar to the pendulum bob, but also we shall see its motion is governed by the very same equation, that of simple harmonic motion.

Further, the moving parcel is disturbing its surrounding environment, which will attempt to adjust back into its original state. This adjustment takes the place of gravity waves which propagate away from the original point of disturbance. In the simplest limit, the parcel oscillations are purely vertical and the adjustment waves move only horizontally. Thus, gravity waves are transverse rather than longitudinal waves like sound waves.

2.4.1 The buoyancy term

Equation (2.2), repeated below, describes the vertical motion of an air parcel when mixing, diffusion, moisture and friction can be neglected:

$$\frac{dw}{dt} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - g \tag{2.23}$$

The right hand side of (2.23) is the hydrostatic equation, and is exactly equal to zero when the atmospheric state is precisely hydrostatic. Hydrostatic balance means the air has no vertical accelerations, which means velocity is constant, and most likely *zero*. Therein lies the rub: the average state of any spatially extensive slice of the atmosphere is so close to hydrostatic balance that (2.23) would be useless for calculating the vertical acceleration of that area. (This is because the left hand side would be the small difference between two very large terms, and thus subject to huge relative errors.) However, perturbations from this state, such as might exist in the interior of narrow cumulus clouds, might well depart from hydrostatic balance by a large degree, but this merely serves to reinforce the notion that the mean, hydrostatically balanced state should be removed from this equation.

Therefore, we subject (2.23) to a perturbation analysis, though a somewhat more sophisticated one than we employed for sound waves. We take each variable embodied in (2.1)-(2.4) and break them into mean and perturbation parts:

$$u(x,z,t) = \bar{u}(z) + u'(x,z,t),$$

$$w(x,z,t) = \bar{w}(z) + w'(x,z,t),$$

$$p(x,z,t) = \bar{p}(z) + p'(x,z,t),$$

$$\rho(x,z,t) = \bar{\rho}(z) + \rho'(x,z,t),$$

$$\theta(x,z,t) = \bar{\theta}(z) + \theta'(x,z,t).$$

To simplify matters, we can pretend mean density is constant with height, so $\bar{\rho} = \rho_0$, a constant. Owing to the exponential dependence of density on height, this approximation can only be applied to a very shallow atmospheric layer. In addition, we can make the analysis easier by taking the mean horizontal wind to be calm. A mean flow only Doppler shifts the gravity waves, and we can add this effect in at the derivation's end. Finally, the mean state is a function of height alone, and is further assumed to be in precise hydrostatic balance and vertically motionless so $\bar{w} = 0$ and

$$\frac{d\bar{p}}{dz} = -\rho_0 g.$$

The perturbation analysis makes the right hand side of (2.23) become:

$$-\frac{1}{\rho}\frac{\partial p}{\partial z} - g = -\frac{1}{\rho_0 + \rho'}\left[\frac{d\bar{p}}{dz} + \frac{\partial p'}{\partial z}\right] - g.$$

Using the binomial expansion (2.6), we obtain:

$$-\frac{1}{\rho}\frac{\partial p}{\partial z} - g = -\frac{1}{\rho_0}\frac{d\bar{p}}{dz}\left[1 - \frac{\rho'}{\rho_0}\right] - \frac{1}{\rho_0}\frac{\partial p'}{\partial z} - g.$$

Applying the hydrostatic equation reduces the buoyancy term further to:

$$-\frac{1}{\rho}\frac{\partial p}{\partial z} - g \quad = \quad -\frac{1}{\rho_0}\frac{\partial p'}{\partial z} - \frac{\rho'}{\rho_0}g. \tag{2.24}$$

Now take (2.10) and apply the perturbation method to it:

$$\ln(\bar{\theta} + \theta') = \gamma^{-1}\ln(\bar{p} + p') - \ln(\rho_0 + \rho') + constants.$$

Using (2.5) yields:

$$\ln\left[\bar{\theta}\{1 + \frac{\theta'}{\bar{\theta}}\}\right] \quad = \quad \gamma^{-1}\ln\left[\bar{p}\{1 + \frac{p'}{\bar{p}}\}\right] - \ln\left[\rho_0\{1 + \frac{\rho'}{\rho_0}\}\right] + constants. \tag{2.25}$$

Separating the values within the brackets and applying (2.5) yields:

$$\ln\bar{\theta} + \frac{\theta'}{\bar{\theta}} \quad = \quad \gamma^{-1}\ln\bar{p} + \gamma^{-1}\frac{p'}{\bar{p}} - \ln\rho_0 - \frac{\rho'}{\rho_0} + constants. \tag{2.26}$$

The base state itself satisfies a balance obtained by replacing the variables in (2.10) with their mean values:

$$\ln\bar{\theta} = \gamma^{-1}\ln\bar{p} - \ln\rho_0 + constants.$$

Subtracting this equation from (2.26) leaves us with:

$$\frac{\theta'}{\bar{\theta}} \approx \gamma^{-1}\frac{p'}{\bar{p}} - \frac{\rho'}{\rho_0}.$$

With the ideal gas law, we can see that the first term on the right hand side is the adiabatic speed of sound, and so (after rearranging a bit):

$$\frac{\rho'}{\rho_0} \approx \frac{1}{c_s^2}\frac{p'}{\rho_0} - \frac{\theta'}{\overline{\theta}}. \tag{2.27}$$

Equation (2.27) may be interpreted as follows: A parcel, having properties p, ρ, and θ, is displaced from its original position. The differences between the parcel's and environment's properties are represented by the perturbation values, p', ρ' and θ'. These perturbation properties are related by (2.27). Of special importance is the parcel's density perturbation relative to the environment. If the parcel is less dense ($\rho' < 0$), it will wish to rise from its new position. The equation shows its density perturbation is a function of its pressure and (potential) temperature discrepancies with respect to its surroundings.

Recall that one of the standard air parcel assumptions is that the parcel is in mechanical equilibrium with the environment (i.e., $p' = 0$). Equation (2.27) suggests the time scale associated with the attainment of mechanical equilibrium involves the sound wave speed. The faster the speed of sound is, the more rapidly this equilibrium is brought about. Another way of saying this is: the more incompressible the medium is, the less "slack" that is permitted to occur at any instant of time. In the incompressible limit, $c_s \to \infty$ and the pressure perturbation disappears. Even in a very compressible fluid such as air, the adjustment is so quick that it may be safely neglected.

This is the justification for the air parcel approach. Thus, we can reasonably drop the p' term, leaving us with:

$$\frac{\rho'}{\rho_0} \approx -\frac{\theta'}{\overline{\theta}}, \tag{2.28}$$

and thus an air parcel that is warmer than its surroundings ($\theta' > 0$) is also less dense ($\rho' < 0$). Therefore, density differences may be determined from temperature properties alone. After applying (2.28) to (2.24), we can see that we have transformed the right hand side of (2.23), creating (and neglecting the approximation signs):

$$\frac{dw}{dt} = -\frac{1}{\rho_0}\frac{\partial p'}{\partial z} + g\frac{\theta'}{\overline{\theta}}. \tag{2.29}$$

2.4.2 The gravity wave phase speed

At this point, we apply the perturbation method to the remaining equations in the set (2.1)-(2.4) and, after neglecting products of perturbations, we find:

$$\frac{\partial u'}{\partial t} = -\frac{1}{\rho_0}\frac{\partial p'}{\partial x} \tag{2.30}$$

$$\frac{\partial w'}{\partial t} = -\frac{1}{\rho_0}\frac{\partial p'}{\partial z} + g\frac{\theta'}{\bar{\theta}} \tag{2.31}$$

$$\frac{\partial u'}{\partial x} + \frac{\partial w'}{\partial z} = 0 \tag{2.32}$$

$$\frac{\partial \theta'}{\partial t} + w'\frac{d\bar{\theta}}{dz} = 0. \tag{2.33}$$

Consistent with our previous assumptions, we are taking the atmosphere to be incompressible (and therefore shallow), and that is why the continuity equation (2.32) has been reduced to such a simple form. Take the horizontal derivative of (2.31) and subtract from it the vertical derivative of (2.30) to obtain:

$$\frac{\partial}{\partial t}\left[\frac{\partial w'}{\partial x} - \frac{\partial u'}{\partial z}\right] - \frac{g}{\bar{\theta}}\frac{\partial \theta'}{\partial x} = 0. \tag{2.34}$$

Using (2.32) and (2.33), θ' and u' can be eliminated from the above equation, leaving an expression in one variable, w':

$$\frac{\partial^2}{\partial t^2}\left[\frac{\partial^2 w'}{\partial x^2} + \frac{\partial^2 w'}{\partial z^2}\right] + N^2\frac{\partial^2 w'}{\partial x^2} = 0, \tag{2.35}$$

where the Brunt-Väisälä frequency, N, has been defined to be

$$N \equiv \sqrt{g\frac{d\ln\bar{\theta}}{dz}}.$$

Note that 2.35 is the equation of a simple harmonic oscillator, having precisely the same form as the pendulum equation you saw in physics class.

Again, wave-like solutions are assumed, this time in the variable w', which is presumed to have a two-dimensional spatial structure:

$$w' = Ae^{i(kx+mz-\omega t)},$$

where m is the vertical wavenumber, related to the vertical wavelength L_z by:

$$m = \frac{2\pi}{L_z},$$

and ω is the gravity wave frequency, related to the oscillation period P by:

$$\omega = \frac{2\pi}{P}.$$

Substituting this into (2.35) yields a quadratic equation in ω:

$$\omega^2(k^2 + m^2) - N^2k^2 = 0.$$

Solving for the frequency yields:

$$\omega = \pm\frac{Nk}{\sqrt{k^2 + m^2}}. \tag{2.36}$$

Note again there are two frequencies, representing waves that travel outward on either side of the original disturbance. If we had taken the mean horizontal flow, $\bar{u}$, to be nonzero (and constant with height) rather than calm, we would have instead ended up with:

$$\omega = \bar{u}k \pm \frac{Nk}{\sqrt{k^2 + m^2}}. \tag{2.37}$$

Gravity waves propagate both horizontally and vertically. The horizontal phase speed, c_x is simply:

$$c_x = \frac{\omega}{k},$$

while the vertical phase speed is:

$$c_z = \frac{\omega}{m}.$$

For the former, then, we can see that the horizontal propagation speed of a gravity wave is:

$$c_x = \bar{u} \pm \frac{N}{\sqrt{k^2 + m^2}}. \tag{2.38}$$

This is interpreted as follows: A parcel is displaced and disturbs the environment. The environment adjusts to the disturbance by issuing gravity waves that propagate rightward and leftward relative to the disturbance, with phase speed c_x. The disturbance is embedded in a flow of speed $\bar{u}$ relative to the ground, and so the ground-relative propagation speeds of the gravity wave are $c_x = \bar{u} + \frac{N}{\sqrt{k^2+m^2}}$ and $c_x = \bar{u} - \frac{N}{\sqrt{k^2+m^2}}$. The downstream moving wave speed is enhanced and the upstream wave speed is slowed. If the wave speed is equal and opposite to the flow speed, the gravity wave can be held stationary relative to the ground.

2.4.3 Sound and gravity waves, combined

Backing up a bit, consider again the environment's response to a vertically displaced parcel, which is expanding or contracting depending on which direction it was pushed. The environment responds in two ways: by locally compressing and expanding horizontally — which is accomplished by sound waves – and by oscillating upwards and downwards — which is accomplished by gravity waves. Both take place simultaneously, but the sound wave adjustment is far faster than the gravity wave equilibration.

One way this can be seen by comparing the sound and gravity wave phase speeds, for typical conditions. We saw earlier that the sound speed was about 331 m s^{-1} in an isothermal atmosphere. It doesn't vary much from this value, though. If we presume standard tropospheric conditions, the Brunt-Väisälä frequency N is about 0.01 s^{-1}. If we take $m = 0$ in the gravity wave equation, and assume the wave has a horizontal wavelength of about 10 km or so (reasonable assumptions), then the gravity wave phase speed is about 16 m s^{-1}, far, far slower.

The sound wave adjustment really is just the atmosphere's ability to avoid having internal "holes" develop in it. Physically, that is important, but practically it has nothing to do with the phenomena we usually wish to observe and/or simulate. Instead, we can presume the atmosphere's very good at its job and that the acoustic adjustment is virtually instantaneous. This is always true unless we are considering flow speeds that are close to the speed of sound; i.e., supersonic or nearly supersonic flows.

Consider an air parcel traveling at about 10 m s^{-1}. As it moves, it is "assaulting" neighboring parcels, which have to adjust to its presence. They will compress or expand, and the signal speed associated with this adjustment is very fast — it travels at the speed of sound. Unless the parcel is also traveling close to the speed of sound, the adjustment is so fast, and of such a small amplitude, that we can just take it for granted. (The amplitude is small because the adjustment is rapid; the "slackness" just can't build up under these conditions.) The parcel also causes the environment to adjust gravitationally, through gravity waves. Since the gravity wave phase speed is close in magnitude to the flow speed, however, the process is relatively slow — and significantly affected by the flow itself. Thus, gravity waves may have large amplitudes and cannot be neglected in this example.

The problem is, as we shall see, the efficiency of our numerical model is determined by the speed of the fastest moving signal supported by the equations. To keep the numerical integration stable, a time step has to be chosen based on this fastest signal speed. Sound waves are unimportant to the phenomena we wish to simulate, but since they move so quickly, they are the waves that determine the model time step. This makes solving the equations as we've expressed them so far terribly inefficient. We'll have to do something about this, and what we do will be based on the simple fact that sound and gravity wave adjustments are separate, of very different time scales, and thus don't interact very much.

Chapter 3

Derivation of the fully compressible model framework

In this chapter, we develop the basic equations of the fully compressible model, still assuming an adiabatic, dry, nondiffusive and frictionless atmosphere on a flat, nonrotating planet. However, to make the addition of moisture easier, we'll carry virtual temperature even though the moisture content of this atmosphere is assumed to be zero. We will start from scratch using base state environment assumptions and approximations appropriate for a cloud (convection-scale) model, though we will make use of analyses previously presented.

We start with these equations:

$$\frac{du}{dt} = -\frac{1}{\rho}\frac{\partial p}{\partial x} \tag{3.1}$$

$$\frac{dw}{dt} = -\frac{1}{\rho}\frac{\partial p}{\partial z} - g \tag{3.2}$$

$$\frac{d\theta}{dt} = 0 \tag{3.3}$$

$$\frac{d\rho}{dt} = -\rho\nabla\cdot\vec{V} \tag{3.4}$$

$$p = \rho R_d T_v \tag{3.5}$$

along with the definition of nondimensional pressure

$$\pi = \left[\frac{p}{p_0}\right]^{\frac{R_d}{c_{pd}}}. \tag{3.6}$$

There are four tasks to perform:

- Get the pressure gradient accelerations in terms of nondimensional pressure π.

- Perform a perturbation analysis of the pressure accelerations about a reasonable model mean state.
- Demonstrate the advantages of nondimensional over dimensional pressure.
- Replace the continuity equation with a pressure tendency equation.

Finally, the fully compressible model equations as we'll solve them are presented.

3.1 The pressure gradient acceleration terms

First, expand (3.6) with logs and work it into an equation relating the vertical derivatives of dimensional and nondimensional pressure:

$$\begin{aligned} \frac{c_{pd}}{R_d} \ln \pi &= \ln p - \ln p_0 \\ \frac{\partial \ln p}{\partial z} &= \frac{c_{pd}}{R_d} \frac{\partial \ln \pi}{\partial z}. \end{aligned} \tag{3.7}$$

Now we work on the right hand side (RHS) of (3.2). Using the ideal gas law (3.5), we can write this part of the equation as:

$$-R_d T_v \frac{\partial \ln p}{\partial z} - g.$$

Using (3.7) and recalling that $T = \theta\pi$ so that $T_v = \theta_v \pi$, we obtain:

$$\frac{dw}{dt} = -c_{pd}\theta_v \frac{\partial \pi}{\partial z} - g. \tag{3.8}$$

Similarly, (3.1) becomes

$$\frac{du}{dt} = -c_{pd}\theta_v \frac{\partial \pi}{\partial x}, \tag{3.9}$$

and our pressure gradient acceleration terms now employ π.

3.2 Perturbation analysis applied to the pressure accelerations

We now assume the mean state is solely a function of height, so that

$$\pi(x, z, t) = \bar{\pi}(z) + \pi'(x, z, t),$$

etc., and further that the mean state is hydrostatically balanced, which means:

$$\frac{d\bar{\pi}}{dz} = -\frac{g}{c_{pd}\bar{\theta}_v}. \tag{3.10}$$

Perform the perturbation analysis on (3.8), remembering to eliminate products of perturbations as they appear. This yields:

$$\begin{aligned}\frac{dw}{dt} &= -c_{pd}(\bar{\theta}_v + \theta'_v)\frac{\partial}{\partial z}(\bar{\pi} + \pi') - g \\ &= -c_{pd}\bar{\theta}_v\frac{d\bar{\pi}}{dz} - c_{pd}\theta'_v\frac{d\bar{\pi}}{dz} - c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial z} - g \\ &= -c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial z} + g\frac{\theta'_v}{\bar{\theta}_v}. \end{aligned} \tag{3.11}$$

For the x-direction, we wind up with:

$$\frac{du}{dt} = -c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial x}. \tag{3.12}$$

It is noted here that Lipps and Hemler (1982) advocated rewriting (3.11) to put the mean virtual potential temperature inside the vertical derivative; i.e.,

$$\frac{dw}{dt} = -c_{pd}\frac{\partial\bar{\theta}_v\pi'}{\partial z} + g\frac{\theta'_v}{\bar{\theta}_v}.$$

This was done to improve energy conservation, and could be justified by the fact that the vertical variation of the virtual potential temperature is relatively small. Durran (1989) discusses why this works (p. 1459).

3.3 Advantage of π over p

In our discussion of gravity waves in Chapter 2, our perturbation analysis yielded:

$$\frac{dw}{dt} = -\frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} - g\frac{\rho'}{\bar{\rho}}, \tag{3.13}$$

and we saw we could further expand the buoyancy term as:

$$-g\frac{\rho'}{\bar{\rho}} = -\frac{g}{\gamma}\frac{p'}{\bar{p}} + g\frac{\theta'_v}{\bar{\theta}_v} \tag{3.14}$$

$$= -\frac{g}{\bar{c_s}^2}\frac{p'}{\bar{\rho}} + g\frac{\theta'_v}{\bar{\theta}_v} \tag{3.15}$$

so that our vertical equation of motion could also be written as:

$$\frac{dw}{dt} = -\frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} - \frac{g}{\bar{c_s}^2}\frac{p'}{\bar{\rho}} + g\frac{\theta'_v}{\bar{\theta}_v}. \tag{3.16}$$

There are a few changes in the above equations from Chapter 2. Virtual temperature has been used in place of temperature, and the mean density has been assumed to be a function of height in the equations above. The sound speed is still

$$\bar{c_s}^2 = \frac{c_{pd}}{c_{vd}} R_d \bar{\theta}_v \bar{\pi} \tag{3.17}$$

but the use of the overbar symbol acknowledges it depends solely on base state values.

Note that the dimensional pressure perturbation p' shows up not only in the pressure gradient acceleration term, but also in the buoyancy term. This is a problem because, as we shall see, it is actually quite difficult to come up with exact predictions of p' (or π', for that matter) in the model. Instead, we get pressure perturbations only to within an unknown, arbitrary constant. That doesn't matter if only pressure *gradients* are required, since then the constant becomes irrelevant. Recall that by invoking the mechanical equilibrium assumption, we could eliminate the pressure perturbation from the buoyancy term, and we finally ended up with:

$$\frac{dw}{dt} = -\frac{1}{\bar{\rho}} \frac{\partial p'}{\partial z} + g \frac{\theta_v'}{\bar{\theta}_v}. \tag{3.18}$$

Ostensibly, (3.11) is the counterpart of (3.18), based on their similar appearance. However, we can show that (3.11) is at the same level of approximation as (3.16), which means we avoided the arguable mechanical equilibrium assumption. Note, then, that using π' in place of p' allows us to eliminate the presence of the pressure perturbation from the buoyancy term without having to resort to the mechanical equilibrium assumption. This is the advantage of using the nondimensional pressure, since now not having unique values of pressure perturbation doesn't cost us so much in terms of accuracy.

The balance of this section is devoted to proving that (3.11) and (3.16) are at the same level of approximation. If we take the defintion of π, (3.6), take logs, apply the perturbation method, employ the $\ln(1+x) \approx x$ approximation and then remove the base state (all things we've done before), we can produce this approximation:

$$\frac{\pi'}{\bar{\pi}} \approx \frac{R_d}{c_{pd}} \frac{p'}{\bar{p}}.$$

By further using the ideal gas law, and recognizing that $\bar{T_v} = \bar{\theta}_v \bar{\pi}$, we generate

$$\pi' = \frac{p'}{c_{pd} \bar{\rho} \bar{\theta}_v}.$$

Now, take the vertical derivative of this expression, which first gives us:

$$c_{pd} \frac{\partial \pi'}{\partial z} = \left[\frac{\bar{\rho} \bar{\theta}_v \frac{\partial p'}{\partial z} - p' \frac{d \bar{\rho} \bar{\theta}_v}{dz}}{\bar{\rho}^2 \bar{\theta}_v^2} \right]$$

Multiplying through by $\bar{\theta}_v$ and expanding the vertical derivative involving density and potential temperature yields:

$$c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial z} = \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} - \frac{p'}{\bar{\rho}}\left[\frac{d\ln\bar{\theta}_v}{dz} + \frac{d\ln\bar{\rho}}{dz}\right]. \tag{3.19}$$

Note the left hand side (LHS) is the pressure acceleration term in (3.11). So this term is not equal to the dimensional vertical pressure acceleration alone; there is an extra term, which looks like something from the buoyancy term in (3.14) prior to the application of the mechanical equilibration assumption.

To see that this surmise is indeed correct, we set out to manipulate that extra term. Starting with the ideal gas law for the mean state, which can also be cast as:

$$\bar{p} = \bar{\rho}R_d\bar{\pi}\bar{\theta}_v$$

and first taking logs and then differentiating with respect to height, we find:

$$\frac{d\ln\bar{p}}{dz} = \frac{d\ln\bar{\rho}}{dz} + \frac{d\ln\bar{\pi}}{dz} + \frac{d\ln\bar{\theta}_v}{dz}.$$

Replace the two vertical pressure gradient terms using the hydrostatic equation forms involving $\bar{p}$ and $\bar{\pi}$. Manipulate what remains into something that can replace the extra term in (3.19). First, we have:

$$\frac{d\ln\bar{\theta}_v}{dz} + \frac{d\ln\bar{\rho}}{dz} = \frac{1}{\bar{\pi}}\frac{g}{c_{pd}\bar{\theta}_v} - \frac{\bar{\rho}g}{\bar{p}}$$

and finally we wind up with

$$\frac{d\ln\bar{\theta}_v}{dz} + \frac{d\ln\bar{\rho}}{dz} = -\frac{g}{\bar{c_s}^2}. \tag{3.20}$$

Subbing this into (3.19), we see

$$c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial z} = \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} + \frac{g}{\bar{c_s}^2}\frac{p'}{\bar{\rho}}.$$

Therefore, the entire RHS in (3.11) is:

$$-c_{pd}\bar{\theta}_v\frac{\partial\pi'}{\partial z} + g\frac{\theta_v'}{\bar{\theta}_v} = -\frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} - \frac{g}{\bar{c_s}^2}\frac{p'}{\bar{\rho}} + g\frac{\theta_v'}{\bar{\theta}_v}. \tag{3.21}$$

Thus, it is seen that the right hand sides of (3.11) and (3.16) are at the same level of approximation.

3.4 Converting the continuity equation into a pressure tendency equation

3.4.1 Derivation

As mentioned before, we do not need prognostic equations for both density and pressure, owing to the ideal gas law. Because perturbation pressure appears in important terms in the model, and because we've already replaced the buoyancy term's perturbation density with θ_v', it is convenient to replace our prognostic equation for density [the continuity equation, (3.4)] with an equation for the nondimensional pressure tendency.

Using the ideal gas law we can manipulate (3.6) into:

$$\pi^{\frac{c_{vd}}{R_d}} = \frac{\rho R_d \theta_v}{p_0}. \tag{3.22}$$

Take the substantial derivative of this expression and manipulate it into this form:

$$\begin{aligned} \frac{c_{vd}}{R_d}\pi^{\frac{c_{vd}}{R_d}-1}\frac{d\pi}{dt} &= \frac{\rho R_d \theta_v}{p_0}\left[\frac{1}{\rho}\frac{d\rho}{dt} + \frac{1}{\theta_v}\frac{d\theta_v}{dt}\right] \\ &= \pi^{\frac{c_{vd}}{R_d}}\left[\frac{1}{\rho}\frac{d\rho}{dt} + \frac{1}{\theta_v}\frac{d\theta_v}{dt}\right] \end{aligned} \tag{3.23}$$

where the latter expression was obtained through usage of (3.22). Divide through by $\pi^{\frac{c_{vd}}{R_d}}$ and rearranging a little yields this expression:

$$\frac{d\pi}{dt} = \frac{R_d \pi}{c_{vd}}\left[\frac{1}{\rho}\frac{d\rho}{dt} + \frac{1}{\theta_v}\frac{d\theta_v}{dt}\right] \tag{3.24}$$

At this point, use the continuity equation (3.4) to replace the density derivative on the RHS. Recalling that the adiabatic speed of sound is $c_s^2 = \frac{c_{pd}}{c_{vd}} R_d \theta_v \pi$, we see that

$$\frac{R_d \pi}{c_{vd}} = \frac{c_s^2}{c_{pd}\theta_v}. \tag{3.25}$$

Using (3.25), we have:

$$\frac{d\pi}{dt} = -\frac{R_d \pi}{c_{vd}}\nabla \cdot \vec{V} + \frac{c_s^2}{c_{pd}\theta_v}\frac{d\theta_v}{dt}. \tag{3.26}$$

Now subject (3.26) to a perturbation analysis, recalling that the mean state is a function of height alone, except we're only going to (explicitly) expand π for now. The LHS of (3.26)

becomes:

$$\begin{aligned}\frac{d\pi}{dt} &= \frac{\partial\pi}{\partial t} + u\frac{\partial\pi}{\partial x} + w\frac{\partial\pi}{\partial z} \\ &= \frac{\partial\pi'}{\partial t} + u\frac{\partial\pi'}{\partial x} + w\frac{\partial\pi'}{\partial z} + w\frac{d\bar{\pi}}{dz} \\ &= \frac{\partial\pi'}{\partial t} + \vec{V}\cdot\nabla\pi' + w\frac{d\bar{\pi}}{dz}. \end{aligned} \tag{3.27}$$

The RHS of (3.26) becomes:

$$-\frac{R_d\bar{\pi}}{c_{vd}}\nabla\cdot\vec{V} - \frac{R_d\pi'}{c_{vd}}\nabla\cdot\vec{V} + \frac{c_s^2}{c_{pd}\theta_v^2}\frac{d\theta_v}{dt}. \tag{3.28}$$

One term from each side may be combined in the following fashion:

$$\left[-\frac{R_d\bar{\pi}}{c_{vd}}\nabla\cdot\vec{V} - w\frac{d\bar{\pi}}{dz}\right] \Rightarrow -\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}\left[\nabla\cdot\bar{\rho}\bar{\theta}_v\vec{V}\right], \tag{3.29}$$

which accomplishes an important simplification. Note the mean state sound speed defined in (3.17) has been employed.

It is easier to show (3.29) is true by going backwards. First, apply the vector chain rule to the RHS of (3.29):

$$\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}\left[\bar{\rho}\bar{\theta}_v\nabla\cdot\vec{V} + \vec{V}\cdot\nabla\bar{\rho}\bar{\theta}_v\right]$$

Recognizing the mean fields vary only vertically, we obtain:

$$\frac{\bar{c_s}^2}{c_{pd}\bar{\theta}_v}\nabla\cdot\vec{V} + \frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}w\frac{d\bar{\rho}\bar{\theta}_v}{dz}$$

Use (3.25) and split the vertical derivative in the second term to yield:

$$\underbrace{\frac{R_d\bar{\pi}}{c_{vd}}\nabla\cdot\vec{V}}_{A} + \frac{R_d\bar{\pi}}{c_{vd}\bar{\rho}\bar{\theta}_v}w\left[\bar{\rho}\frac{d\bar{\theta}_v}{dz} + \bar{\theta}_v\frac{d\bar{\rho}}{dz}\right]$$

which can be manipulated into:

$$A + \frac{R_d\bar{\pi}}{c_{vd}}w\left[\frac{d\ln\bar{\theta}_v}{dz} + \frac{d\ln\bar{\rho}}{dz}\right]. \tag{3.30}$$

Now, the term in square brackets has been seen before, in (3.20). Use (3.20) to replace the density derivative in (3.30) above. We should find that the two terms with the vertical temperature derivatives exactly cancel, leaving us with:

$$A - \frac{R_d\bar{\pi}}{c_{vd}}w\frac{g}{\bar{c_s}^2}$$

Take this expression, the hydrostatic equation (3.10) and the base state sound speed (3.17), and we finally end up with

$$-\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}\left[\nabla\cdot\bar{\rho}\bar{\theta}_v\vec{V}\right]=-\frac{R_d\bar{\pi}}{c_{vd}}\nabla\cdot\vec{V}-w\frac{d\bar{\pi}}{dz},$$

which concludes the proof.

3.4.2 Interpretation

This is the pressure tendency equation we have derived:

$$\frac{\partial\pi'}{\partial t}=\underbrace{-\vec{V}\cdot\nabla\pi'}_{[1]}\underbrace{-\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}\left[\nabla\cdot\bar{\rho}\bar{\theta}_v\vec{V}\right]}_{[2]}-\frac{R_d\pi'}{c_{vd}}\nabla\cdot\vec{V}+\underbrace{\frac{c_s^2}{c_{pd}\theta_v^2}\frac{d\theta_v}{dt}}_{[3]}. \tag{3.31}$$

The left side, of course, represents the local perturbation pressure rate of change at a single location. Three terms on the RHS have been highlighted. These terms influence the local pressure tendency through:

Term 1 *Advection of pressure.* "The future pressure here depends on the present pressure there". The local pressure rises or falls depending on what the wind is carrying with it. Advection travels with the flow velocity, with a time scale given by $\frac{L}{V}$, where L is some measure of horizontal length (such as the grid spacing) and V is the flow speed. Since $V\approx 10$ m s^{-1} or so, pressure advection is not rapid.

Term 2 *Acoustic adjustment.* "The future pressure here depends on the present wind field here (the dynamics)". The local pressure rises or falls depending (primarily) on the local mass convergence. This signal travels at the sound wave speed, so its time scale is $\frac{L}{c_s}$ (i.e., quite short).

Term 3 *Diabatic heating.* "The future pressure here depends on the thermodynamics."

Rewrite (3.31), incorporating all terms on the RHS except term [2] into catch-all proxy f_π, and we have the equation in the form we plan to use in the model:

$$\frac{\partial\pi'}{\partial t}+\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2}\left[\nabla\cdot\bar{\rho}\bar{\theta}_v\vec{V}\right]=f_\pi. \tag{3.32}$$

In their 3D model, Klemp and Wilhelmson (1978, *J. Atmos. Sci.*, p. 1070) argued that the terms embodied in f_π appear to have little effect on important physical processes operating in convection, and thus set the term to zero. (They remarked that this would be formally justified, but the paper they referred to never appeared — at least to my knowledge.) The

chief consequence of neglecting f_π appears to be that unique values of pressure are no longer obtained; instead, the field is predicted only to within an unknown constant. Since we have successfully avoided having to use raw values of π', this is not a concern. We will follow Klemp and Wilhelmson and set f_π to zero.

3.5 The fully compressible model equations

These are the equations we've developed thusfar (after performing a compatible perturbation analysis on (3.3):

$$\frac{\partial u}{\partial t} = -\vec{V} \cdot \nabla u - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} \tag{3.33}$$

$$\frac{\partial w}{\partial t} = -\vec{V} \cdot \nabla w - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + g\frac{\theta'_v}{\bar{\theta}_v} \tag{3.34}$$

$$\frac{\partial \theta'}{\partial t} = -\vec{V} \cdot \nabla \theta' - w\frac{d\bar{\theta}}{dz} \tag{3.35}$$

$$\frac{\partial \pi'}{\partial t} = -\frac{\bar{c_s}^2}{\bar{\rho} c_{pd} \bar{\theta}_v^2} \left[\nabla \cdot \bar{\rho}\bar{\theta}_v \vec{V} \right] \tag{3.36}$$

This is the fully compressible model framework we will use for our model. While this framework represents a convenient and relatively easily programmed equation set, it should be noted it is not optimal from an energy and mass conservation standpoint, especially when moisture is included. Bryan and Fritsch (2002, MWR) discuss improvements to this set of equations.

Chapter 4

Numerical solution of partial differential equations, Part I: The basics and the upstream scheme

4.1 Introduction

Partial differential equations (PDEs) may be classified into three types, based on their form and character: *parabolic*, *hyperbolic* and *elliptic*. The first two will be illustrated with the aid of Fig. 1, which depicts initial and subsequent fields for an initially bell-shaped perturbation.

Parabolic equations have the basic form

$$\frac{\partial u}{\partial t} = K_x \frac{\partial^2 u}{\partial x^2} \tag{4.1}$$

(where K_x is a positive constant) and are characterized by diffusion. Figure 1 shows that after being subjected to diffusion for a long time, the perturbation will be completely smoothed away. During its amplitude decrease, the original "pile" of material itself did not propagate, though it did spread horizontally. In addition, the total quantity of material, given by the area under the curve, has been conserved. In contrast, hyperbolic equations, having the form

$$\frac{\partial u}{\partial t} = -c \frac{\partial u}{\partial x}, \tag{4.2}$$

are characterized by *advection*, translation without change of amplitude or shape (see Fig. 1). In (4.2), c is a constant propagation velocity, the speed at which the perturbation will translate (rightward for $c > 0$.)

Elliptic equations often have forms like

$$\nabla^2 u = F \tag{4.3}$$

and are often diagnostic relations (i.e., no time derivatives appear). An example of an elliptic equation in dynamic meteorology is the relationship between streamfunction and vorticity. Elliptic equations are straightforward to solve if u is known and F is desired, but what if we have F and need u? In that case, it can be difficult, if not impossible, to obtain exact values of u given the F field, since having F does not inform us directly about u, or even the gradient of u, but rather its second derivatives. Because of this, u anywhere depends (to a certain degree) on F *everywhere*, and thus F *anywhere* affects u everywhere, again to a degree.

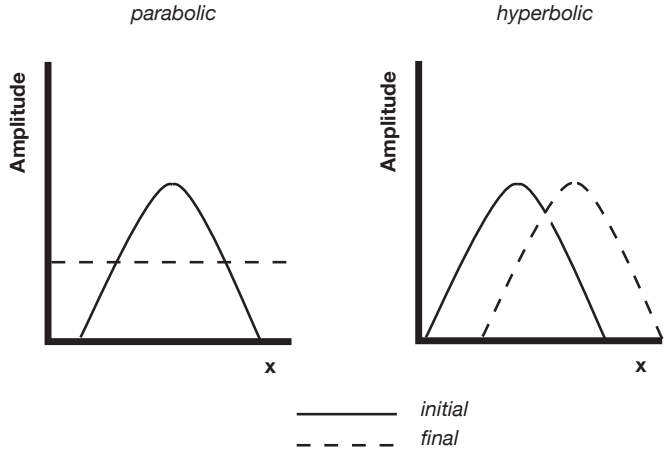

Figure 4.1: Parabolic and hyperbolic (presuming $c > 0$) type solutions.

Our set of equations are a mixed set of hyperbolic and parabolic equations, but there is an elliptic equation in the mix as well if the anelastic or incompressible approximations are made. Even if the problem being simulated is truly inviscid, however, the numerical solution may not be. The numerical scheme employed may introduce an artificial source of smoothing into the solution, or we may find the need of intentionally introduce artificial diffusion in order to keep the solution stable.

In this chapter, we examine the finite difference (FD) approximation to the simple one-dimensional (1D) inviscid advection equation (4.2), which we'll rewrite more simply as

$$u_t + cu_x = 0 \tag{4.4}$$

We appreciate that in discretizing space and time, creating a grid spacing Δx and a time step Δt, we introduce errors that should increase as Δx and Δt get larger. We have to hope, however, that as Δx and Δt go to zero that our numerical solution converges on the correct solution. The Lax-Richtmyer theorem tells us that such convergence is assured if the numerical scheme is both *consistent* and *stable*. Consistency means we are solving the correct problem; stability means the solution will not grow unbounded with time.

4.2 Consistency

At least initially, we'll examine consistency and stability using the *upstream*, or upwind scheme as our practical example, not because the upstream scheme is good (actually, it's pretty bad) but owing to its simplicity and pedagogical value. The upstream approximation to (4.4) is:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\left[\frac{u_j^n - u_{j-1}^n}{\Delta x}\right] = 0, \tag{4.5}$$

where c must be nonnegative in this example so the wave is moving from left to right. In the above, u_j^n represents the variable value at grid point j at time n, which we take to be the current spatial location and the present time.

The upstream scheme is forward-in-time and upstream-in-space. The time derivative is computed from the present [time n] to the future [time $n+1$], while the spatial derivative is computed between here [grid point j] upstream to there [grid point $j-1$], where the wave is coming from. If c changes sign, the spatial derivative will have to be revised so that the derivative is computed upstream. The upstream scheme is "explicit" because (4.5) may be rewritten leaving the only unknown value — u_j^{n+1} — on the LHS. The rest are known.

Is (4.5) a consistent approximation to (4.4)? We assess this with Taylor series expansions, and the point to be expanded about is the current grid point value at the present time; i.e., u_j^n. Therefore, we have:

$$u_j^{n+1} = u_j^n + u_t\Delta t + u_{tt}\frac{(\Delta t)^2}{2} + u_{ttt}\frac{(\Delta t)^3}{3!} + \cdots,$$

and

$$u_{j-1}^n = u_j^n - u_x\Delta x + u_{xx}\frac{(\Delta x)^2}{2} - u_{xxx}\frac{(\Delta x)^3}{3!} + \cdots.$$

(How do you know how many terms to keep in the Taylor series? Generally you don't know for certain, so keep enough terms that you don't run out later when you truncate.)

Substitute these expansions into (4.5), then rearrange in such a way as to reproduce the LHS of (4.4) so we can see what's left over. We have

$$\frac{1}{\Delta t}\left[u_j^n + u_t\Delta t + u_{tt}\frac{(\Delta t)^2}{2} + u_{ttt}\frac{(\Delta t)^3}{3!} - u_j^n\right] + \frac{c}{\Delta x}\left[u_j^n - u_x\Delta x + u_{xx}\frac{(\Delta x)^2}{2} - u_{xxx}\frac{(\Delta x)^3}{3!} - u_j^n\right].$$

Simplify and distribute the Δt and Δx, moving the terms resembling the original PDE to the LHS. We end up with:

$$u_t + cu_x = \underbrace{-\frac{\Delta t}{2}u_{tt} + \frac{c\Delta x}{2}u_{xx} - \frac{(\Delta t)^2}{3!}u_{ttt} - \frac{c(\Delta x)^2}{3!}u_{xxx} + O[(\Delta t)^3, (\Delta x)^3]}_{\tau}. \tag{4.6}$$

The remaining terms of the Taylor series had terms that contained the cubic or higher powers of the grid spacing and time step, and we concatenated them into the final term.

According to the original PDE, (4.4), **the RHS of (4.6) should be zero**, but this isn't likely to be true, at least for finite values of Δt and Δx. What remains there is *truncation error*, τ. The scheme is consistent, though, if and only if the truncation error vanishes as Δt and Δx go to zero. This is indeed the case for the upstream scheme.

For nonzero Δt and Δx, it is seen that we are NOT exactly solving (4.4), because the truncation error τ is nonzero. We can manipulate the truncation term to gain some physical insight on the problem we are actually (unavoidably, if not inadvertently) solving. First, perform $\frac{\partial}{\partial t}$(4.6) to yield:

$$u_{tt} + cu_{xt} = -\frac{\Delta t}{2}u_{ttt} + \frac{c\Delta x}{2}u_{xxt} - \frac{(\Delta t)^2}{3!}u_{tttt} - \frac{c(\Delta x)^2}{3!}u_{xxxt} + O[(\Delta t)^3, (\Delta x)^3].$$

Then take $-c\frac{\partial}{\partial x}$(4.6), producing

$$-cu_{tx} - c^2u_{xx} = +\frac{c\Delta t}{2}u_{ttx} - \frac{c^2\Delta x}{2}u_{xxx} + c\frac{(\Delta t)^2}{3!}u_{tttx} + \frac{c^2(\Delta x)^2}{3!}u_{xxxx} + O[(\Delta t)^3, (\Delta x)^3].$$

Add these last two expressions together and solve for u_{tt}:

$$u_{tt} = c^2u_{xx} + \Delta t\left[-\frac{u_{ttt}}{2} + \frac{c}{2}u_{ttx}\right] + \Delta x\left[\frac{c}{2}u_{xxt} - \frac{c^2}{2}u_{xxx}\right] + O[(\Delta t)^2, (\Delta x)^2]. \tag{4.7}$$

Similarly (skipping the gory details and explicitly keeping fewer terms), we can produce:

$$u_{ttt} = -c^3u_{xxx} + O[\Delta t, \Delta x] \tag{4.8}$$

$$u_{ttx} = c^2u_{xxx} + O[\Delta t, \Delta x] \tag{4.9}$$

$$u_{xxt} = -cu_{xxx} + O[\Delta t, \Delta x]. \tag{4.10}$$

Substitute (4.8-4.10) into (4.7) and plug the result into (4.6) to get a rewritten expression for the truncation error on the RHS:

$$u_t + cu_x = \frac{c\Delta x}{2}\left[1 - \frac{c\Delta t}{\Delta x}\right]u_{xx} + O[(\Delta t)^2, (\Delta x)^2] \tag{4.11}$$

(for convenience, only the leading term in τ has been explicitly retained). Again, it is seen that $\tau \to 0$ as Δt and $\Delta x \to 0$, so the scheme is consistent. More importantly, it is seen for nonzero τ, the equation we are actually solving numerically is

$$u_t + cu_x = f(u_{xx}).$$

The leading part of the truncation error is a diffusion term [compare to (4.1)]. Recall that the advection equation should move the initial disturbance along at constant speed

without change of shape. It is seen that the fatal flaw of the upstream scheme is that it contains large implicit diffusion, which is present for numerical reasons owing to the finite discretization. Integrated for a sufficiently long period, the upstream scheme will diffuse any initial perturbation away, which makes it useless for long-term integrations.

Figure 2 presents a qualitative look at the performance of the upstream scheme. A single sine wave has been advected to the right at constant speed c for some period of time. The "true (analytic) solution" has the wave having moved exactly at speed c without change in amplitude or shape. Owing to the extreme inherent diffusiveness of the upstream scheme, the amplitude of the wave has been greatly damped. However, the wave is in about the right place; that is, it has been advected at about the right speed. The figure also shows how an alternate method, the *leapfrog scheme* (to be introduced presently), would fare on this problem. Note the leapfrog solution has properly preserved the wave's original amplitude but tends to advect the wave a bit too slowly. Thus, this scheme has little *amplitude error*, but does have some degree of *phase error*.

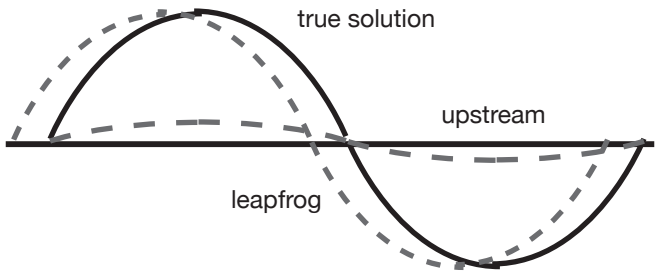

Figure 4.2: Upstream and leapfrog solutions compared to the true (analytic) solution.

4.3 Stability

4.3.1 The analytic solution

To assess a scheme's stability, we need to compare the scheme's solution to the analytic solution. In (4.4), we assume the analytic result has wave-like solutions of the form:

$$u(x,t) = Ae^{i(kx-\omega_a t)} \tag{4.12}$$

in the same manner as we did when we examined sound and gravity waves in Chapter 2. In (4.12), the subscript "a" denotes "analytic". Recall that, by definition, the phase speed is defined as

$$c_a = \frac{\omega_a}{k}. \tag{4.13}$$

Unlike any previous situation, we recognize that the frequency in (4.12) may be complex. In that event, the real part of the frequency determines the phase speed; i.e., $c_a = \frac{\omega_{aR}}{k}$.

Substituting (4.12) into (4.4) yields the analytic frequency equation

$$-i\omega_a + cik = 0.$$

As c and the horizontal wave number k are real numbers, it is seen that the analytic frequency ω_a is purely real. This expression can be rearranged to show that $c = \frac{\omega_a}{k}$. Since according to (4.13), the RHS of this expression is the analytic phase speed, we see that for the analytic solution, c_a and the advection speed c from (4.4) are the same. That is, in the analytic solution, the wave translated to the right at speed c. Note further that the propagation speed is independent of the wavelength because k cancels out (leaving $c_a = c$).

Equation (4.12) holds at a particular time t, representing the present time. What will the solution look like at a future time, $t+n\Delta t$, being n time steps in advance of the present time? We want this future solution expressed in terms of the present time's solution. Substitute $t + n\Delta t$ for t in (4.12), and we find:

$$\begin{aligned} u(x, t + n\Delta t) &= Ae^{i(kx-\omega_a[t+n\Delta t])} \\ &= Ae^{i(kx-\omega_a t)}e^{-i\omega_a n\Delta t}. \end{aligned}$$

Using (4.12) again and defining the *change function* for the analytic solution, λ_a, as:

$$\lambda_a = e^{-i\omega_a \Delta t} \tag{4.14}$$

then the future solution is related to the present one by:

$$u(x, t + n\Delta t) = u(x, t)\lambda_a^n. \tag{4.15}$$

Consider the analytic frequency ω_a. We've already shown the analytic frequency is purely real. However, in the more general case, the frequency is potentially a complex number. The true solution can translate and its amplitude can change as a function of time. Translation is determined by the real part of ω_a, while amplitude change (growth or decay) is represented by the imaginary part of ω_a. Because ω_a can be complex, λ_a can be also.

Recognizing that ω_a might be complex, we write it as:

$$\omega_a = \omega_{aR} + i\omega_{aI}.$$

Then

$$\lambda_a = e^{-i\omega_a \Delta t} = e^{-i\omega_{aR}\Delta t}e^{\omega_{aI}\Delta t}$$

Using Euler's relation to expand the RHS' first exponential yields:

$$\lambda_a = \underbrace{e^{\omega_{aI}\Delta t}\cos\omega_{aR}\Delta t}_{real\ part} - \underbrace{ie^{\omega_{aI}\Delta t}\sin\omega_{aR}\Delta t}_{imaginary\ part}. \tag{4.16}$$

We can write the change function λ_a in *polar form*; i.e.,

$$\lambda_a = |\lambda_a| e^{i\theta_a}$$

where

$$\begin{aligned} |\lambda_a| &= \sqrt{\lambda_{aR}^2 + \lambda_{aI}^2} \\ \theta_a &= \arctan \frac{\lambda_{aI}}{\lambda_{aR}}. \end{aligned}$$

The expression θ_a is termed the *analytic phase function*. Here,

$$\begin{aligned} |\lambda_a| &= \sqrt{e^{2\omega_{aI}\Delta t}(\cos^2 \omega_{aR}\Delta t + \sin^2 \omega_{aR}\Delta t)} \\ &= e^{\omega_{aI}\Delta t}, \end{aligned}$$

because the term under the square root sign is merely unity, and

$$\begin{aligned} \theta_a &= \arctan \left[\frac{-e^{\omega_{aI}\Delta t} \sin \omega_{aR}\Delta t}{e^{\omega_{aI}\Delta t} \cos \omega_{aR}\Delta t} \right] \\ &= \arctan \left[-\tan \omega_{aR}\Delta t \right] \\ &= \arctan \left[\tan(-\omega_{aR}\Delta t) \right] \\ &= -\omega_{aR}\Delta t. \end{aligned}$$

Therefore, the polar form of the change function may be written as:

$$\lambda_a = |\lambda_a| e^{-i\omega_{aR}\Delta t} = e^{\omega_{aI}\Delta t} e^{-i\omega_{aR}\Delta t}. \tag{4.17}$$

Now consider at this point (4.15) together with (4.17). We want to know how the future state of the solution relates to the present one. This is accomplished through the change function λ_a. But λ_a has two parts, as shown by (4.17). The exponential function with the real part of the analytic frequency, ω_{aR}, merely represents sinusoidal oscillation in space and translation in time at phase speed $c_a = \frac{\omega_{aR}}{k}$.

In contrast, the exponential with the imaginary frequency component, ω_{aI}, represents amplitude change as a function of time. If in fact $\omega_{aI} > 0$, then *exponential growth* occurs, and the physical situation is unstable. Thus, we term the absolute value (complex conjugate) of the change function, $|\lambda_a|$, as the *amplification factor*.

To see this, substitute (4.17) into (4.15). We find:

$$u(x, t + n\Delta t) = u(x, t)|\lambda_a|^n e^{-i\omega_{aR} n\Delta t}.$$

Note the amplification factor is raised to the power n, the number of time steps we have advanced subsequent to the present time. If this factor is greater than one, growth in time is swift.

Again, we've already showed that for the analytic solution, ω_a is purely real, so $\omega_{aI} = 0$ and thus $|\lambda_a| = 1$. The true solution merely translates at speed $c = c_a$ without change of amplitude. The analytic phase function, then, is simply

$$\theta_a = -\omega_{aR}\Delta t = -kc_a\Delta t. \tag{4.18}$$

The finite difference scheme's solution, however, may have a different, and complex, frequency. We examine the upstream scheme's frequency and phase function next.

4.3.2 The finite difference scheme

The finite difference (FD) analogue to the preceeding will be indicated with the subscript "*d*". The finite difference frequency, ω_d, may be complex, so:

$$\omega_d = \omega_{dR} + i\omega_{dI}.$$

The FD change function is

$$\lambda_d = e^{-i\omega_d\Delta t}$$

and may be complex, so:

$$\lambda_d = \lambda_{dR} + i\lambda_{dI}.$$

In polar form, the change function is:

$$\lambda_d = |\lambda_d|e^{i\theta_d}$$

where the FD amplification factor is $|\lambda_d| = e^{\omega_{dI}\Delta t}$ and the FD phase function is $\theta_d = \arctan\left[\frac{\lambda_{dI}}{\lambda_{dR}}\right]$. Completing the analogy, we write the FD phase function may be rewritten as:

$$\theta_d = \arctan\frac{\lambda_{dI}}{\lambda_{dR}} = -\omega_{dR}\Delta t = -kc_d\Delta t \tag{4.19}$$

where c_d is now the translation speed of the numerically simulated wave.

In the analytic solution, we found $|\lambda_a| = 1$. This is not necessarily true for the FD scheme's simulated wave. Thus, the case of $|\lambda_d| \neq 1$ represents FD *amplitude error*. Further, the FD and analytic phase speeds, c_d and c_a, are not necessarily the same. The case of $c_d \neq c_a$ represents FD *phase error*, which can be expressed by the ratio:

$$\frac{\theta_d}{\theta_a} = \frac{-kc_d\Delta t}{-kc_a\Delta t} = \frac{c_d}{c_a}. \tag{4.20}$$

Recall that for the analytic solution, c_a was independent of k, so all waves propagated at the same speed (c), irrespective of wavelength. In general, it will be seen that the phase error

in the FD solution is a function of wavelength — specifically, that the schemes translate longer waves better than shorter ones. Consider then an initial condition composed of a sum of waves of different wavelengths, which combine to produce a composite wave with a certain shape. In the analytic solution, all waves translate together at the same speed, so the composite shape is preserved. In the FD scheme, however, since different waves have different phase errors, the composite wave will start to "unravel" as it propagates, with the unavoidable result that its shape will not be preserved. Since, in general, different waves move at different speeds in the FD solution, the numerical result will also suffer from *dispersion error*.

We need to find expression for $|\lambda_d|$ and θ_d for the upstream scheme. We'll do this by employing the so-called "modified" von Neumann stability analysis. This technique is easier to apply than the "regular" von Neumann approach but is not as powerful, and can yield necessary conditions for stability but not sufficient conditions. For what we're going to examine, however, the regular and modified techniques yield the same results, so we might as well do it the easier way.

The upstream scheme FD approximation (4.5), rewritten in terms of the unknown future forecast u_j^{n+1}, is:

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{\Delta x}\left[u_j^n - u_{j-1}^n\right]. \tag{4.21}$$

Assume a wave-like solution representing the FD analogue of (4.12):

$$u_j^n = Ae^{i(kx-\omega_d t)}. \tag{4.22}$$

Note the frequency ω_d in the above. Equation (4.21) also contains u_j^{n+1} and u_{j-1}^n, which may be gotten from (4.22) by substituting $t+\Delta t$ for t and $x-\Delta x$ for x, respectively. Then, u_j^{n+1} is:

$$\begin{aligned} u_j^{n+1} &= \underbrace{Ae^{i(kx-\omega_d t)}}_{u_j^n}\underbrace{e^{-i\omega_d \Delta t}}_{\lambda_d} \\ &= u_j^n \lambda_d \end{aligned}$$

and the comparable expression for u_{j-1}^n is:

$$u_{j-1}^n = u_j^n e^{-ik\Delta x}.$$

Upon substituting these expressions into (4.21), one finds:

$$u_j^n \lambda_d = u_j^n - \frac{c\Delta t}{\Delta x}\left[u_j^n - u_j^n e^{-ik\Delta x}\right].$$

The common u_j^n cancels. Let $c' = \frac{c\Delta t}{\Delta x}$ for convenience, and note that owing to Euler's relation

$$e^{-ik\Delta x} = \cos k\Delta x - i \sin k\Delta x.$$

Therefore,

$$\lambda_d = \underbrace{1 - c' + c' \cos k\Delta x}_{\lambda_{dR}} \underbrace{-ic' \sin k\Delta x}_{\lambda_{dI}}. \tag{4.23}$$

The square of the FD amplification factor is given by:

$$|\lambda_d|^2 = \left[\lambda_{dR}^2 + \lambda_{dI}^2\right]$$

which using (4.23) above can be shown to be:

$$|\lambda_d|^2 = 1 + 2c'(1 - c')\left[\cos k\Delta x - 1\right]. \tag{4.24}$$

Now, if $|\lambda_d|^2 > 1$, the solution grows with time, and thus the situation is unstable. Thus, in order to avoid instability, we require $|\lambda_d|^2 < 1$, and (4.24) shows this to be conditioned upon:

$$c' \equiv \frac{c\Delta t}{\Delta x} \leq 1. \tag{4.25}$$

Equation (4.25) is often termed the Courant-Fredrichs-Lewy (CFL) criterion. Some physical insight can be gained from turning that expression upside down, producing

$$s = \frac{\Delta x}{c\Delta t}$$

This can be thought of as a *sampling rate*. $s = 1$ means the wave propagating at speed c will move one grid point Δx in one time step Δt. Our analysis shows that sampling too slowly (small s or large c') yields instability; the solution blows up. Intuition tells you that a higher sampling rate (larger s or smaller c') should be an improvement — the more rapidly you sample something, the more smoothly it appears to move between samples. Actually, it happens that in the FD world, sampling more frequently isn't always a good idea, and this will now be seen to be true for the upstream scheme, at least.

Recall that the upstream scheme's fatal flaw is the presence of implicit diffusion in the scheme (recall Fig. 2). This diffusion can be seen in $|\lambda_d|$. From (4.24), it is seen that if $c' = 1$, then $|\lambda_d| = 1$, and there is no amplitude error. The solution neither grows nor decays; the FD scheme does what it is supposed to, at least with respect to the amplitude. However, if a higher sampling rate is chosen, such that $c' < 1$, then (4.24) shows that $|\lambda_d| < 1$ and the solution decays with time.

Normally, the smaller your time step, the more expensive the computations. Thus, you typically want to adopt the largest Δt that yields a stable solution. With the upstream scheme, this turns out to be the right thing to do, because choosing the largest stable time step helps alleviate the implicit diffusion problem. This is ostensibly easy to do in this sample problem; you know c. However, in practice the advection problems one encounters

are multi-dimensional, having non-constant flow speeds and nonlinear advection terms. In that event, choosing the "right" time step for a given grid spacing is quite impossible.

For $c' \neq 1$, the amplitude will either grow or decay for all waves, because $|\lambda_d| \neq 1$. However, all waves are not affected equally. Due to the $\cos k\Delta x$ term in (4.24), $|\lambda_d|$ varies with wavenumber k, even for the same c'. It turns out that the shortest waves always handled worst. Figure 3 (left side) shows the amplitude error $|\lambda_d|$ as a function of wavelength and normalized wavenumber ($\frac{k\Delta x}{\pi}$, which varies between 0 and 1).

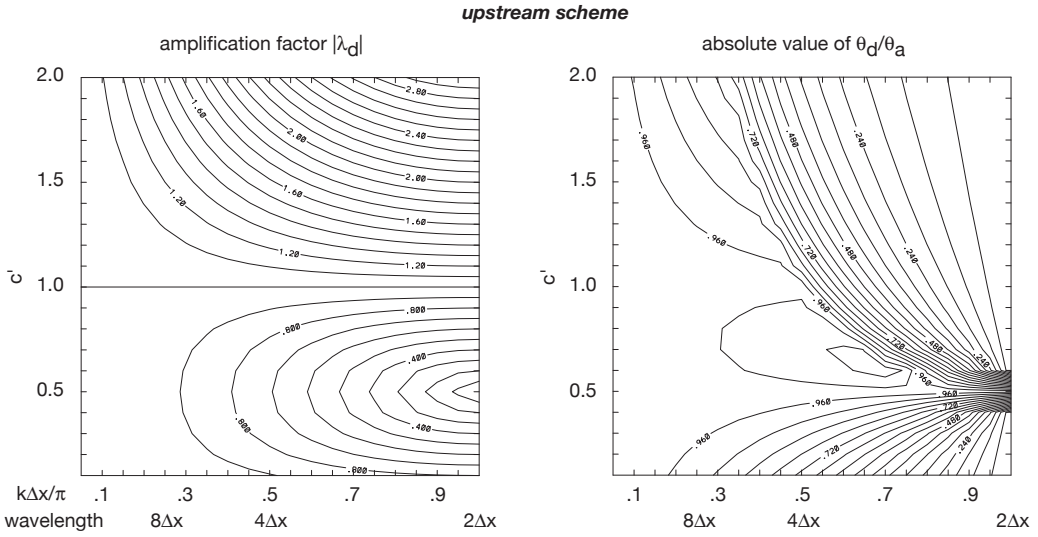

Figure 4.3: Amplitude error (left) and absolute phase error (right), plotted as a function of c' and wavelength and normalized wavenumber.

We must appreciate that the smallest resolvable wave has a wavelength of $L_x = 2\Delta x$. Since k is $\frac{2\pi}{L_x}$, $k\Delta x = \pi$ for the $2\Delta x$ wave and $\cos k\Delta x = \cos \pi =$ -1. In this situation, (4.24) shows that

$$|\lambda_d|^2 = 1 - 4c'(1 - c').$$

It is clear that if $c' > 1$, $|\lambda_d|^2 > 1$, and because

$$u(x, t + n\Delta t) = u(x, t)\lambda_d^n.$$

[the FD analogue of (4.15)], it is seen the amplitude will grow swiftly with time. Indeed, one should see that due to the presence of $\cos k\Delta x$ in the equation, it is the shortest waves that will amplify the fastest. For $c' < 1$, amplitudes will decrease because $|\lambda_d|^2 < 1$, but the shortest waves damp the fastest. Interestingly, the damping rate reaches a maximum for all wavelengths at $c' = 0.5$, and then diminishes for decreasing c'. At $c' = 0.5$, $|\lambda_d|^2 = 0$ for the $2\Delta x$ wave; it will vanish in just one time step.

Now we consider the phase error of the upstream scheme. Take (4.18) and, recalling that

for the analytic solution $c_a = c$, rewrite it as:

$$\theta_a = -k\Delta x \frac{c\Delta t}{\Delta x} = -c' k \Delta x,$$

where all we did was to first multiply and divide by Δx and then employ the definition of c'. The real and imaginary parts of the FD change function λ_d were revealed in (4.23); plug these into (4.19) to yield:

$$\theta_d = \arctan \frac{-c' \sin k\Delta x}{1 - c' + c' \cos k\Delta x}.$$

Finally, use these expressions to form the phase error ratio defined in (4.20):

$$\frac{\theta_d}{\theta_a} = \frac{\arctan \frac{-c' \sin k\Delta x}{1-c'+c' \cos k\Delta x}}{-c' k \Delta x}. \tag{4.26}$$

The phase error for the upstream scheme is shown on the right side of Fig. 3. It turns out that, in this aspect, the upstream scheme is imperfect even at $c' = 1$. At this value, waves with grid wavelengths $L_x < 4\Delta x$ are not advected properly. (For $c' = 1$, the advection speeds of these waves are too slow, though that can't be seen directly in the figure because the absolute value of the phase error is shown.) For $c' < 0.5$, all wavelengths are advected too slowly compared to the analytic solution. Again, this is contrary to to the intuition that higher sampling rates are better.

Furthermore, phase error is worst for the shortest waves. At any value of c', the smallest resolvable wave, $2\Delta x$, *does not move* relative to the grid at all. Recall that in the analytic solution, all waves move at the same speed c so the feature being advected retains its original shape. In the FD solution, the dependence of phase error on wavelength means that a feature composed of a superposition of waves with different wavelengths will unravel while being advected due to dispersion error.

The lesson to be learned from both sources of error is that the shortest waves tend to be mishandled, which implies: (a) do not invest much importance into these waves (owing to their poor handling); and (b) make sure you choose your grid spacing to properly resolve the phenomenon you are trying to advect. For example, if your primary wave of interest is 10 km wide, choose Δx on the order of 1 km, so your feature is $10\Delta x$.

Chapter 5

Numerical solution of partial differential equations, Part II: The leapfrog scheme

In Chapter 4, it was seen that the upstream scheme was a consistent approximation to the hyperbolic PDE

$$u_t + cu_x = 0, \tag{5.1}$$

but it suffered from considerable amplitude error, making it essentially useless in practice. A superior alternative is the leapfrog method, in which the time and space derivatives are replaced with centered approximations. One version of the leapfrog scheme applied to (5.1) is

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} = -c\left[\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right]. \tag{5.2}$$

Note that this scheme, too, is explicit and involves three time levels (n, $n+1$ and $n-1$) and three points in space (j, $j+1$ and $j-1$). The time and space derivatives are centered around the here/now point, u_j^n, and are computed over intervals of $2\Delta t$ and $2\Delta x$. As a result, note that although we wish to create a forecast at the present point at time $n+1$ (i.e., u_j^{n+1}), that forecast *does not involve* the present value at this point! That is to say, u_j^n does not appear in the equation — at least not in the advection term.

In finite differencing, it is typically found that centered schemes are superior in accuracy to one-sided schemes (like the upstream scheme). Generally, the more points that are included in the approximation of a derivative, the greater the accuracy, but whether the points are deployed in a centered or one-sided manner also matters. The order of accuracy is determined by the lowest power of the grid and time spacing in the truncation error term. In equation (4.6) of Chapter 4, it was seen that the lowest power of Δt and Δx for the upstream method was one. Thus, that scheme was first-order accurate. It will be seen that the truncation

term for (5.2) has powers of two for both Δt and Δx, so it is second-order accurate for both time and space, even though it also explicitly employs only two points in each derivative. Other factors being equal, the second-order approximation would be superior to a first-order scheme because as Δt and/or Δx go to zero, Δt^2 and Δx^2 do so faster.

Although often applied to the entire approximation shown in (5.2), the term "leapfrog" actually refers only to the way the *time* differencing is handled. Our present value of the variable being predicted is u_j^n. To make a forecast for time $n+1$, we're actually backtracking to time $n-1$ and then *leaping over* time n all the way to time $n+1$. When we wish to forecast for time $n+2$, we will start at time n and leapfrog over time $n+1$. See Fig. 5.1.

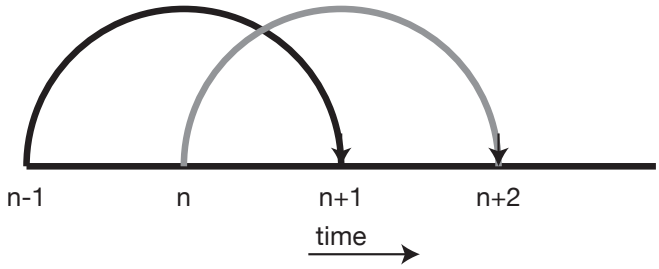

Figure 5.1: "Leapfrog" time stepping.

By using additional points, it is possible to construct still higher order approximations to the space and/or time derivatives, producing an even higher order version of (5.2). While this is often done for the spatial derivative, it is very rarely attempted with the leapfrog time term, for reasons that will become readily apparent! (Higher accuracy in time while formally retaining two time levels is possible with Runge-Kutta schemes; see Model Task 0B.) A second-order time but fourth-order space leapfrog scheme would look like this:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} = c\left[\frac{-u_{j+2}^n + u_{j-2}^n + 8u_{j+1}^n - 8u_{j-1}^n}{12\Delta x}\right].$$

This fourth-order spatial derivative requires four points, distributed symmetrically about the grid point the gradient is being computed for.

As the order of the approximation is increased, it becomes more challenging to implement these schemes, especially in the vicinity of boundaries. Near a boundary, the fourth-order approximation cannot be used, because points well outside the domain would be required, so the second-order scheme is employed there instead. In theory, using a less accurate scheme at any point within the domain degrades the accuracy of the entire solution, especially if the feature being simulated is close to the boundary. In practice, however, model domains are much wider than they are deep, and so the lateral boundaries are usually much farther from the features of interest than the upper and lower boundaries. Thus, in 2D or 3D problems, fourth-order spatial differencing, when employed, is usually restricted to the horizontal derivatives; the vertical gradients are computed using second-order approximations.

There is a significant downside to using leapfrog time integration. We saw that the analytic solution for (5.1) consisted of a single feature, translating at speed c, to the right if $c > 0$. There was only one solution, befitting the first degree derivative u_t. However, an n-level finite difference scheme has $n-1$ degrees of freedom and *there is one solution for each degree of freedom.* Only one of these convolved solutions can be physically realistic – the "physical mode" – the others are "computational modes".

The upstream scheme used a two time level approximation for u_t, and so it also yielded only one (generally not very good) answer. The leapfrog, in contrast, utilizes three time levels, and thereby results in *two solutions.* One is potentially the correct answer; the other is most definitely *wrong.* So why are we doing this?

We want to use higher order (and especially centered) approximations for spatial derivatives, because they are more accurate. Generally, they advect features better, especially the small scale waves. But, as we will demonstrate, enhancing the accuracy of the time integration can create artificial solutions that are not at all desirable if it is accomplished by including more time levels into the scheme. Is the solution to use something that is centered and at least second-order in space but with a two time level integrator like the upstream scheme? This would seem to be a reasonable compromise. Unfortunately, such schemes are not stable, so they are useless.

5.1 Consistency and stability of the second-order leapfrog scheme

We will now show the leapfrog scheme (5.2) is consistent and second-order. We need to take Taylor series expansions about the here/now point u_j^n. For the time derivative, this becomes:

$$\begin{aligned}
\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} &= \frac{1}{2\Delta t}\left[2\Delta t u_t + \frac{2}{3!}\Delta t^3 u_{ttt} + \cdots\right] \\
&= u_t + \frac{1}{3!}\Delta t^2 u_{ttt} \\
&= u_t + O(\Delta t^2).
\end{aligned}$$

Thus, our leapfrog time approximation is equivalent to the derivative we're trying to solve (u_t), plus truncation error that is proportional to the square of the time step. Note the truncation error goes to zero as Δt does, so the time derivative approximation is consistent. For the space approximation, we can use the same technique to show:

$$c\left[\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right] = cu_x + O(\Delta x^2),$$

showing the spatial approximation to also be second-order accurate and consistent.

Note that, unlike the upstream scheme, the leapfrog truncation error has derivatives with odd degrees (e.g, u_{xxx}) instead of even degrees. Even degree derivatives constitute diffusion processes, while odd degrees represent propagation. Thus, we anticipate the leapfrog scheme will have little or no amplitude error (at least while stable) but may have some problem with feature phase speeds.

To assess the second-order leapfrog scheme's stability and error characteristics, we'll again employ the modified von Neumann method. First, we write the scheme in explicit form:

$$u_j^{n+1} = u_j^{n-1} - \frac{c\Delta t}{\Delta x}\left[u_{j+1}^n - u_{j-1}^n\right]. \tag{5.3}$$

We again assume solutions of the form:

$$u_j^n = Ae^{i(kx-\omega_d t)}$$

Using previously employed definitions, we see that:

$$\begin{aligned} u_j^{n+1} &= u_j^n\lambda \\ u_j^{n-1} &= u_j^n\lambda^{-1} \\ u_{j+1}^n &= u_j^n e^{ik\Delta x} \\ u_{j-1}^n &= u_j^n e^{-ik\Delta x}, \end{aligned}$$

where the subscript "d" has been dropped from λ because it is clear we're talking about the FD approximation here.

Substituting these expressions into (5.3) yields the following equation:

$$\lambda = \lambda^{-1} - \frac{c\Delta t}{\Delta x}\left[e^{ik\Delta x} - e^{-ik\Delta x}\right].$$

Multiply through by λ and note that Euler's relation allows us to simplify the bracketed term in this way:

$$\begin{aligned} e^{ik\Delta x} - e^{-ik\Delta x} &= \cos k\Delta x + i\sin k\Delta x \\ &= \cos k\Delta x + i\sin k\Delta x - \cos k\Delta x + \sin k\Delta x \\ &= 2i\sin k\Delta x \end{aligned}$$

Rearranging the result, we wind up with this quadratic:

$$\lambda^2 + i2c'\sin k\Delta x\lambda - 1 = 0,$$

where we have again defined $c' = \frac{c\Delta t}{\Delta x}$. For convenience, let $a = c'\sin k\Delta x$, which results in:

$$\lambda^2 + i2a\lambda - 1 = 0. \tag{5.4}$$

Now, remember that λ is our change function, how we relate the future state of the solution to the present state. Note the change function here is a quadratic, meaning there are two solutions. Thus, there are two future states of the wave, even though the analytic version admits only one result. These two states are convolved as they are added together.

Solving quadratics containing complex numbers can be a little tricky. As an aside, we provide below solutions for complex equations like

$$\lambda^2 + (A + Bi)\lambda + (C + Di) = 0,$$

where A, B, C, and D are real numbers. As shown by Kurihara (1965; see Appendix 2), the two solutions are:

$$\begin{aligned}
\lambda_+ &= \left[\frac{-A}{2} + \frac{1}{2\sqrt{2}}\sqrt{R + \sqrt{R^2 + I^2}}\right] + i\left[\frac{-B}{2} + \frac{1}{2\sqrt{2}}\frac{I}{\sqrt{R + \sqrt{R^2 + I^2}}}\right] \\
\lambda_- &= \left[\frac{-A}{2} - \frac{1}{2\sqrt{2}}\sqrt{R + \sqrt{R^2 + I^2}}\right] + i\left[\frac{-B}{2} - \frac{1}{2\sqrt{2}}\frac{I}{\sqrt{R + \sqrt{R^2 + I^2}}}\right],
\end{aligned}$$

where

$$\begin{aligned}
R &= A^2 - B^2 - 4C \\
I &= 2AB - 4D.
\end{aligned}$$

In our equation, $A = D = 0$, $B = 2a$, and $C = -1$, so $R = -4(a^2 - 1)$ and $I = 0$. Therefore, our two change function solutions simplify down to:

$$\begin{aligned}
\lambda_+ &= \sqrt{1 - a^2} - ia \\
\lambda_- &= -\sqrt{1 - a^2} - ia,
\end{aligned}$$

and so we can write the solutions jointly as:

$$\lambda = -ia \pm \sqrt{1 - a^2}. \tag{5.5}$$

The interpretation of (5.5) depends upon whether $\sqrt{1 - a^2}$ is real or imaginary. Recall that

$$a = \frac{c\Delta t}{\Delta x} \sin k\Delta x,$$

and note that the maximum absolute value of sin is 1. Thus, if $\frac{c\Delta t}{\Delta x} \leq 1$ then $a \leq 1$ as well and as a result $\sqrt{1 - a^2}$ is real. In that case,

$$\lambda = \pm \underbrace{\sqrt{1 - a^2}}_{real} - \underbrace{ia}_{imaginary}, \tag{5.6}$$

and

$$\begin{aligned} |\lambda|^2 &= \left[\sqrt{1-a^2}\sqrt{1-a^2}+a^2\right] \\ &= 1-a^2+a^2 \\ &= 1. \end{aligned}$$

This is true for both change function solutions, $\lambda_{+,-}$, for any values of Δt and Δx so long as $\frac{c\Delta t}{\Delta x} \leq 1$. Therefore, *so long as it is stable, the leapfrog scheme has no amplitude error.* It is the advantage of centered schemes in general that they possess no implicit diffusion like the one-sided methods do. (We know that again $\frac{c\Delta t}{\Delta x} \leq 1$ is the stability criterion for the leapfrog scheme, as it was for the upstream method, because if we exceed this value, $|\lambda| > 1$ and the solution will blow up).

5.2 Physical and computational modes, and dealing with "time-splitting"

Before specifically considering the phase error of the leapfrog scheme, we have to see the consequences of using a three time-level approximation to a first-degree derivative. Again, we write the change function in polar form:

$$\lambda = |\lambda| e^{i\theta},$$

where it is recalled that the phase function is defined as:

$$\theta = \arctan \frac{\lambda_I}{\lambda_R}. \tag{5.7}$$

However, we've *two* solutions now, involving roots λ_+ and λ_-, and phase functions θ_+ and θ_-. These solutions are related through polar form (see Appendix A) as:

$$\lambda_+ = |\lambda| e^{-i\theta_-} \tag{5.8}$$

$$\lambda_- = |\lambda| e^{i(\theta_- + \pi)}. \tag{5.9}$$

Note the above expressions employed the phase function defined for the negative root. This was done for convenience since it can be shown that $\theta_- = -\theta_+$ (see Appendix A). From here on, the subscript on θ will be neglected.

We're only interested in how the scheme behaves when stable, and while stable $|\lambda|$ is always unity, so we can ignore it. Also (again, see Appendix A), $\theta = \arcsin(a)$. Recall at this point that we've routinely been assuming wave-like solutions of the form

$$u_j^n = A e^{i(kx - \omega_d n\Delta t)} = A\lambda^n e^{ikx}. \tag{5.10}$$

In this version, we are relating the solution at time n, u_j^n, to the solution at time 0, which is $n\Delta t$ time steps in the past. The equation for u_j^0 is simply a version of the above with $n = 0$, i.e.,

$$u_j^0 = Ae^{ikx},$$

and using the properties of the exponential function we again appreciate that (5.10) can be expressed as

$$u_j^n = A\lambda^n e^{ikx}.$$

But since we now have *two* solutions, we need to revise the above expression to:

$$u_j^n = (A_1\lambda_+^n + A_2\lambda_-^n)e^{ikx}. \tag{5.11}$$

One of these components may not be right, but the other is definitely wrong, and it remains to identify the mitigate the bad solution.

Quick recap of the preceding: We're looking at the how the leapfrog FD scheme handles waves. The state of the wave at the present time and spatial position is u_j^n. It turns out that for the leapfrog scheme, the solution consists of two convolved parts, given by the two terms on the RHS of (5.14), and having combined (total or net) amplitude A. We will identify one part with the desired solution and call it the *physical mode in time*. We will see the other solution, which we will term the *computational mode in time*, has the very curious habit of *changing sign every time step*. This portion of the solution is entirely artificial and we are going to do our best to get rid of it.

5.2.1 Behavior of the computational mode

The amplitudes A_1 and A_2 can be determined from the initial condition (IC), and this is covered in detail in the following subsection. However, we do not have to go through all those details to recognize which of the two solutions is the undesirable one. So, in this subsection, we will take a short cut to the answer.

We will start by expanding out (5.11) into

$$u_j^n = A_1\lambda_+^n e^{ikx} + A_2\lambda_-^n e^{ikx}. \tag{5.12}$$

Now we employ (5.8) and (5.9), recognizing that $|\lambda| = 1$. So the above expression becomes

$$u_j^n = A_1 e^{-i\theta n} e^{ikx} + A_2 e^{i(\theta+\pi)n} e^{ikx}$$

after λ_+ and λ_- are raised to the nth power. Note that $e^{i(\theta+\pi)n} = e^{i\theta n}e^{i\pi n}$. Further note that since $e^{i\pi}$ = -1 then $e^{i\pi n} = (\text{-}1)^n$, so that our equation becomes

$$u_j^n = A_1 e^{-i\theta n} e^{ikx} + (-1)^n A_2 e^{i\theta n} e^{ikx}. \tag{5.13}$$

We can say three things about this equation: First, there are two solutions convolved, represented by the amplitudes A_1 and A_2. Second, since the first term has $e^{-i\theta n}$ and the other has $e^{\theta n}$, and θ is proportional to c_d, the two parts are moving in *opposite directions*. Finally, owing to (-1)^n, the sign of the A_2 component is *flipping every time step*. That is clearly wrong, so the second term is the computational mode in time, leaving the first to be the physical mode.

Because of the flipping, the computational mode will manifest itself as $\mathbf{2\Delta}t$ **noise**. This is illustrated in Fig. 5.2. Suppose at some time, we have curvilinear feature in our domain that looks like that in the top panel. The feature is translating rightward. The bottom panel illustrates how that feature appears at location j as a function of time as it passes by. Superimposed on the feature is amplifying $2\Delta t$ noise, representing the influence of the computational mode on the physical solution. This mode itself can be thought of as two solutions that are in the process of decoupling and diverging away from each other, one represented by the even time steps, and the other by the odd ones. This is sometimes known as "time splitting", a term that is also used in a very different manner, representing a technique for integrating acoustically-active and inactive terms using different time steps for efficiency.

5.2.2 Initialization of the computational mode

Now we consider the details of determining the amplitudes of the physical and computational modes. From the preceding, we appreciate that $A = A_1 + A_2$, but we will find it more convenient to write this as $A_1 = A - A_2$. Thus, we can revise (5.13) as

$$u_j^n = (A - A_2)e^{-i\theta n}e^{ikx} + (-1)^n A_2 e^{i\theta n}e^{ikx}. \tag{5.14}$$

We determine the amplitude of the computational mode, A_2, by specifying how the time integration is started. Recall the leapfrog centered time derivative looks like this:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t}.$$

Usually, we don't have u_j^{n-1} at the initial time, only u_j^n. However, we can start the model integration out with a forward-time and centered-space scheme instead. This scheme is unstable, but we're only going to use it for one time step. Thus, for the first step:

$$\frac{u_j^1 - u_j^0}{\Delta t} = -c\left[\frac{u_{j+1}^0 - u_{j-1}^0}{2\Delta x}\right]. \tag{5.15}$$

Solve (5.15) for u_j^1 and then sub in the IC, $u_j^0 = Ae^{ikx}$. Then proceed using the modified

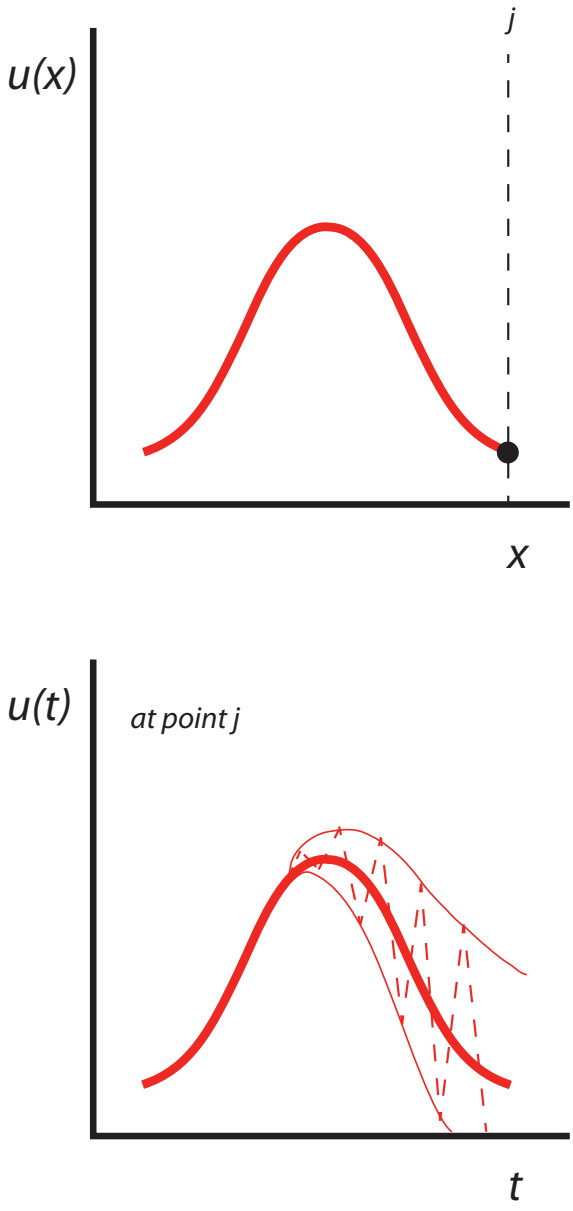

Figure 5.2: Top: a wave-like feature present in the domain at some time. Bottom: How the solution varies at point j identified in the top panel, presuming the feature is translating to the right. The thin, dashed lines represent the combination of the physical mode in time (the thick red curve) and the computational mode, which can be thought of as the decoupling of two solutions.

von Neumann method as before. We find:

$$\begin{aligned} u_j^1 &= u_j^0 - \frac{c\Delta t}{2\Delta x}\left[u_{j+1}^0 - u_{j-1}^0\right] \\ &= Ae^{ikx} - \frac{c\Delta t}{2\Delta x}Ae^{ikx}\left[e^{ik\Delta x} - e^{-ik\Delta x}\right] \\ &= Ae^{ikx} - \frac{c\Delta t}{\Delta x}Ae^{ikx}(i\sin k\Delta x) \\ &= Ae^{ikx}\left[1 - ic'\sin k\Delta x\right] \\ u_j^1 &= Ae^{ikx}\left[1 - ia\right]. \end{aligned} \tag{5.16}$$

We have again used the shorthands $c' = \frac{c\Delta t}{\Delta x}$ and $a = c'\sin k\Delta x$.

Now backtrack to (5.14) to look at the first time step after the initial time (i.e., $n = 1$). So

$$u_j^1 = (A - A_2)e^{-i\theta}e^{ikx} + (-1)A_2e^{i\theta}e^{ikx}.$$

Equate the preceding with (5.16). The common e^{ikx} term cancels and what remains can be written as:

$$A[1 - ia] = Ae^{-i\theta} - A_2[e^{-i\theta} + e^{i\theta}].$$

Note that $e^{-i\theta} + e^{i\theta} = 2\cos\theta$, and that $\theta = \arcsin(a)$, so that $\sin\theta = a$. Therefore, the expression can be further simplified to:

$$A(1 - \cos\theta) = -2A_2\cos\theta,$$

which can be solved for A_2 to yield:

$$A_2 = \frac{-A(1 - \cos\theta)}{2\cos\theta}.$$

Using this expression for A_2 in (5.14) first results in:

$$u_j^n = \left[A + \langle\frac{A(1 - \cos\theta)}{2\cos\theta}\rangle\right]e^{-i\theta n}e^{ikx} - (-1)^n\langle\frac{A(1 - \cos\theta)}{2\cos\theta}\rangle e^{i\theta n}e^{ikx},$$

which can be rearranged into the following form:

$$u_j^n = Ae^{ikx}\left[\langle\frac{(1 + \cos\theta)}{2\cos\theta}\rangle e^{-i\theta n} - (-1)^n\langle\frac{(1 - \cos\theta)}{2\cos\theta}\rangle e^{i\theta n}\right]. \tag{5.17}$$

This exposition effectively illustrates that the computational mode can exist from the very initial time. That does not mean that this component will grow to become problematic, but in practice with complicated, nonlinear problems, the mode does become excited and will soon grow to consume the physical solution if not mitigated.

5.2.3 Mitigation of the computational mode

To rid ourselves of the computational mode, we can exploit the tendency of the mode to alternate sign from time step to time step. In Fig. 5.3, it is presumed we can isolate the computational mode and show it for two adjacent time steps. The physical mode probably didn't change much between these two time levels, but the computational mode flipped over. So, if we average the two times together in some fashion, creating a new estimate for one of the times involved, the physical part of the solution won't be changed much but the computational part will be nearly eliminated.

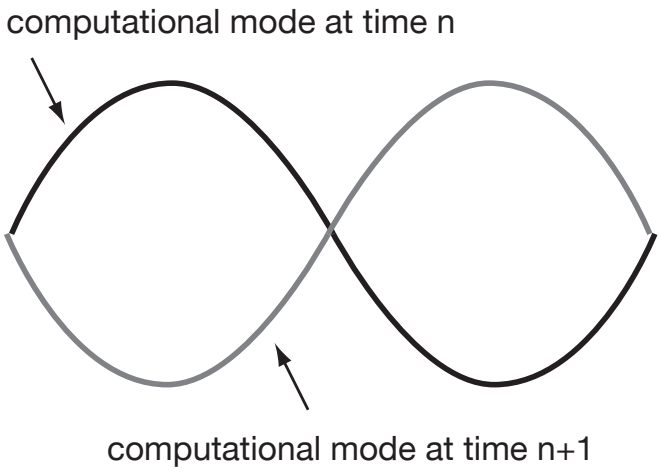

Figure 5.3: The computational mode flips every time step.

Thus, we can counteract the computational mode by introducing artificial time diffusion (or time smoothing) into our solution. Essentially, we are forcing into our system a term like u_{tt}. At a minimum, three time levels will actually be involved. However, if we want to perform our diffusion centered in time (and we do, for the sake of accuracy), this means that we need to know the forecast for time $n+1$ before we can apply smoothing centered at time n.

This can be accomplished by mixing pre- and post-smoothed values for our forecast variable, using the algorithm described below. Each time step contains the following two tasks:

- Use the leapfrog scheme to solve for the future value u_j^{n+1}, creating a preliminary, unsmoothed estimate.
- Revise the present value of u_j^n to adjusted value $\dot{u}_j^n$, the adjustment being indicated by the inclusion of a dot, by applying time smoothing involving the new future estimate (u_j^{n+1}), the present time's unsmoothed estimate (u_j^n), and the value from the past $(\dot{u}_j^{n-1})$, which has already been smoothed.

This can be implemented in the following manner, termed the Robert-Asselin time filter (after Robert 1966; Asselin 1972):

$$u_j^{n+1} = \dot{u}_j^{n-1} - c'(u_{j+1}^n - u_{j-1}^n) \tag{5.18}$$

$$\dot{u}_j^n = u_j^n + \frac{\epsilon}{2}\left[u_j^{n+1} - 2u_j^n + \dot{u}_j^{n-1}\right] \tag{5.19}$$

Equation (5.18) is the regular leapfrog scheme, used to create a temporary estimate for u_j^{n+1}. Then, we go back and revise the present time's estimate [in (5.19)] by smoothing it in a time-centered fashion. In (5.19), ϵ is a small diffusion coefficient, usually taken to be something like 0.1.

This approach is commonly applied but not without justifiable criticism. One of the most serious is that the mean of the three time values employed in the filter is not preserved by the smoothing function. An alternate method for addressing this problem that has been proposed in recent years (e.g., Williams 2009; Amezcua et al. 2011) can preserve the three time level mean by smoothing both time n and time $n+1$ in each application. This revised filter can be represented by the following, where now two dots indicate a value that has been altered for a second time:

$$u_j^{n+1} = \ddot{u}_j^{n-1} - c'(\dot{u}_{j+1}^n - \dot{u}_{j-1}^n) \tag{5.20}$$

$$\ddot{u}_j^n = \dot{u}_j^n + \frac{\epsilon\alpha}{2}\left[u_j^{n+1} - 2\dot{u}_j^n + \ddot{u}_j^{n-1}\right] \tag{5.21}$$

$$\dot{u}_j^{n+1} = u_j^{n+1} - \frac{\epsilon(1-\alpha)}{2}\left[u_j^{n+1} - 2\dot{u}_j^n + \ddot{u}_j^{n-1}\right] \tag{5.22}$$

The new parameter in this strategy, α, takes on a value between 0 and 1, inclusive, with $\alpha = 1$ representing the original Robert-Asselin filter. Williams (2009) notes that while $\alpha = 0.5$ preserves the three time level mean exactly, a slightly higher value ($\alpha = 0.53$) when $\epsilon = 0.2$ does the least damage to the accuracy of the physical mode.

5.3 Phase error

Now, finally, we consider the phase error of the leapfrog scheme. Actually, we're only interested in the error exhibited by the physical mode. In the above, we should have come to realize that the physical mode was represented by the + root given by (5.8). The phase function for this root is (see Appendix A):

$$\theta_{d+} = -\arcsin\left[c' \sin(k\Delta x)\right].$$

The subscript "*d*" has reappeared since we're going to compare this to the analytic phase function, found previously to be:

$$\theta_a = -c'k\Delta x.$$

Figure 3 plots the ratio of $\frac{\theta_{d+}}{\theta_a}$ as a function of normalized wavenumber and c'. It can be seen that the phase error is not a strong function of c' but varies tremendously with wavelength. As in the case of the upstream scheme, the shorter waves are handled worst, and the smallest resolvable wave (2Δx) does not move at all. As a result, the leapfrog FD solution suffers from dispersion error.

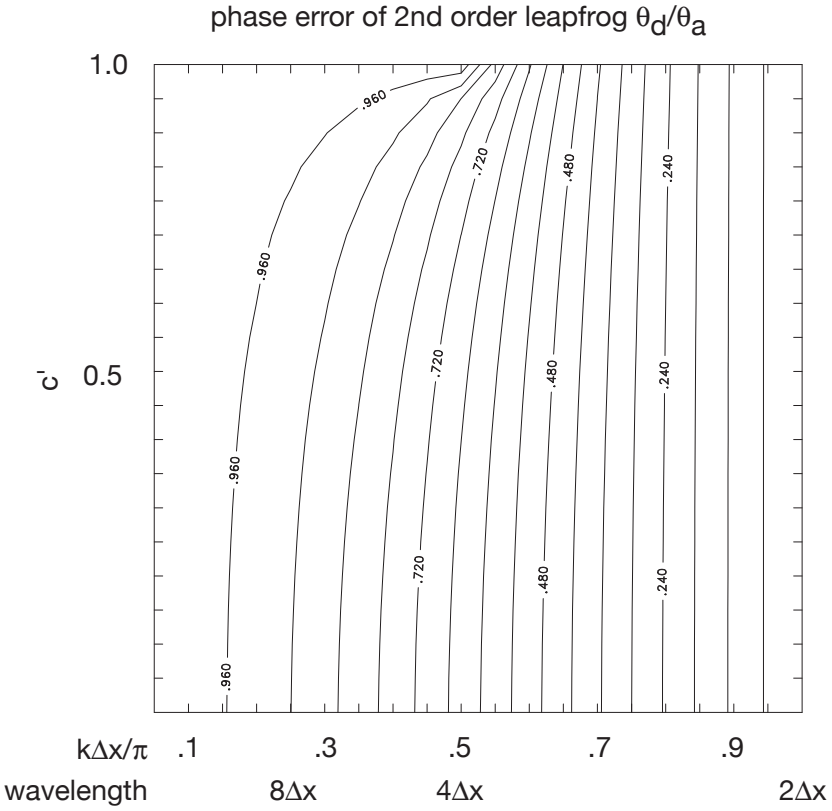

Figure 5.4: Phase error of the second-order leapfrog scheme.

5.4 The 2D leapfrog scheme

In the one-dimensional case, the stability criterion for both the upstream and leapfrog schemes was $\frac{c\Delta t}{\Delta x} \leq 1$. The quantity $\frac{c\Delta t}{\Delta x}$ is often termed the Courant number. The "rule" that the Courant number must be less than one is not set in stone, however. It is dependent on the order of accuracy of the scheme, the dimensionality of the problem and the grid layout. Generally, higher order schemes, multi-dimensional geometries and more sophisticated grid layouts (i.e., grid staggering) result in more restrictive stability conditions.

A case in point is the 2D leapfrog scheme, analyzed below. Let A be the variable being advected, by a two-dimensional flow field with flow components being u and v in the x and

y directions, respectively. Then the 2D leapfrog may be written in explicit form as:

$$A_{ij}^{n+1} = A_{ij}^{n-1} - \frac{u\Delta t}{\Delta x}\left[A_{i+1,j}^{n} - A_{i-1,j}^{n}\right] - \frac{v\Delta t}{\Delta y}\left[A_{i,j+1}^{n} - A_{i,j-1}^{n}\right]. \tag{5.23}$$

The variables A, u and v are all defined at the same grid points; i.e., the grid is unstaggered. Since the flow is presumed to vary in speed through the domain, this problem is not easy to solve. However, we recognize that the stability of the scheme will be challenged most in that section of the domain where the flow is moving fastest. Thus, we can replace u and v by their maximum values. For simplicity, take this maximum to be c in both directions. Further simplification can be realized by forcing the grid spacing to be uniform. Take this spacing to be Δ.

We assume 2D solutions now of the form:

$$A_{ij}^{n} = \hat{A}e^{i(kx+ly-\omega t)}$$

where l is the wavenumber in the y direction, related to the y direction wavelength L_y through

$$l = \frac{2\pi}{L_y}.$$

Inserting the assumed solution into the 2D leapfrog scheme results in:

$$\lambda = \frac{1}{\lambda} - \frac{c\Delta t}{\Delta}\left[(e^{ik\Delta x} - e^{-ik\Delta x}) + (e^{il\Delta y} - e^{-il\Delta y})\right]$$

Using Euler's relation and rearranging (after multiplying through by λ) results in:

$$\lambda^2 + 2i\frac{c\Delta t}{\Delta}\left[\sin(k\Delta x) + \sin(l\Delta y)\right]\lambda - 1 = 0$$

If we let a to be $\frac{c\Delta t}{\Delta}\left[\sin(k\Delta x) + \sin(l\Delta y)\right]$, our quadratic is

$$\lambda^2 + 2ia\lambda - 1 = 0$$

yet again, with the same roots previously presented in (5.5). Identical also is the fact that if $\sqrt{1-a^2}$ is real, then $|\lambda| = 1$ and the 2D leapfrog scheme has no amplitude error. Therefore, the stability criterion is $a^2 \leq 1$ yet again.

However, now that stability criterion implies a more onerous restriction on the time step. Since

$$a = \frac{c\Delta t}{\Delta}\left[\sin(k\Delta x) + \sin(l\Delta y)\right]$$

and we need $a^2 \leq 1$, then this means:

$$\frac{c^2\Delta t^2}{\Delta^2}\left[\sin(k\Delta x) + \sin(l\Delta y)\right]^2 \leq 1.$$

The maximum value of sin is one, so the largest the term in square brackets can be is two. Thus, we really have:

$$\frac{4c^2\Delta t^2}{\Delta^2} \leq 1$$

which implies that

$$\frac{c\Delta t}{\Delta} \leq \frac{1}{2}.$$

That is, the stability criterion is twice as restrictive. If we had also staggered the grid, putting u, v and A at different locations, the criterion would be still more restrictive. If we need to add time smoothing (to control the computational mode) and/or spatial smoothing (to restrain small-scale computational noise — a future topic), the time step we are allowed to employ will be smaller still.

The derivation of the phase error for the 2D leapfrog scheme is skipped, but again the scheme handles the shortest waves the worst. The only major difference is now the smallest resolvable waves ($2\Delta x$ and $2\Delta y$) do in fact translate relative to the grid. They are not stationary, as in the 1D case. Unfortunately, they also move *backwards*, against the wind, which will be demonstrated in Model Task 4. That is even more reason to ignore the shortest waves, or even to remove them from the solution.

Chapter 6

Dynamical frameworks

In Chapter 3, we derived the fully compressible (FCOM) model framework below

$$\frac{\partial u}{\partial t} = -\vec{V} \cdot \nabla u - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} \tag{6.1}$$

$$\frac{\partial w}{\partial t} = -\vec{V} \cdot \nabla w - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + g\frac{\theta'_v}{\bar{\theta}_v} \tag{6.2}$$

$$\frac{\partial \theta'}{\partial t} = -\vec{V} \cdot \nabla \theta' - w\frac{d\bar{\theta}}{dz} \tag{6.3}$$

$$\frac{\partial \pi'}{\partial t} = -\frac{\bar{c_s}^2}{\bar{\rho} c_{pd} \bar{\theta}_v^2} \left[\nabla \cdot \bar{\rho}\bar{\theta}_v \vec{V} \right] \tag{6.4}$$

In this chapter, we consider ways of solving this system of equations. These equations contain advective processes, sound waves and gravity waves. As noted earlier, sound waves play a dynamically important role in the atmosphere, but we need not be concerned with the details of how and when they act, and why. We want them to do their work and stay out of the way.

However, as noted earlier, the presence of sound waves seriously complicates the efficient solution of these equations. In Chapter 4 and Chapter 5, we saw that the need to maintain stability places limits on how large we can choose the model time step to be. For the upstream and second-order leapfrog schemes applied to the one-dimensional constant advection problem, the time step was constrained by the ratio of the grid spacing to the advection velocity. As the complexity of the FD scheme grows — with grid staggering, time smoothing, multi-dimensions, etc. — the limits on the time step becomes even larger.

It becomes more restrictive still as terms are added to the equations that represent non-advective phenomena (i.e., waves). In general, this time step is limited by the speed of the fastest moving signal in the model, regardless whether that signal speed represents the (advecting) flow or a wave. Indeed, they are additive: if the flow speed is uniform at 30 m

s^{-1} and the fastest wave is propagating in the direction of the flow at 50 m s^{-1}, then the fastest moving signal in the model — the signal that limits the time step — is really 80 m s^{-1}.

In the FCOM system above, the fastest moving signal is the sound wave ($\approx$ 350 m s^{-1}), and thus it happens that the greatest expense is being caused by the least important aspect of the model physical framework. There are three approaches we can take, beyond the simple "grin and bear it". These are:

1. Adopt "time-splitting", a technique that identifies the acoustically active and inactive terms in the equations and solves them on different time steps, each chosen to maintain stability[1]. Time-splitting was suggested in Marchuk (1974), a Russian textbook, and pioneered by Klemp and Wilhelmson (1978).

    - Advantage: Relatively economical.
    - Disadvantage: Complex implementation, small impact on the model physics.

2. Artifically *slow down* the sound waves, by treating the sound wave speed as a free parameter and discounting it. This is sometimes called the "quasi-compressible" (QCOM) or "super-compressible" approach and essentially allows the atmosphere to have more slackness.

    - Advantage: Efficient, easiest and most straightforward to code.
    - Disadvantage: Does more violence to the model physics.

3. Artificially *speed up* the sound waves... all the way to infinite phase speed. After all, the acoustic adjustment is already very rapid, why not make it instantaneous? This is the basis of the *anelastic approximation*.

    - Advantage: Eliminates something you really didn't want anyway, and has some desirable properties as well as a simple continuity equation.
    - Difficult to implement, as it makes pressure a diagnostic (as opposed to prognostic) variable. Very inefficient in 3D models.

6.1 Time-splitting

Time-splitting is based on the recognition that sound waves severely constrain the model time step but also that these waves manifest themselves most prominently in the pressure

[1]Caution: as noted in Chapter 5, the term "time-splitting" is also often used to describe the odd-even time step solution separation that occurs in the leapfrog scheme owing to the computational mode.

and divergence terms. The local time rate of change of velocity (e.g., u_t) is driven by two relatively separate phenomena: rapidly fluctuating acoustic waves, and slower moving (advective and gravity-driven) features that result in slower local fluctuations. Given that acoustic waves exist due to the slackness of the atmosphere regarding pressure and density, it is unsurprising that the "acoustically active" terms are those involving pressure. In time-splitting, we seek to keep the model stable by integrating the acoustically active terms on a small time step, while economizing by integrating the (relatively) inactive terms on a larger time step. Thus, the model contains two separate time steps, a large step Δt and a small (sound) time step $\Delta\tau$.

When we solve an equation like

$$u_t + cu_x = 0$$

with a three time level method like the second-order leapfrog scheme, we can think of the problem in this way: We want to jump forward from time $n-1$ (or $t-\Delta t$) to time $n+1$ (or $t+\Delta t$), and we do this by *holding the advection term* (cu_x) *constant at its value at the center of that time interval* (time n or t). This is because the advection term is evaluated at time n. We've a simple advection problem, and our time step Δt is determined by speed of the flow accomplishing the advection (c). Thus, the advective time scale is $\frac{\Delta x}{c}$

Now consider a more complicated problem, such as that offered in (6.1). The time tendency of u is not only a function of advection (the first term on the RHS) but also of the pressure gradient. The advection term again carries with it an advective time scale (now being $\frac{\Delta x}{u}$) that, left to itself, would determine the model time step. But the equation set now also admits acoustic wave solutions which, owing to their rapid speed, act to place a much more severe restriction on the time step.

Sound waves move so fast that advection doesn't affect their propagation very much at all. Thus, the advection term can still be computed on a relatively long time scale, that scale given by the flow speed (resulting in Δt). But the pressure gradient acceleration has to be computed over a relatively smaller time step, owing to the rapid propagation of sound waves. The time step chosen to keep sound waves stable is designated as $\Delta\tau$.

Note we could very well integrate the advection term on the small time step as well. It's a waste of resources since the advection term shouldn't be changing very much over the small time step. In fact, it should be changing so little that it could be presumed to be constant to a reasonable degree of accuracy. In fact, that's precisely what we will do.

Equation (6.1) can be written in this simple fashion:

$$u_t = -PGA - f_u,$$

where PGA is the pressure gradient acceleration term, and f_u represents all remaining terms (here, it is just the advection term; in a more complete model, f_u also would contain Coriolis,

diffusion and friction terms). We want to step from time $t - \Delta t$ to time $t + \Delta t$. The term f_u is evaluated at the intermediate (here/now) point, time t, and held constant through the time inetrval span $2\Delta t$, just as before. However, now we need to solve the PGA term over a smaller time step. This will be done using a forward stepping scheme, each leap being of time length $\Delta\tau$, starting from time $t - \Delta t$.

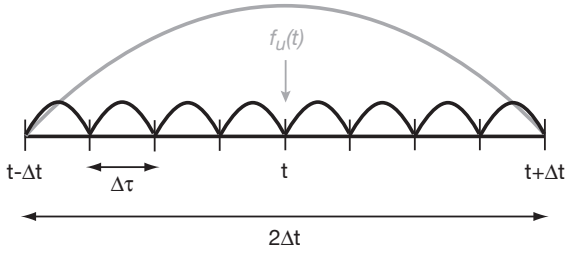

Figure 6.1: "Time-splitting".

It is recalled that the forward-time, center-space approximation is unstable for advection. Yet, although each small time step is done in the forward fashion, this isn't unstable for advection because as far as advection is concerned, it is still centered in time. For each "now" point (time t) we start Δt into the past to go to Δt into the future. This means, however, that half of the small time stepping is redone each (large) time step. To get from time t to time $t + \Delta t$, we start at time $t - \Delta t$ and do the small time stepping past time t all the way to time $t + \Delta t$. Then, to get to time $t + 2\Delta t$, we start back at time t and do the small time stepping over yet another time interval of length $2\Delta t$, redoing the stepping in the interval between times t and $t + \Delta t$. This isn't terribly efficient, but needs to be done to keep advection stable, and (unlike especially some of the advection terms) the PGA is computationally simple, involving relatively few arithmetic operations.

This time-splitting approach is also applied to the PGA term in the w equation (6.2) and the pressure equation (6.4). In KW's original strategy, then, the potential temperature equation (6.3) was integrated on the large time step, Δt. As mesoscale and synoptic scale models became deeper, extending well into the stratosphere, modelers discovered that their longer time step was being limited by that layer's great stability. Skamarock and Klemp (1992) noted that the w equation's buoyancy term and the θ' equation's vertical advection term could also be easily incorporated into the small time stepping scheme, thereby reliving another constraint on Δt.

In a mesoscale model, the vertical grid width Δz is typically smaller than the horizontal length Δx. Sound waves propagate vertically as well as horizontally, so it is the *vertical* grid length that really determines the small time step. Recall that we don't care how the sound waves move, as long as they do their job and stay out of the way. Further efficiency can be

achieved by treating the vertical pressure gradient terms in an *implicit* fashion.

Up to now, we have only considered explicit schemes, methods that allow us to integrate the model one grid point at a time. In an explicit scheme, the future solution at a given point depends only on the present and past values at other points, and *not on the future value at any other grid point*. In an implicit scheme, a set of grid points (such as those in a single model column) have to be solved simultaneously, because the forecast at a given point also depends on the forecast at other points. This will be elaborated upon later when the anelastic system is considered.

6.2 Quasi-compressibility

The quasi-compressible approach to solving (6.1)-(6.4) is brutally simple. Since sound waves serve to limit the time step owing to their great propagation speed, one can adopt a larger time step by simply choosing to artifically restrain the sound speed. Thus, c_s is treated as a free parameter, and is artifically discounted. As noted above, this is the simplest approach to implement, because it avoids the complexity inherent in implementing either time-splitting (in the fully compressible framework) or solving the diagnostic pressure equation (in the anelastic framework).

How low can c_s go? The equations admit both sound and gravity waves, but they don't interact much owing to their very different phase speeds. However, as the sound waves get slower, the possibility for inappropriate interaction becomes larger. Keeping advection and gravity wave time scales the same, but decreasing the sound speed, is tantamount to making the system you are simulating closer and closer to the supersonic limit. When the flow goes supersonic, the entire character of the solution changes, and the equations are no longer hyperbolic (or mixed hyperbolic/parabolic) in nature. For one thing, shock waves become possible.

Although the QCOM approach is becoming more popular, I am not aware of any really detailed examination of the errors associated with its adoption. One thing that can easily be done is to perform your experiment with a variety of sound speeds, including some relatively small ones, and watch for the differences that crop up owing to your making the model atmosphere more compressible and your flow speeds more nearly supersonic.

6.3 The anelastic approximation

In the anelastic approximation, we actually attain some efficiency (with respect to the time step, anyway) by speeding up the sound waves... to infinite speed. Take (6.4) and rewrite it

as the following:

$$\frac{1}{\bar{c}_s^2}\frac{\partial \pi'}{\partial t} = -\frac{1}{\bar{\rho} c_{pd} \bar{\theta}_v^2}\left[\nabla \cdot \bar{\rho}\bar{\theta}_v \vec{V}\right].$$

Remember we have already made some approximations, such as presuming that since sound waves are so fast, advection of pressure has a negligible influence on the local perturbation pressure tendency. For simplicity, presume an isentropic base state (no gradients of $\bar{\theta}_v$), yielding:

$$\frac{1}{\bar{c}_s^2}\frac{\partial \pi'}{\partial t} = -\frac{1}{\bar{\rho} c_{pd} \bar{\theta}_v}\left[\nabla \cdot \bar{\rho}\vec{V}\right].$$

Interpret this in the following way: local perturbation pressure fluctuations are caused by local mass convergence (per unit volume), itself driven by gradients in the wind field. When convergence occurs, $\nabla \cdot \bar{\rho}\vec{V} < 0$ and the pressure tends to rise. In our approximation, there is a direct cause-effect relationship between convergence/divergence and pressure fluctuations. However, there is a time lag between the onset of convergence (say) and when the pressure starts to rise. That lag represents the compressibility of the medium, and its time scale is determined by the sound speed c_s.

Now, perhaps we wish to remove this time lag, by increasing the sound speed. If we force $c_s \to \infty$, the pressure tendency term vanishes, and we're left with:

$$\nabla \cdot \bar{\rho}\vec{V} = 0 \tag{6.5}$$

This is known as the *anelastic continuity equation*, and was originally derived (at least formally) by Ogura and Phillips (1962). Before, the quantity shown in (6.5) wasn't forced to vanish, and its departure from zero drove the pressure tendency. Now, however, it is constrained to be zero each and every time step, at each and every location. Increasing the mass convergence locally now doesn't cause the future pressure to increase, instead it causes the pressure to increase *now*. There is no more slackness.

Viewed in this way, it is seen that pressure is no longer an property independent of the wind field. Rather, it is entirely dependent upon — and determined by — the wind (and buoyancy) fields. Pressure is no longer a prognostic variable in its own right.

What's more, the pressure essentially becomes an implicit quantity. When we perturb the atmosphere somewhere, the surrounding environment has to adjust to it. As discussed before, it adjusts in a variety of fashions. One adjustment is carried by acoustic waves, which quickly spread out away from the source of the perturbation. As time goes on, more and more points, located farther and farther away, "feel" the impact of the original perturbation; its spread is limited only by the sound speed.

However, if we cause the sound wave to propagate infinitely fast, that means *every point in the atmosphere feels the perturbation instantaneously.* That means, we can't determine the

pressure perturbation at any one location without knowing the pressure everywhere else at the same time. In other words, we can't solve pressure point-by-point, we have to determine the pressure simultaneously for all the grid points. It is an implicit problem.

6.3.1 The anelastic pressure equation

So, how do we get pressure, now that we've eliminated the pressure tendency? We can derive a diagnostic relation for pressure in the following way. First, note that (6.5) can be written as:

$$\frac{\partial \bar{\rho} u}{\partial x} + \frac{\partial \bar{\rho} w}{\partial z} = 0 \tag{6.6}$$

To simplify the derivation, (6.1) and (6.2) are rewritten as:

$$\frac{\partial u}{\partial t} = -ADV(u) - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} \tag{6.7}$$

$$\frac{\partial w}{\partial t} = -ADV(w) - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + B, \tag{6.8}$$

where $ADV(u)$ and $ADV(w)$ represent the advection terms of u and w, respectively, and B designates the buoyancy term. (In a more complex model, the ADV terms would also include diffusion, friction, and Coriolis terms.) Multiply (6.7) by $\bar{\rho}$ and take the horizontal derivative. Multiply (6.8) by $\bar{\rho}$ and take the vertical derivative. Upon adding the equations, we have:

$$\frac{\partial}{\partial t}\left[\frac{\partial \bar{\rho} u}{\partial x} + \frac{\partial \bar{\rho} w}{\partial z}\right] = -\frac{\partial}{\partial x}\left[\bar{\rho} ADV(u)\right] - \frac{\partial}{\partial z}\left[\bar{\rho} ADV(w)\right] - \bar{\rho} c_{pd}\bar{\theta}_v \frac{\partial^2 \pi'}{\partial x^2} - \frac{\partial}{\partial z}\bar{\rho} c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + \frac{\partial \bar{\rho} B}{\partial z} \tag{6.9}$$

The LHS of (6.9) was produced by interchanging the order of the time and space derivatives. Note it yields the time derivative of (6.6), a quantity that should be zero at every time. Thus, the LHS of (6.9) is zero, at least analytically. In practice, this term will differ from zero by round-off error, and that would have to be accounted for or such errors would accumulate with time[2]

If we rearrange what remains, we get a pressure equation that looks like this:

$$\frac{\partial^2 \pi'}{\partial x^2} + \frac{1}{\bar{\rho} c_{pd}\bar{\theta}_v}\frac{\partial}{\partial z}\bar{\rho} c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} = -\frac{1}{\bar{\rho} c_{pd}\bar{\theta}_v}\left[\frac{\partial \bar{\rho} ADV(u)}{\partial x} + \frac{\partial \bar{\rho} ADV(w)}{\partial z}\right] + \frac{1}{\bar{\rho} c_{pd}\bar{\theta}_v}\frac{\partial \bar{\rho} B}{\partial z}. \tag{6.10}$$

It looks complicated owing to the dependence of $\bar{\rho}$ and $\bar{\theta}_v$ on height. (Note that we previously assumed an isentropic base state in getting the anelastic continuity equation. That said, the

[2]So, in practice, this term evaluated at time $n-1$ is retained and moved to the RHS to account for truncation errors in making the previous forecast, but is presumed to hold exactly at time $n+1$ and thus vanish. We cannot make the $n+1$ computation yet anyway, unless we pursue an iterative approach to solving for the pressure perturbation.

equation can — and usually is — applied in a nonisentropic atmosphere, because neglect of the vertical gradient of θ_v can be justified by scale analysis. This results in some error, of course.)

6.3.2 Simplification and interpretation

In a constant density and potential temperature atmosphere, the pressure equation has the simple form of

$$\nabla^2 \pi' = F,$$

where F represents everything on the RHS. Note two things about this equation. First, it is elliptic, and has no time derivatives. Second, it depends on the gradients of the advection and buoyancy terms, which are evaluated at the present time. Thus, *we use the wind (and temperature, through buoyancy) information from the present time to compute the pressure at the present time.* In essence, the adjustment is infinitely fast.

If you have π' at every point, and need to compute F, that is a direct and simple problem, since you just discretize the Laplacian in FD form and solve it. Here, however, we have F and need to compute π'. We still discretize the Laplacian in FD form, but now it should be appreciated that we can't solve for a given point without also solving for its adjacent points, and we can't solve for those adjacent points without also solving for their adjacent points... and so on. Thus, every point depends on every other point, at least to some degree.

To demonstrate how we solve this kind of problem, we can simplify the equation further, to a single dimension. Consider the equation

$$\frac{\partial^2 \pi'}{\partial x^2} = F.$$

Discretize the Laplacian with a centered scheme, resulting in (after dropping the primes):

$$\frac{\pi_{i+1} - 2\pi_i + \pi_{i-1}}{\Delta x^2} = F_i.$$

It is seen that a point i directly depends on points $i-1$ and $i+1$, but not directly on any other point. However, every point depends on its neighbors so, ultimately, this will lead us to the boundaries. As a consequence, we can't solve this problem for any given grid point without applying some boundary conditions (BCs); that is, we need to either know π at the endpoints, or at the very least specify the horizontal gradient of π there. To keep it simple, we will presume we know the value of π. This is called a Dirichlet BC. If we instead presume a value for the first derivative of π, that is known as a Neumann BC.

We can express the above in the form of a matrix:

$$\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \vdots \\ \pi_{N-1} \\ \pi_N \end{bmatrix} = \Delta x^2 \begin{bmatrix} F_1 - \frac{\pi_0}{\Delta x^2} \\ F_2 \\ F_3 \\ \vdots \\ F_{N-1} \\ F_N - \frac{\pi_{N+1}}{\Delta x^2} \end{bmatrix} \tag{6.11}$$

There are a total of N interior points (that is, points that do not involve a boundary value in the Laplacian, i.e., those with an index between 2 and $N-1$). Note that there is a regular set of entries in the leftmost matrix, being values 1, -2 and 1 for all but the points just inside the boundaries. Even for this simple problem it is seen that most of the entries of this tridiagonal matrix are zero. The rightmost matrix is the vector of known values of F. The boundary values π_0 and π_{N+1} have to be prescribed (for Dirichlet conditions) and are known, and thus these values are moved into the F matrix. The matrix can be solved by Gaussian elimination. (Actually, the regular nature of the leftmost matrix means there are also more efficient, direct methods of solution as well.)

Here is a simple little example, involving a total of 5 points, three unknowns and two boundary points (so N=3). Take Δx=1 and presume the boundary π values to be zero. Thus, we have:

$$\begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$$

Applying Gaussian elimination on these F values yields these values: π_1=-13/4; π_2=-7/2; π_3=-11/4. These can be substituted into the original equation to verify that indeed these values of π, along with the boundary conditions, do indeed generate the given values of F.

The simple example shows that the pressure values have to be obtained all at once, since each point depends on its neighbors, and also helps to illustrate how important the BCs are. Changing the BCs slightly would result in a radically different solution (try that).

The simple 1D problem is very easy to solve. In 2D, it is more difficult. Note that for N unknown points, the π matrix is N by N. In 2D, say we have a domain of NX and NZ unknown points. Thus, the matrix is $NX \cdot NZ$ by $NX \cdot NZ$. Say our domain is 300 by 30 points. Then the π matrix is 9000 $\times$ 9000! In 3D, the array is dimensioned $NX \cdot NY \cdot NZ$ by $NX \cdot NY \cdot NZ$. This is quickly getting out of hand.

However, notice that the π matrix has a very regular structure. That is true in 2D and 3D as well, though of course the structures in those situations are more complicated. For a large number of unknowns, the matrix is also very *sparse*; most of the entries are zero. We can make use of the sparseness and use an algorithm that requires us only to save

the nonzero matrix values. This saves a lot on storage, but the problem is still becoming quite cumbersome to solve. In 2D, the anelastic model is pretty competitive with the fully compressible framework, but in 3D it lags well behind, owing to the number of computations required to solve the diagnostic pressure equation.

6.3.3 Decomposition of the anelastic pressure perturbation field

The anelastic pressure equation does have one very nice property. (The derivation of this pressure decomposition is covered in the next subsection.) Since the Laplacian operator is linear, the pressure can be cleanly separated into components. In general form, we have:

$$\nabla^2 \pi' = F_b + F_d,$$

where F_b is the vertical derivative of the buoyancy term and F_d represents the derivatives of the advection (and other) terms. The subscripts b and d are chosen because the terms represent *buoyancy* and *dynamic* forcings, respectively. The derivatives of the advection terms affect pressure dynamically, through convergence and divergence. An example of this is cyclostrophic balance, where a local circulation generates low pressure inside the circulation.

The buoyancy term affects pressure through heating and cooling. An example of this: You know a positively buoyant parcel will rise. To do so, it must push surrounding air out of the way. Part of the buoyancy provokes a pressure response through the buoyancy term. The vertical gradient of buoyancy above the parcel's center is negative, and that in itself produces high pressure. Below the parcel center, the gradient is positive and low pressure is produced.

Again, since the Laplacian operator is linear, we can separate the total pressure into its dynamic and buoyant parts. Thus, we can define *dynamic pressure*, π'_d, as the solution of:

$$\nabla^2 \pi'_d = F_d,$$

and *buoyancy pressure*, π'_b, as the solution of:

$$\nabla^2 \pi'_b = F_b.$$

The total pressure π' is simply the sum of the two component pressures. This can't be done without the anelastic pressure equation, and is a very useful property of the anelastic framework.

6.3.4 Derivation of the anelastic pressure decomposition

While nondimensional pressure π' has theoretical and computational advantages, perturbation p' is employed here as it makes for a somewhat more straightforward presentation. The

perturbation method has already been applied to pressure, temperature θ and density ρ in the equations of motion listed below and the term in the w equation inversely proportional to the sound speed squared has already been removed. $B \equiv g\frac{\theta'}{\theta}$ is the buoyancy. Equation 6.15 is the traditional anelastic continuity equation.

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} - w\frac{\partial u}{\partial z} - \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial x} \tag{6.12}$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} - w\frac{\partial v}{\partial z} - \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial y} \tag{6.13}$$

$$\frac{\partial w}{\partial t} = -u\frac{\partial w}{\partial x} - v\frac{\partial w}{\partial y} - w\frac{\partial w}{\partial z} - \frac{1}{\bar{\rho}}\frac{\partial p'}{\partial z} + B \tag{6.14}$$

$$0 = \frac{\partial \bar{\rho}u}{\partial x} + \frac{\partial \bar{\rho}v}{\partial y} + \frac{\partial \bar{\rho}w}{\partial z} \tag{6.15}$$

Equation 6.15 can also be written as

$$0 = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} + w\frac{d\ln\bar{\rho}}{dz}. \tag{6.16}$$

The derivation starts with multiplying equations (6.12)-(6.14) by $\bar{\rho}$, differentiating with respect to x, y and z, respectively, and adding the result. Recalling that the order of differentiation can be interchanged, the left hand side (LHS) of the new combined equation is

$$\frac{\partial}{\partial t}\left[\frac{\partial \bar{\rho}u}{\partial x} + \frac{\partial \bar{\rho}v}{\partial y} + \frac{\partial \bar{\rho}w}{\partial z}\right],$$

which is zero owing to (6.15). The right hand side (RHS) expands out to

$$\begin{aligned}
&-\frac{\partial \bar{\rho}u}{\partial x}\frac{\partial u}{\partial x} - \bar{\rho}u\frac{\partial^2 u}{\partial x^2} - \frac{\partial \bar{\rho}v}{\partial x}\frac{\partial u}{\partial y} - \bar{\rho}v\frac{\partial^2 u}{\partial x\partial y} - \frac{\partial \bar{\rho}w}{\partial x}\frac{\partial u}{\partial z} - \bar{\rho}w\frac{\partial^2 u}{\partial x\partial z}\\
&-\frac{\partial \bar{\rho}u}{\partial y}\frac{\partial v}{\partial x} - \bar{\rho}u\frac{\partial^2 v}{\partial x\partial y} - \frac{\partial \bar{\rho}v}{\partial y}\frac{\partial v}{\partial y} - \bar{\rho}v\frac{\partial^2 v}{\partial y^2} - \frac{\partial \bar{\rho}w}{\partial y}\frac{\partial v}{\partial z} - \bar{\rho}w\frac{\partial^2 v}{\partial y\partial z}\\
&-\frac{\partial \bar{\rho}u}{\partial z}\frac{\partial w}{\partial x} - \bar{\rho}u\frac{\partial^2 w}{\partial x\partial z} - \frac{\partial \bar{\rho}v}{\partial z}\frac{\partial w}{\partial y} - \bar{\rho}v\frac{\partial^2 w}{\partial y\partial z} - \frac{\partial \bar{\rho}w}{\partial z}\frac{\partial w}{\partial z} - \bar{\rho}w\frac{\partial^2 w}{\partial z^2}\\
&+\frac{\partial \bar{\rho}B}{\partial z} - \nabla^2 p'.
\end{aligned}$$

Using the shorthand of $u_x \equiv \frac{\partial u}{\partial x}$, $u_{xx} \equiv \frac{\partial^2 u}{\partial x^2}$, etc., further manipulation results in

$$\nabla^2 p' = -\bar{\rho}\left[u_x^2 + v_y^2 + w_z^2\right]$$
$$-\bar{\rho}u\frac{\partial}{\partial x}\left[\frac{\partial \bar{\rho}u}{\partial x} + \frac{\partial \bar{\rho}v}{\partial y} + \frac{\partial w}{\partial z} + w\frac{d\ln\bar{\rho}}{dz} - w\frac{d\ln\bar{\rho}}{dz}\right]$$
$$-\bar{\rho}v\frac{\partial}{\partial y}\left[\frac{\partial \bar{\rho}u}{\partial x} + \frac{\partial \bar{\rho}v}{\partial y} + \frac{\partial w}{\partial z} + w\frac{d\ln\bar{\rho}}{dz} - w\frac{d\ln\bar{\rho}}{dz}\right]$$
$$-\bar{\rho}w\frac{\partial}{\partial z}\left[\frac{\partial \bar{\rho}u}{\partial x} + \frac{\partial \bar{\rho}v}{\partial y} + \frac{\partial w}{\partial z} + w\frac{d\ln\bar{\rho}}{dz} - w\frac{d\ln\bar{\rho}}{dz}\right]$$
$$-2\bar{\rho}\left[v_x u_y + w_x u_z + v_z w_y\right] - \frac{d\bar{\rho}}{dz}\left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right] + \frac{\partial \bar{\rho}B}{\partial z}.$$

In the above, the expression $w\frac{d\ln\bar{\rho}}{dz}$ was added and subtracted to the second, third and fourth terms of the RHS. Equation (6.15) can be invoked in those terms and in other places, leaving us with

$$\nabla^2 p' = -\bar{\rho}\left[u_x^2 + v_y^2 + w_z^2\right] - 2\bar{\rho}\left[v_x u_y + w_x u_z + v_z w_y\right] + \bar{\rho}\frac{d^2\ln\bar{\rho}}{dz^2} + \frac{\partial \bar{\rho}B}{\partial z}.$$

The first three terms on the RHS are the dynamic pressure forcing F_d, the last term in the buoyancy forcing F_b. Again, since the ∇^2 operator is linear, the equation may be separated into nonoverlapping dynamic and buoyancy pressure perturbations, creating the following

$$\nabla^2 p' = F = F_d + F_b$$

$$\nabla^2 p'_d = F_d$$
$$\nabla^2 p'_b = F_b$$

where $p' = p'_d + p'_b$. The dynamic pressure perturbation can be further decomposed into linear (the term with the squares) and nonlinear (the cross-derivatives) parts. This was used by Rotunno and Klemp (1982) to explain why supercell storms split, and why the split members tended to move to the right and left of the mean wind through the cloud-bearing layer.

Model Task #6 discusses how the anelastic pressure equation can be solved in the model, and how the dynamic and buoyancy pressure parts can be isolated.

Chapter 7

Nonlinear computational instability

7.1 Origin of the instability

In Model Task #4, we saw the consequences of the leapfrog scheme's poor handling of the shorter waves in solving the simple, 2D linear constant advection problem

$$u_t + c_x u_x + c_y u_y = 0,$$

where c_x and c_y were specified constants. Short waves that were originally hiding in the initial condition were revealed owing to dispersion error, causing the feature being advected to change shape (and making the solution look noisy). Otherwise, the fact that the short waves got exposed did no harm. They did not grow unbounded, at least as long as we satisfied the stability condition.

The short waves didn't do any harm because the problem we solved was linear. In a *nonlinear* problem, the small scale waves do pose a threat to our solution, and can grow unbounded (or, at least grow to the point where they dominate — and destroy — the simulation) even when the stability condition is satisfied. In this situation, they do not grow because their amplification factors exceed unity. Instead, they grow because they spuriously accumulate energy through a phenomenon called *aliasing*.

Aliasing can only occur when the equation being solved contains nonlinear terms, such as the uu_x term in

$$u_t + uu_x = 0,$$

because such terms can cause the creation of new wavelengths that did not exist in the initial condition. A simple, 1D example: Suppose the solution for u at the present time consists of a single sine wave with wavenumber k; i.e.,

$$u = \sin kx.$$

The spatial derivative of this is:

$$u_x = k \cos kx,$$

so the nonlinear term can take the original sine wave and produce

$$uu_x = k \sin kx \cos kx = \frac{1}{2} k \sin 2kx;$$

i.e., a wave with twice the wavenumber (*half the wavelength*) of the original wave. In this way, energy can be transferred to the smaller scales. One could start off with only relatively large wavelengths, but owing to nonlinear interaction, waves with progressively smaller and smaller wavelengths (relative to the grid) will be produced.

And this will continue... at least until the grid can no longer represent the waves being generated by the nonlinear terms. In reality, this downscale transfer continues until very, very small wavelengths are generated that are dissipated via friction. This ditty, attributed to the numerical weather prediction pioneer, Lewis F. Richardson, expresses this phenomenon:

> Big whirls have little whirls that feed on their velocity. Little whirls have lesser whirls, and so on — to viscosity.

The downscale transfer does not cause problems in reality because there exists a sink for the energy at the smallest scales — viscous dissipation.

In the finite difference world, however, this downscale transfer is frustrated, and actually halted, by the fact that the smallest resolvable wave is one with twice the grid interval. (In terms of wavenumber, the smallest wave — or largest possible wavenumber — is $k_{max} = \frac{\pi}{\Delta x}$.) You can think of this smallest wave situation as a boundary condition, a rigid wall or, even better, a mirror. If nonlinear interaction results in the generation of a wave that cannot be resolved on the grid, it will "reflect" back into the resolved wavelengths. This is aliasing. Instead of continuing down towards viscosity, the wave and its energy remain in the numerically resolvable wavelengths.

As a specific example, consider a wave with wavelength $\frac{8}{3}\Delta x$. This is resolvable; there are three peaks in every eight grid points[1] . Two of these waves combine to produce a wave with half the original wavelength, or $\frac{4}{3}\Delta x$. This newly created wave is smaller than $2\Delta x$ and cannot be resolved. The grid, however, sees this as a $4\Delta x$ wave (see Fig. 1). The wave has aliased, or reflected, back into the resolvable grid... and has been given a wavelength that is actually *longer* than the original wave pair had. In reality, this wavelength should have been much smaller and kept on the path to frictional dissipation. In the model, the wave and its energy has not been "lost" and instead has been remained in the vicinity of the smallest resolvable wave.

[1]This may seem like an odd wavelength, but in a Fourier sense it would appear in any domain with N being a multiple of 8.

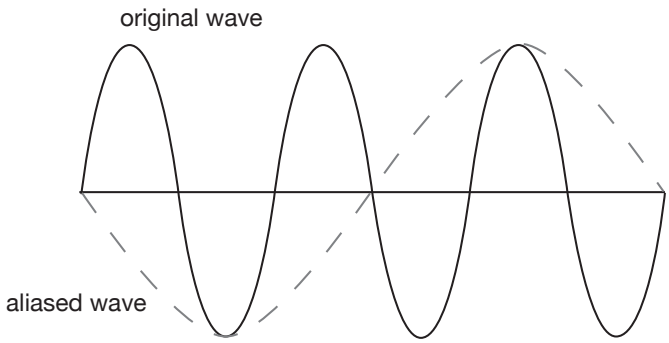

Figure 7.1: Aliasing of an unresolvable wave.

The problem isn't that the energy wasn't lost as it should have been (though that's unrealistic in itself), it's that owing to aliasing energy will accumulate in the shorter waves. To see this, we examine a more general case, in which a pair of waves with potentially different wavenumbers (k_1 and k_2) interact. The multiplication of two such sine waves can yield (using standard trigonometric relations):

$$\sin k_1 x \sin k_2 x = \frac{1}{2}\left[\cos\left(k_1 - k_2\right)x - \cos\left(k_1 + k_2\right)x\right].$$

Two new wavenumbers, having values $k_1 - k_2$ and $k_1 + k_2$, are generated.

Note the latter combination yields a wavelength that is shorter than what the original waves had. Eventually, waves with $k > k_{max}$ will occur and be misrepresented by the FD grid, aliasing back onto the resolved grid. Let

$$\hat{k} = k_1 + k_2.$$

Then:

$$\cos \hat{k}x = \cos\left[2k_{max} - (2k_{max} - \hat{k})\right]x.$$

Recall $k_{max} = \frac{\pi}{\Delta x}$ and use a trig formula for the cosine of a difference to obtain

$$\cos \hat{k}x = \cos(\frac{2\pi}{\Delta x}x)\cos\left[\left(\frac{2\pi}{\Delta x} - \hat{k}\right)x\right] + \sin(\frac{2\pi}{\Delta x}x)\sin\left[\left(\frac{2\pi}{\Delta x} - \hat{k}\right)x\right].$$

At a *given grid point* $x{=}j\Delta x$, $\sin \frac{2\pi}{\Delta x} j\Delta x = 0$ and $\cos \frac{2\pi}{\Delta x} j\Delta x = 1$, so

$$\cos \hat{k} j\Delta x = \cos\left[\left(2k_{max} - \hat{k}\right) j\Delta x\right].$$

The above expression shows that we cannot distinguish between a wavenumber $\hat{k}$ and its counterpart $2k_{max} - \hat{k}$, because their cosines are the same. This becomes important when

the wavenumber $\hat{k} > k_{max}$, which is a wave that is too small to resolve on the grid. In this situation, the wave aliases back onto the grid, and is erroneously seen instead as

$$k^* = 2k_{max} - \hat{k}.$$

See Fig. 2. The wavenumber k_{max} is the symmetry point, and acts like a mirror. Nonlinear interaction produces an unresolvable wave with wavenumber k (henceforth dropping the hat). The distance (in wavenumber space) that the wave resides on the unresolvable side of k_{max} is the same distance to which the wave actually appears to be on the resolvable side.

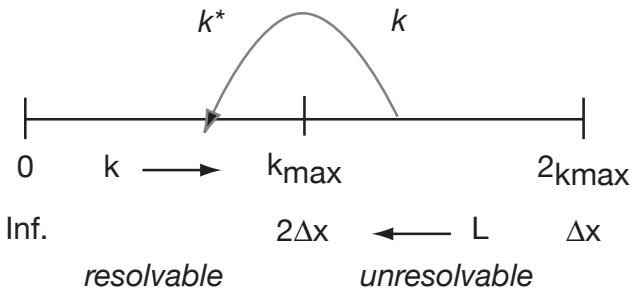

Figure 7.2: How an unresolvable wave reappears on the grid.

Let's reconsider the $\frac{4}{3}\Delta x$ wave, which we saw earlier to be produced as a result of the multiplication of two $\frac{8}{3}\Delta x$ waves. This wave's wavenumber is $k = \frac{3\pi}{2\Delta x}$, which is unresolvable. Thus, it appears on the grid as $k^* = 2k_{max} - k$, or $\frac{\pi}{2\Delta x}$, which is the same as $\frac{2\pi}{4\Delta x}$, i.e., a $4\Delta x$ wave.

Nonlinear instability was first encountered by Norman Phillips, considered the father of modern numerical weather prediction. In a 1956 experiment, Phillips discretized the vorticity equation and, starting from an atmosphere at rest, he integrated his model for 30 simulated days. The simulation came to an end at that point as energy in the smallest resolvable wavelengths grew unbounded. Thinking he had a problem with truncation error, which he knew to be controlled by the time step, he repeated the experiment with a smaller Δt. However, the simulation still came to a catastrophic end, after about the same number of time steps. This told Phillips he wasn't violating any linear stability condition, and that truncation error wasn't the culprit.

What had occurred is that nonlinear interaction was generating progressively smaller and smaller waves, even though small waves weren't contained in the initial condition. Again, in reality, these waves — and their energy — should cascade down to the very smallest scales, where they would be extinguished by friction. In the model, however, the waves and their energy cannot "escape", and once they become unresolvable, they alias back into the resolvable waves and accumulate at the smallest resolvable scales. The growth of energy

in these waves is completely spurious, but their amplification destroyed the solution and brought the simulation to an early and miserable end.

Phillips addressed this problem by periodically decomposing his fields into waves via Fourier analysis and removing those components with wavelengths of $4\Delta x$ and shorter (i.e., $k > \frac{1}{2}k_{max}$). If there is no spectral energy in wavelengths $L < 4\Delta x$, the advection term cannot cause aliasing to occur. With energy precluded from accumulating at the smallest resolvable scales, Phillips' simulation did not come to a catastrophic end, confirming his hypothesis regarding the source of the spurious energy amplification. Later, it became common to remove energy at small scales via smoothing (see next subsection) instead, and waves of $3\Delta x$ and shorter (i.e., $k > \frac{2}{3}k_{max}$) were specifically targeted. Interactions involving wavelengths between 3 and $4\Delta x$ will still cause aliasing, but the aliased wave will fall between 2 and $3\Delta x$, and be subjected to smoothing anyway. It is recalled that the FD schemes typically handle these small waves very poorly anyway, and now we've a very good excuse for removing them from the solution.

7.2 Controlling nonlinear instability through smoothing

The most common way of eliminating the small scale waves that participate in aliasing is through the addition of a computational filter — artificial diffusion — to the model equations. This approach takes advantage of the fact that a diffusion term, like $\frac{\partial^2 u}{\partial x^2}$, acts most strongly on the shortest waves. Recall that the even derivatives represent diffusion. In practice, any even degree derivative can be employed as a short-wave filter (smoother), though the higher the degree, the more narrowly the smoothing effect is concentrated on the smallw aves you wish to eradicate. However, higher degree terms get harder to implement.

The simplest diffusion term is zero-degree, as shown on the RHS of this equation:

$$u_t + uu_x = -au.$$

The coefficient a controls the rate at which waves are extinguished. When u represents velocity, it is often termed "Rayleigh friction"; when u is temperature, it is called "Newtonian cooling". This diffusion term is totally nonselective: all waves are relaxed back to zero amplitude at the same rate. Thus, this term is not very useful for controlling small scale noise.

To see this, we discretize the equation using the leapfrog scheme. We neglect the advection term and focus on the action of the diffuser. The FD version of the equation is:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} = -au_j^{n-1}.$$

Note the diffusion term is computed at time n-1. This is necessary for stability. Assume the same solution we've done countless times before and perform a stability analysis, solving for the amplification factor. In this case, we wind up with

$$|\lambda|^2 = 1 - 2a\Delta t. \tag{7.1}$$

In this case, $|\lambda| < 1$ means the amplitude of the wave is being extinguished. The rate of damping is controlled by the parameter a. Suppose you take $a = \frac{1}{2\Delta t}$; in that case $|\lambda| = 0$, and the wave vanishes upon one application of the diffuser.

Note that the amplitude factor is **not** a function of wavenumber — k does not show up on the RHS. Thus, this term damps out all wavelengths equally, at the same rate. Greater scale selectivity is attained by using a higher order even degree derivative. Adding a second-degree diffuser makes the equation look like:

$$u_t + uu_x = au_{xx}.$$

This is discretized (again, neglecting the advection term) as:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} = a\left[\frac{u_{j+1}^{n-1} - 2u_j^{n-1} + u_{j-1}^{n-1}}{\Delta x^2}\right].$$

Again, the diffuser is implemented at time n-1. You may note that the sign on the RHS is opposite from what it was for the zero-degree term. This sign alternates between (even) degrees.

Assuming the standard solution again produces the following:

$$\lambda = \frac{1}{\lambda} + \left[\frac{a2\Delta t}{\Delta x^2}\right]\frac{1}{\lambda}\left[e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right] \tag{7.2}$$

$$|\lambda|^2 = 1 + \left[\frac{a2\Delta t}{\Delta x^2}\right]\left[2\cos k\Delta x - 2\right] \tag{7.3}$$

Now the amplification (dissipation) factor is a function of wavelength, though k, and this diffuser will hit the smaller waves harder than the longer ones. To see this, let's assume we want to completely eradicate $2\Delta x$ waves in one application. (This is overkill; we are not usually so desperate.) So, we want $|\lambda| = 0$ for $k{=}\frac{2\pi}{2\Delta x}$. Since now $k\Delta x = \pi$, we have

$$0 = 1 + \left[\frac{2a\Delta t}{\Delta x^2}\right]\left[2\cos\pi - 2\right].$$

Solving this we find:

$$\frac{2a\Delta t}{\Delta x^2} = \frac{1}{4}$$

results in the complete removal of $2\Delta x$ waves in one step. In this case, $\frac{a}{\Delta x^2}{=}\frac{1}{8\Delta t}$

How does our hitting the $2\Delta x$ waves harm the longer waves we are interested in preserving? Let's see how an $8\Delta x$ wave would be affected by the same diffuser, using the value of a that eradicated the $2\Delta x$ waves. Now we apply the same equation, but with $k\Delta x = \frac{\pi}{4}$ and set $\frac{2a\Delta t}{\Delta x^2} = \frac{1}{4}$. We find:

$$\begin{aligned} |\lambda|^2 &= 1 + \frac{2a\Delta t}{\Delta x^2}\left[2\cos\frac{\pi}{4} - 2\right] \\ &= 1 - \frac{1}{4}(1.293) \\ &= 0.68, \end{aligned}$$

or $\lambda = 0.83$, which is rather serious dissipation of a wave we wish to keep. Each time step, the wave will lose roughly 20% of its amplitude.

We can attain greater scale selectivity by going to a still higher degree diffuser, such as in:

$$u_t + uu_x = -au_{xxxx}.$$

The amplification factor for this diffuser (again, applied at time n-1) is:

$$|\lambda|^2 = 1 - \frac{2a\Delta t}{\Delta x^4}\left[6 - 8\cos(k\Delta x) + 2\cos(2k\Delta x)\right]. \tag{7.4}$$

The discretization of the fourth-degree diffuser is more complicated still, involving even more points:

$$-au_{xxxx} = -a\left[u_{j+2}^{n-1} - 4u_{j+1}^{n-1} + 6u_j^{n-1} - 4u_{j-1}^{n-1} + u_{j-2}^{n-1}\right]\Delta x^{-4}.$$

This will prove difficult to apply near the boundaries, unless the periodic condition is employed. Typically, owing to the proximity of the upper and lower boundaries, only second-degree diffusers are applied in the vertical direction. Assuming non-periodic horizontal boundaries, if the fourth-degree diffuser is used for the interior grid points, the second-degree scheme will have to be used at the points located just inside of the boundaries (owing to the lack of information available outside of the boundaries).

We can visualize the relative performance of the second- and fourth-degree diffusers by plotting their damping rates [from equations (7.3) and (7.4)] as a function of $k\Delta x$. In Fig. 7.3, the damping rates have been normalized by the values needed to completely remove $2\Delta x$ waves in one time step (one application). This is overkill, but it helps us see how more scale-selective the fourth-order smoother is. A sixth-degree diffuser would be even more surgical, at the cost of still greater complexity. The WRF model has sixth-order smoothing available as an option.

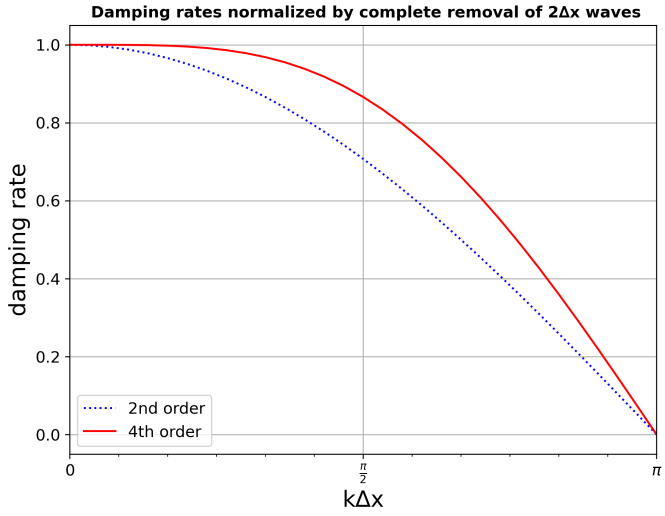

Figure 7.3: Damping rate $|\lambda|$ vs. $k\Delta x$ for the second- and fourth-order diffusers, configured for complete removal of $2\Delta x$ waves in one application.

Chapter 8

Moisture and microphysics

In this chapter, we take the equations in the form we discretized them in Task #5 and modify them to account for moisture and computational diffusion. First, we'll discuss how to handle cloud microphysics in a relatively simple manner. Here, "simple" refers to the many approximations and assumptions we must make; even a "simple" microphysical parameterization is actually quite complicated.

First, let's recap and extend the base state. Our initial environment is a function solely of height. We need column vectors to store $\bar{u}$, $\bar{\theta}$, $\bar{q_v}$, $\bar{p}$, $\bar{\pi}$, and density at the u and w locations, $\bar{\rho}_u$ and $\bar{\rho}_w$. The $\bar{u}$ vector is added so we can implement a base state horizontal wind, with or without vertical shear. To save repetitive calculations, we can also define a mean state virtual potential temperature as:

$$\bar{\theta}_v = \bar{\theta}\left[1 + .61\bar{q_v}\right].$$

We need to modify and augment terms in the original set of equations, and add three new equations, to account for moisture substances in the model. Let the water vapor mixing ratio perturbation be termed q_v', representing the departure from the height-dependent mean state. Now we need to handle condensed water substance. In the simplest approach, condensed water takes two forms: free-floating cloud droplets and larger rain drops that fall relative to still air. Let the cloud water mixing ratio be designated q_c and the rain water mixing ratio be q_r.

8.1 Cloud and rain water

In our simple microphysical scheme, vapor condenses to form floating cloud droplets, which are all assumed to be the same size. Condensation of vapor to cloud water is presumed to

occur instantaneously, in whatever amount deemed necessary to remove any and all supersaturation. This will be performed using the saturation adjustment technique you used to compute the CAPE in Model Task #2 (see later). Conversely, when the relative humidity dips below 100%, any cloud droplets present are presumed to evaporate instantly as well, at least until the humidity again reaches 100%. Cloud water evaporation is also handled by the saturation adjustment technique.

The cloud droplets then slowly self-aggregate into larger rain drops in a process called *autoconversion*. In the Kessler (1969) parameterization, one of the very simplest treatments of cloud microphysics, it is assumed that when the cloud water mixing ratio, q_c, locally exceeds a certain threshhold value, q_{c0} (usually taken to be something like 1x10^{-3} g g^{-1}), then the cloud water mass exceeding that critical concentration converts to rain water at a specified constant rate k_1^{-1}. This can be expressed as:

$$A = \begin{cases} k_1 \left[q_c - q_{c0}\right] & q_c > q_{c0}; \\ 0 \qquad \text{otherwise.} \end{cases}$$

where A represents the amount of cloud water autoconverted. The autoconversion rate is typically presumed to be on the order of 15 min or so, so $k_1 \approx 0.001$ s^{-1} is usually adopted.

Rain water represents larger drops that not only evaporate relatively slowly in the presence of subsaturated air, but also fall relative to still air. The rain drops collect cloud droplets they encounter in their path; this process is termed *accretion*. Unlike cloud droplets, we can't really assume all rain drops are the same size and fall at the same rate. In any given grid box that contains rain water, we will have a mix of rain drop sizes, ranging from very small to huge. The size and mass of a rain drop determines not only how quickly it falls but also how large an area the drop sweeps out as it falls, which itself determines how many cloud droplets the rain drop will encounter and accrete.

Yet, we also can't track all the different possible sizes of rain drops separately. All we really will have is a value for the rain water mixing ratio at each individual grid point. Knowing the mass of rain water present in a particular box, we have to assume a relationship between the rain drop size and number concentration. That is, if we have 10 grams of rain water per kilogram of dry air (i.e., q_r = 10x10^{-3} g g^{-1}), some fraction of that mass will represent small drops, some fraction medium size drops, with the balance being large drops, all related in some simple (but hopefully not completely unrealistic) fashion. Then, we further simplify the problem by presuming we can treat *all* the rain drops as being represented by the mass-weighted average of all the drops that are presumed to exist in the grid box.

8.1.1 The Marshall-Palmer distribution

Marshall and Palmer (1948) studied the sizes of the rain drops they collected and plotted the number collected versus the drop diameter. They found that, except for the very smallest drop diameters, there was an exponential decrease in number concentration as diameter increased. Most microphysical parameterizations are based on this finding. Let N_D be the number of drops within a volume of given diameter D. The exponential dependence can be expressed as:

$$N_D = N_0 e^{-\lambda D}, \tag{8.1}$$

where N_0 is the distribution's intercept (the drop number concentration at essentially zero diameter) and $-\lambda$ is the slope of the distribution (see Fig. 1). λ is a positive number, so the number of drops at a given size decreases with increasing size.

The total number of drops of all sizes, N, is obtained by integrating (8.1) over the range of diameters, from zero to infinity. This, of course, represents the area under the curve depicted in Fig. 1. Thus:

$$\begin{aligned} N &= \int_0^\infty N_D dD \\ &= \int_0^\infty N_0 e^{-\lambda D} dD \\ &= -\frac{N_0 e^{-\lambda D}}{\lambda}\Big]_0^\infty \\ &= \frac{N_0}{\lambda}. \end{aligned} \tag{8.2}$$

However, we still need to figure out how to specify N_0 and λ.

The mass of an individual raindrop of diameter D [$\mathcal{M}_D$] is its density (ρ_l) times its physical volume ($\mathcal{V}_D$):

$$\mathcal{M}_D = \rho_l \mathcal{V}_D.$$

If the raindrop is spherical (not a bad choice), the volume dependence on diameter D or radius R is:

$$\mathcal{V}_D = \frac{\pi}{6} D^3 = \frac{4}{3} \pi R^3.$$

The total rain water mass, M, expressed per unit volume of space (as in a grid box) is simply the mass of a drop of diameter D, multiplied by the number of drops of that diameter present,

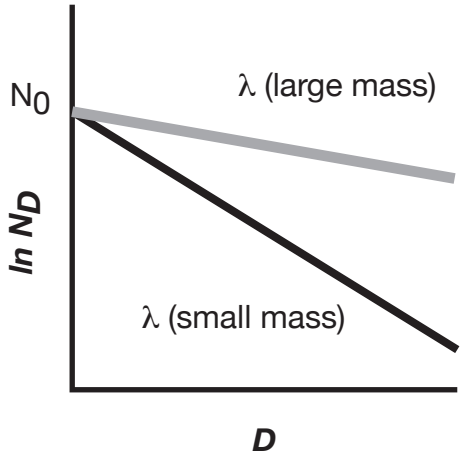

Figure 8.1: Exponential size distribution, showing dependence of slope (λ) on total rain water mass content.

summed over all possible diameters. This is expressed as:

$$\begin{aligned} M &= \int_0^\infty \mathcal{M}_D N_D dD \qquad (8.3) \\ &= \int_0^\infty \rho_l \frac{\pi}{6} D^3 N_0 e^{-\lambda D} dD \qquad (8.4) \\ &= \rho_l \frac{\pi}{6} N_0 \int_0^\infty D^3 e^{-\lambda D} dD. \end{aligned}$$

This can be a nasty integral to solve, but in this case there's a trick, owing to the integration limits of zero and infinity. We note that:

$$\int_0^\infty D^{n-1} e^{-\lambda D} dD = \frac{\Gamma(n)}{\lambda^n},$$

where Γ is the "gamma function" which has the property that

$$\Gamma(n+1) = n\Gamma(n),$$

and it is useful to keep in mind that $\Gamma(2)$=1.0. Applying this trick results in the total mass being expressed as:

$$M = \frac{\rho_l N_0 \pi}{\lambda^4}. \qquad (8.5)$$

Now, the total rain water per unit volume in the box is simply our grid point rain water value, $\bar{\rho} q_r$, which is presumed to be spread equally through the grid box. Using this, we can solve (8.5) for the slope parameter:

$$\lambda = \left[\frac{\rho_l N_0 \pi}{\bar{\rho} q_r} \right]^{\frac{1}{4}}. \qquad (8.6)$$

In most parameterizations, N_0 is taken to be fixed and known, but with different values for each species (such as raindrops, snow crystals, graupel particles and hailstones) that are presumed to be exponentially distributed with respect to size[1]. This means that the slope parameter directly depends upon the rain water content in the grid box ($\bar{\rho}q_r$), since that's the only quantity left to vary in (8.6). As the rain water content increases, the slope increases (becomes less negative, or closer to horizontal). Note that means incremental increases in rain content really comes from assumed growth in the number of large drops.

8.1.2 Terminal velocity

Picture a cap cloud sitting astride a mountain peak. These clouds form when stable air is forced to rise over the mountain barrier and can be remarkably persistent. That the cap clouds neither fall down upon the mountain nor trail off downstream tells us two important things. First, if the cloud has a fall, or settling, velocity, it must be very slow. But why can't gravity bring the cloud down? Second, we know that air must be flowing *through* the cloud, even though the cloud itself does not appear to be moving. This means that both condensation and evaporation of cloud water must be very rapid processes.

Now picture an isolated cumulus cloud that is heavily precipitating. Beneath the cloud base, you may be able to observe the precipitation shaft. These raindrops were formed within the cloud, but then fell out, and many of them will survive to reach the ground, even though the surrounding air is subsaturated (otherwise, cloud base wouldn't be *above* the ground). This tells us two other important things: that raindrops do have an appreciable fall speed with respect to still air, and that in contrast to cloud droplets, raindrops evaporate slowly.

Both cloud and raindrops are affected by gravity, which naturally wants to make them fall towards the ground. However, as an object (drop) starts to fall, it encounters frictional resistance from the air it has to pass through. The faster the object moves, the greater the air's drag on the object. This leads to a maximum possible fall speed, the *terminal velocity*. Terminal velocity is achieved when the drag force on the drop equals the opposing gravity force.

Let us first consider small droplets, like cloud droplets, and see why their final speed is so small that it is indeed negligible. For spherical droplets with small diameters, Stokes' drag law provides an expression for the drag force F:

$$F = 6\pi\mu R v. \tag{8.7}$$

In the above, μ is the viscosity of air and v is the object's velocity (in this case, directed downward). The gravity force acting on an object of mass $\mathcal{M}$ is simply $\mathcal{M}g$, where $g \approx 9.8$ m

[1]As examples, Marshall and Palmer determined $N_0 = 0.08$ cm^{-4} [8×10^6 m^{-4}] for their raindrops, and Lin et al. (1983) used 0.03 cm^{-4} for their snow species.

s^{-2}. Again, for a sphere of liquid water, the mass is $\frac{4}{3}\pi R^3 \rho_l$. Equating the drag and gravity forces yields the condition under which the velocity reaches its terminal value (v=V_D):

$$6\pi\mu R V_D = \frac{4}{3}\pi R^3 \rho_l g.$$

Solving for V_D, we find:

$$V_D = \frac{2R^2 \rho_l g}{9\mu}.$$

Taking air viscosity to be $\mu \approx 2\text{x}10^{-5}$ kg m^{-1} s^{-1} and a cloud droplet radius of about 10^{-5} m, along with $\rho_l = 10^3$ kg m^{-3}, yields a terminal velocity for cloud droplets of about 10^{-2} m s^{-1}. This is negligible even in still air, and is why we presume the cloud droplets are free-floating.

Stokes' law (8.7) above really only applies to small droplets. For larger drops, it has to be revised into the following:

$$F = 6\pi\mu R v \left[\frac{C_D Re}{24}\right], \tag{8.8}$$

where C_D is the (nondimensional) aerodynamic drag coefficient and Re is the Reynolds number, a ratio between inertial and viscous forces. At small Re, viscosity is dominant and $C_D Re \approx 24$, even though C_D itself is not constant. For large Re, viscosity cannot oppose the motion very effectively, and the drag coefficient C_D tends to approach a constant value of 0.4-0.6[2].

The Reynolds number may be written as:

$$Re = \frac{2V_D R\bar{\rho}}{\mu}$$

when v has reached V_D. Plugging this into (8.8), we find the new balance is

$$6\pi\mu R V_D \left[\frac{2C_D V_D R\bar{\rho}}{24\mu}\right] = \frac{4}{3}\pi R^3 \rho_l g.$$

Note V_D now appears twice on the LHS. Solving for the terminal velocity yields:

$$V_D^2 = \frac{8}{3}\frac{g}{C_D}\frac{\rho_l}{\bar{\rho}}R.$$

Thus, the terminal velocity depends on the square root of the drop size. Dropping a few constants, and replacing radius R with diameter D produces:

$$V_D = k g^{\frac{1}{2}} \left[\frac{\rho_l}{\bar{\rho}}\right]^{\frac{1}{2}} D^{\frac{1}{2}}, \tag{8.9}$$

[2]Lin et al. (1983) used $C_D = 0.4$ for rain and 0.6 for hail.

where k merely aggregates some of the constants. This is the equation we will be using for a single rain drop.

Equation (8.9) clearly shows that the terminal velocity of a drop is dependent upon the drop's diameter. We presume there is an entire range of rain drop diameters represented within the total mass of rain water in a given grid box. We can't track them all, so we hope we can get away with making all the drops fall at a uniform speed... determined by the mass weighted terminal velocity, which we'll call $\widehat{V}_T$. We need to take the terminal velocity — different for each diameter D — multiply it by the mass associated with drops of that diameter, multiply again by the number of drops we have for that particular diameter, and sum that over all the possible diameters. Then, we need to normalize this by the total mass, given by (8.3). That is to say:

$$\widehat{V}_T = \frac{\int_0^\infty V_D \mathcal{M}_D N_D dD}{\int_0^\infty \mathcal{M}_D N_D dD}.$$

The demoninator of the above evaluates to (8.5). The numerator reduces down, at first, to:

$$k \left[g \frac{\rho_l}{\bar{\rho}} \right]^{\frac{1}{2}} \frac{\pi}{6} \rho_l N_0 \int_0^\infty D^{\frac{7}{2}} e^{-\lambda D} dD.$$

We solve the integral with the help of the Gamma function. After dividing by (8.5) and simplifying, we find:

$$\widehat{V}_T = k \left[g \frac{\rho_l}{\bar{\rho}} \right]^{\frac{1}{2}} \Gamma(4.5) \lambda^{-\frac{1}{2}}, \tag{8.10}$$

our mass-weighted rain water terminal velocity. Note that it depends on the air density and the slope parameter (itself a function of rain water content).

The above equation was derived from first principles, and is used to compute the terminal velocity of some species in microphysics schemes, such as for graupel particles in the South Dakota scheme (Lin et al. 1983; hereafter "LFO"). Often, however, empirical equations constructed using curve fitting are also employed. As an example, LFO, among others, used a relationship like this for for the terminal velocity of a rain drop of diameter D:

$$V_D = a D^b \left[\frac{\bar{\rho}_0}{\bar{\rho}} \right]^{1/2},$$

where a and b are empirically determined constants, and $\bar{\rho}_0$ is the air density at the surface. A frequently used source for these constants is Locatelli and Hobbs (1974). The air density ratio is a fallspeed correction introduced by Foote and du Toit (1969), and permits terminal velocities to increase with altitude, other factors being equal.

8.1.3 Accretion of cloud droplets by rain water

Because rain drops fall relative to still air and cloud droplets do not, rain drops can encounter — and collect — cloud particles. Raindrops can encounter cloud droplets even in strong updrafts, where both kinds of condensed water are actually rising. The key point is that the difference between settling velocities for a given raindrop and cloud particle is the fall speed of the raindrop, and this is the encounter velocity for the drop.

A single raindrop with diameter D settling relative to still air (and cloud droplets) at speed V_D sweeps out a volume determined by the drop's cross-sectional area, which is $\frac{1}{4}\pi D^2$. During its fall (relative to cloud particles), it encounters cloud droplets which are distributed through the grid box with content $\bar{\rho}q_c$. Upon colliding with a cloud droplet, the droplet may stick to the raindrop, or it may not, depending on how sticky or "efficient" the drop is. This is quantified as the collection efficiency, ϵ_D.

Thus, the rate of mass increase for the raindrop of diameter D can be expressed as:

$$\frac{d\mathcal{M}_D}{dt} = \epsilon_D \frac{\pi D^2}{4} V_D \bar{\rho} q_c.$$

Integrate this expression over all raindrop sizes, and call it B (units $\mathrm{kg}_w\ \mathrm{kg}_{air}^{-1}\ \mathrm{s}^{-1}$) for convenience. This yields:

$$\begin{aligned} B &= \frac{d}{dt}\int_0^\infty \mathcal{M}_D N_D dD \\ &= \epsilon_D \frac{\pi}{4} k g^{\frac{1}{2}} \left[\frac{\rho_l}{\bar{\rho}}\right]^{\frac{1}{2}} \bar{\rho} q_c N_0 \int_0^\infty D^{\frac{5}{2}} e^{-\lambda D} dD. \end{aligned}$$

In the above, we used our equation for the terminal velocity of a single drop of diameter D.

Using the Gamma function yet again allows us to quickly solve the integral, producing:

$$B = \epsilon_D \frac{\pi}{4} k g^{\frac{1}{2}} \left[\frac{\rho_l}{\bar{\rho}}\right]^{\frac{1}{2}} \bar{\rho} q_c N_0 \left[\frac{\Gamma(3.5)}{\lambda^{\frac{7}{2}}}\right].$$

This is the way the equation is presented in LFO, and it shows that accretion depends on the cloud water mass available per unit volume ($\bar{\rho}q_c$). It also, logically, depends on the rain water content, which is hiding in the slope parameter. Owing to (8.6), we know that

$$\lambda^{\frac{7}{2}} = \left[\frac{\rho_l N_0 \pi}{\bar{\rho} q_r}\right]^{\frac{7}{8}}.$$

Using this in the equation for B and rearranging, we see that the accretion of cloud water by rain is

$$B \propto \epsilon_D N_0^{\frac{1}{8}} \bar{\rho} q_c q_r^{\frac{7}{8}} \left[\frac{\rho_l}{\bar{\rho}}\right]^{-\frac{3}{8}}.$$

The Kessler parameterization involves a simplified version of this expression, often rendered in models predicting mixing ratios as

$$B = k_2 q_c q_r^{\frac{7}{8}}, \tag{8.11}$$

with the accretion rate, k_2, usually taken to be 2.2. It is not clear this is a completely faithful representation of the Kessler formulation, however. First, note that in Kessler (1967, 1969) and Kessler and Bumgarner (1971), the prognostic equations for condensed water were written in terms of water content (specifically, g_w/m^3) instead of mixing ratios. The accretion rate coefficient (units $g_w/m^3/s$) was presented as

$$k_2' m_c M_r^{\frac{7}{8}}, \tag{8.12}$$

in which m_c and M_r are the cloud and rain water contents, and $k_2' = k_2 E N_0^{1/8}$, with k_2 = 6.96E-4 in unspecified units, E is an efficiency taken to be unity, and N_0 again is the intercept of the raindrop size distribution, adopted as 10^7 m^{-4}.

We need to ascertain the units of k_2' and rewrite Kessler's expression for condensate expressed as mixing ratios. Working backwards, we find the units of k_2' have to be $m^{7/8}$ $g_w^{7/8}$ s^{-1}, which with Kessler's specifications becomes $\hat{k}_2 = 2.2$ $m^{7/8}$ $kg_w^{7/8}$ s^{-1} (note notation change) after conversion to mks. The prognostic equations for cloud and rain water contents are, in part:

$$\frac{dM_r}{dt} = -\frac{dm_c}{dt} = \hat{k}_2 m_c M_r^{7/8}.$$

Unitwise, we know that $m_c = \bar{\rho} q_c$ and $m_r = \bar{\rho} q_r$. So substituting and also dividing by air density yields

$$\frac{dq_r}{dt} = -\frac{dq_c}{dt} = \hat{k}_2 q_c \left[\bar{\rho} q_r\right]^{7/8}. \tag{8.13}$$

which differs from the usual formulation (8.11) by the $\bar{\rho}^{7/8}$ term. Using (8.11) with $k_2 = 2.2$ essentially inflates the accretion rate because $\bar{\rho}^{7/8} < 1$. Thus, for our model, it appears we should formulate the accretion rate as

$$B = \hat{k}_2 q_c \left[\bar{\rho} q_r\right]^{7/8} \tag{8.14}$$

instead when $\hat{k}_2 = 2.2$. It may suffice to write $B = 2.2 \bar{\rho} q_c q_r^{7/8}$.

8.1.4 Evaporation of rainwater

A raindrop's mass will increase or decrease depending on whether vapor is diffusing towards or away from it. The direction in which vapor diffuses depends upon the diffusivity of vapor and the gradients in the vapor field. We can try to model this using a simple diffusion

equation, involving the second derivative of the vapor field. Let the vapor density (vapor mass per unit volume) be ρ_v, then a simple diffusion equation for vapor is:

$$\frac{\partial \rho_v}{\partial t} = \delta_v \nabla^2 \rho_v,$$

where δ_v is the diffusivity coefficient of vapor in air.

Consider a spherical drop of radius R. The Laplacian in this equation is three-dimensional, but by recognizing that the raindrop is spherical, and we're concerned with the vapor gradients radially outward from the drop, some simplification can be attained by casting the equation in spherical coordinates. Skipping some details, this leads to an expression relating the vapor density at some point r away from the drop $[\rho_v(r)]$ in terms of the vapor density far from the drop $[\rho_v(\infty)]$ and the vapor gradient in the radial direction between the drop's outer surface $[\rho_v(R)]$ and far away. This is:

$$\rho_v(r) = \rho_v(\infty) - \frac{R}{r}\left[\rho_v(\infty) - \rho_v(R)\right].$$

The time rate of change of the drop's mass $\mathcal{M}$ will depend upon the drop's surface area ($4\pi R^2$; over which evaporation is taking place), the vapor gradient radially away from the drop and the vapor diffusivity. This can be written as:

$$\frac{d\mathcal{M}}{dt} = 4\pi R^2 \delta_v \frac{d\rho_v}{dr}.$$

Using the previously obtained result allows us to rewrite this as:

$$\frac{d\mathcal{M}}{dt} = 4\pi R \delta_v \left[\rho_v(\infty) - \rho_v(R)\right].$$

This does not provide sufficient information to solve the problem. We need to make some assumptions. First, we presume that air very near the drop is saturated. We know that the saturation vapor density is a function (and a strong one, at that) of temperature T, which at the outer edge of the raindrop has the value $T(R)$. So

$$\rho_v(R) = \rho_{vs}\left[T(R)\right].$$

If a location farther away is subsaturated, such that $\rho_v(\infty) < \rho_{vs}(R)$, vapor will diffuse away form the drop and the drop's mass will be decreased as its liquid evaporates to vapor form. This causes latent cooling — or heat to be extracted from the surrounding air.

If we can assume that the latent cooling is drawn from the air by conduction (diffusion of heat in air), then we can equate the latent cooling rate owing to evaporation with the diffusion of heat through the air. The latent cooling is proportional to the drop's mass change, with

the proportionality being L_v, the latent heat of vaporization. The diffusivity of heat in air is quantified by δ_T, and equating the two yields:

$$L_v \frac{d\mathcal{M}}{dt} = 4\pi R \delta_T \left[T(R) - T(\infty) \right].$$

The RHS of the above was constructed via analogy with vapor diffusion. The left side of the expression is the cooling rate owing to drop mass loss; the right side shows that the heat required to evaporate the vapor comes from the surrounding air via diffusion.

The key points in the above analysis are that evaporation depends upon the difference in vapor density near and away from the drop and the ability of both vapor and heat to diffuse through the air. By combining all of the above expressions, it can be shown that this analysis leads to an evaporation rate equation of the form:

$$\frac{d\mathcal{M}}{dt} = \frac{4\pi R(S-1)}{fcn(\delta_T) + fcn(\delta_v)},$$

where S is the relative humidity (expressed as a fraction), and $S < 1$ means subsaturation. So, in that case $S - 1 < 0$ and the drop mass decreases.

This equation has to be integrated over all drop sizes, of course. Additional complexity arises if the drop is evaporating while also falling. The rush of air past an evaporating drop "ventilates" the drop and increases its mass loss. Kessler expressed the evaporation rate, E, in the following way:

$$E = \frac{1}{\bar{\rho}} \left[\frac{(1 - \frac{q_v}{q_{vs}}) C_{vent} (\bar{\rho} q_r)^{0.525}}{2.03 \times 10^4 + \frac{9.58 \times 10^6}{\bar{p} q_{vs}}} \right], \tag{8.15}$$

where C_{vent} is the ventilation factor

$$C_{vent} = 1.6 + 30.39 (\bar{\rho} q_r)^{0.2046},$$

and $\bar{p}$ is the mean pressure in Pa. (The preceding differs from Kessler; I've converted the units into the *mks* system.)

8.1.5 Reflectivity

In this subsection, we take the opportunity to generalize our discussion to include other precipitation species other than raindrops (i.e., frozen particles such as snow, graupel and hail). These will also be presumed to be exponentially distributed, with the distribution for water species x given by

$$n(D_x) = N_{0x} e^{-\lambda_x D_x}, \tag{8.16}$$

where $n(D_x)$ is the number of particles of size D_x, and N_{0x} and λ_x are the intercept and slope of the particle distribution, respectively. The mass of an individual particle of density

ρ_x, presumed spherical, is

$$m(D_x) = \frac{1}{6}\pi\rho_x D_x^3. \tag{8.17}$$

As seen before, the total number of particles N_x is obtained by integrating (8.16) over the entire range of sizes, as in

$$N_x = \int_0^\infty n(D_x)dD_x = \frac{N_{0x}}{\lambda_x}, \tag{8.18}$$

and total mass M_x is obtained in an analogous fashion as

$$M_x = \int_0^\infty m(D_x)n(D_x)dD_x, \tag{8.19}$$

which yields

$$M_x = \frac{\pi\rho_x N_{0x}}{\lambda_x^4}. \tag{8.20}$$

The total mass per unit volume, presumed spread equally through the volume, is $M_x = \bar{\rho}q_x$, where $\bar{\rho}$ is the air density and q_x is the mixing ratio of species x. Thus, (8.20) may be solved for the slope parameter

$$\lambda_x = \left[\frac{\pi\rho_x N_{0x}}{\bar{\rho}q_x}\right]^{\frac{1}{4}}. \tag{8.21}$$

Often, though not always, the intercept N_{0x} is taken to be constant, making the slope parameter specified uniquely by the species mass content. That assumption, though convenient, is in itself questionable (Cotton and Anthes 1989), and so-called double moment parameterizations (e.g., Ferrier 1994) have arisen in part owing to these concerns. Still, in LFO, the intercept for snow (N_{0s}) was fixed at 0.03 cm^{-4}; many studies followed their lead. Snow density is often considered to have a very low density, on the order of 0.1 g cm^{-3}.

For frozen water species, the question arises whether the diameters in the foregoing expressions should represent *actual* or *melted* diameters. Unfortunately, there appears to be considerable confusion between the two in the literature, and this can make a sizable difference with respect to particles such as low density snow. As an example, LFO's formulation employed actual diameters in their derivations of fallspeeds and conversion rates, but they appear to have directly relied on Gunn and Marshall's (1958) work that employed melted diameters without appreciating the distinction. Two papers by Passarelli (1978a,b) discuss the difference between the two.

Let us consider the issue of melted vs. actual diameters in a little more detail. Let the *melted* diameter of a frozen particle be designated as $\hat{D}_x$. The exponential distribution of melted diameters (8.16) is now written as

$$n(\hat{D}_x) = \hat{N}_{0x}e^{-\hat{\lambda}_x\hat{D}_x}, \tag{8.22}$$

where intercept and slope parameters for the distribution of melted particles are being used. An equivalent distribution based on actual diameter D_x should logically be subject to two

constraints: the distributions contain the same number of particles, and represent the same total mass. These constraints yield relationships between the actual and melted diameter distributions' slopes and intercepts given by

$$\frac{\hat{N}_{0x}}{\hat{\lambda}_x^4}\rho_x = \frac{N_{0x}}{\lambda_x^4}\rho_w \tag{8.23}$$

$$\frac{\hat{N}_{0x}}{\hat{\lambda}_x} = \frac{N_{0x}}{\lambda_x}, \tag{8.24}$$

where ρ_w is the density of liquid water. Two further relationships may be identified. If the particles are spherical, then $\rho_w \hat{D}_x^3 = \rho_x D_x^3$. Also, $n(\hat{D}_x)d\hat{D}_x = n(D_x)dD_x$. These were also invoked by Passarelli (1978a).

The preceding results in these relationships between actual and melted particle distribution parameters:

$$\lambda_x = \hat{\lambda}_x \left(\frac{\rho_x}{\rho_w}\right)^{\frac{1}{3}}, \tag{8.25}$$

and

$$N_{0x} = \hat{N}_{0x} \left(\frac{\rho_x}{\rho_w}\right)^{\frac{1}{3}}. \tag{8.26}$$

Passarelli (1978b) presented these relationships in his equation (14). This means that if the slope of the *melted* snow particle exponential distribution was $\hat{N}_{0s} = 0.03$ cm^{-4}, as often cited, the slope of the equivalent distribution based on actual particle diameters would be 0.014 cm^{-4}. That difference might not seem like much, but by itself it is sufficient to alter accretion of cloud water by snow rates by more than 10%.

Additionally, and perhaps more seriously, values for intercepts employed in simulations of warm season deep convection are largely drawn from observations derived from very different weather phenomena, such as frontal and mountain wave clouds, and winter weather events. Samples from deep convection are not all that common, but we note that Musil et al. (1976) found snow intercept (N_{0s}) values far smaller than what appears to have become the "standard" value. It might make more sense to adopt the double-moment that effectively renders the distribution intercept into a prognostic variable. This adds a large amount of complexity and expense to the problem, however.

Now we turn to the main topic of this subsection. The reflectivity Z_x of a particular species of water is a function of the sixth power of the particle diameter D_x is

$$Z_x = \int_0^\infty D_x^6 n(D_x) dD_x. \tag{8.27}$$

For ice species, the equivalent reflectivity $Z_{ex} = \alpha Z_x$ where α is the ratio of dielectric constants of ice and water. For water, $Z_{ex} = Z_x$. After equivalent reflectivities for each species are summed (forming Z_e), total reflectivity in decibels is computed as $10\log_{10}(Z_e)$.

We have not yet specified whether actual or melted particle diameters are being used. Dye et al. (1974) note that "backscattering is proportional to the mass of the particle so that the measured diameter must be corrected for the density". Therefore, we proceed under the assumption Z_x is based on *melted* diameters. Using $\hat{D}_x$ in place of D_x in (8.27) and noting that $n(D_x)dD_x = n(\hat{D}_x)d\hat{D}_x$, we have

$$Z_x = \int_0^\infty \hat{D}_x^6 n(\hat{D}_x) d\hat{D}_x = \Gamma(7)\frac{\hat{N}_{0x}}{\hat{\lambda}_x^7}, \tag{8.28}$$

where Γ is the Gamma function. Substitution of the actual particle distribution [i.e., employing (8.25) and (8.26)] results in

$$Z_x = \int_0^\infty D_x^6 n(D_x) dD_x = \Gamma(7)\frac{N_{0x}}{\lambda_x^7}\left(\frac{\rho_x}{\rho_w}\right)^2. \tag{8.29}$$

This functional form was presented by Passarelli (1978a) and used by Fovell and Ogura (1988), among others.

For compatible slope and intercept values, (8.28) and (8.29) should – and do – result in the same values. For a given melted diameter intercept $\hat{N}_{0x}$, the equivalent N_{0x} is smaller, increasing the magnitude of λ_x^{-7}. However the correction factor $\left(\frac{\rho_x}{\rho_w}\right)^2$ compensates for this. Therefore, with identical values of Z, the Z_e values are also the same. According to Smith (1984), the appropriate value of α in this case is 0.224 [from his equation (8)] since the reflectivity was effectively based on melted diameters[3].

Though it may not be obvious at first sight, this strategy for computing ice equivalent reflectivities is consistent with that advanced by Smith (1984). In that paper, two strategies for computing Z_e from Z were presented [Smith's equations (8) and (10)], depending on whether melted or actual particle diameters were employed in the calculation of Z. This puts the effective correction factor in the dielectric constant rather than in the computation of Z itself. Smith presumed a value of 0.92 g cm^{-3} for ice, so his correction term would be $(0.92)^2 = 0.846$, the ratio between the diaelectric constants for his (8) and (10). The present treatment is more general as it does not presume a specific value for ice species density ρ_x. The density of snow is usually taken to be much smaller than that.

From the preceding it is suggested that reflectivity should be based on melted diameters, and if actual diameters are used, the correction factor $\left(\frac{\rho_x}{\rho_w}\right)^2$ should appear. This indicates that Fovell and Ogura's (1988) equation (1) [with α=.224] is appropriate when actual particle diameters are used in the calculation, but not when melted diameters are used. There are examples in the literature of studies that used the correction term along with the melted diameters. That combination produces nice (and realistic!) reflectivities, but appears to

[3]Fovell and Ogura (1988) used 0.213, itself ultimately based on the widely propagated error discussed by Smith (1984).

be based on an inconsistency. Improper inclusion of the term results in reflectivities that are fully 20 dBZ lower (since $10\log_{10}\left(\frac{\rho_s}{\rho_w}\right)^2$ = -20 when ρ_s = 0.1 g cm^{-3}). Models tend to create overly large reflectivities, but there must be something else (such as excessively large conversion rates and/or inappropriate size distribution intercepts) to blame.

8.2 The equations with moisture and diffusion added

Thus, our equations (in flux form; see Model Task #5) are now:

$$\frac{\partial u}{\partial t} = -\frac{\partial uu}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} uw}{\partial z} - c_{pd}\bar{\theta}_v\frac{\partial \pi'}{\partial x} + K_x\frac{\partial^2 u}{\partial x^2} + K_z\frac{\partial^2 (u-\bar{u})}{\partial z^2} \tag{8.30}$$

$$\frac{\partial w}{\partial t} = -\frac{\partial uw}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} ww}{\partial z} - c_{pd}\bar{\theta}_v\frac{\partial \pi'}{\partial z} + g\left[\frac{\theta'}{\bar{\theta}} + .61q_v' - q_L\right] + K_x\frac{\partial^2 w}{\partial x^2} + K_z\frac{\partial^2 w}{\partial z^2} \tag{8.31}$$

$$\frac{\partial \theta'}{\partial t} = -\frac{\partial u\theta'}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} w\theta'}{\partial z} - w\frac{d\bar{\theta}}{dz} + \frac{L_v}{c_{pd}\bar{\pi}}\left[C-E\right] + K_x\frac{\partial^2 \theta'}{\partial x^2} + K_z\frac{\partial^2 \theta'}{\partial z^2} \tag{8.32}$$

$$\frac{\partial \pi'}{\partial t} = -\frac{\bar{c_s}^2}{\bar{\rho} c_{pd}\bar{\theta}_v^2}\left[\bar{\rho}\bar{\theta}_v\frac{\partial u}{\partial x} + \frac{\partial \bar{\rho}\bar{\theta}_v w}{\partial z}\right] + K_x\frac{\partial^2 \pi'}{\partial x^2} + K_z\frac{\partial^2 \pi'}{\partial z^2} \tag{8.33}$$

$$\frac{\partial q_v'}{\partial t} = -\frac{\partial uq_v'}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} wq_v'}{\partial z} - w\frac{d\bar{q_v}}{dz} + K_x\frac{\partial^2 q_v'}{\partial x^2} + K_z\frac{\partial^2 q_v'}{\partial z^2} - C + E \tag{8.34}$$

$$\frac{\partial q_c}{\partial t} = -\frac{\partial uq_c}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} wq_c}{\partial z} + K_x\frac{\partial^2 q_c}{\partial x^2} + K_z\frac{\partial^2 q_c}{\partial z^2} + C - A - B \tag{8.35}$$

$$\frac{\partial q_r}{\partial t} = -\frac{\partial uq_r}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho}\left[w - \widehat{V}_T\right] q_r}{\partial z} + K_x\frac{\partial^2 q_r}{\partial x^2} + K_z\frac{\partial^2 q_r}{\partial z^2} + A + B - E, \tag{8.36}$$

where q_L is the total condensed water mixing ratio ($q_c + q_r$), and K_x and K_z are the computational diffusion coefficients in the x and z directions, respectively. In the above, we have:

- Restored virtual temperature to the pressure gradient acceleration terms, the w equation's buoyancy term, and to the π' equation;
- Added an equation for q_v' that mimics the handling of θ';
- Added equations for q_c and q_r;
- Added a term in the vertical advection of q_r to account for the terminal velocity of rain water ($\widehat{V}_T$);
- Revised the w equation's buoyancy term to make it simpler to calculate (see below);
- Added the buoyancy drag owing to condensed water in the buoyancy term;

- Added terms governing condensation/evaporation of vapor (C) into the temperature and moisture substance equations (including simplifications discussed in Appendix A);
- Added terms governing the conversion of cloud water to rain water (A) and accretion of cloud water by rainwater B into the q_c and q_r equations;
- Added terms handling the evaporation of rain water (E) into the temperature and q_r equations;
- Added computational diffusion terms to restrain nonlinear instability (see Chapter 7).

8.2.1 Some comments on the above equations

Note that the computational diffusion is applied solely to *perturbations from the mean state*, not the mean state itself. That doesn't matter for horizontal diffusion, since the mean state is a function only of height. However, since computational diffusion is completely artificial, we don't want it to be altering the base state of the atmosphere.

The equations above are written in terms of perturbation potential temperature, vapor and pressure, but in terms of the full field u. This is an arbitrary choice. Note on the right sides of the equations, we often have to use perturbations of temperature, pressure and moisture (as in the pressure gradient and buoyancy terms, for example), but we only need perturbation u only in the u equation's vertical diffusion term. If we solved explicitly for u' instead of full u, we'd have to constantly add the mean state back in when doing advection. Conversely, if we solved for full θ or π, for example, we'd have to be constantly subtracting out the mean state. So, it's just easier to do it the way shown.

We saw in Model Task # 4 in particular that negative values can be produced in the wake of an originally positive definite initial condition. In the same way, advection of water substance can create unphysical negative mixing ratios. These need to be handled somehow, prior to computing microphysics. There are several ways this can be done (see also Model Task #6), the simplest being setting negative values of q_c, q_r and $\bar{q}_v + q'_v$ to zero. That, however, represents a spurious source of moisture in the model. Aside from implementing more sophisticated, strictly positive-definite advection schemes, another way might be to compute the total domain mass content of negative and positive water substance (for species x, this would be the sum of $\bar{\rho} q_x \Delta x \Delta z$ over all real grid points) where $q_x > $ and < 0, respectively, zero all negative q_x values, and reducing each positive q_x value by the ratio of the negative and positive mass contents. That way, negative mixing ratios are removed but the total domain mass content of q_x is not changed.

The cloud microphysics is reflected on the equations' right hand sides by autoconversion and accretion of cloud water (A and B), which are sinks for cloud water and sources for

rain water; condensation/evaporation of vapor (C), which is a sink for vapor and a source for cloud water when $C > 0$; and rainwater evaporation (E), which is a sink for rain and a source for vapor. Terms C and E involve latent heat release or absorption, and so these terms also show up in the θ' equation.

Recall that raindrops are presumed to fall at their mass-weighted terminal velocity. Thus, we need an additional mass flux term in the q_r equation. This term is shown combined with vertical advection to try to reduce computational errors. Consider an updraft. If the vertical advection and fall velocity terms were computed separately, we could have one term forcing rain water to rise some distance, and then the other term forcing some of the rain to fall back down some distance. It makes sense to combine these terms, if it is practical (code-wise) to do so.

Now we need to consider the saturation adjustment. We already utilized the cloud model's adjustment technique in computing the CAPE as part of Model Task #2. In the model time stepping, the vapor-cloud saturation adjustment is performed last, because it is presumed to operate instantaneously. First, advection of vapor and condensed water, including the terminal velocity term for rainwater, is computed. The microphysical conversions (autoconversion and accretion of cloud water, and rain evaporation) are also calculated. This yields a new forecast at time n+1 for vapor, q_c and q_r.

This new state may be subsaturated or supersaturated with respect to vapor. As the last step, the time n+1 forecast for vapor and cloud water are readjusted. If the grid box is supersaturated with respect to vapor, cloud water is produced. If the grid box is subsaturated and cloud water is present, some (or all) of the cloud droplets are evaporated. Doing this last insures that the forecasted field is not thermodynamically inconsistent.

8.2.2 Rewriting the buoyancy term

The w equation's buoyancy term made use of the following set of approximations. We should have had $\frac{\theta'_v}{\bar{\theta}_v}$ there. Virtual temperature is:

$$\theta_v = \theta(1 + .61 q_v)$$

and thus for the mean state it is

$$\bar{\theta}_v = \bar{\theta}(1 + .61 \bar{q_v}).$$

Apply logs to the former expression, yielding

$$\ln \theta_v = \ln \theta + \ln(1 + .61 q_v).$$

Recall that $\ln(1+x) \approx x$ when $x << 1$. Since the vapor mixing ratio is pretty small, $.61q_v << 1$ and thus $\ln(1+.61q_v) \approx 0.61q_v$. Therefore,

$$\ln \theta_v \approx \ln \theta + .61q_v.$$

Now perform a perturbation analysis on the above equation. In the usual fashion, we first get

$$\ln\left[\bar{\theta}_v\left[1+\frac{\theta'_v}{\bar{\theta}_v}\right]\right] \approx \ln\left[\bar{\theta}\left[1+\frac{\theta'}{\bar{\theta}}\right]\right] + .61\bar{q_v} + .61q'_v.$$

Again using $\ln(1+x) \approx x$, we find that

$$\ln \theta_v + \frac{\theta'_v}{\bar{\theta}_v} \approx \ln \bar{\theta} + \frac{\theta'}{\bar{\theta}} + .61\bar{q_v} + .61q'_v.$$

Subtract out the mean state, and we're left with:

$$\frac{\theta'_v}{\bar{\theta}_v} \approx \frac{\theta'}{\bar{\theta}} + .61q'_v,$$

which we used in (8.31) above. The condensed water appears in the buoyancy term owing to the drag law.

8.2.3 The latent heating term in the θ' equation

The latent heating or cooling due to condensation and evaporation appears in the equation for perturbation potential temperature (8.32) as

$$\frac{L_v}{c_{pd}\bar{\pi}}\left[C - E\right],$$

where C is the condensation of vapor ($C > 0$ for vapor to cloud water transformation), E is the evaporation of rain water and L_v is the latent heat of vaporization. This simple expression is the end product of a long chain of assumptions and approximations. This term is derived below.

The first law of thermodynamics may be written as:

$$\delta Q = c_v d(mT) + pdV, \tag{8.37}$$

where c_v is the specific heat at constant volume, V is volume, p is the total air pressure and T is temperature. We recognize that total pressure is the sum of the partial pressures of dry air, p_d, and water vapor, e, and that both dry air and water vapor are ideal gases. The total mass, m, is the sum of the dry air and vapor masses, m_d and m_v, as well as the mass

of condensed liquid, m_l. Heat exchange is due to a change in the vapor mass, with vapor decrease resulting in heat release. That is:

$$\delta Q = -L' dm_v$$

where L' is the latent heating factor.

We expand the RHS of (8.37), producing

$$-L' dm_v = c_{vd} d(m_d T) + c_{vv} d(m_v T) + c_l d(m_l T) + p_d dV + e dV. \tag{8.38}$$

In the above, c_{vd} and c_{vv} are the constant volume specific heats of dry air and water vapor, and c_l is the specific heat of liquid water. We recognize that the dry air mass, m_d, is constant. Also, since both dry air and water vapor are ideal gases, and the dry air and vapor mixture occupy the same volume and have the same temperature, we can write their respective gas laws as:

$$\begin{aligned} p_d V &= m_d R_d T \\ eV &= m_v R_v T, \end{aligned}$$

where R_d and R_v are the dry air and vapor gas constants. Differentiating these two gas laws and rearranging them yields:

$$\begin{aligned} p_d dV &= -V dp_d + m_d R_d dT \\ e dV &= -V de + m_v R_v dT + R_v T dm_v, \end{aligned}$$

since m_d is constant.

These expressions are useful in rewriting (8.38) which, after expansion, produces:

$$\begin{aligned} -L' dm_v &= c_{vd} m_d dT + c_{vv} m_v dT + c_{vv} T dm_v + c_l m_l dT + c_l T dm_l \\ &\quad -V dp_d + m_d R_d dT - V de + m_v R_v dT + R_v T dm_v \ . \end{aligned} \tag{8.39}$$

Recall that $c_{vd} + R_d = c_{pd}$ for dry air, and $c_{vv} + R_v = c_{pv}$ for vapor. Also, the vapor loss is the condensed liquid's gain, so $dm_l = -dm_v$. This brings us to:

$$-L' dm_v = c_{pd} m_d dT + c_{pv} m_v dT + c_l m_l dT + (c_{pd} - c_l) T dm_v - V dp.$$

In the above, we recombined the dry air and vapor partial pressures back into total pressure in the RHS' last term.

Now divide the above by the dry air mass, m_d. We define the mixing ratio for a given substance as the mass of the substance divided by the dry air mass. Thus, q_v, the vapor mixing ratio, is $\frac{m_v}{m_d}$, and the liquid water mixing ratio is $q_l = \frac{m_l}{m_d}$. After rearrangement, we have:

$$-\left[L' + (c_{pv} - c_l) T\right] dq_v = c_{pd} dT + c_{pv} q_v dT + c_l q_l dT - \alpha_d dp \tag{8.40}$$

where α_d is $V\ m_d^{-1}$. Let

$$L_v = L' + (c_{pv} - c_l)T \tag{8.41}$$

be defined as the *latent heat of vaporization*. The factor L' is nearly constant, but L_v is obviously a linear function of temperature.

We can perform further rearrangement with the help of the ideal gas law, starting with

$$\begin{aligned} pV &= mRT \\ &= m_d R_d T + m_v R_v T \\ p\alpha_d &= R_d T + q_v R_v T \\ \alpha_d &= \frac{R_d}{p}\left[1 + \frac{R_v}{R_d} q_v\right] T. \end{aligned}$$

We had divided through by m_d, and produced an expression for α_d that may be substituted into (8.40) above. After also dividing by T and collecting some terms, we come to:

$$\begin{aligned} -\frac{L_v}{T} dq_v &= c_{pd}\left[1 + \frac{c_{pv}}{c_{pd}} q_v\right] d\ln T \\ &\quad -R_d\left[1 + \frac{R_v}{R_d} q_v\right] d\ln p + c_l q_l d\ln T. \end{aligned} \tag{8.42}$$

To use the above in our θ equation, we need to get it in terms of potential temperature. We recall potential temperature is defined as:

$$\theta \equiv T\left[\frac{1000}{p}\right]^{\frac{R_d}{c_{pd}}}$$

where p is the total pressure. It may not have been obvious that this was the right or best definition of θ, but actually this particular derivation does show this to be the case. Expand the above and differentiate to yield:

$$c_{pd} d\ln\theta = c_{pd} d\ln T + R_d d\ln p.$$

Using this in (8.42), and rearranging yet again, leads us to our final, unapproximated moist adiabatic thermodynamic equation:

$$\begin{aligned} c_{pd} d\ln\theta &= -\left[1 + \underbrace{\frac{R_v}{R_d} q_v}_{I}\right]^{-1} \\ &\quad \left[\underbrace{\frac{L_v}{T} dq_v}_{A} + \left[\underbrace{c_l q_l d\ln T}_{B} + \underbrace{c_{pd} q_v \left[\frac{c_{pv}}{c_{pd}} - \frac{R_v}{R_d}\right] d\ln T}_{C}\right]\right]. \end{aligned} \tag{8.43}$$

We can do a scale analysis on (8.43), using some reasonable values for the variables involved. The vapor and dry air gas constants are about 452 and 287 J kg^{-1} K^{-1}, respectively, and $L_v \approx 2.5 \times 10^6$ J kg^{-1}. Let T be 300 K, and allow 10 g kg^{-1} of vapor to condense, meaning -$dq_v = dq_l = 10^{-2}$ kg kg^{-1}. Take the warming resulting from this to be about 25 K. Then, we see Term I is about 0.16, making a small but potentially significant contribution to the first term on the RHS (because it is being added to 1). The dominant term is clearly Term A, which evaluates to roughly 83. Terms B and C are relatively much smaller, being 3.5 and 0.2 respectively, dwarfed by Term A.

If we elect to neglect Term I as well as Terms B and C, we find, after rearrangement

$$d\theta = \frac{L_v \theta}{c_{pd} T} dq_v.$$

Using the definition $T{=}\theta\pi$, and differentiating with respect to time gives us

$$\frac{d\theta}{dt} = \frac{L_v}{c_{pd}\pi} \frac{dq_v}{dt}.$$

The time rate of change of vapor mass (due to condensation of vapor or evaporation of liquid) yields latent heating or cooling of approximately the specified amount. In (2.5), the rate of vapor mass change was written as $C - E$ (where C is now condensation or evaporation of cloud water), which we obtain from doing the saturation adjustment. It is not uncommon to take L_v as a constant, neglecting the previously mentioned temperature dependence.

Actually, the term in (2.5) contains one last assumption. $\bar{\pi}$ is used in place of π because we linearized the nondimensional pressure. Hopefully, it is seen that we made quite a few approximations to reach that simple form used in (8.32).

Part II

Model Tasks

Chapter 9

Model Task #0: Solving the 1D wave equation

In this task, we will program the 1D linear wave equation in a domain with periodic boundaries. This task will illustrate the basics of model initialization, time stepping, application of boundary conditions (BCs), amplitude and phase errors and their dependence on wavelength and time step, and avoidance of roundoff errors. This is called Model Task #0 in part because it does not directly lead to the cloud model.

For simplicity, the example code developed herein prints out data that is copied into Excel for plotting. Excel will not be useful for subsequent model tasks.

9.1 The upstream or upwind scheme

9.1.1 The scheme

As discussed in Chapter 4, the 1D wave equation can be written as

$$u_t = -cu_x, \tag{9.1}$$

where c is a constant wave speed and the wave translates to the right when $c > 0$. The upstream approximation is forward in time and upwind in space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} - c\left[\frac{u_i^n - u_{i-1}^n}{\Delta x}\right], \tag{9.2}$$

which in this form is only valid when c is non-negative. As there is only one unknown on the RHS, (9.2) can be written in explicit form as

$$u_i^{n+1} = u_i^n - \frac{c\Delta t}{\Delta x}\left[u_i^n - u_{i-1}^n\right]. \tag{9.3}$$

9.1.2 Implementation

The upstream method is a two time-level scheme, and we will call the present value u and forecast value up. We will also create and track the true solution, $uexact$. All three are dimensioned NX. At the start of the program, we should initialize parameters and constants such as NX, current simulation time $time$, time index n, time step dt (Δt), space index i, grid spacing dx (Δx), and trigonometric π, etc., and create arrays u, up, and $uexact$.

Our first case will be initialized with a single, rightward-moving wave of specified wavelength such that an integral number of waves fits in the domain. The lateral boundaries will be periodic, so the wave will pass out the right side and enter back in on the left. The exact solution will perfectly preserve its original shape and return to its initial position within the domain in a certain number of time steps. We will compare our numerical solution to this truth.

We will take the initial time as 0 seconds, so the time index n starts at 0. We need to provide the initial condition for u for all i. As a precaution, future value up should be initialized with zeroes for all i. **In the example below, the space index i runs from 1 to NX, inclusive.** This indexing conforms with standard Fortran practice, in which an array dimensioned NX has array elements numbered $1 \leq i \leq NX$. Thus, $i = 1$ and $i = NX$ will represent boundary points. Languages such as Python and C++ use zero-based indices, so the NX dimensioned array elements range from $0 \leq i \leq NX$-1.

Once this is completed, the main program loop consists of these actions:

- Increment the time index n. Thus, the first forecast corresponds to $n = 1$.
- Use the upstream scheme to create up for interior points $1 < i < NX$, overwriting the previously stored values.
- Apply the boundary conditions on up at $i = 1$ and $i = NX$.
- Set for the next time step. This involves transferring the contents of up into u for all i, and updating the current time. One obvious but poor way of doing the latter is $time = time + dt$ (see below).
- Compute the exact solution $uexact$.
- Write outputs for u and $uexact$, if appropriate.
- If $time < timend$, loop; otherwise, exit.

Trigonometric π can be easily computed to machine precision with

$$trigpi = 4. * \arctan(1.0).$$

I call this variable *trigpi* to differentiate it from the model's nondimensional pressure variable, π. Simply adding dt to *time* every time step can cause roundoff errors to accumulate. Over a long integration, compare the results of these two lines of code:

```
time1 = time1 + dt
time2 = float(n)*dt
```

Especially for certain non-integer values of dt, significant differences will emerge.

For periodic boundaries (in Fortran convention), the NX-2 points between 2 and NX-1, inclusive, will be termed the *real* points. The remaining ("fake") points serve to facilitate handling of the boundary points. After *up* is computed, the boundary conditions can be applied like this:

```
u(1) = u(nx-1)
u(nx) = u(2)
```

At this point, all entries in *up* have been supplied with information. When setting for the next time step, simply transfer the entire contents of *up* into u. When handled in this manner, we do not need to treat the real u points closest to the boundaries separately or specially when computing *up* for the next time step.

9.1.3 Test problem

For our test problem, set $c = 1.0$ m/s, $dt = 1.0$ s, and $dx = 1.0$ m, so $c' \equiv \frac{c\Delta t}{\Delta x} = 1.0$. Let $NX = 52$, so there are 50 real points, make the initial wavelength to be $L = 50\Delta x$, and take $timend = 50$ s. This means we will execute the model for precisely one revolution, and should obtain the modeled and exact solutions as shown in Fig. 9.1. As revealed by Fig. 4.3, the upstream scheme has no significant error for this configuration.

We can craft our initial condition for u using code like this

```
       wavelength = 50  ! i.e., 50dx
       amp = 1.0        ! amplitude
       do i=2,nx-1      ! Loop over real points
        xi=float(i-2)
        u(i) = amp*sin(2*xi*trigpi/wavelength)
       enddo
! enforce periodic boundary conditions
       u(1)=u(nx-1)
       u(nx)=u(2)
```

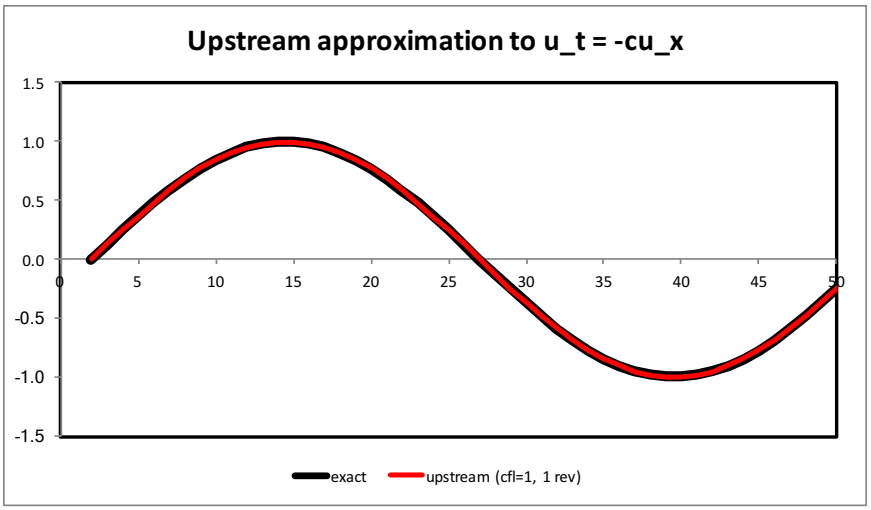

Figure 9.1: Upstream scheme solution (red curve) for test problem ($50\Delta x$ wave with c$'$ = 1.0) after one revolution, shown with exact solution (black curve).

The exact solution for u at time step n can be computed using

```
       do i=2,nx-1
! exact solution at any time n
         xei = float(i-2)-c*n*dt
         uexact(i) = amp*sin(2*xei*trigpi/wavelength)
       enddo
```

9.1.4 Errors

The exact solution of the linear wave equation translates precisely at speed c without change of shape or amplitude. Even if the initial condition consists of a set of waves with different wavelengths and amplitudes, the combined waveform translates as a coherent unit because each component's amplitude is preserved and they share the same phase speed. As discussed earlier, *amplitude* error occurs in this case if the wave's original amplitude is not preserved and *phase* error occurs if the simulated wave translates too quickly or slowly. Figure 4.3 showed that, for the upstream scheme, amplitude and phase errors were a function of wavelength and c'.

Specifically, that figure revealed that simulated wave amplitude decays when $c' < 1$ and grows when $c' > 1$, with shorter waves always handled worse than longer ones. Both the decay and growth are exponential, which means for $c' < 1$, the wave is disappearing and for $c' > 1$ the model is unstable. Counterintuitively, sampling the wave propagation better (i.e., selecting $c' < 1$) actualy makes the scheme worse. For $c' < 1$, the poorest performance for

all wavelengths is at $c' = 0.5$. At that c' value, the amplitude factor for $4\Delta x$ waves is 0.7, which means the wave would lose 30% of its amplitude *every time step*. That wave would not persist very long.

The exact solution is nondispersive but the upstream scheme is not since simulated phase speeds vary with wavelength. Some wavelengths move faster than c while most move more slowly. For long wavelengths, the phase error is small. For our test problem, we selected $c' = 1$, so there was no amplitude error, and $L = 50\Delta x$, so the phase error was very small and would only emerge from much longer time integrations.

9.1.5 Upstream scheme experiments (Model Task #0A)

We will manipulate c' by altering the time step. Conduct these experiments:

1. Try values of $dt < 1.0$ for the $50\Delta x$ wave. Observe the amplitude and phase error after one revolution. Compare your results to Fig. 4.3.

2. Try values of $c' > 1$, by increasing dt. How does the wave amplitude change after one revolution? What happens if you intergrate the model longer? This is linear instability.

3. Change the initial condition's wavelength to something smaller, like $5\Delta x$ (wavelength = 5), and set $c' = 0.5$. From Fig. 4.3, we expect to lose about 20% of the amplitude per time step. Does that happen? Plot the maximum wave value vs. time[1].

4. Compare simulations for the $50\Delta x$ wave with $c' = 0.5$ and 0.25. Run each simulation for 50 seconds. According to Fig. 4.3, the amplitude error is supposed to be worse for $c' = 0.5$. Is it? If not, why not? (Note a 50 sec integration requires 100 time steps with $dt = 0.5$ second, and 200 time steps for $dt = 0.25$ sec.)

5. According to Fig. 4.3, the $2\Delta x$ wave isn't supposed to move at all, for any value of c'. Does it move? If it does, why does it?

6. Now create a initial condition consisting of two combined waves. Specifically, combine a 50 and $10\Delta x$ wave, each with initial amplitude of 1.0, and set $c' = 0.5$. Anticipate what the result would look like after 1 revolution before running the model and checking your guess. (If you wish to explore other wave combinations, keep in mind you need to have an integral number of waves in the initial condition.)

For #5 above, note that a $2\Delta x$ wave is easily "lost" on a grid. You can try this modified code:

[1]Because of the periodic BCs, your initial condition has to have an integer number of waves in the domain, or the BC will mishandle the wave. To explore some wavelengths, it may be necessary to change NX

```
xi=float(i-2)
if(wavelength.eq.2) xi=xi+0.5
u(i) = amp*sin(2*xi*trigpi/wavelength)
```

9.2 The leapfrog and RK3 schemes

The upstream scheme was seen to suffer from substantial amplitude error when $c' \neq 1$. We next examine two alternatives, the second-order leapfrog and third-order Runge-Kutta (RK3) schemes, which will be added to your code for Model Task #0B. It is noted that the terms "leapfrog" and "RK" actually refer to how they accomplish *time* integration. You still have to specify how the spatial discretization is handled. We will examine versions of these schemes that are 2nd order in space.

9.2.1 The leapfrog scheme

The second-order leapfrog scheme (see Chapter 5) is centered about the "here/now" point in both time and space, which means *three* time levels (n-1, n, and n+1) and three spatial points (i-1, i, and i+1) are involved:

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} = -c \left[\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \right]. \tag{9.4}$$

Note an odd feature of this scheme: the forecast for the present spatial location (u_i^{n+1}) does **not** depend on that point's current value (u_i^n). Can you imagine that this might induce some interesting behavior?

$$u_i^{n+1} = u_i^{n-1} - \frac{c\Delta t}{\Delta x} \left[u_{i+1}^n - u_{i-1}^n \right]. \tag{9.5}$$

As with the upstream scheme, (9.4) can be written in explicit form:

We will call the future, present, and past values of the prognostic variable *up*, *u*, and *um*, respectively. The Fortran code for this could look like:

```
up(i) = um(i) - c*(d2t/d2x)*(u(i+1)-u(i-1))
```

in which *d*2*x* and *d*2*t* will be defined presently.

As before, the initial time will be 0 seconds, and the time index n starts at 0, so the first forecast is for $n = 1$. Since this is a three time level scheme, we ostensibly need *two* values, for $n = 0$ and $n =$ -1, at the start. Where does the value for $n =$ -1 come from? Usually,

we fudge it. The easiest (and usually, the only practical) way to start the scheme is to leap by Δt instead of $2\Delta t$ for the first time step. We can code this efficiently via equating the values for time indices 0 and -1, as will be seen below. As a consequence, and as illustrated in Fig. 9.2, the first time step will be handled differently from all remaining steps, using a forward-time, center-space scheme that is actually unstable. However, it is only used once.

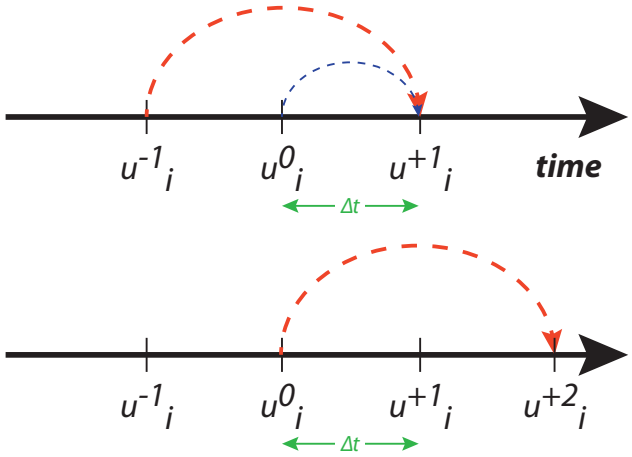

Figure 9.2: Starting off the three time level leapfrog scheme.

The initial condition is provided for both real and fake points and into both u and um (for convenience, as will be seen). Define $d2x = dx + dx$. **However, define $d2t = dt$ to start.** In this way, the first forecast will appear to use u and um over $2\Delta t$ to make the first forecast, but in fact the scheme just uses u and steps ahead only dt. Then, at the conclusion of the first time step, redefine $d2t = dt + dt$. Indeed, you can safely redefine $d2t$ at the conclusion of every time step, as the code execution is cheap. In this way, the handling of the first time step can be accomplished without any if statements.

Your programming logic could look like the following. Note $d2t$ is updated at the end of every time step, and setting for the next time step involves moving u into um and up into u.

```
! Initialize constants
      d2x = dx + dx
      d2t = dt
! Initialize u(i,k) over all real and fake points
[code here]
! Fudge um - this loop could be replaced with a single assignment
      do i=1,nx
        um(i,k) = u(i,k)
      enddo
```

```
! In the time stepping loopÉ  Since um = u and d2t = dt to start, this
!  is really a forward-time step, not a leapfrog step first time through.
      do i=2,nx-1
       up(i) = um(i)-(c*d2t/d2x)*(u(i+1)-u(i-1))
      enddo
! Take care of boundary points
[code here]

! Set for new time step
      do i=1,nx
        um(i) = u(i)    ! Present time becomes past
        u(i)  = up(i)   ! Future time becomes present
        up(i) = 0.      ! Start with a clean slate
      enddo
! Update d2t at end of time step.  Can do every time step; does not hurt
      d2t = dt + dt
```

Perform the same test problem as you did for the upstream scheme. With $c' = 1$ there should be no appreciable difference between the simulated and exact solution after one revolution.

9.2.2 Discussion

As discussed in Chapter 5, an advantage of the leapfrog scheme is that it has no amplitude error as long as it is stable, which for the current problem means $c' \leq 1$. However, as seen in Fig. 5.4, the scheme has phase error that is wavelength-dependent, and so has dispersion error. Short waves are again less well handled and, in this case, the $2\Delta x$ waves do not move. Generally, odd-order schemes do better with phase while even-order ones do better with amplitude.

That being said, the phase error of the leapfrog scheme is not too bad for long waves, while the amplitude error of the upstream method is unacceptable. You should verify the result of the experiment shown in Fig. 9.3 with your Model Task #1 code. Again, a single wave of $50\Delta x$ is advected, but now with $c' = 0.5$ and for 10 revolutions. You should find a small phase lag in the leapfrog solution. The phase error in the upstream result is quite small, but the wave has almost disappeared by this time.

It is noted in passing that in more complex, realistic problems, the computational mode in time discussed in Chapter 5 is also a concern.

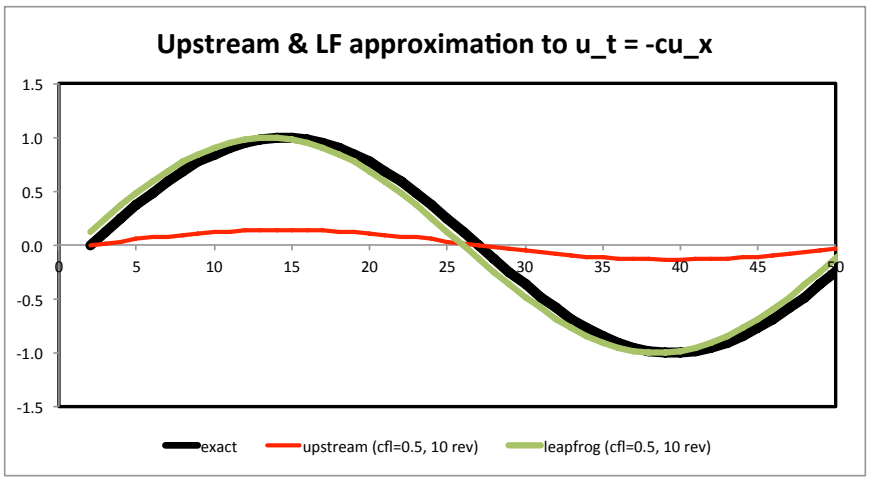

Figure 9.3: Similar to Fig. 9.3 but with $c' = 0.5$ and after 10 revolutions. Upstream (red), leapfrog (green) and exact (black) solutions shown.

9.2.3 An RK3 scheme

RK is a family of predictor-corrector schemes, including RK2, RK3, and RK4 (2nd through 4th order accurate, respectively). Additionally, there is no single RK scheme of a given order. As with the leapfrog, the RK label refers to how it integrates in *time*; you also need to choose a form of spatial differencing. The leapfrog scheme attained 2nd order accuracy in time by centering the tendency about time n (leaping from n-1 to n+1), over an interval of $2\Delta t$. An RK2 scheme attains 2nd order accuracy in time by re-computing the tendency halfway between times n and n+1. Thus, it is still centered in time, but not around time n.

For this task, we will try out Wicker and Skamarock's (2002) RK3 scheme, which recomputes the temporal tendency between times n and n+1 twice, for a total of three estimates of the forecast. That scheme, which was adopted for the Weather Research and Forecasting (WRF) model's Advanced Research WRF core[2], was tested with spatial derivatives that were 3rd, 4th, 5th, or 6th order accurate. For simplicity, however, we will match this with spatial differencing that is only 2nd order accurate, but suffices for the moment as the emphasis is on the time stepping.

First, we will write the advection term at i for time n with this centered in space formulation:

$$\text{adv}(u_i^n) = -c \left[u_{i+1}^n - u_{i-1}^n \right] / 2\Delta x. \tag{9.6}$$

Note the minus sign is incorporated into this definition. We will make three estimates of the forecast value. For the first (see Fig. 9.4), we combine the advection at time n with the

[2]The Model for Prediction Across Scales (MPAS) uses RK2 instead of RK3 as the default time integrator. The loss of accuracy was considered to be small compared to other operations and RK2 permits longer stable time steps (Bill Skamarock, personal communication, 2021).

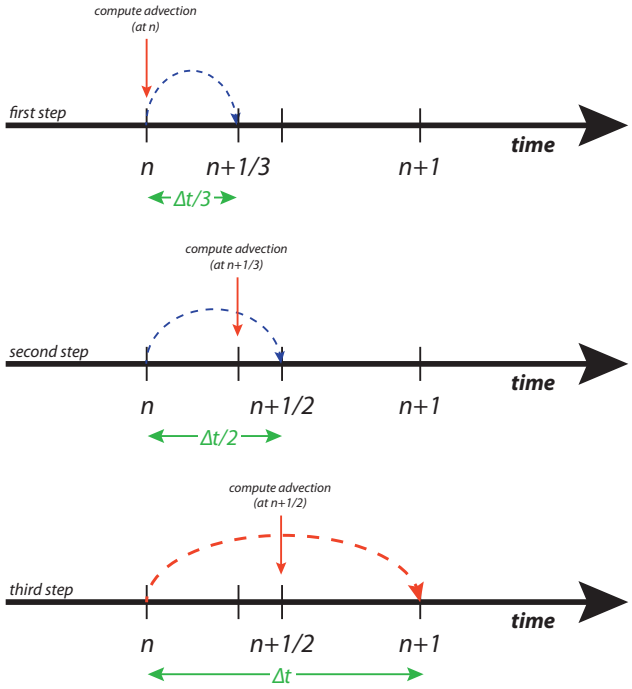

Figure 9.4: Time stepping in our RK3 scheme.

present time, but only jump ahead $\Delta t/3$, creating intermediate estimate u_i^*:

$$u_i^* = u_i^n + \frac{\Delta t}{3}\text{adv}(u_i^n). \tag{9.7}$$

If we had jumped ahead by the full time step, we would have been using a forward time, center space scheme, which is absolutely unstable. Instead, we will recompute advection using our new temporary estimate u_i^*, valid at time $n+\frac{1}{3}$, and use that to obtain our second estimate, u_i^{**}:

$$u_i^{**} = u_i^n + \frac{\Delta t}{2}\text{adv}(u_i^*). \tag{9.8}$$

Although advection was computed at time $n+\frac{1}{3}$, we went back to the start time n and stepped ahead by $\Delta t/2$, which means this time the advection term was not centered. Now, using this second estimate, we recompute advection a third time, now valid at time $n+\frac{1}{2}$, and combine that with the present value to obtain the forecast for time $n+1$:

$$u_i^{n+1} = u_i^n + \Delta t\text{adv}(u_i^{**}). \tag{9.9}$$

In the end, we end up with a scheme that is centered in time, but between times n and n+1.

9.2.4 Leapfrog and RK3 scheme experiments (Model Task #0B)

For Model Task #0B, add leapfrog and RK3 options to your code. For this task, the RK3 should be mated with second order spatial differencing, as in the discussion above. You can add better versions of this scheme later if you wish. Here are some experiments to try. **Only the bolded ones are required.**

1. We stated the forward in time, centered in space scheme was absolutely unstable. You can prove that to yourself by coding and running it.

2. **Do an experiment using upstream, LF, and our RK3 schemes with $c' = 1$ and $L = 50\Delta$x, executed for 10 revolutions. You should find the upstream and LF solutions are nearly exact, but our RK3 has phase lag even at c'=1. Send me this plot.**

3. **The RK3 is more costly to compute compared to the leapfrog, but it also permits longer time steps, and can stay stable for some $c' > 1$. Try runs with larger Δt. At what time step value does the RK3 scheme become unstable? Make a plot of maximum wave amplitude after 20 revolutions vs. Δt, for time steps between 1 and 2 sec, and send it to me.**

4. What happens when you make the initial condition wavelength smaller, like 25 or $10\Delta x$. How do the leapfrog and RK3 perform?

5. Our RK3 isn't optimal. What would happen if you adopted a higher order spatial differencing? (See Wicker and Skamarock, 2002, MWR, p. 2089).

6. Durran's (2010) text presents two, even more complex RK alternatives (p. 53). How well do they perform?

Chapter 10

Model Task #1: Setting up the base state

10.1 Vertical grid arrangement

Our model will have five prognostic variables: horizontal velocity u, vertical velocity w, potential temperature θ, water vapor mixing ratio q_v, and nondimensional pressure π. First, we will set up the initial environmental values ("base state") for each of these variables, plus air density ρ. The base state will vary only in the vertical direction and will be assumed to be in hydrostatic balance. MKS values will be used at all times.

In finite differencing, we subdivide the model domain into a set of grid volumes (or grid areas in 2D). Once accomplished, it remains to specify values of the variables for each grid volume. Where in space should the variables be defined, or thought of as existing? The simplest procedure might be to declare all variables to represent points residing at the center of each grid box. In this arrangement, each variable value is though of as representing a grid volume average, and the most logical place to think of this average existing is at the center.

Instead, our model will employ a *staggered* grid arrangement – specifically, Arakawa's "C" grid – which is a natural for mesoscale models. The scalar variables (θ, π, q_v) are still placed at the grid center but the velocity components are arrayed along the volume or area edges (see Fig. 1). In this arrangement, the velocities represent flows across the boundaries.

The grid box width and depth will be called Δx and Δz, respectively. Note that u and the scalar variables are at the same physical height levels, while w is displaced $0.5\Delta z$ above and below. These two height levels will be called the u and w heights, respectively. In this arrangement, we only really need boundary conditions in the vertical direction on w, and we will assume the boundaries are rigid, flat plates such that $w = 0$ there.

The only density our model will need is base state density — at least explicitly. Density, a scalar variable, is most naturally defined at the grid box center with the other scalars (the u height level). However, since density is so important, we will find it useful to also define density at the w height levels. The base state densities at the u and w height levels will be called RHOU and RHOW, respectively, for coding purposes. The base states for the other variables will be called UB (for $\bar{u}$), TB (for $\bar{\theta}$), QB (for $\bar{q_v}$), and PIB (for $\bar{\pi}$). Mean w is zero.

The vertical grid index will be termed k, and all base state arrays will be vectors with dimension NZ, as there are a total of NZ grid points (see Figs. 1 and 2). *Since the grid is staggered, note that a given k-index value, say "k = 5", refers to different physical heights for the w and u/scalar variables!*. To facilitate coding, we will include one fictitious grid volume at the top and bottom of each model column. It will be seen that this permits simpler and more efficient coding of our equations at a small cost of increased storage. (Fictitious grids will be added along the horizontal boundaries as well, for the same reason.)

In **Fortran** (Fig. 1), the index for an array dimensioned NZ begins at index 1 and runs to NZ, inclusive. Thus, the real boundaries for w will be located at $k = 2$ and NZ, and w will be zero at both. Therefore, the lowest and highest physical u and scalar points are located at $k = 2$ and NZ-1, respectively. The scalar/u grid points at $k = 1$ and NZ are fictitious, as is the w point for k=1.

In **C++** and other zero-based index languages (Fig. 2), array indices start at 0 and thus the maximum index for an array dimensioned NZ is actually NZ-1. So, we will enforce w to be zero at $k = 1$ and NZ-1 and the lowest and highest u and scalar points are located at $k = 1$ and NZ-2. The scalar/u grid points at $k = 0$ and NZ-1 are fictitious, as is w at k=0. *Note that in an array dimensioned NZ the point k=NZ does not exist* – at least, not within the range of the array under examination. The attempt to access array location NZ in an array dimensioned NZ is the single most common coding error I've seen.

10.2 Base state temperature and moisture

The base state environment described similar to that employed by Weisman and Klemp (1982, MWR, p. 504) and represents conditions common to the Midwestern United States during the spring season. Let z_{TR} and T_{TR} be the height level (12000 m) and temperature (213 K) of the tropopause, with θ_{TR} being the potential temperature there (343 K). Let z_T be the distance of a given scalar location above the model surface. In **Fortran**, the model surface is the $k = 2$ w point in Fortran and so the physical height of the first scalar point is $(k - 1.5)\Delta z$. In **C++**, the model surface is the $k = 1$ w point and so the physical height of the first scalar point is $(k - 0.5)\Delta z$. The Weisman and Klemp vertical mean potential temperature ($\bar{\theta}$) is:

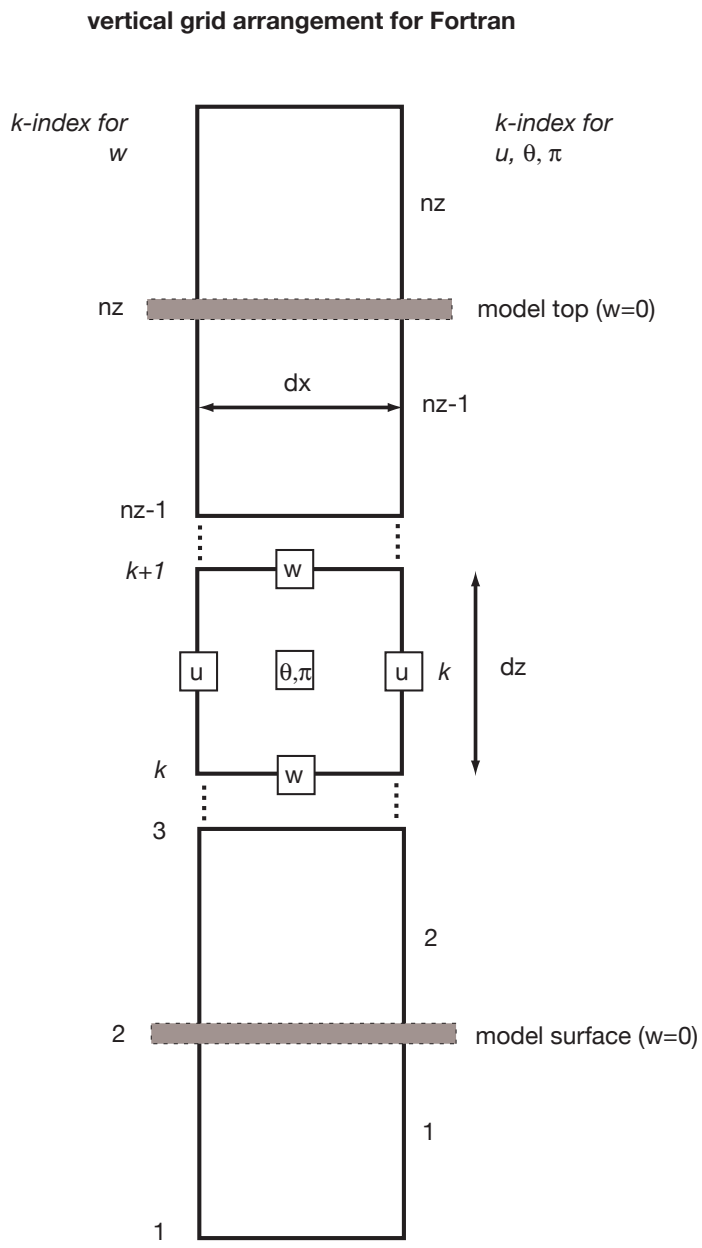

Figure 10.1: Vertical grid arrangement for **Fortran** code. The first and last real w points are at $k = 2$ and NZ, respectively. The first and last real scalar/u points are at $k = 2$ and NZ-1. Note presence of two fictitious grid boxes, located just above and below the model's vertical boundaries.

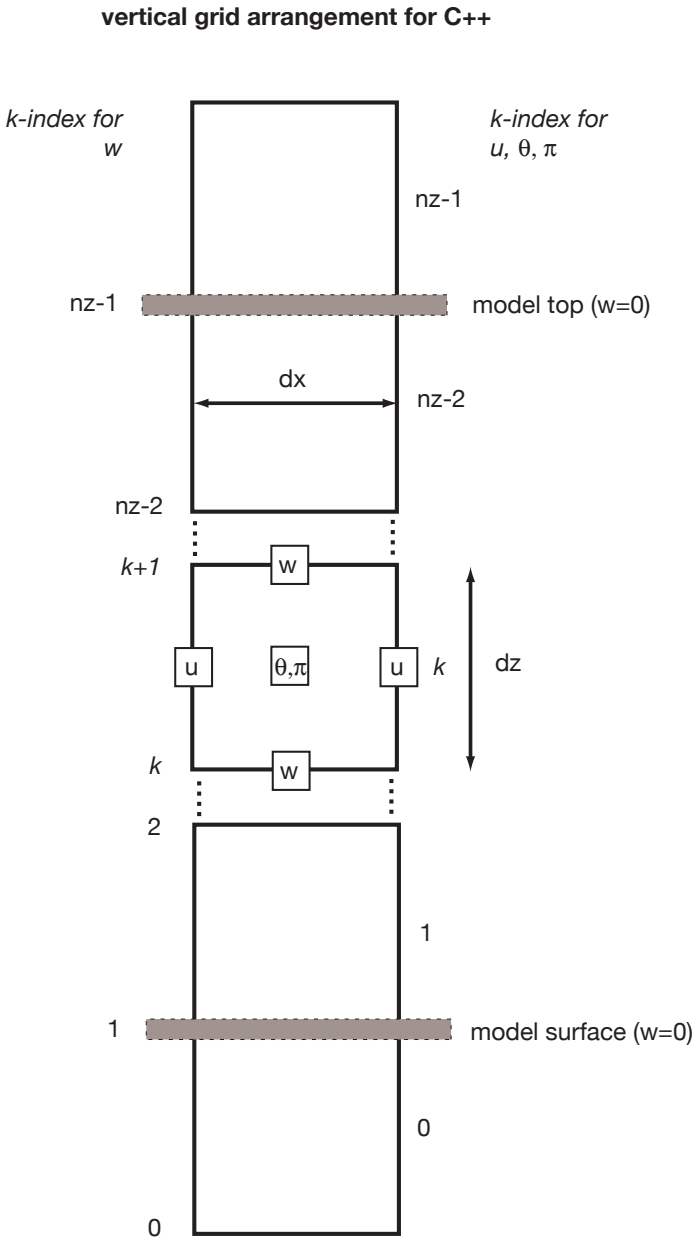

Figure 10.2: Vertical grid arrangement for **C++** and other zero-based index language code. The first and last real w points are at $k = 1$ and NZ-1, respectively. The first and last real scalar/u points are at $k = 1$ and NZ-2. Note presence of two fictitious grid boxes, located just above and below the model's vertical boundaries.

$$\bar{\theta} = \begin{cases} 300 + 43\left[\frac{z_T}{z_{TR}}\right]^{1.25} & z_T \leq z_{TR}; \\ \theta_{TR} \exp\left[\frac{g(z_T - z_{TR})}{c_{pd} T_{TR}}\right] & z_T > z_{TR}. \end{cases}$$

where $g = 9.81$ m s^{-1}, and $c_{pd} = 1004$ J kg^{-1} K^{-1}. Let the base state mean vapor mixing ratio, $\bar{q_v}$ (kg kg^{-1}), be given by:

$$\bar{q_v} = \begin{cases} 0.0161 - 0.000003375 z_T & z_T \leq 4000 \text{ m}; \\ 0.0026 - 0.00000065(z_T - 4000) & 4000 < z_T \leq 8000 \text{ m}; \\ 0 \qquad z_T > 8000 \text{ m}. \end{cases}$$

Take NZ=40 and $\Delta z = 700$ m and compute both. Then use these data to create virtual potential temperatures, which should also be contained in an array of dimension NZ and positioned at the scalar heights, using $\bar{\theta_v} = \bar{\theta}\,[1. + 0.61\bar{q_v}]$. I'll call this array TBV.

10.3 Derived quantities

Next, the base state nondimensional pressure ($\bar{\pi}$) has to be computed[1]. Pressure is nondimensionalized as:

$$\pi = \left[\frac{p}{p_0}\right]^{\frac{R_d}{c_{pd}}},$$

where $R_d = 287$ J kg^{-1} K^{-1} and $p_0 = 100000$ N m^{-2} (1000 mb). The hydrostatic equation, written in terms of π is:

$$\frac{d\bar{\pi}}{dz} = -\frac{g}{c_{pd}\bar{\theta_v}}.$$

We will obtain $\bar{\pi}$ as a function of height by integrating this equation upwards from the surface, starting with a supplied surface pressure which will be taken to be 96500 N m^{-2} (965 mb). The basic concept is illustrated in Fig. 10.3. This surface pressure (PSURF) is recorded at the lowest real w point ($k = 2$ in **Fortran**, $k = 1$ in **C++**). The first task is then to compute $\bar{\pi}$ at the first real scalar point, located $0.5\Delta z$ above the surface. The mean virtual potential temperature, $\bar{\theta_v}$, will be taken to be its value at the first real scalar point and presumed constant below that point. This can be coded in **Fortran** as:

```
xk = rd/cpd
pisfc  = (psurf/p0)**xk
pib(2) = pisfc - grav*0.5*dz/(cpd*tbv(2))
```

[1]The advantage of avoiding dimensional pressure is discussed in Sec. 3.3.

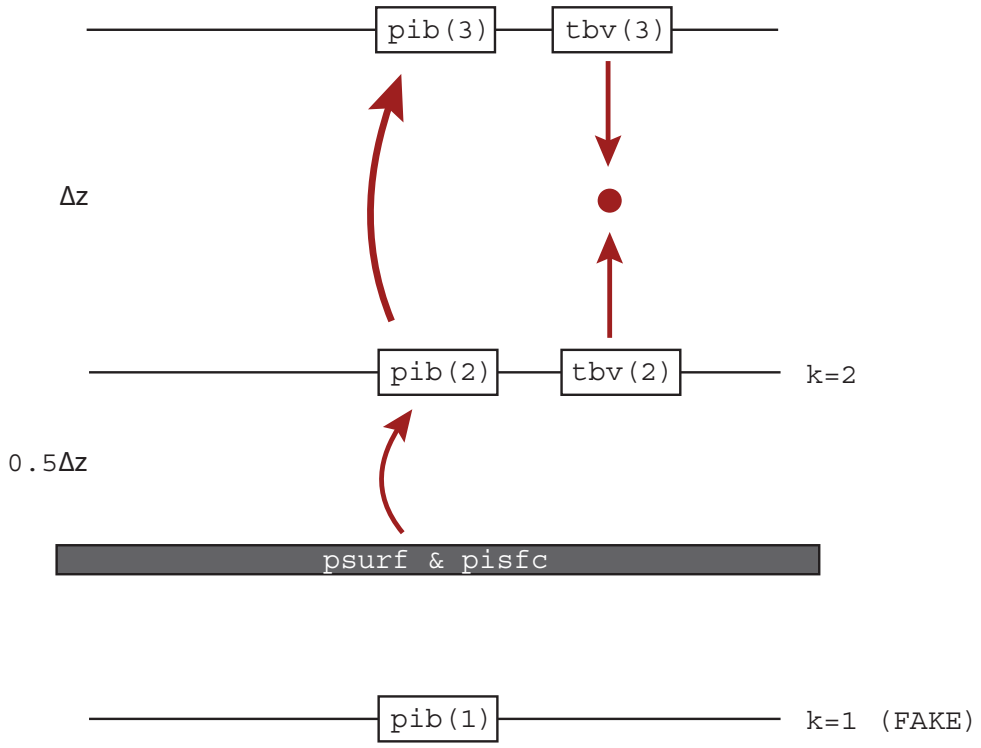

Figure 10.3: Computation of nondimensional pressure at scalar levels. The black block illustrates the model surface.

In **C++**, the code might look like this:

```
xk = rd/cpd;
pisfc = pow((psurf/p0),xk);
pib[1] = pisfc - grav*0.5*dz/(cpd*tbv[1]);
```

Now we need to integrate the hydrostatic equation up through the rest of the column. Each step will be of height dz as we traverse from one scalar point to the next. From the hydrostatic equation, we know that the pressure at the top of a designated layer depends on two things: the pressure at the layer bottom, and the mean virtual temperature of the layer. To get the mean $\bar{\theta}_v$ for a layer between two scalar heights, we need to average the TBV values at the layer top and bottom. In **Fortran**, we're integrating from $k = 3$ to NZ-1 because NZ-1 represents the uppermost real scalar point (one-half Δz below the model top at w position $k = NZ$). (If the desired DO loop increment is unity, it need not be explicitly coded.) In **C++**, we traverse from $k = 2$ to NZ-2, inclusive.

```
do k = 3, nz-1
  tbvavg = 0.5*(tbv(k)+tbv(k-1))
  pib(k) = pib(k-1) - grav*dz/(cpd*tbvavg)
enddo

for(k = 2; k <= nz-2; k++){
  tbvavg = 0.5*(tbv[k]+tbv[k-1]);
  pib[k] = pib[k-1] - grav*dz/(cpd*tbvavg);
}
```

You may find it useful to compute mean dimensional pressure at each scalar level as well.

Once $\bar{\pi}$ is obtained, mean state density $\bar{\rho}$ has to be computed from a form of the ideal gas law (with pressure nondimensionalized):

$$\bar{\rho} = \frac{p_0 \bar{\pi}^{\frac{c_{vd}}{R_d}}}{R_d \bar{\theta}_v}$$

where $c_{vd} = c_{pd}$ - R_d = 717 J kg^{-1} K^{-1}. Density at the scalar height levels (RHOU) is easily obtained. You get the surface value of RHOW (rhow(2) or rhow[1]) by knowing the surface pressure, and RHOWs farther aloft by averaging pairs of RHOU values to the w height level. From Fig. 1, it is seen this is written as:

```
rhow(k) = 0.5*(rhou(k) + rhou(k-1))
```

Finally, compute the relative humidity at each scalar height level, using a form of Tetens' equation (see Soong and Ogura 1973, JAS; Murray 1967, JAM) to get the saturation mixing ratio for the base state, $\bar{q}_{vs}$:

$$\bar{q}_{vs} = \frac{380}{\bar{p}} \exp\left[\frac{17.27(\bar{T} - 273.)}{(\bar{T} - 36.)}\right]$$

where $\bar{p}$ and $\bar{T}$ are base state pressure (N m^{-2}) and temperature (K). Since you are carrying nondimensional pressure and potential temperature instead, these variables need to be converted. Using the definition of π:

$$p = p_0 \pi^{\frac{c_{pd}}{R_d}}$$

and the relationship between T and θ is:

$$T = \theta\pi.$$

Compute $\bar{q}_{vs}$ for each scalar grid point and use it to calculate the relative humidity.

10.4 Results for some fields

In creating the following, I employed these values for model constants: $g = 9.81$ m s^{-1}; c_{pd} = 1004 J kg^{-1} K^{-1}; $R_d = 287$ J kg^{-1} K^{-1}; and $c_{vd} = 717$ J kg^{-1} K^{-1}.

z(km)	tb(K)	qb(g/kg)	rhou(kg/m^3)	RH (%)	pib(ndim)	p(mb)	T(deg. C)
0.35	300.52	14.92	0.108851E+01	88.78	0.978590E+00	927.08	20.93
1.05	302.05	12.56	0.102329E+01	96.08	0.956077E+00	854.59	15.63
1.75	303.88	10.19	0.959954E+00	99.96	0.933657E+00	786.52	10.57
2.45	305.90	7.83	0.898970E+00	98.72	0.911346E+00	722.71	5.63
3.15	308.08	5.47	0.840510E+00	89.18	0.889156E+00	663.00	0.78
3.85	310.38	3.11	0.784646E+00	66.11	0.867096E+00	607.21	-4.02
4.55	312.79	2.24	0.730747E+00	62.94	0.845181E+00	555.20	-8.78
5.25	315.30	1.79	0.679410E+00	66.96	0.823428E+00	506.80	-13.52
5.95	317.89	1.33	0.630783E+00	67.54	0.801845E+00	461.83	-18.25
6.65	320.56	0.88	0.584796E+00	61.11	0.780434E+00	420.11	-22.97
7.35	323.30	0.42	0.541370E+00	41.13	0.759196E+00	381.46	-27.70
8.05	326.11	0.00	0.500413E+00	0.00	0.738135E+00	345.70	-32.44

```
 8.75 328.97  0.00   0.461731E+00  0.00   0.717253E+00 312.68 -37.19
 9.45 331.90  0.00   0.425375E+00  0.00   0.696554E+00 282.24 -41.96
10.15 334.88  0.00   0.391250E+00  0.00   0.676039E+00 254.21 -46.76
10.85 337.91  0.00   0.359259E+00  0.00   0.655707E+00 228.46 -51.58
11.55 340.99  0.00   0.329307E+00  0.00   0.635558E+00 204.83 -56.43
12.25 346.96  0.00   0.298942E+00  0.00   0.615673E+00 183.27 -59.54
12.95 358.28  0.00   0.267244E+00  0.00   0.596277E+00 163.85 -59.52
13.65 369.97  0.00   0.238910E+00  0.00   0.577493E+00 146.50 -59.50
14.35 382.04  0.00   0.213581E+00  0.00   0.559303E+00 130.98 -59.47
15.05 394.51  0.00   0.190940E+00  0.00   0.541687E+00 117.11 -59.45
15.75 407.38  0.00   0.170700E+00  0.00   0.524628E+00 104.71 -59.43
16.45 420.68  0.00   0.152607E+00  0.00   0.508109E+00  93.62 -59.40
17.15 434.40  0.00   0.136433E+00  0.00   0.492111E+00  83.71 -59.37
17.85 448.58  0.00   0.121974E+00  0.00   0.476619E+00  74.84 -59.35
18.55 463.22  0.00   0.109049E+00  0.00   0.461616E+00  66.92 -59.32
19.25 478.33  0.00   0.974942E-01  0.00   0.447088E+00  59.84 -59.29
19.95 493.94  0.00   0.871648E-01  0.00   0.433019E+00  53.51 -59.26
20.65 510.06  0.00   0.779307E-01  0.00   0.419394E+00  47.84 -59.23
21.35 526.71  0.00   0.696756E-01  0.00   0.406200E+00  42.78 -59.20
22.05 543.89  0.00   0.622957E-01  0.00   0.393422E+00  38.26 -59.17
22.75 561.64  0.00   0.556982E-01  0.00   0.381049E+00  34.21 -59.14
23.45 579.97  0.00   0.498000E-01  0.00   0.369067E+00  30.59 -59.10
24.15 598.89  0.00   0.445270E-01  0.00   0.357463E+00  27.36 -59.07
24.85 618.44  0.00   0.398128E-01  0.00   0.346226E+00  24.47 -59.03
25.55 638.62  0.00   0.355982E-01  0.00   0.335344E+00  21.88 -58.99
26.25 659.46  0.00   0.318303E-01  0.00   0.324806E+00  19.57 -58.95
```

Chapter 11

Model Task #2: Assessing convective instability

The present task is to assess the convective instability of the sounding we created in Task #1. This instability, called Convective Available Potential Energy (CAPE), is usually gauged for an insulated parcel originating in the lower troposphere that is subsequently raised without mixing with — nor experiencing resistance from — the environment. If the parcel can become positively buoyant, it will convert the potential energy represented by its buoyancy (CAPE) into kinetic energy of motion. In actuality, the parcel will experience some degree of mixing with its surroundings, and has to push air residing above it out of the way. Thus, the CAPE is an overestimate of the true amount of kinetic energy that might be realized by the parcel.

11.1 Lifting and adjusting a parcel on the model grid

Figure 11.1 sets up the problem and presents a hypothetical situation. In the *grid view* we are defining a parcel at the lowest *real* scalar grid point by specifying its potential temperature and water vapor mixing ratio. Call these θ_p and q_{vp}. (We're using these properties because they are conserved, at least until the parcel saturates.) Once defined, we will raise the parcel, grid point by grid point, until we reach the model's topmost scalar point. There are two parts to this problem: first, we compute potential temperature and vapor content along the parcel's path, and then we assess its instability (if any) relative to its surroundings. We use the fundamental air parcel assumption that the parcel's pressure equals that of its surroundings at all times.

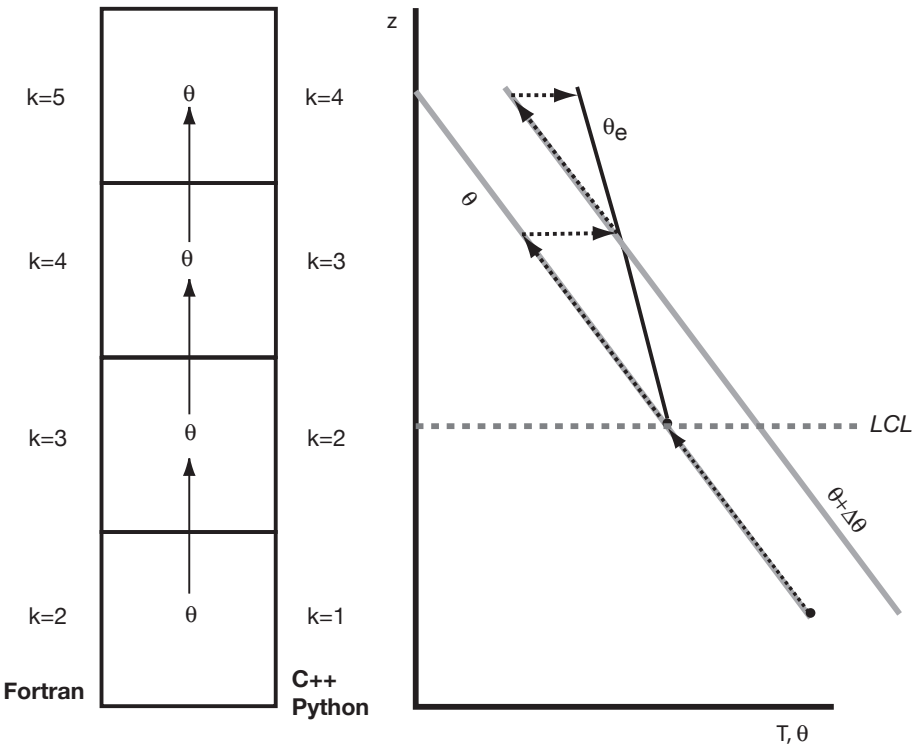

Figure 11.1: Grid and Stuve diagram views of an ascending parcel. On this figure, θ represents parcel potential temperature, θ_p.

Call the lowest scalar point index k_1; this is $k = 2$ in Fortran and $k = 1$ in C++. When we raise the parcel to the next grid point up ($k = k_1+1$), its θ_p and q_{vp} are (tentatively) preserved. Figure 11.1 also sketches what this would look like on a thermodynamic chart like a Stuve diagram. We have ascended one grid point along a dry adiabat. At this juncture, we test to see if the parcel has become saturated during this displacement. This is done by computing the parcel's saturation mixing ratio, q_{vsp}, a function of its potential temperature θ_p and (nondimensional) pressure $\bar{\pi}$ (the environment value at the parcel's height) given by[1]:

$$q_{vsp} = \frac{380}{p_0 \bar{\pi}^{\frac{c_{pd}}{R_d}}} \exp\left[\frac{17.27(\theta_p \bar{\pi} - 273.)}{(\theta_p \bar{\pi} - 36.)}\right],$$

where it is recalled that $T = \theta\pi$. In the example shown, the parcel is assumed to have just become saturated at $k = k_1+1$, so $q_{vsp} = q_{vp}$, so the lifting condensation level (LCL) is at this height level.

Further lifting should now be moist adiabatic, with the parcel continually condensing sufficient vapor to prevent supersaturation at any and every infinitesimal vertical "step". Note that since pressure varies along the parcel path, this is impossible to handle with precise accuracy in a model (the model's framework is Eulerian, not Lagrangian, anyway). However, it is not terribly inaccurate to first raise the saturated parcel *dry adiabatically* – which of course produces a supersaturated parcel) – and then adjust the parcel back to 100% relative humidity at the new height level. This effectively discretizes the saturated adiabatic process into two separate steps, dry adiabatic expansion followed by isobaric saturation adjustment, as shown on the *Stuve view* in Fig. 11.1. Naturally, this approximation gets worse as the vertical grid interval gets larger.

We appreciate that if the parcel is supersaturated, i.e., $q_{vp} > q_{vsp}$, the amount of condensation actually realized is *less* than the difference $q_{vp} - q_{vsp}$. This is because of the temperature dependence of the saturation mixing ratio. The parcel is supersaturated because it carries more vapor than it can hold at its temperature. However, as the supersaturated parcel's vapor condenses, heat is released that raises the parcel's temperature, increasing its ability to hold vapor. Thus, during the process of condensation, q_{vp} decreases (as vapor condenses) but q_{vsp} increases (owing to latent heating).

We compute the isobaric saturation adjustment in the following way[2]: The amount of vapor that can be condensed from a supersaturated parcel starting with pre-adjustment values of

[1] In avoiding carrying both dimensional and nondimensional pressure, the equation got more complicated. You may decide to create a vector array for both.

[2] The present approach is something of a compromise between the saturation adjustments proposed by Asai (1965) and Soong and Ogura (1973).

potential temperature θ_p, vapor mixing ratio q_{vp} and saturation mixing ratio q_{vsp} is:

$$C = \frac{q_{vp} - q_{vsp}}{1 + \phi}$$

where ϕ is defined as:

$$\phi = q_{vsp}\left[\frac{17.27 \cdot 237 L_v}{c_{pd}(\theta_p \bar{\pi} - 36.)^2}\right]$$

and L_v is the latent heat of vaporization ($\approx 2.5 \times 10^6$ J kg^{-1}). The vapor loss due to condensation is then

$$q_{vp} = q_{vp} - C,$$

and the heat released raises the parcel's potential temperature according to:

$$\theta_p = \theta_p + \frac{L_v}{c_{pd}\bar{\pi}} C.$$

At the conclusion of this adjustment, we should have a parcel potential temperature and vapor mixing ratio that represents exact saturation. It is worth demonstrating that is indeed the case.

11.2 Computing CAPE and CIN on the model grid

CAPE is an integrated property, defined as:

$$CAPE = \int_{z_{LFC}}^{z_{EQL}} g\left[\frac{\theta_{pv} - \bar{\theta}_v}{\bar{\theta}_v}\right] dz$$

and represents the "positive area" on a thermodynamic diagram when both the environmental sounding and the parcel path are plotted. As the parcel rises, it may find itself less dense than the environment at some point. The height where this first occurs is called the level of free convection (LFC). Eventually, the parcel will become more dense than the surrounding environment; this height is termed the equilibrium level (abbreviated EQL) and is the first guess at cloud top. CAPE is computed between these two levels.

CIN, in contrast, is the "negative area" existing between the parcel's initial location (z_0) and the LFC, so it is defined as

$$CIN = \int_{z_0}^{z_{LFC}} g\left[\frac{\theta_{pv} - \bar{\theta}_v}{\bar{\theta}_v}\right] dz.$$

CIN is integrated negative buoyancy, but is often presented as a positive quantity. The definition here produces negative values when CIN is present. Note that these formulae

use *virtual potential temperatures* for the parcel and environment (θ_{pv} and $\bar{\theta}_v$, respectively). These are defined as:

$$\theta_{pv} = \theta_p(1 + 0.61 q_{vp}),$$

and

$$\bar{\theta}_v = \bar{\theta}(1 + 0.61 \bar{q_v}).$$

Using virtual temperatures allows us to incorporate the effect of moisture on parcel density.

Our present application computes the CAPE and CIN of the Weisman-Klemp sounding for a surface-based parcel; i.e., commencing at the lowest scalar level above the model ground. Although we should probably develop code that can handle complex situations – such as soundings that possess one or more layers with negative buoyancy above the first LFC – the present sounding will not require special treatment. Our specific example will assign potential temperature (300.52 K) and mixing ratio values (11.5 g kg^{-1}) to the parcel representing initial subsaturation (use only kg kg^{-1} in doing your model calculations), from which the parcel's θ_{pv} can be computed. (Note this parcel is negatively buoyant even at its origin, as it possesses less vapor than the environment at the first scalar level.) At each model level above the surface, we have already computed parcel virtual potential temperature, θ_{pv}, and we have the environmental value $\bar{\theta}_v$ as well. It is time to compute CAPE and CIN.

Let the integrand of CAPE and CIN be called b, for buoyancy. We have b at each model level[3]. CAPE can be computed using the trapezoidal rule, so the fractional value between model levels k and $k-1$ would be

$$\frac{b_k + b_{k-1}}{2} \Delta z,$$

as illustrated in Fig. 11.2. This straightforward if the parcel is positively (or negatively) buoyant at both levels. It is likely, however, that the LFC and EQL will fall between model levels, as also shown in Fig. 11.2, so there will one or two layers that have to be treated differently. These special layers will be identifiable by a change in sign of b between the bottom and top.

As an example, in the layer encompassing the LFC, the positive area would be

$$\text{areapos} = \frac{b_k + 0}{2}(z_k - z_{LFC}),$$

[3]This presumes the parcel CAPE and CIN are computed after parcel buoyancy is determined at each level. That is not necessary, as once you have parcel b at level k, you can compute the CAPE and/or CIN in the layer between levels k and $k-1$. In other words, you do not really need to allocate an array for the parcel buoyancy.

where z_{LFC} is the height of the LFC (at which point parcel buoyancy is zero). z_{LFC} can be found via linear interpolation:

$$z_{LFC} = z_k - \frac{\Delta z}{b_k - b_{k-1}}(b_k - 0) = z_k - \beta \Delta z,$$

where β is the fraction of the layer possessing positive buoyancy. This permits us to express the positive area as

$$\text{areapos} = \frac{b_k}{2}\beta \Delta z.$$

The negative area below the LFC contributes to CIN, and would be

$$\text{areaneg} = \frac{b_{k-1}}{2}(1 - \beta)\Delta z.$$

So, we might have some code for the LFC layer that looks like the following:

```
btop = b( k )                ! buoyancy at top of current layer
bbot = b(k-1)                ! buoyancy at bottom of current layer

if(btop.ge.0..and.bbot.lt.0.)then  ! we have isolated the LFC layer
   frac = btop/(btop - bbot)       ! fraction of layer that is positively buoyant
   areapos = 0.5*btop*frac*dz      ! positive area according to trapezoidal rule
   areaneg = 0.5*bbot*(1.-frac)*dz ! negative area according to trapezoidal rule
   zlfc = z(k) - dz*frac/1000.     ! height of LFC in km
[...]
```

In the above, z is an array of model scalar level heights (in km) and $zlfc$ is the height of the LFC. Something similar would have to be done for the layer encompassing the EQL. In my code, I consider four mutually exclusive possibilities for each model layer: that it contains the LFC (as in the above code), that it contains the EQL, that it is positively buoyant through the layer, and that it is negatively buoyant through the layer. The latter contributes to CIN only if the layer resides below the LFC. (Do not add the negative buoyancy above the LFC to CIN.)

My output is included at the end of this chapter.

11.3 What is CAPE missing?

The units of CAPE (and CIN) are J kg^{-1} or, equivalently, m^2 s^{-2}. The latter suggests that CAPE is a (vertical) velocity squared. In our discussion of gravity waves, we rewrote the

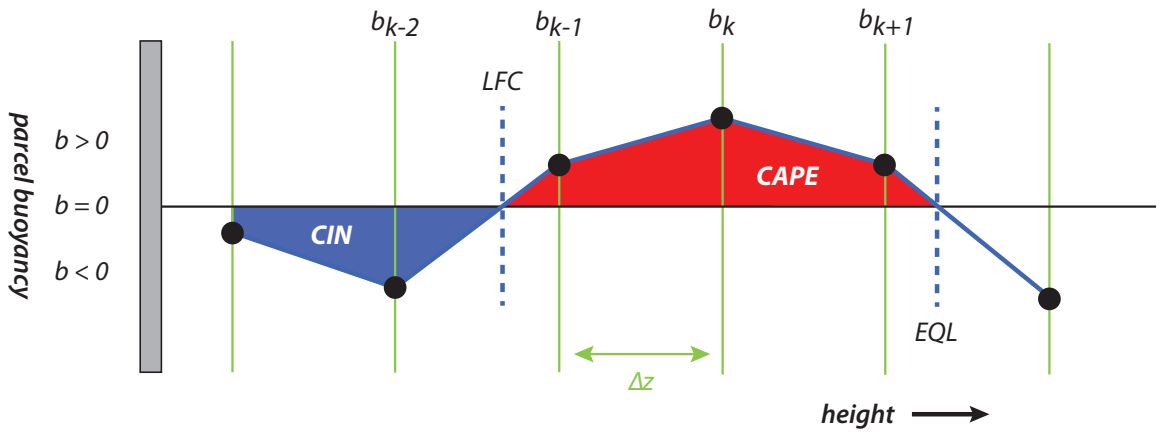

Figure 11.2: Parcel buoyancies on the model vertical grid (oriented horizontally for convenience). In the hypothetical example illustrated, the parcel starts with some small negative buoyancy at its origin.

vertical equation of motion into a form similar to that below (though now with moisture):

$$\frac{dw}{dt} = -\frac{1}{\bar{\rho}}\frac{\partial p'}{\partial x} + g\frac{\theta_v'}{\bar{\theta}_v}.$$

The hydrostatically balanced mean state has already been removed and the primed values represent the difference between the parcel and its surrounding environment. Given the air parcel assumption of mechanical equilibrium, $p' = 0$, and the equation is revised to:

$$\frac{dw}{dt} = g\frac{\theta_{pv} - \bar{\theta}_v}{\bar{\theta}_v} \tag{11.1}$$

Note the right hand side already looks like CAPE; all it needs is to be integrated between the LFC and EQL height levels. If the motion is strictly one-dimensional (in the vertical) and steady with time, then

$$\frac{dw}{dt} = \frac{dw}{dz}\frac{dz}{dt} = w\frac{dw}{dz} = \frac{1}{2}\frac{d(w^2)}{dz}.$$

Integrate (11.1), using the rightmost expression above to replace the left hand side of (11.1). So, after rearranging:

$$\begin{aligned} CAPE &= \frac{1}{2}\int_{z_{LFC}}^{z_{EQL}} d(w^2) \\ &= \frac{1}{2}w_{EQL}^2 - w_{LFC}^2 \\ &= \frac{1}{2}w_{EQL}^2 \end{aligned} \tag{11.2}$$

where it has been assumed that $w = 0$ at the LFC (not a bad assumption). Thus, the vertical velocity at the estimated cloud top (EQL) depends on the vertically integrated parcel positive buoyancy such that:

$$w_{EQL} = \sqrt{2CAPE}.$$

For a CAPE of 2000 J kg^{-1}, such as might be found in a late spring Midwestern environment during the afternoon, this predicts a parcel would strike cloud top with a vertical velocity of over 63 m s^{-1}. Two things are unrealistic about this value. First, it is too large. Second, maximum vertical velocity is not likely to be found at cloud top, but rather somewhere farther below. (Can you explain why?) Indeed, by the time a real parcel approaches cloud top, its velocity will be much reduced from the largest value it had previously attained, and may even be reasonably close to zero. (Again, can you explain why?)

11.4 Results

In the results tabulated below, height `z` is in km, pressure `p` in mb, `thv_env` and `thv_prcl` are environmental and parcel virtual potential temperatures in K, `qv_prcl` is the parcel vapor content in g kg^{-1}, `CAPE` and `CIN` are in J kg^{-1}, and `buoybot` and `buoytop` are parcel buoyancies at the bottom and top of the present layer, in m s^{-2}. CIN is presented as a negative quantity. These results were obtained with a parcel initial potential temperature defined as 305.2 K, parcel initial mixing ratio of 11.5 g/kg, and the gravity acceleration and latent heat of vaporization were defined as 9.81 m/s/s and 2.5E6 J/kg/K, respectively.

You certainly can anticipate that the accuracy of your CAPE and CIN calculations depends on resolution, and applicability of your estimates depends on your initial parcel properties. What happens if you increase NX and decrease Δz correspondingly? What happens if you give the parcel different properties, such as something representative of a deeper layer?

Figure 11.3 shows the sounding as rendered by the `metpy` Skew-T package. Although provided with the same initial parcel properties, the CAPE and CIN values computed by this routine do not agree with we have obtained here. This is in part due to the differences in the calculation procedure, definitions used (e.g., `metpy`'s CAPE and CIN do not use virtual temperature), and perhaps precision.

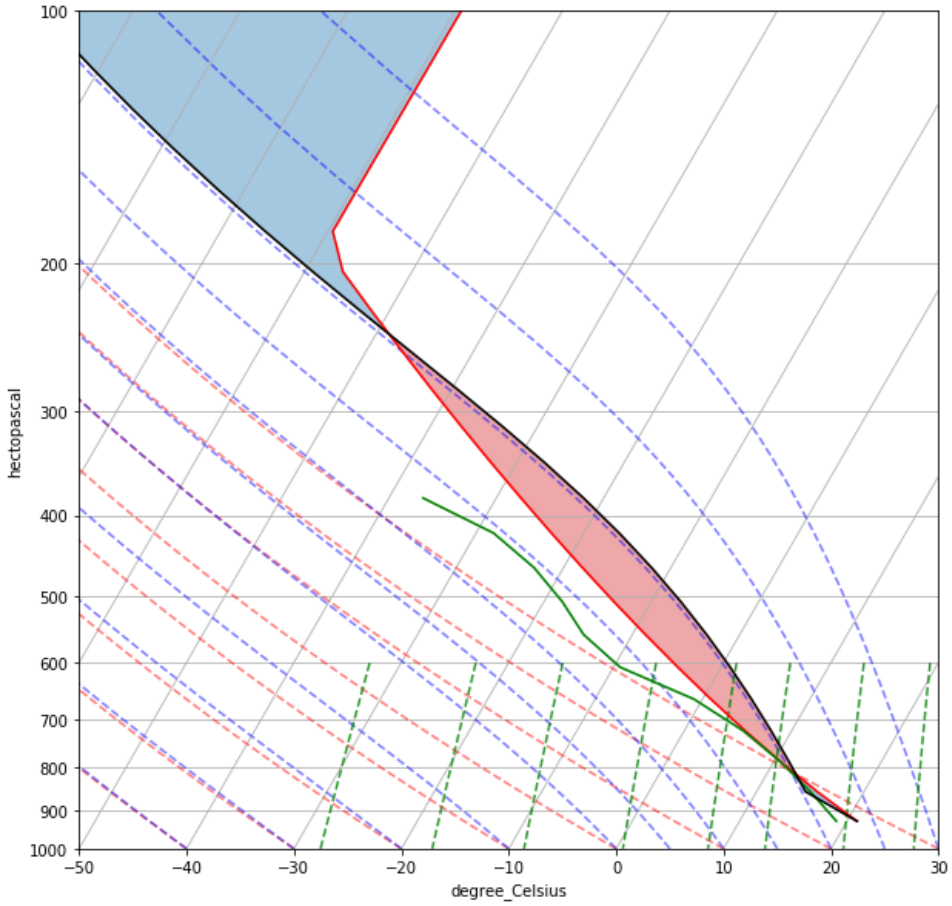

Figure 11.3: Our sounding as visualized using `metpy`.

```
  [partial results]
  initial parcel potential temperature:    300.52      K
  initial parcel vapor mixing ratio:       11.50      g/kg
    z     p    thv_env  thv_prcl   qv_p      cape     cin     buoybot   buoytop
  1.05   854.6  304.36  302.63   11.50       0.0   -26.6    -0.020    -0.056
  1.75   786.5  305.76  306.00   10.15       0.3   -43.8    -0.056     0.008
[...]
  3.85   607.2  310.97  316.29    6.23     194.9   -43.8     0.122     0.168
  4.55   555.2  313.22  319.50    5.07     322.5   -43.8     0.168     0.197
[...]
  8.75   312.7  328.97  332.77    0.64    1134.2   -43.8     0.162     0.113
  9.45   282.2  331.90  333.67    0.38    1192.2   -43.8     0.113     0.052
 10.15   254.2  334.88  334.26    0.21    1205.8   -43.8     0.052    -0.018
[...]
 13.65   146.5  369.97  335.01    0.00    1205.8   -43.8    -0.638    -0.927
 14.35   131.0  382.04  335.03    0.00    1205.8   -43.8    -0.927    -1.207
 15.05   117.1  394.51  335.03    0.00    1205.8   -43.8    -1.207    -1.479
[...]
 24.85    24.5  618.44  335.03    0.00    1205.8   -43.8    -4.322    -4.496
 25.55    21.9  638.62  335.03    0.00    1205.8   -43.8    -4.496    -4.663
 26.25    19.6  659.46  335.03    0.00    1205.8   -43.8    -4.663    -4.826
  Vertically integrated CAPE   1205.8 J/kg  CIN is    -43.8 J/kg
  LFC detected at  1.67 km
  EQL detected at  9.97 km
```

Chapter 12

Model Task #3: Grid setup, initial condition and visualization

In this task, we set up the model's two-dimensional (2D) grid, specify the initial condition, and get a start on model visualization. The dry version of our model will have four 2D prognostic variables, u, w, θ' and π', and carry a minimum of five one-dimensional (1D) arrays for the base state temperature, nondimensional pressure and zonal horizontal velocity ($\bar{\theta}$,$\bar{\pi}$, and $\bar{u}$), along with two arrays for mean density at the u and w levels. Sample code for getting started with the GrADS visualization package is also included. My code adds a fifth 2D prognostic variable, for meridional horizontal velocity v, in case you wish to add Coriolis accelerations to the 2D model.

12.1 Grid setup

Figure 11.1 shows the grid setup. Remember, in each grid box the velocity components are defined on the outer edges and the scalar variables reside at the center. The physical domain is completely surrounded by a set of artificial grid boxes; this just makes coding the model a little easier. There are a total of NX points in the x direction (index i) and NZ points in the z direction (index k).

In the horizontal direction, the physical boundaries are u points and are located at i-index locations 2 and NX for Fortran (Fig. 11.1) or 1 and NX-1 for C++ and other zero-based index languages (Fig. 11.2). In the vertical, the physical boundaries coincide with w points at index locations 2 and NZ in Fortran or 1 and NZ-1 in C++. We will force w to be zero at these points, this corresponding to rigid, free-slip (frictionless) boundaries. For this task,

we'll take the domain to be laterally periodic, though we will likely want to reconsider this later.

First, we need to set up the required arrays. We will eventually be using the leapfrog scheme, a **three time level** method. Thus, for each prognostic variable — u, w, θ' and π' — we need to carry three arrays, corresponding to the past, present and future values. For the u variable, for example, these will be called `up`, `u` and `um`, respectively. There is a more storage-efficient way of handling this, but that is not our present concern.

So, we have something like this for Fortran

```
c set parameters
c grid dimensions
      parameter(nx=83,nz=42)
c grid box lengths
      parameter(dx=400.,dz=400.)
c variable arrays
      dimension thp(nx,nz),th(nx,nz),thm(nx,nz)
      dimension wp(nx,nz),w(nx,nz),wm(nx,nz)
      dimension up(nx,nz),u(nx,nz),um(nx,nz)
      dimension pip(nx,nz),pi(nx,nz),pim(nx,nz)
c base state vectors for theta, vapor, dim. pres., ndim. pres.,
c      and density at u and w levels
      dimension tb(nz),qb(nz),pb(nz),pib(nz),rhou(nz),rhow(nz)
```

or for C++

```
// grid dimensions
const int nx = 83;
const int nz = 42;

// grid box lengths
const double dx = 400.;
const double dz = 400.;

// base state vectors
double tb[nz], qb[nz], pb[nz], pib[nz], rhou[nz], rhow[nz];

// prognostic arrays
double thp[nx][nz], th[nx][nz], thm[nx][nz];
double up[nx][nz], u[nx][nz], um[nx][nz];
double wp[nx][nz], w[nx][nz], wm[nx][nz];
double pip[nx][nz], pi[nx][nz], pim[nx][nz];
```

To simplify things, we'll consider a **dry and neutral environment**, so set `tb` to 300.0 at every vertical point and set `qb` to zero. (Don't throw away the code that created the sounding used in Model Tasks 1 and 2; you may want to reuse that later.)

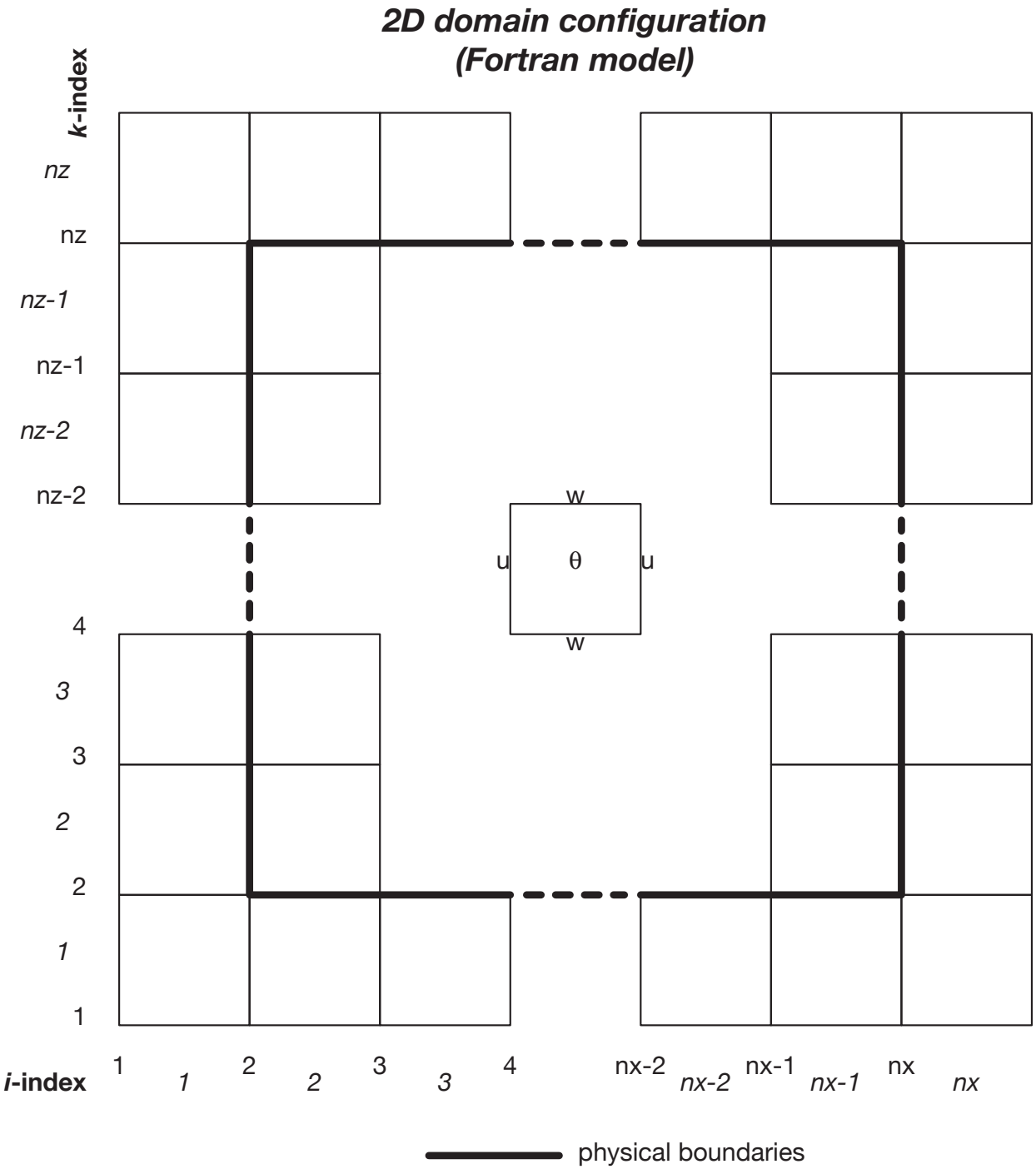

Figure 12.1: Model grid arrangement for Fortran.

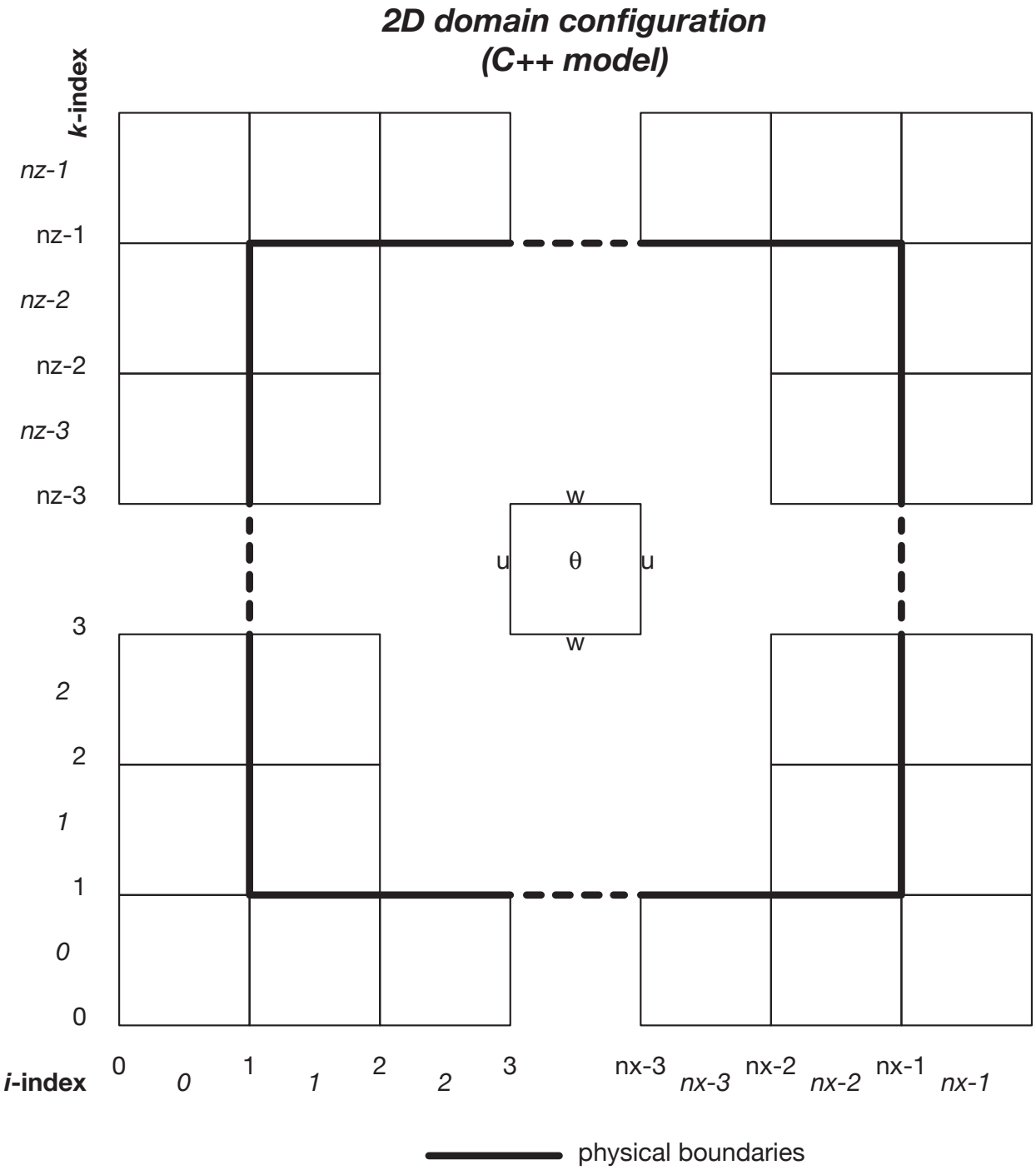

Figure 12.2: Model grid arrangement for C++ and other zero-based index languages.

Second, make sure the base state arrays are handled correctly at the artificial points above and below the physical grid. Since $w = 0$ at the physical boundaries, there can be no transport through the boundary. Thus, we can specify the temperature and pressure at these artificial points to be anything we want; it won't make a difference. For reasons better appreciated later, it is best to take the mean state fields to possess no gradient across the boundaries. We've already set up `tb` and `pib` (for example) for the real scalar points (`k = 2, NZ-1` in Fortran or `k = 1, NZ-2` in C++). Now we need to do:

```
TB(1)=TB(2)
TB(NZ)=TB(NZ-1)
PIB(1)=PIB(2)
PIB(NZ)=PIB(NZ-1)
RHOU(1)=RHOU(2)
RHOU(NZ)=RHOU(NZ-1)
```

or

```
tb[0] = tb[1];
tb[nz-1] = tb[nz-2];
pib[0] = pib[1];
pib[nz-1] = pib[nz-2];
rhou[0] = rhou[1];
rhou[nz-1] = rhou[nz-2];
```

12.2 Initial condition

Next we set up the initial perturbation. We'll define a radially symmetric thermal, consisting of a positive potential temperature perturbation. Take the horizontal radius, $RADX$, to be 4000 m and the vertical radius, $RADZ$, to also be 4000 m. The amplitude of the thermal is $\delta = 3$ K. Center the thermal at $ZCNT$= 3000 m above the ground. The equations are:

$$rad = \sqrt{\left[\frac{(z_T - ZCNT)}{RADZ}\right]^2 + \left[\frac{\Delta x(i - IMID)}{RADX}\right]^2} \tag{12.1}$$

and

$$TH(i,k) = \begin{cases} \frac{1}{2}\delta\left[\cos(rad \cdot \pi) + 1\right] & rad \leq 1; \\ 0 & \text{otherwise.} \end{cases} \tag{12.2}$$

$TH(i,k)$ represents the potential temperature perturbation at grid points i and k at the present time n. In the above π is 3.14159..., not nondimensional pressure. The best way of using trigonometric π in a program is to let the computer calculate it to machine precision. This can be done with

```
      TRIGPI=4.*ATAN(1.0)
```

(This is the same function in C++). As we saw earlier, the height at a scalar point z_T, is defined differently in Fortran and C++; in the former, it is $\Delta z\,(k-1.5)$ while in the latter it is $\Delta z\,(k-0.5)$.

`IMID` represents the middle θ point of the domain. For `NX` odd, it is located at point `(NX+1)/2` in the Fortran domain (starting with $i=1$) or `(NX-1)/2` in the C++ domain (starting with $i=0$). Using a symmetric thermal precisely at the domain's midpoint will help us test our code once the full model's time stepping loop is implemented in Model Task 5. For an initially calm environment, a symmetric perturbation placed at the exact center of the domain will generate a solution which must remain symmetric about the domain center (for scalars and w; the solution will be antisymmetric for u)... unless there are coding errors.

The leapfrog scheme, having three time levels, has to be started off not only with the present time value at each grid point, but also with the past time value (time level n-1). We do not generally have this information, so we typically assign the same perturbation to `THM(i,k)`. This actually causes some problems; we'll see this later.

This thermal perturbation will disturb the atmosphere. Normally, this will provoke both sound waves and gravity waves, but because we are assuming a neutral atmosphere, only the former will appear. We can attempt to construct a perturbation pressure field in balance with it. The perturbation hydrostatic equation may be written in continuous and discrete representations as:

$$\frac{\partial \pi'}{\partial z} \;=\; \frac{g}{c_{pd}}\frac{\theta'}{\bar{\theta}^2} \tag{12.3}$$

and

```
      TUP = TH(i,k+1)/(TB(k+1)**2)
      TDN = TH(i,k  )/(TB(k  )**2)
      PI(i,k) = PI(i,k+1)-0.5*(G/CPD)*(TUP+TDN)*DZ
```

or

```
tup = th[i][k+1]/(TB[k+1]*TB[k+1]);
tdn = th[i][ k ]/(TB[ k ]*TB[ k ]);
pi[i][k] = pi[i][k+1]-0.5*(g/cpd)*(tup+tdn)*dz;
```

In C++, remember to declare `tup` and `tdn` as doubles.

We have to assume a value (such as zero) for π' somewhere. It is easiest to choose this to be at the topmost real π' point (`PI(i,nz-1)`), so actually we need to integrate (12.3) *downward*. Thus, the above code has to be rearranged and remapped to fit this new use. We can do that in this fashion:

```
      DO i = 2,nx-1
c set the assumed point, and the fake point above it
       PI(i,nz-1) = 0.
       PI(i,nz   ) = 0.
c integrate perturbation hydrostatic equation downward
       DO k = nz-2, 2, -1
        TUP = TH(i,k+1)/(TB(k+1)**2)
        TDN = TH(i,k  )/(TB(k  )**2)
        PI(i,k) = PI(i,k+1)-0.5*(G/CPD)*(TUP+TDN)*DZ
       ENDDO
c get the artificial point below the grid
       PI(i,1)=PI(i,2)
      ENDDO

c finesse the first PIM values, over all points (real and fake)
      DO K = 1,nz
       DO i = 1,nx
        PIM(i,k) = PI(i,k)
       ENDDO
      ENDDO
```

```
          for(i = 1; i <= nx-2; i++){
 // set the assumed point, and the fake point above it
            pi[i][nz-2] = 0.;
            pi[i][nz-1] = 0.;
            for(k = nz-3; k >= 1; k--){
              tup = th[i][k+1]/(TB[k+1]*TB[k+1]);
              tdn = th[i][ k ]/(TB[ k ]*TB[ k ]);
              pi[i][k] = pi[i][k+1]-0.5*(g/cp)*(tup+tdn)*dz;
            } // end of for k
 // get the artificial point below the grid
            pi[i][0] = pi[i][1];
            } // end of for i

 // finesse the first PIM values, over all points (real and fake)
          for(i = 0; i <= nz-1; i++){
           for(k = 0; k <= nx-1; k++){
             pim[i][k] = pi[i][k];
           }
          }
```

In the above, I am using `nz1` and `nz2` to represent `nz-1` and `nz-2`, respectively. Of course, those constants have to be declared somewhere in the code.

You may wish to plot dimensional pressure perturbation p' instead. Recall from Chap. 3 that π' and p' are related by:

$$p' = \pi' c_{pd} \bar{\rho} \bar{\theta}_v.$$

This can be coded as:

```
PPRT(i,k) = PI(i,k)*CPD*RHOU(k)*TB(k)*(1.+0.61*QB(k))
```

in Fortran. Of course, for now we're neglecting moisture. However, just in case you want to add moisture later, you might as well be prepared for it.

Finally, make plots of the initial θ' and π' (or p') fields. Figure 11.3 presents initial fields for θ' and p', made using the GrADS package, described in the next section. Vertical profiles of θ' (in blue) and p' (in red) through the center of the thermal are presented in Fig. 11.4.

12.3 Visualization with GrADS

If you are programming in a language (such as Fortran) that does not natively support visualization, you have a number of options available. In this section, the GrADS (Grid Analysis and Display System) package is described. GrADS available for Mac, Linux/Unix and Windows. The GrADS home page is

`http://cola.gmu.edu/grads/grads.php`

As a visualization environment, GrADS has several significant advantages: it is free, easily installed, very scriptable, operable in interactive and batch modes, and can produce publication-quality graphics. Many of its downsides are due to its roots in climate science. GrADS natively and naively treats the horizontal dimension as longitude or latitude. It also has difficulty handling time increments less than 1 minute.

With GrADS, you will have two files for each model run: a control file, and a data file. The control file, with suffix `.ctl` is a text file that describes the contents of the binary `.dat` file. On the course web page, I provide sample Fortran code that can be used (or adapted) to create both files, a template that illustrates the sequence of events involved, and a sample GrADS script. These are also reproduced farther below.

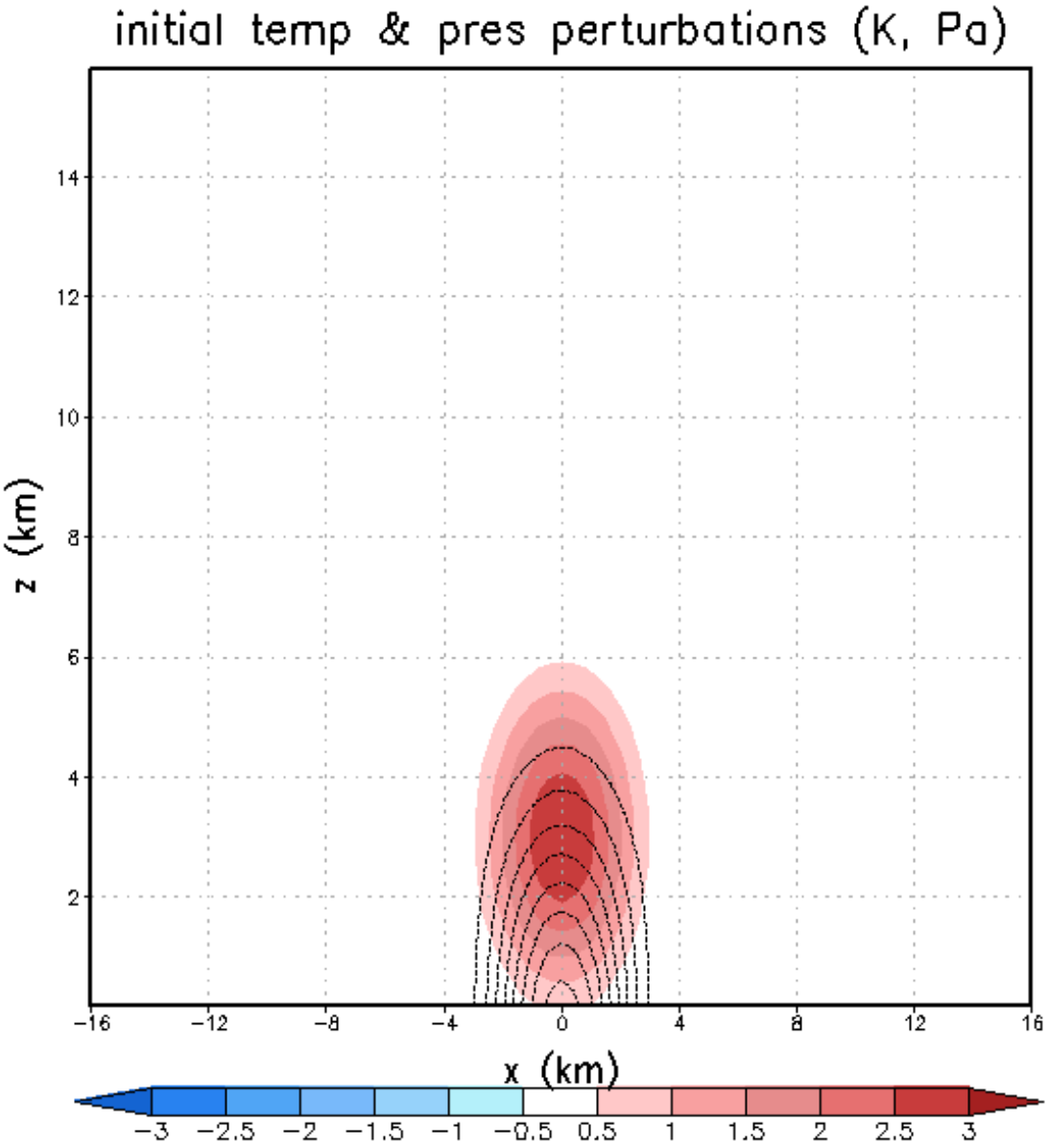

Figure 12.3: Initial potential temperature (shaded as shown, in K) and dimensional pressure (50 Pa contours, starting with -50 Pa) perturbation fields for our initial thermal.

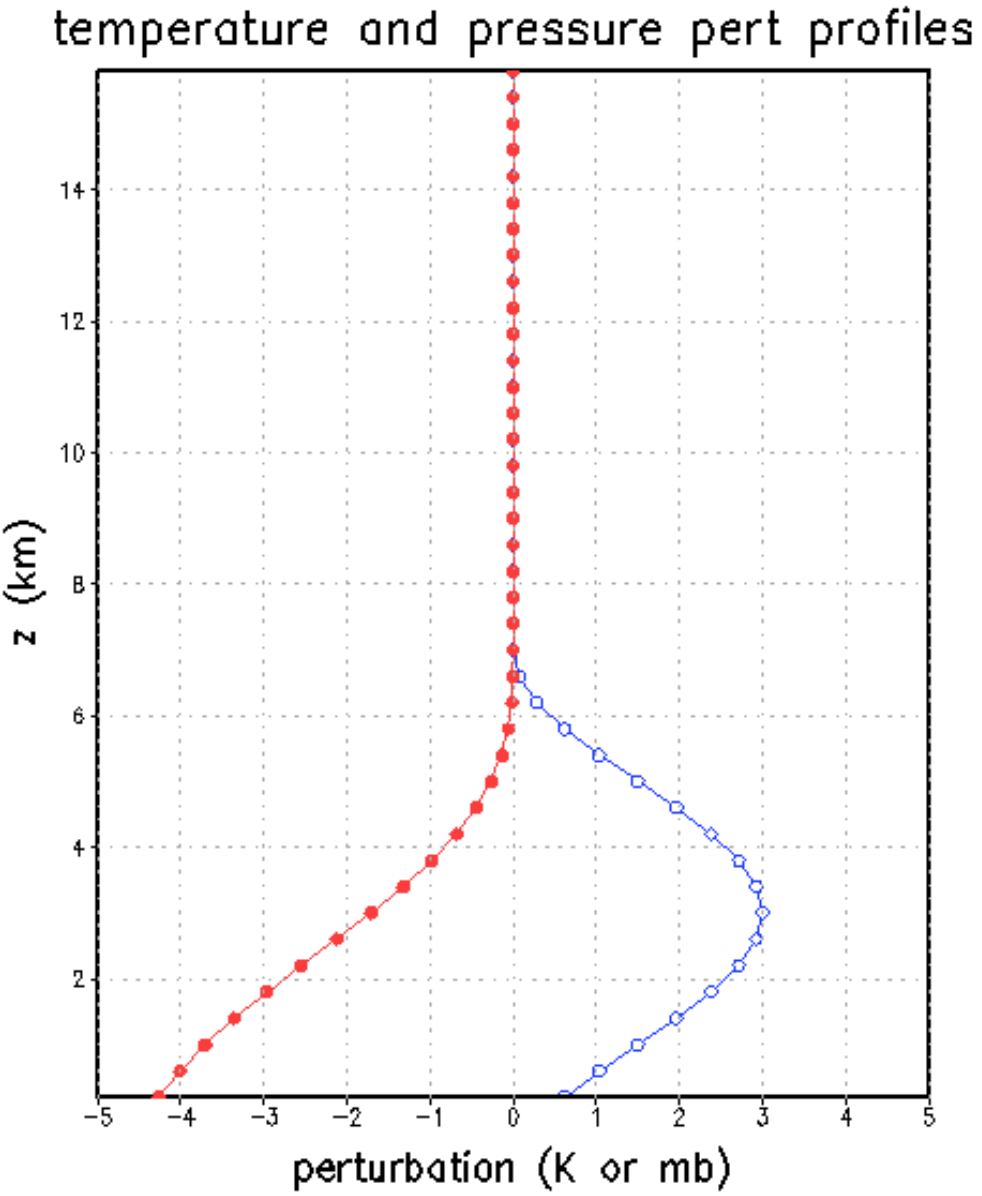

Figure 12.4: Vertical profiles of initial potential temperature (K; blue) and dimensional pressure (mb; red) up through the center of the initial thermal. Millibars are used for convenience.

Particularly useful GrADS-related links include:

Documentation index: `http://cola.gmu.edu/grads/gadoc/gadocindex.html`
User guide and scripting basics: `http://cola.gmu.edu/grads/gadoc/users.html`
The GrADS control file: `http://cola.gmu.edu/grads/gadoc/descriptorfile.html`
Controlling colors: `http://cola.gmu.edu/grads/gadoc/colorcontrol.html`
The GrADS script library: `http://cola.gmu.edu/grads/gadoc/library.html`

The example program and GrADS-based subroutines printed out below are also available on the course website.

```
c GrADS-enabled example model code for ATM562 (Fortran 77 version)
c http://www.atmos.albany.edu/facstaff/rfovell/ATM562/grads_example_code_augmented.f
c version of 10/24/2018

c this template illustrates a procedure to write data to GrADS files:
c  one GrADS .dat file is created, and TWO GrADS .ctl files
c  the first .ctl file is temporary, to help you visualize model
c  output during a long run, or if a model crash prevents creation
c  of the final .ctl file (which is done last)

      program task3

c -----------------------------------------------------------------
c  your non-executable code, including parameter, dimension, data
c  NOTE my GrADS code expects certain arrays, including a base state
c   mean wind (ub) and a 3D array for north-south velocity (v) that
c   can be utilized for other variables for models lacking Coriolis
c  ADDITIONALLY the leapfrog scheme needs three time levels;
c   I designate these as thp, th, and thm, for perturbation theta
c  SO your code may include the following:
c -----------------------------------------------------------------
c base state vectors
      dimension tb(nz),qb(nz),pb(nz),pib(nz),rhou(nz),rhow(nz),ub(nz)
c 2D prognostic arrays - 3 time levels
      dimension thp(nx,nz),th(nx,nz),thm(nx,nz)   ! pot. temp. pert.
      dimension wp(nx,nz),w(nx,nz),wm(nx,nz)      ! vert. vel.
      dimension up(nx,nz),u(nx,nz),um(nx,nz)      ! zonal horiz. vel.
      dimension vp(nx,nz),v(nx,nz),vm(nx,nz)      ! merid. horiz. vel.
      dimension qvp(nx,nz),qv(nx,nz),qvm(nx,nz)   ! water vapor
      dimension pip(nx,nz),pi(nz,nz),pim(nx,nz)   ! ndim pert. pres.

[...]

c -----------------------------------------------------------------
c  end of non-executable code here
c -----------------------------------------------------------------

c -----------------------------------------------------------------
c  your model setup code is here
c -----------------------------------------------------------------
c  * declare your time step, grid spacings (dt,dx,dz)
c  * declare your model end time, plotting interval (timend,nplt)
c  * set up your 1D base state (tb,pib,rhou,rhow,ub)
c  * set your model 2D initial condition (thermal, etc.)
c examples... you need to change these

      dt=5.0                  ! seconds
      timend=1000             ! seconds
      nplt=60                 ! plot every 60 time steps, or 300 sec

[...]

c -----------------------------------------------------------------
c  GrADS initialization is here
```

```
c ---------------------------------------------------------------
c  * set your casename (will create 'casename'.ctl, 'casename'.dat)
c  * create a temporary control (*.ctl) file
c  * write initial condition to data (*.dat) file
c EXAMPLES... you need to change these

      casename='test' ! declare 'character*80 casename' above
      iplt = 0        ! counter for plotting calls
      byteswap = 0    ! if byteswapping not needed, set = 0
      igradscnt =999  ! grads counter - dummy value to start

c  * initial call to write_to_grads_ctl sets up temporary control file

      call write_to_grads_ctl(casename,nx,nz,dt,dx,dz,
     1 igradscnt,nplt,byteswap) ! create the temporary control file

c  * write out model data at initial time - i.e., call dumpgrads

      igradscnt =0    ! reset grads counter
       call dumpgrads(igradscnt,u,v,w,th,pi,qv,rhou,tb,pib,ub,qb,
     1   nx,nz,cpd)

c ---------------------------------------------------------------
c  model integration loop STARTS here
c ---------------------------------------------------------------
c  * update plot counters
c  * integrate your equations, take care of boundary conditions
c  * set for next time step
c example...

       iplt = iplt + 1


c ---------------------------------------------------------------
c  decide if it's time to plot...
c ---------------------------------------------------------------
c  * if it's time to plot, call dumpgrads and reset plot counter
c example...

      if(iplt.eq.nplt)then
       call dumpgrads(igradscnt,u,v,w,th,pi,qv,rhou,tb,pib,ub,qb,
     1   nx,nz,cpd)
       iplt=0
      endif



c ---------------------------------------------------------------
c  model integration loop ENDS here
c ---------------------------------------------------------------
c  * go back to start of model integration loop unless it's timend
```

```
c ---------------------------------------------------------------
c  main routine ENDS here
c ---------------------------------------------------------------
c  * before you stop model, create FINAL grads control file
c  * this control file will have the proper number of times listed

      call write_to_grads_ctl(casename,nx,nz,dt,dx,dz,
     1 igradscnt,nplt,byteswap)

      stop
      end

c ---------------------------------------------------------------
c  after your main routine, include GrADS subroutines:
c    subroutine dumpgrads
c    subroutine write_to_grads_dat
c    subroutine write_to_grads_ctl
c  * you can also keep them in separate files and link to them
c  * EXAMPLE:   gfortran task3.f grads_routines.f
c ---------------------------------------------------------------
c ===============================================================
```

The code below contains Fortran subroutines for creating and writing to GrADS control and data files. Note the call to `dumpgrads` requires 5 2D arrays (`u, v, w, th, pi`), representing the 4 model prognostic variables and an array nominally for meridional velocity `v`; five 1D arrays with base state values (`rhou, tb, pib, ub, qb`); and the specific heat at constant pressure (`cpd`), which is used to compute the dimensional pressure perturbation. Although our model will be strictly 2D initially, the north-south velocity (`v`) is included in the event that you wish to include Earth rotation later. This array, which is hardcoded at the scalar grid location, can also be used for other purposes.

```
c ================================================================
c GrADS routines for ATM562 (Fortran 77 version)
c http://www.atmos.albany.edu/facstaff/rfovell/ATM562/grads_routines_augmented.f
c this version also carries water vapor 2D field (qv) and base state (qb)
c version of 10/9/2018

c ======= OVERVIEW ===============================================
c subroutine dumpgrads expects the following to be passed as input
c   igradscnt - a counter
c   nx, nz - array dimensions
c   u, v, w, th, pi - 2D arrays dimensioned (nx, nz)
c   rhou, tb, pib, ub, qb - 1D base state arrays dimensioned (nz)
c   cpd - specific heat at constant pressure
c
c with this input, this routine writes
c   full and perturbation u, th, and pi; full v; and
c     perturbation pressure (in mb)

c subroutine write_to_grads_ctl writes out the ctl file

c EXTRA/OPTIONAL:
c to add more fields to the GrADS output:
c   (1) alter dumpgrads to pass more fields, or compute them in dumpgrads
c   (2) add calls to write_to_grads_dat in dumpgrads for new fields
c   (3) update ngradsvars (# of variables written out) in write_to_grads_ctl
c   (4) add code in write_to_grads_ctl to describe the new field


c ----------------------------------------------------------------
c subroutine dumpgrads
c ----------------------------------------------------------------
      subroutine dumpgrads(igradscnt,u,v,w,th,pi,qv,rhou,tb,pib,ub,
     1 qb,nx,nz,cpd)
      dimension u(nx,nz),v(nx,nz),w(nx,nz),th(nx,nz),pi(nx,nz),
     1 qv(nx,nz)
      dimension rhou(nz),tb(nz),pib(nz),ub(nz),qb(nz)
      dimension temp(nx,nz)
      dimension zero(nz)

      nxm=nx
      nzm=nz
      do k=1,nzm
       zero(k)=0.
      enddo

c some calls pass a non-zero base state array to augment to the 2D field
      call write_to_grads_dat(u,nxm,nzm,nx,nz,zero,0.,1)    ! full U is predicted
      call write_to_grads_dat(u,nxm,nzm,nx,nz,ub,-1.,1)     ! pert u = U-ub
      call write_to_grads_dat(v,nxm,nzm,nx,nz,zero,0.,0)    ! v def. at scalar point

      call write_to_grads_dat(w,nxm,nzm,nx,nz,zero,0.,2)
      call write_to_grads_dat(th,nxm,nzm,nx,nz,tb,1.,0)     ! full pot temp
      call write_to_grads_dat(th,nxm,nzm,nx,nz,zero,0.,0)   ! pert pot temp
      call write_to_grads_dat(qv,nxm,nzm,nx,nz,qb,1.,0)    ! full qv
```

```
      call write_to_grads_dat(qv,nxm,nzm,nx,nz,zero,0.,0)   ! pert qv
      call write_to_grads_dat(pi,nxm,nzm,nx,nz,pib,1.,0)    ! full ndim pressure
      call write_to_grads_dat(pi,nxm,nzm,nx,nz,zero,0.,0)   ! pert ndim pressure
      do i=1,nxm
       do k=1,nzm
        temp(i,k)=0.01*rhou(k)*cpd*tb(k)*pi(i,k)
       enddo
      enddo
      call write_to_grads_dat(temp,nxm,nzm,nx,nz,zero,0.,0)  ! pert pressure in mb
      igradscnt=igradscnt+1
      print *,' done writing grads write number ',igradscnt
      return
      end

c --------------------------------------------------------------------
c subroutine write_to_grads_dat
c --------------------------------------------------------------------
      subroutine write_to_grads_dat(array,nxx,nzx,nxg,nzg,zmean,zfactor,
     1 iavg)
      dimension array(nxx,nzx),zmean(nzx)
      dimension dummy(nxg,1,nzg)
      do i=1,nxg
       do k=1,nzg
        dummy(i,1,k)=0.
       enddo
      enddo
      if(iavg.eq.0)then ! no averaging needed
       do i=1,nxg
        do k=1,nzg
         dummy(i,1,k)=array(i,k)+zfactor*zmean(k)
        enddo
       enddo
      else if(iavg.eq.1)then ! average in x-direction
       do k=1,nzg
        do i=1,nxg-1
        dummy(i,1,k)=0.5*(array(i+1,k)+array(i,k))+zfactor*zmean(k)
        enddo
       enddo
      else if(iavg.eq.2)then ! average in z-direction
       do i=1,nxg
        do k=1,nzg-1
         dummy(i,1,k)=0.5*(array(i,k+1)+zfactor*zmean(k+1)
     1                    +array(i, k )+zfactor*zmean( k ))
        enddo
       enddo
      else if(iavg.eq.3)then ! average from strfcn point
       do i=2,nxg-1
        do k=2,nzg-1
         top_pair=0.5*(array(i+1,k+1)+array(i,k+1))+zfactor*zmean(k+1)
         bot_pair=0.5*(array(i+1, k )+array(i, k ))+zfactor*zmean( k )
         dummy(i,1,k)=0.5*(top_pair+bot_pair)
        enddo
       enddo
      endif ! iavg
```

```
      do k=2,nzg-1
       write(66) ((dummy(i,j,k),i=2,nxg-1),j=1,1)
      enddo
      return
      end



c ------------------------------------------------------------------
c subroutine write_to_grads_ctl
c ------------------------------------------------------------------
      subroutine write_to_grads_ctl(casename,nxg,nzg,dt,dx,dz,
     1 igradscnt,ipltint,byteswap)

      character*4 ctl,dat
      character*80 casename,ctlfile,datfile

      close(44)
      close(66)

      ngradsvars=11
      print *,' writing grads control file'
c      if(icall.eq.0)then ! first call
c create names for ctl, dat files
       knx=index(casename,' ')
       write(ctl,'(a4)') '.ctl'
       write(dat,'(a4)') '.dat'
       ctlfile=casename(1:knx-1)//ctl
       datfile=casename(1:knx-1)//dat
       write(6,*) 'casename is ',casename
       write(6,*) 'ctlfile is ',ctlfile
       write(6,*) 'datfile is ',datfile
       open(44,file=ctlfile,status='unknown')
       open(66,file=datfile,status='unknown',
     1   form='unformatted')
c      endif ! icall

c

      gradstinc=float(ipltint)*dt/60.  ! minutes
      igradstinc=ifix(gradstinc) ! if not integer number of min, time counted wrong
      print *,' nxg= ',nxg,' nzg= ',nzg,' gradstinc ',gradstinc

      write(44,900) datfile ! no underscores in ctl, dat file names
 900  format('DSET ^',a80)
      write(44,901)
 901  format('TITLE ATM 562 model',/,
     1 'OPTIONS sequential',/,'UNDEF -9.99E33')
      if(byteswap.eq.1) write(44,902)
 902  format('OPTIONS byteswapped')
c      write(44,100) nxg-2,dx/2/1000.,dx/1000. ! dx in km
      write(44,100) nxg-2,-1*float(nxg-2)*dx/2/1000.+0.5*dx/1000.,
     1 dx/1000. ! dx in km
 100  format('XDEF ',i3,' LINEAR ',2f10.5,/,'YDEF 1 LINEAR 1.0 1.0')
```

```
      write(44,101) nzg-2
 101  format('ZDEF ',i3,' levels')
      do k=2,nzg-1
       write(44,102) (float(k)-1.5)*dz/1000.
 102   format(f10.3)
      enddo
      if(igradstinc.lt.1)then
       print *,' *** WARNING - since plotting interval less ',
     1  'than one minute, GrADS will not report time ',
     1  'correctly.'
       print *,' *** Plot interval recorded as 1 min in ',
     1  'control file'
       igradstinc=1
      endif
      write(44,103) igradscnt,igradstinc
 103  format('TDEF ',i5,' LINEAR 00:00Z01JAN2000 ',i3,'mn')
      write(44,104) ngradsvars
 104  format('VARS ',i4)
      write(44,105) nzg-2
 105  format('u  ',i3,' 00   horizontal velocity')
      write(44,106) nzg-2
 106  format('upr',i3,' 00   pert horizontal velocity')
      write(44,199) nzg-2
 199  format('v  ',i3,' 00   north-south velocity')

      write(44,107) nzg-2
 107  format('w  ',i3,' 00   vertical velocity')
      write(44,108) nzg-2
 108  format('th ',i3,' 00   potential temperature')
      write(44,109) nzg-2
 109  format('thpr',i3,' 00   pert potential temperature')
      write(44,118) nzg-2
 118  format('qv ',i3,' 00   vapor mixing ratio')
      write(44,119) nzg-2
 119  format('qvpr',i3,' 00   pert vapor mixing ratio')
      write(44,110) nzg-2
 110  format('pi ',i3,' 00   ndim pressure')
      write(44,111) nzg-2
 111  format('pipr',i3,' 00  pert ndim pressure')
      write(44,112) nzg-2
 112  format('pprmb',i3,' 00  pert pressure in millibars')
      write(44,999)
 999  format('ENDVARS')
      close(44)

      return
      end
c =====================================================================
```

Here is the GrADS script that produced Fig. 11.3. Scripts can be a lot simpler than this. Note this script calls two others: `cbarn.gs`, to draw the colorbar, and `rgbset.gs`, to define the colors used. Those scripts are also available on the course web page.

```
* example GrADS plot script for model task 3
* ATM562
* http://www.atmos.albany.edu/facstaff/rfovell/ATM562/plot_init_cond.gs
* version of 10/9/2015

* define some nice colors via the rgbset script
'run rgbset.gs'

* set background color white and clear screen
'set display color white'
'clear'

* set map projection off
'set mproj off'

* this formats the virtual page
'set vpage 0 8.5 0.5 8.5'

* smooth contours. enabled until switched off.
'set csmooth on'

* ----------- make temperature perturbation plot -----------------
* set contour label, make thp plot, draw title and axes
* declare a color-shaded plot
'set gxout shaded'

* define colors and contour levels
* colors are as defined in rgbset.gs
'set clevs -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1.0 1.5 2.0 2.5 3'
'set ccols   49  47  45 44 43 42 0 0 62 63 64 65 67 69'


* this next line turns off the grads logo
'set grads off'
* override default GrADS axis labels
'set xaxis -16 16 4'
'd thpr'

* draw the colorbar. requires script cbarn.gs
'run cbarn 1 0 5 0.18'

* reset graphics output to contour
'set gxout contour'

* ----------- make pressure perturbation plot -----------------
* set contour color and contour interval
'set ccolor 1'
'set cint 50'
```

```
* suppress contour labels
'set clab off'
* this next line tries to suppress the zero contour
'set black 0 0'
* plot pprmb but convert to Pascals
'd pprmb*100'

* draw titles and labels
'draw title initial temp & pres perturbations (K, Pa)'
'draw xlab x (km)'
'draw ylab z (km)'


* ----------- make a PNG plot --------------------------------
'gxprint init_tp_pp.png'

* finish up
'set clab on'
```

The next script plots the vertical profiles of temperature and perturbation pressure.

```
* example GrADS plot script for model task 3
* ATM562
* http://www.atmos.albany.edu/facstaff/rfovell/ATM562/plot_vert_profile.gs
* version of 10/9/2015

* set background color white and clear screen
'set display color white'
'clear'

* set map projection off
'set mproj off'

* this formats the virtual page
'set vpage 0 8.5 0.5 8.5'

* smooth contours. enabled until switched off.
'set csmooth on'

* set location at domain center
'set lon 0'

* set range for horizontal plot
'set vrange -5 5'
'set grads off'

* plot temperature perturbation profile. color it blue
'set ccolor 4'
'd thpr'

* plot pressure perturbation profile (in mb). color ir red
```

```
'set ccolor 2'
'd pprmb'

* titles and labels
'draw title temperature and pressure pert profiles'
'draw xlab perturbation (K or mb)'
'draw ylab z (km)'

* ----------- make a PNG plot ---------------------------------
'gxprint init_tp_pp_profile.png'
```

Chapter 13

Model Task #4: Implementing the leapfrog scheme

As an intermediate step towards building the 2D model, this task gets you to implement the leapfrog scheme for a simple 2D case with constant advection velocity. This task demonstrates how to implement a time stepping loop for a simple equation. We will take the Model Task #3 code, add an cone-shaped initial condition for one of the variables, and build the loop. The resulting simulation will well illustrate the phenomenon of dispersion error.

13.1 2D linear advection

Specifically, we're going to solve the simple constant advection equation

$$v_t = -c_x v_x - c_z v_z,$$

where c_x and c_z are the advection speeds in the x and z directions, respectively. I have selected the variable v because I am already writing this field to the GrADS output files, and am defining it at the scalar location, but am not yet using it for anything. The leapfrog approximation to this is (written in explicit form):

$$v_{i,k}^{n+1} = v_{i,k}^{n-1} - c_x \frac{2\Delta t}{2\Delta x}\left[v_{i+1,k}^{n} - v_{i-1,k}^{n}\right] - c_z \frac{2\Delta t}{2\Delta z}\left[v_{i,k+1}^{n} - v_{i,k-1}^{n}\right].$$

The indices i and k are for the x and z directions. There is a reason why I kept the factor of 2 in both the numerator and denominator in the advection terms (watch for this below). This can be implemented in **Fortran** and **C++** as:

```
 vp(i,k)=vm(i,k)-cx*d2t*rd2x*(v(i+1, k )-v(i-1, k ))
1               -cz*d2t*rd2z*(v( i ,k+1)-v( i ,k-1))

 vp[i][k]=vm[i][k]-cx*d2t*rd2x*(v[i+1][ k ]-v[i-1][ k ])
                  -cz*d2t*rd2z*(v[ i ][k+1]-v[ i ][k-1]);
```

where `d2t` is $2\Delta t$, `d2x` is $2\Delta x$, `d2z` is $2\Delta z$, `cx` is c_x and `cz` is c_z, along with:

```
rd2x=1./d2x
rd2z=1./d2z
```

This turns some frequent divisions into multiplications, which are faster and more efficient. So, we've three arrays, `vp`, `v`, and `vm`, each dimensioned `nx` by `nz`. I have adopted `nx=nz`=51, so for **Fortran** the domain center is at index 26, and for **C++**/Python at index 25.

To facilitate this example, we will implement *periodic boundary conditions in both directions*. This means there are only `nx-2` and `nz-2` unique points. The **Fortran** `do` loops will span `i`= 2, `nx-1` in the x direction and `k` =2, `nz-1` in the z direction. The **C++** `for` loops will span `i` =1, `nx-2` and `k` =1, `nz-2`. Upon completion of these calculations, we need to fill out the non-unique points in the arrays, using coding like:

```
do k=2,nz-1
 v( 1,k)=v(nx-1,k)
 v(nx,k)=v(   2,k)
enddo
do i=1,nx
 v(i, 1)=v(i,nz-1)
 v(i,nz)=v(i,   2)
enddo

for(k=1; k<= nz-2; k++){
  v[ 0  ][k] = v[nx-2][k];
  v[nx-1][k] = v[ 1  ][k];
  }
for(i=0; i<= nx-1; i++){
  v[i][ 0  ] = v[i][nz-2];
  v[i][nz-1] = v[i][ 1  ];
  }
```

Take `dx` and `dz` to be 100 m, `dt` to be 50 sec, and `cx` = `cz` = 1 m s^{-1}. The initial perturbation will resemble Task #3's, but with `radx` = `radz` = 1000, `delt` = 10, and placed in the domain center. Because of the doubly periodic domain, the simulated cone should return to the domain center in 4900 sec. You can use code like this for **Fortran** and **C++**:

```
c Fortran
c imid and jmid are coordinates of domain midpoint
      imid=(nx+1)/2
      kmid=(nz+1)/2
c parameters for initial perturbation
      delt=10.
      radx=1000.
      radz=1000.
c trigpi is trigonometric pi
      trigpi=4.*atan(1.0)
      do i=2,nx-1
      do k=2,nz-1
       rad=sqrt((dz*(k-kmid)/radz)**2
     1         +(dx*(i-imid)/radx)**2)
       if(rad.ge.1.)then
        v(i,k)=0.
       else
        v(i,k)=.5*delt*(cos(trigpi*rad)+1.)
       endif
c we don't have VM at the initial time, so we fake it
       vm(i,k)=v(i,k)
      enddo
      enddo


// C++
// imid and jmid are coordinates of domain midpoint
      imid=(nx-1)/2;
      kmid=(nz-1)/2;
// parameters for initial perturbation
      delt=10.;
      radx=1000.;
      radz=1000.;
// trigpi is trigonometric pi
      trigpi=4.*atan(1.0);
      for(i = 1; i<= nx-2; i++{
         for(k = 1; k <= nz-2; k++){
           rad=sqrt(pow(dz*(k-kmid)/radz,2)
                   +pow(dx*(i-imid)/radx,2);
           if(rad >= 1.)
              v[i][k] = 0.;
           else
              v[i][k] = .5*delt*(cos(trigpi*rad)+1.);
// we don't have um at the initial time, so we fake it
           vm[i][k] = v[i][k];
         } // end of for k
      } // end of for i
```

This perturbation shouldn't be nonzero near the boundaries, but take care of the boundary points anyway, just to be safe:

```
c Fortran
c periodic BCs
      do k=2,nz-1
       v(1,k)=v(nx-1,k)
       v(nx,k)=v(2,k)
       vm(1,k)=vm(nx-1,k)
       vm(nx,k)=vm(2,k)
      enddo
c so now 1,nx and 2,nz-1 has been done
      do i=1,nx
       v(i,1)=v(i,nz-1)
       v(i,nz)=v(i,2)
       vm(i,1)=vm(i,nz-1)
       vm(i,nz)=vm(i,2)
      enddo
c and that took care of the remaining points
```

```
// C++
// periodic BCs
      for(k = 1; k <= nz-2; k++){
         v[ 0  ][k] = v[nx-2][k];
         v[nx-1][k] = v[ 1  ][k];
         vm[ 0  ][k] = vm[nx-2][k];
         vm[nx-1][k] = vm[ 1  ][k];
       }
// so now 0,nx-1 and 1,nz-2 are done
      for(i = 0; i <= nx-1; i++){
         v[i][ 0  ] = v[i][nz-2];
         v[i][nz-1] = v[i][ 1  ];
         vm[i][ 0  ] = vm[i][nz-2];
         vm[i][nz-1] = vm[i][ 1  ];
      }
// and that took care of the remaining points
```

Get ready to integrate. I like to set counters for both the time step and the model time. Also, remember since we dont have time `n-1` starting out, we need to do the first time step with a *forward time and center space scheme*. We can fake this easily, without much extra coding, by taking `vm` = `v` at the model start [we already did that above] and initially taking `d2t` = `dt` instead of $2\Delta t \ldots$ but just for the first time step. So we can code (in Fortran; similar code for C++):

```
c get ready to integrate
c N is my time step counter; start at N=0
      n=0
c TIMENOW is my model time keeper; start at TIMENOW=0
      timenow=0.
c TIMEND is my integration stop time: 4900 sec
      timend=4900
c My grid spacings
```

```
      d2x = dx + dx
      d2z = dz + dz
c but D2T is DT to start!
      d2t = dt
```

Now, build the integration loop. There's many ways of implementing a loop; one of the most straightforward is to do something like this:

```
c Fortran
c Get ready, get set - GO

1000  n = n + 1                                  ! update the timestep counter
      timenow = float(n)*dt                      ! update the time

c Are we through yet?  If so, bail out
      if(timenow.gt.timend) go to 1001

c
c ******************************
c The integration code goes here..
c ******************************
c

c Here is the end of the integration loop.  First, make sure we fix D2T.
c This gets done at the end of the first model time, and every time
c  thereafter - no big deal; it avoids an IF statement and doesn't slow
c  the model down enough to stress over
      d2t = dt + dt

c Loop back to label 1000
      goto 1000

c At TIME=TIMEND, the loop exits to here
1001  continue

c Close up shop..
```

```
// C++
// Get ready, get set - GO
      double temp = timend/dt;
      nmax = (int) temp;

      while(n < nmax)
      {
       n++;                                  // update the timestep counter
       timenow = n*dt;                       // update the model time

           /* integration code goes here */

           /* here is the end of the loop.  First, we make sure we
           fix D2T.  This gets done at the end of the first model time,
```

```
            and every time thereafter - no big deal; it avoids an IF statement
            and doesn't slow the model down enough to stress over */

          d2t = dt + dt;

        } // end of time stepping loop

// Close up shop..
```

As I said, there are more elegant ways of doing this main loop, but I find this inelegant way to be quite readable. However, note in the above that I'm handling specification of the current model time with something like `timenow = n*dt` instead of `timenow = timenow + dt`. The latter is more straightforward but can result in the accumulation of truncation error. (Try this with a fractional time step like 0.4 and see what happens.) That can be important if you need to use the `timenow` variable to trigger processes (including stopping the model integration) elsewhere in the code.

In the integration loop, do the forecast over the unique points, then clean up the boundary points, and finally get set for the next time step. Notice that since `d2t` is really `dt` and `vm` is `v` for the first time step, the leapfrog code really is implementing the forward time scheme for the first time step. Also note that we'll require `v` values outside of the ranges 2, `nx`-1 and 2, `nz`-1 (or 1, `nx`-1 and 1, `nz`-1 for zero-based index languages), but since we take care of the boundary points each time step, we can implement the leapfrog code without a lot of messy IF statements.

```
c Fortran
c In the integration loop:

c Loop over the unique points
  do k=2,nz-1
  do i=2,nx-1
       vp(i,k)=vm(i,k)-cx*d2t*rd2x*(v(i+1,k)-v(i-1,k))
     1                 -cz*d2t*rd2z*(v(i,k+1)-v(i,k-1))
       enddo
       enddo
c That was easy.  Now take care of the boundary points for VP
       do k=2,nz-1
        vp(1,k)=vp(nx-1,k)
        vp(nx,k)=vp(2,k)
       enddo
       do i=1,nx
        vp(i,1)=vp(i,nz-1)
        vp(i,nz)=vp(i,2)
       enddo
```

```
c Finally, get set for new time step.  For the next step,
c  V becomes VM and VP becomes V.  Note we do this over ALL points
c  since the boundaries are already fixed up.
  do k=1,nz
  do i=1,nx
       vm(i,k)=v(i,k)
       v(i,k)=vp(i,k)
      enddo
      enddo

c Now is it time for a plot?  If so, call GrADS subroutine here.


// C++
// In the integration loop:

// Loop over the unique points
      for(i = 1; i <= nx-2; i++){
        for(k = 1; k <= nz-2; k++){
          vp[i][k]=vm[i][k]-cx*d2t*rd2x*(v[i+1][ k ]-v[i-1][ k ])
                       -cz*d2t*rd2z*(v[ i ][k+1]-v[ i ][k-1]);
        }
      }
// That was easy.  Now take care of the boundary points for VP
      for(k = 1; k <= nz-2; k++){
         vp[ 0  ][k] = vp[nx-2][k];
         vp[nx-1][k] = vp[ 1  ][k];
       }
      for(i = 0; i <= nx-1; i++){
         vp[i][ 0  ] = vp[i][nz-2];
         vp[i][nz-1] = vp[i][ 1  ];
      }

// Finally, get set for new time step.  For the next step,
//  V becomes VM and VP becomes V.  Note we do this over ALL points
//  since the boundaries are already fixed up.
      for(i = 0; i <= nx-1; i++){
      for(k = 0; k <= nz-1; k++){
         vm[i][k] = v [i][k];
         v [i][k] = vp[i][k];
      }
      }

// Now is it time for a plot?  If so, call GrADS routine here.
```

13.2 Results of example integration

Figure 13.1 shows the initial condition and the forecasted fields at 1500 sec intervals. The contoured field is the leapfrog solution, superimposed on the exact solution, represented by the color shaded field. The true feature and its finite difference counterpart are being

advected in the "northeast" direction. Owing to the double periodicity assumption, the cone reenters the domain at the southwest corner, and the cone's virtual neighbors are also seen in the northwest and southeast quadrants.

Note the quick appearance of small-scale features in the wake and the deformation of the simulated cone. We can consider this as "noise" if we choose, but in reality this represents an integral part of the true cone structure that is being mishandled. Note further that the noise is propagating in the opposite direction to the main feature! For the 2D leapfrog scheme, the phase speed of small-scale waves such as $2\Delta x$ and $2\Delta z$ is not only incorrect in magnitude, but also in direction. Owing to the domain's periodicity, the noise field intersects itself, creating an interference pattern. There is also a small but discernible phase lag.

The simulated cone should remain axially symmetric about a 45° angle, the propagation direction. If your results differ, make sure you've placed the initial perturbation in the exact center of the domain, check your leapfrog scheme code, and re-examine how you handle the boundary conditions.

Note: For this model task, I have slightly modified my GrADS code with respect to how the "vertical" direction is handled. This is the code change in subroutine `write_to_grads_ctl`:

```
c original code - commented out
c      write(44,101) nzg-2
c 101  format('ZDEF ',i3,' levels')
c      do k=2,nzg-1
c       write(44,102) (float(k)-1.5)*dz/1000.
 102   format(f10.3)
c      enddo

c revised code starts
      write(44,1100) nzg-2,-1*float(nzg-2)*dz/2/1000.+0.5*dz/1000.,
     1 dz/1000. ! dz in km
1100  format('ZDEF ',i3,' LINEAR ',2f10.5)
c revised code ends
```

13.3 Specification of the exact solution

In this section, code for generating the exact or true solution is presented. The initial cone feature is recreated each time step, centered on a position that is shifted at speeds c_x and c_z. It is likely there is a more elegant way of accomplishing this, but this approach works.

The main problem is that the cone will exit the domain on one side and re-enter on the

other. Thus, there will be a period of time in which the cone is straddling the boundaries, possibly with "ghost" features in other parts of the domain. (See, for example, Fig. 13.1 at the northwest and southeast corners.) This cannot be handled in the simple manner by which we handled the periodic boundaries in the code.

Figure 13.2 presents the concept employed by my example code for placing the cone representing the exact solution. For simplicity, the cone is presumed to be translating in one direction, towards the right. At the time shown, the cone is in the process of exiting the domain and re-entering on the left side. We can consider this a combination of a physical cone (that is only partly within the domain) and its virtual counterpart (that is also partially in the physical domain).

The x coordinate for the center of the physical cone is `xmid`. This point falls within the domain. `xi` represents any grid point in the domain, whether residing within the cone or not. Since `xmid` is inside the domain and the domain is periodic, the centroid of the virtual cone will reside outside of the physical domain, at location `xmidmirror`. Our arbitrary grid point `xi` has distance `xloc` from the centroid of the physical cone, and distance `xlocmirror` from the virtual cone's center.

For reconstruction of the exact solution, the *smaller* of these two distances should be selected. In the example shown, that would be `xloc` since the virtual cone's center is much farther away. This distance will be used to determine if the grid point lies within the cone. Other points in the domain, however, will be closer to the virtual cone's center, and that distance will be used to determine if the point resides within the cone instead. This will permit reconstruction of the exact solution, even if it is the process of moving across the virtual boundaries.

In the code below, the exact solution is stored in the 2D array called `qv`, utilizing an array I write to my GrADS files that is not currently being used for moisture.

```
c ---------------------------------------------------------------------
c exact solution
c ---------------------------------------------------------------------

      xmid=dx*(nx+1)/2+cx*n*dt              ! Departure of cone centroid
      zmid=dz*(nz+1)/2+cz*n*dt              ! from initial position

      if(xmid.ge.nx*dx) xmid=xmid-(nx-2)*dx ! Passing the periodic
      if(zmid.ge.nz*dz) zmid=zmid-(nz-2)*dz ! boundary

      if(xmid.gt.dx*(nx+1)/2) then          ! The cone's mirror location
        xmidmirror=xmid-(nx-2)*dx           ! on other side of periodic bdry
```

```
else
  xmidmirror=xmid+(nx-2)*dx
endif
if(zmid.gt.dz*(nz+1)/2) then
  zmidmirror=zmid-(nz-2)*dz
else
  zmidmirror=zmid+(nz-2)*dz
endif

qv=0.                          ! start with a clean slate

do i=2,nx-1
do k=2,nz-1
 xi=float(i)*dx                ! Current location
 zk=float(k)*dz

 xloc=((xi-xmid)/radx)**2      ! Location relative to
 zloc=((zk-zmid)/radz)**2      ! domain midpoint

 xlocmirror=((xmidmirror-xi)/radx)**2 ! Distance to mirror beyond the
 zlocmirror=((zmidmirror-zk)/radz)**2 ! periodic boundary

 xloc=amin1(xloc,xlocmirror) ! Take smaller of the
 zloc=amin1(zloc,zlocmirror) ! two coordinates
 rad=sqrt(xloc+zloc)

 if(rad.lt.1.) qv(i,k)=.5*delt*(cos(trigpi*rad)+1.) ! Exact soln.

enddo
enddo
```

13.4 Animations using GrADS

The script included below can be used to create a sequence of PNG files that can be combined into an animated GIF. As written, the script starts making images from the first time (`dis_t = 1`) until the last time in the file (`_endtime`). Hitting the Enter/Return key is needed to advance from frame to frame. That can be eliminated by commenting out the `pull dummy` line in the code.

```
* example GrADS plot script for model task 4
* this version can save individual frames as png images
* ATM562
* http://www.atmos.albany.edu/facstaff/rfovell/ATM562/plot_cone_movie.gs
* version of 10/9/2018

'set display color white'
'clear'
'run rgbset.gs'

* display parameters
'set mproj off'
'set vpage 0. 8.5 0. 8.5'
'set parea 1 7.5 1 7.5'

* save individual plots as png images?
say 'Create png images? (1=yes ; 0=no)'
pull ans

* find final time in grads file
frame = 1
'q file'
rec=sublin(result,5)
_endtime=subwrd(rec,12)
say " endtime is " _endtime

* looping flag
runscript = 1

* start at time 1
dis_t = 1

* ================================================================
* MOVIE LOOP
* ================================================================
while(runscript)
'set t ' dis_t
'clear'

'set grads off'
'set ccolor 15'
'set cint 2.0'
'set black 0 0'
'set cthick 7'
'set gxout shaded'
'set clevs 0 2 4 6 8 10'
'set ccols 0 0 61  63  65  67  69'
'd qv'
'run cbarn.gs'

'set clab off'
'set gxout contour'
'set ccolor 2'
```

```
'set clevs 2 4 6 8 10'
'd qv'

'set clab on'
'set cthick 5'
'set ccolor 1'
'set cthick 6'
'set cint 2.0'
'd v'



* ======================================================================
* FINISH
* ======================================================================
if(ans)
if( frame < 10 )
'gxprint movie00'frame'.png  '
else
if ( frame < 100 )
'gxprint movie0'frame'.png  '
else
'gxprint movie'frame'.png  '
endif
endif
frame=frame+1
endif

* this next line makes you hit return key to advance
pull dummy

if ( dis_t=_endtime )
 runscript=0
endif
dis_t = dis_t + 1
endwhile
```

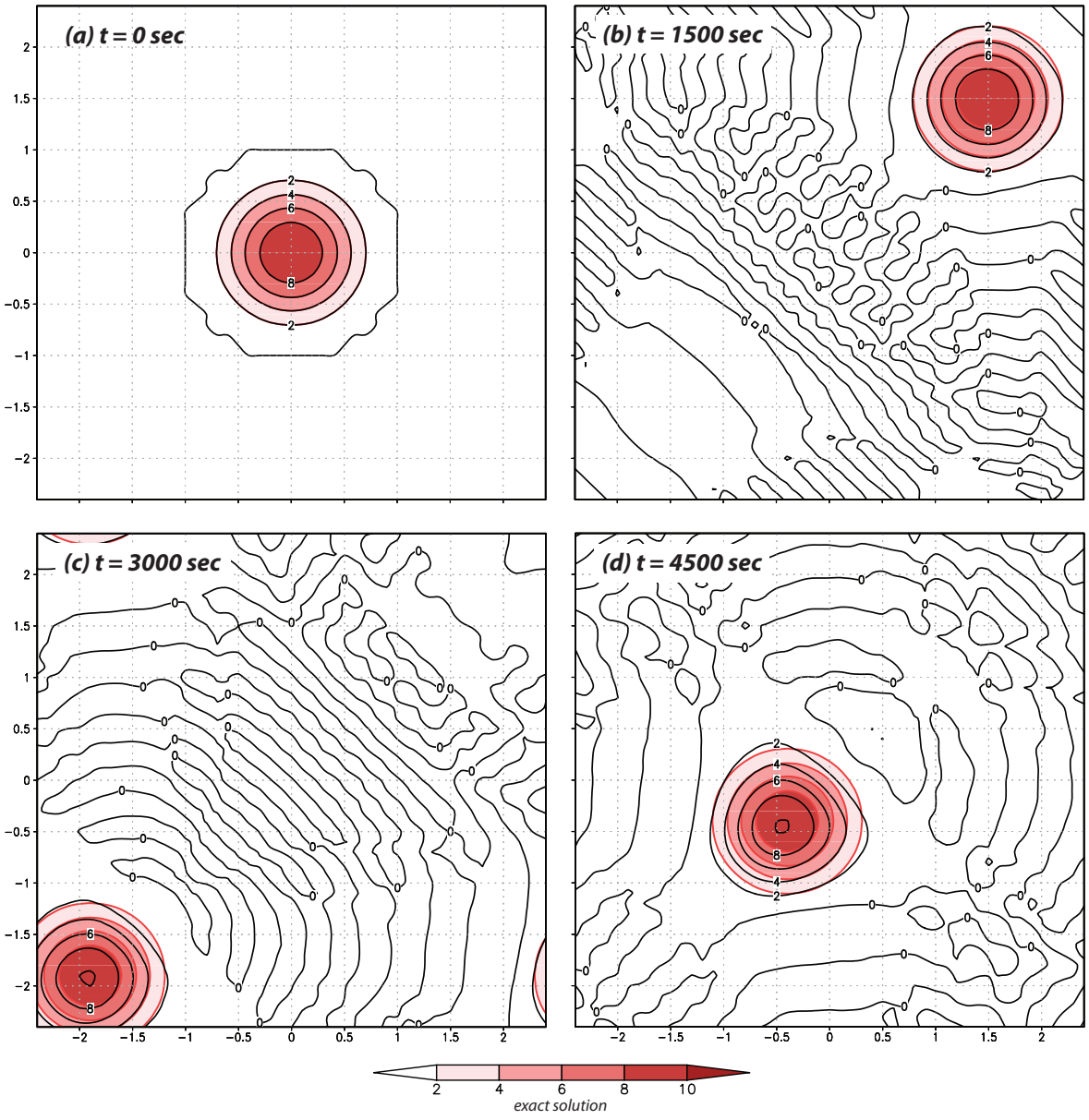

Figure 13.1: Fields at initial time and at 1500 sec intervals thereafter. Color shaded field represents exact solution. Contour interval is 2 units.

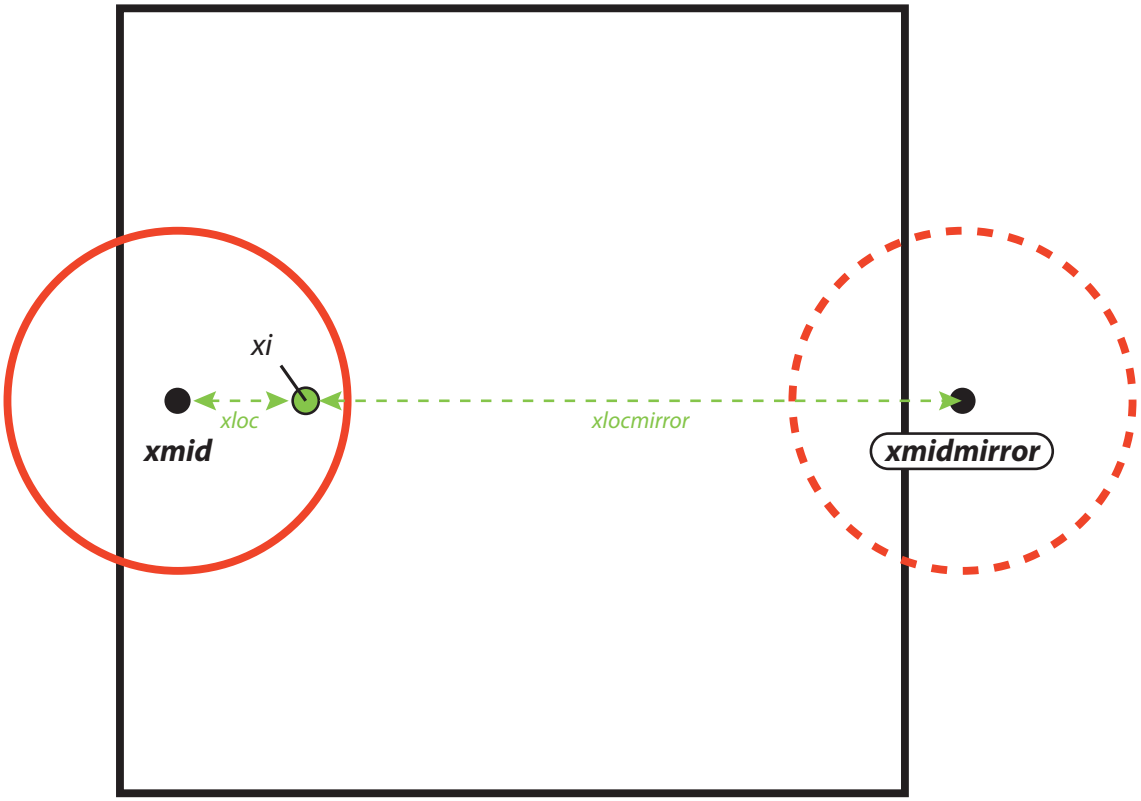

Figure 13.2: Concept for handling the placement of the cone representing the exact solution in a doubly periodic domain, the real (physical) extent of which is indicated by the thick black square.

Chapter 14

Model Task #5: Discretizing the model equations

14.1 Equations and flux form

This task starts up where Task #3 left off. In that task, we had created a neutral, calm and dry base state environment, defined a thermal perturbation as an initial condition and made sure the initial pressure field was in hydrostatic balance with it. In this task we discretize the model equations and perform a sample integration within a domain that is presumed to be periodic in the x direction and confined between flat, rigid plates in the z direction. Please keep in mind that the periodicity assumption is very confining and unrealistic for a finite domain, and can potentially exert an enormous influence on the results obtained. In the next chapter, the handling of open lateral boundaries is discussed.

The model equations we're going to solve (yet again) are:

$$\frac{\partial u}{\partial t} = -\vec{V} \cdot \nabla u - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} \tag{14.1}$$

$$\frac{\partial w}{\partial t} = -\vec{V} \cdot \nabla w - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + g\frac{\theta'}{\bar{\theta}} \tag{14.2}$$

$$\frac{\partial \theta'}{\partial t} = -\vec{V} \cdot \nabla \theta' - w\frac{d\bar{\theta}}{dz} \tag{14.3}$$

$$\frac{\partial \pi'}{\partial t} = -\frac{\bar{c_s}^2}{\bar{\rho} c_{pd} \bar{\theta}_v^2} \left[\nabla \cdot \bar{\rho}\bar{\theta}_v \vec{V} \right] \tag{14.4}$$

At this point, our model does not include moisture but the base state virtual temperature has been retained in the pressure gradient terms and the pressure equation. Incorporation of moisture will require *revising the w equation's buoyancy term* in addition to including at

least one equation for water substance.

Our handling of these equations will parallel that of Wilhelmson and Chen (1982), except that we will only implement second-order approximations. They didn't solve the equations quite in the form presented above. Rather, they rewrote the advection terms into *flux form*. Flux form has some desirable properties, but it is based on a set of assumptions.

Recall from discussion in Chapter 6 that the pressure equation (14.4) reduces to the anelastic continuity equation

$$\nabla \cdot \bar{\rho}\vec{V} = 0 \tag{14.5}$$

when the sound speed becomes infinitely fast and the base state environment is isentropic. Even though the problems we solve rarely satisfy those conditions, (14.5) is often used to rewrite the advection terms into flux form. As an example, take (14.1) and multiply it by the mean density, $\bar{\rho}$:

$$\frac{\partial \bar{\rho}u}{\partial t} = -\bar{\rho}u\frac{\partial u}{\partial x} - \bar{\rho}w\frac{\partial u}{\partial z} - \bar{\rho}c_{pd}\bar{\theta}_v\frac{\partial \pi'}{\partial x}$$

The mean density was brought directly into the time derivative on the LHS because it isn't a function of time (it is only a function of height). Next, use the chain rule to produce:

$$\frac{\partial \bar{\rho}u}{\partial t} = -\frac{\partial \bar{\rho}uu}{\partial x} - \frac{\partial \bar{\rho}uw}{\partial z} + u\left[\nabla \cdot \bar{\rho}\vec{V}\right] - \bar{\rho}c_{pd}\bar{\theta}_v\frac{\partial \pi'}{\partial x}$$

The term in square brackets is simply (14.5), and is zero in an isentropic and anelastic atmosphere. Wilhelmson and Chen's model atmosphere was neither isentropic nor anelastic, but they employed flux form (neglecting the square bracketed term) because they stated the anelastic solution is "the desired one".

Actually, Wilhelmson and Chen did *not* write the θ' equation in flux form because of a concern that then the θ' and π' equations would no longer be independent. It seems that this isn't a problem with the QCOM approximation (or so I recall), though, since we've already neglected the f_π term that was originally in (14.4). (That term had $\frac{d\theta'}{dt}$ in it.) You can write the θ' equation in flux form or not (or code both and see if you can find a difference). In the example problem shown later, flux form was not used in the θ' equation. I have a switch in my model where I can go back and forth between flux form and regular ("advective") form for this equation.

What's missing from this? So far, we have no artificial diffusion of any kind, be it time smoothing to control the leapfrog odd-even time step separation, or spatial smoothing to restrain nonlinear computational instability. Those will need to be added in the future as more complex phenomena are simulated.

14.2 Discretization

Recall we have a staggered grid, with periodic lateral boundaries (applied at u locations `i = 2` and `nx` in Fortran) and rigid lower and upper boundaries (applied at w locations `k = 2` and `nz` in Fortran). Thus, for all fields, the unique points fall in the range `i = 2, nx-1`, inclusive. For u, θ' and π', the unique points in the vertical direction fall between `k = 2, nz-1`, inclusive. For w, the inclusive range is between `3` and `nz-1`, since we don't need to solve for w at `k = 2` and `nz` (it's zero; why bother?).

The flux form equations are:

$$\frac{\partial u}{\partial t} = -\frac{\partial uu}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} uw}{\partial z} - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} \tag{14.6}$$

$$\frac{\partial w}{\partial t} = -\frac{\partial uw}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} ww}{\partial z} - c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} + g\frac{\theta'}{\bar{\theta}} \tag{14.7}$$

$$\frac{\partial \theta'}{\partial t} = -\frac{\partial u\theta'}{\partial x} - \frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} w\theta'}{\partial z} - w\frac{d\bar{\theta}}{dz} \tag{14.8}$$

$$\frac{\partial \pi'}{\partial t} = -\frac{\bar{c_s}^2}{\bar{\rho} c_{pd} \bar{\theta}_v^2}\left[\bar{\rho}\bar{\theta}_v \frac{\partial u}{\partial x} + \frac{\partial \bar{\rho}\bar{\theta}_v w}{\partial z}\right] \tag{14.9}$$

Note that in (14.6)-(14.8) we've divided through by $\bar{\rho}$. Also, the advection of θ' has been written in flux form but vertical advection of mean potential temperature cannot be. If you choose to use advective form for θ', then the equation is simply:

$$\frac{\partial \theta'}{\partial t} = -u\frac{\partial \theta'}{\partial x} - w\frac{\partial \theta'}{\partial z} - w\frac{d\bar{\theta}}{dz} \tag{14.10}$$

In this case, the two vertical advection terms may be combined, if desired.

14.2.1 The u equation

Clarity is more important that efficiency in the following discussion, so feel free to code these equations more compactly. First, we look at the time derivative term, recognizing we'll have to rewrite it into explicit form rather than the form shown here:

$$\frac{\partial u}{\partial t} = \frac{u_{i,k}^{n+1} - u_{i,k}^{n-1}}{2\Delta t}$$

The other time derivatives are handled in the same way. The horizontal advection term in the u equation will be discretized in this manner (see Fig. 1 for grid layout and index assignments):

$$-\frac{\partial uu}{\partial x} = -\frac{1}{\Delta x}\left[\left[\frac{u_{i+1,k}^n + u_{i,k}^n}{2}\right]^2 - \left[\frac{u_{i,k}^n + u_{i-1,k}^n}{2}\right]^2\right]$$

To get the horizontal gradient of u^2 centered at the u point, we first have to average u^2 to the scalar point, and then divide by the distance between the scalar points (Δx). Note that although the difference is not done over $2\Delta x$, it is still centered and second-order accurate. This is one advantage of grid staggering.

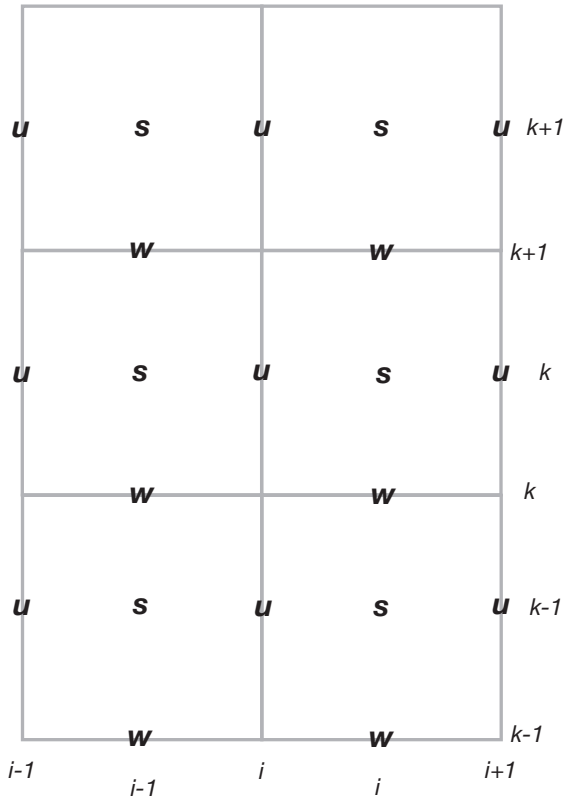

Figure 14.1: Grid staggering and indexing ("s" stands for scalar).

The vertical advection of u will be discretized as follows. The vertical term has to be centered at the here/now u point, which means we have to average u vertically as well as average w horizontally. Note $\bar{\rho}_{u,k}$ is the mean density evaluated at the kth u level, while $\bar{\rho}_{w,k}$ is evaulated at the kth w level. This is more convenient than constantly averaging density in the vertical direction.

$$
\begin{aligned}
-\frac{1}{\bar{\rho}}\frac{\partial \bar{\rho} u w}{\partial z} &= -\frac{1}{\bar{\rho}_{u,k}\Delta z} \\
&\quad \left[\bar{\rho}_{w,k+1}\frac{1}{2}(w^n_{i,k+1}+w^n_{i-1,k+1})\frac{1}{2}(u^n_{i,k+1}+u^n_{i,k})\right. \\
&\quad \left.-\bar{\rho}_{w,k}\frac{1}{2}(w^n_{i,k}+w^n_{i-1,k})\frac{1}{2}(u^n_{i,k}+u^n_{i,k-1})\right]
\end{aligned}
$$

The u equation's pressure gradient acceleration term is easy, a natural for this grid staggering. To center the horizontal derivative at the ith u point, we need the π values to the right and left, being at indices i and $i-1$, respectively:

$$-c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial x} = -c_{pd}\bar{\theta}_{v,k} \frac{1}{\Delta x} \left[\pi'^n_{i,k} - \pi'^n_{i-1,k} \right] .$$

Sample Fortran coding

Note in this sample an attempt has been made to line up the terms on the right hand side, to facilitate reading, understanding, and debugging.

```
c preliminary definitions --- redefined each time step
c  (because d2t is redefined after first time step is completed)
      dtx=d2t/dx
      dtz=d2t/dz
c do loop for U
c loop over unique points: i=2,nx-1 and k=2,nz-1
       do k=2,nz-1
        do i=2,nx-1
         up(i,k)=um(i,k)-.25*dtx*((u(i+1,k)+u(i,k))**2
     1                           -(u(i-1,k)+u(i,k))**2)
     2                  -.25*dtz*(rhow(k+1)*(w(i,k+1)+w(i-1,k+1))
     2                                     *(u(i,k+1)+u(i  ,k))
     3                           -rhow( k )*(w(i,k  )+w(i-1,k  ))
     3                                     *(u(i,k  )+u(i  ,k-1)))/rhou(k)
     4                  -dtx*cp*tbv(k)*(pi(i,k)-pi(i-1,k))
        enddo
       enddo
```

Now enforce the boundary conditions. There is no flow through the rigid upper and lower boundaries (since w there is zero), so it doesn't matter what u is just above and below the model surface. It helps simplify the coding if we just take u to be *zero-gradient* across these two boundaries. Then, we need to enforce periodicity. Boundary conditions for the other variables are discussed in the next section.

```
c zero gradient top and bottom over the unique points
      do i=2,nx-1
       up(i,1)=up(i,2)
       up(i,nz)=up(i,nz-1)
      enddo
c now k=1,nz has been done for the unique points
c now apply periodic lateral boundaries for all k
      do k=1,nz
       up(1,k)=up(nx-1,k)
       up(nx,k)=up(2,k)
      enddo
```

Sample Python coding

A straightforward port of the sample Fortran loop above to Python might result in the following code (`dtx` and `dtz` defined as above.). As Python uses zero-based indexing, the real points extend from 1 to `nx-2` in the horizontal and (for u and scalars) from 1 to `nz-2` in the vertical. However, in Python the *ranges do not include the final value indicated*, so this is coded as `1,nx-1` and `1,nz-1`. The rest of the code closely mimics its Fortran counterpart.

```
for i in range(1,nx-1,1):
    for k in range(1,nz-1,1):
       up[i,k] = um[i,k]
                 -0.25*dtx*(((u[i+1,k]+u[i,k])**2.0)
                           -((u[i-1,k]+u[i,k])**2.0))
                 -0.25*dtz*(rhow[k+1]*(w[i,k+1]+w[i-1,k+1])
                                      *(u[i,k+1]+u[i  ,k  ])
                           -rhow[k  ]*(w[i,k  ]+w[i-1,k  ])
                                      *(u[i,k  ]+u[i  ,k-1])
                                    )/rhou[k]
                 -dtx*cpd*tbv[k]*(pi[i,k]-pi[i-1,k])
```

A major deficiency of Python is that it performs repetitive computational tasks very slowly, and the code sample above is not competitive with Fortran. However, Python can be vectorized, and the code version below executes substantially faster. This code example was provided by Massey Bartolini.

```
up[1:-1,1:-1] = um[1:-1,1:-1]
                -0.25*dtx*(((u[2:  ,1:-1]+u[1:-1,1:-1])**2.0)
                          -((u[ :-2,1:-1]+u[1:-1,1:-1])**2.0))
                -0.25*dtz*(rhow[2:  ]*(w[1:-1,2:  ]+w[ :-2,2:  ])
                                      *(u[1:-1,2:  ]+u[1:-1,1:-1])
                          -rhow[1:-1]*(w[1:-1,1:-1]+w[ :-2,1:-1])
                                      *(u[1:-1,1:-1]+u[1:-1, :-2])
                                    )/rhou[1:-1]
                -dtx*cpd*tvb[1:-1]*(pi[1:-1,1:-1]-pi[:-2,1:-1])
```

The expression `up[1:-1,1:-1]` includes the predicted values of u from $i = 1$ to `nx-2`. The range end specified is "-1", which represents the last item in the sequence `nx-1`, but (again in Python) the last item in the range is not included. In several places, blanks are provided instead of indices, which means the default values [0, referring to both start and end] are being requested. Don't forget to handle the boundary conditions.

14.2.2 The w equation

The w equation's horizontal advection term is:

$$\begin{aligned} -\frac{\partial uw}{\partial x} &= -\frac{1}{\Delta x} \\ &\left[\frac{1}{2}(u^n_{i+1,k}+u^n_{i+1,k-1})\frac{1}{2}(w^n_{i+1,k}+w^n_{i,k})\right. \\ &\left.-\frac{1}{2}(u^n_{i,k}+u^n_{i,k-1})\frac{1}{2}(w^n_{i,k}+w^n_{i-1,k})\right]. \end{aligned}$$

Both u and w have to be averaged, while in the vertical advection term

$$\begin{aligned} -\frac{1}{\bar\rho}\frac{\partial \bar\rho ww}{\partial z} &= -\frac{1}{\bar\rho_{w,k}\Delta z} \\ &\left[\bar\rho_{u,k}\left[\frac{w^n_{i,k+1}+w^n_{i,k}}{2}\right]^2\right. \\ &\left.-\bar\rho_{u,k-1}\left[\frac{w^n_{i,k}+w^n_{i,k-1}}{2}\right]^2\right] \end{aligned}$$

only w has to be averaged. In the pressure gradient acceleration term, $\bar\theta_v$ has to be averaged to the w location:

$$-c_{pd}\bar\theta_v\frac{\partial \pi'}{\partial z} = -c_{pd}\frac{1}{2}\left[\bar\theta_{v,k}+\bar\theta_{v,k-1}\right]\left[\frac{\pi'^n_{i,k}-\pi'^n_{i,k-1}}{\Delta z}\right].$$

Finally, the buoyancy term is:

$$+g\frac{\theta'}{\bar\theta} = g\frac{1}{2}\left[\frac{\theta'^n_{i,k}}{\bar\theta_k}+\frac{\theta'^n_{i,k-1}}{\bar\theta_{k-1}}\right]$$

14.2.3 The θ' equation (in flux form)

In the horizontal advection term, we need to average θ' to the u locations before we can compute the derivative. Thus:

$$-\frac{\partial u\theta'}{\partial x} = -\frac{1}{\Delta x}\left[u^n_{i+1,k}\frac{1}{2}(\theta'^n_{i+1,k}+\theta'^n_{i,k})-u^n_{i,k}\frac{1}{2}(\theta'^n_{i,k}+\theta'^n_{i-1,k})\right].$$

The vertical advection of θ' is similar:

$$-\frac{1}{\bar\rho}\frac{\partial \bar\rho w\theta'}{\partial z} = -\frac{1}{\bar\rho_{u,k}\Delta z}\left[\bar\rho_{w,k+1}w^n_{i,k+1}\frac{1}{2}(\theta'^n_{i,k+1}+\theta'^n_{i,k})-\bar\rho_{w,k}w^n_{i,k}\frac{1}{2}(\theta'^n_{i,k}+\theta'^n_{i,k-1})\right].$$

The final term (for now) is the vertical advection of the mean state. Some people multiply and divide by $\bar{\rho}$, to be consistent with the other terms:

$$-\frac{w}{\bar{\rho}}\frac{d\bar{\rho}\bar{\theta}}{dz} = -\frac{1}{2}\frac{1}{\bar{\rho}_{u,k}}\left[\bar{\rho}_{w,k+1}w^n_{i,k+1}\frac{(\bar{\theta}_{k+1}-\bar{\theta}_k)}{\Delta z} + \bar{\rho}_{w,k}w^n_{i,k}\frac{(\bar{\theta}_k-\bar{\theta}_{k-1})}{\Delta z}\right]$$

I am not aware of a compelling reason for doing that. Note that since this equation is *not* in flux form, it consists of an average of two differences, rather than the difference of two averages.

14.2.4 The π' equation

The RHS of the π' equation has two terms, both multiplied by this coefficient:

$$-\frac{\bar{c_s}^2}{\bar{\rho}c_{pd}\bar{\theta}_v^2} = -\frac{c_s^2}{\bar{\rho}_{u,k}c_{pd}\bar{\theta}^2_{v,k}}$$

Theoretically, $\bar{c_s}$ is a function of height, as seen in Chapter 2, but we will take it to be a constant, externally supplied parameter. The first term is:

$$\bar{\rho}\bar{\theta}_v\frac{\partial u}{\partial x} = \frac{1}{\Delta x}\bar{\rho}_{u,k}\bar{\theta}_{v,k}\left[u^n_{i+1,k}-u^n_{i,k}\right],$$

while the second term is:

$$\frac{\partial\bar{\rho}\bar{\theta}_v w}{\partial z} = \frac{1}{\Delta z}\left[\bar{\rho}_{w,k+1}w^n_{i,k+1}\frac{1}{2}(\bar{\theta}_{v,k+1}+\bar{\theta}_{v,k}) - \bar{\rho}_{w,k}w^n_{i,k}\frac{1}{2}(\bar{\theta}_{v,k}+\bar{\theta}_{v,k-1})\right].$$

14.3 Boundary conditions

As noted above, we are presuming a horizontally periodic domain confined between flat and rigid plates. The periodic condition was illustrated above for the u equation and this is applied to all prognostic variables at the left and right boundaries. Rigidity and flatness imply no flow normal to the plates, and thus the first and last real w points are identically zero (i.e., for Fortran, `w(i,2) = w(i,nz) = 0.` at all `i`). As a consequence, the upper and lower boundary conditions do not actually matter for the other prognostic variables, so it suffices to presume zero-gradient conditions, as illustrated above for the u equation.

When we are finished with the boundary conditions, every (fake as well as real) point in the `up, wp, thp, pip` arrays has been given a value, and we are ready to set for the next time step. The order in which the prognostic equations are computed and boundary conditions applied does not matter, as long as the updating is performed *last* in the time stepping loop.

14.4 A sample integration

Figures 14.2-14.5 show fields of θ', w, u and π' through a sample integration, made with these values listed below (that are consistent with the model design of Model Task 3):

- Model domain of 83 by 42 points (so the model takes little memory).
- Calm, dry, isentropic atmosphere (300 K potential temperature).
- Surface pressure 965 mb.
- 400 m grid spacing in both directions (pretty coarse for this problem).
- Sound speed $c_s = 50$ m s^{-1} (*too slow*, but allows model to run quickly).
- Thermal specifications: 4000 m horizontal and vertical radii, centered at 3000 m above the ground, with maximum amplitude of 3 K.
- 2 sec time step.
- Integrated for 1200 sec.

Each figure shows the fields at the initial time, and at 400, 800 and 1200 sec. We can't make the atmosphere too deep because it's dry adiabatic (do you know why that is?). We can't integrate the model for too long; the thermal will eventually hit the rigid model top. Even before that, the periodic lateral boundaries will cause (physical, not necessarily numerical) problems for this narrow simulation domain. This is because periodic boundaries in an initially calm environment act as rigid walls, and are perfectly reflective.

We can observe the consequences of the reflective walls in all of the model prognostic variables, but it may be most obvious in the perturbation pressure field at the lowest scalar level and when presented as a Hovmöller (time-space) diagram (Fig. 14.6). In Model Task #3, we supplied the initial thermal with a hydrostatically-balanced pressure perturbation field. Close observation of the model integration reveals the initially supplied π' field is not sufficient to prevent sound waves from being excited by the temperature disturbance. Two waves emerge and propagate in opposite directions at speed c_s. After reflecting off the periodic boundaries, the waves return to the domain center, where they reflect yet again.

That Hovmöller diagram also helps us see that our initially supplied π' field was perhaps not very good. We assumed the adjustment was hydrostatic, which means each column was

handled in an independent manner, and only model columns that had some thermal perturbation inserted were affected. This is why the magnitude of the initial pressure perturbation at the surface was large in absolute terms; in reality, the adjustment should have been much broader, resulting in a smaller surface pressure reduction directly beneath the thermal. Note how quickly the initially supplied surface π' changed. This is evidence the adjustment was excessively large. Did it do more harm than good? Try making a simulation without the initial π'.

Also, keep statistics like domain maximum w and θ' every time step, and plot them. Do you see any temporal instability? What happens if you don't hydrostatically balance the initial thermal? If you properly center your initial condition, and have no coding errors, your solution should be perfectly axially symmetric (or, for u, perfectly anti-symmetric).

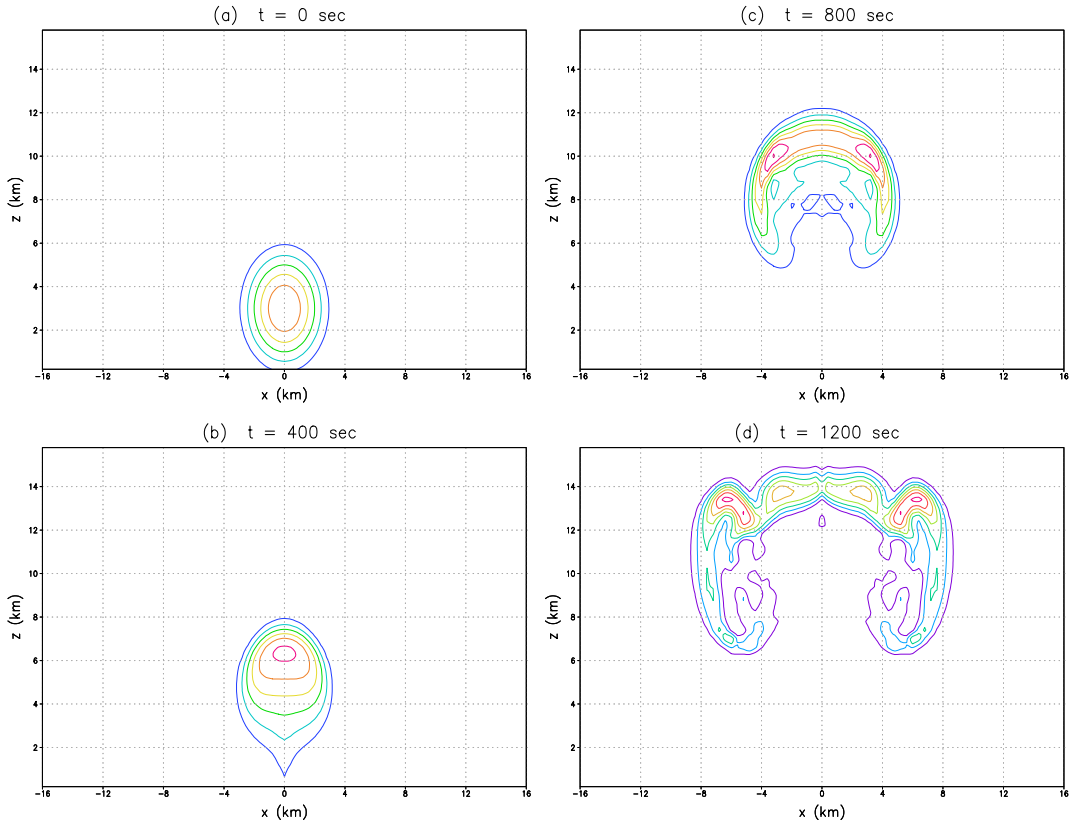

Figure 14.2: Perturbation potential temperature field at initial and three subsequent times. Contour interval 0.5 K; zero contour suppressed.

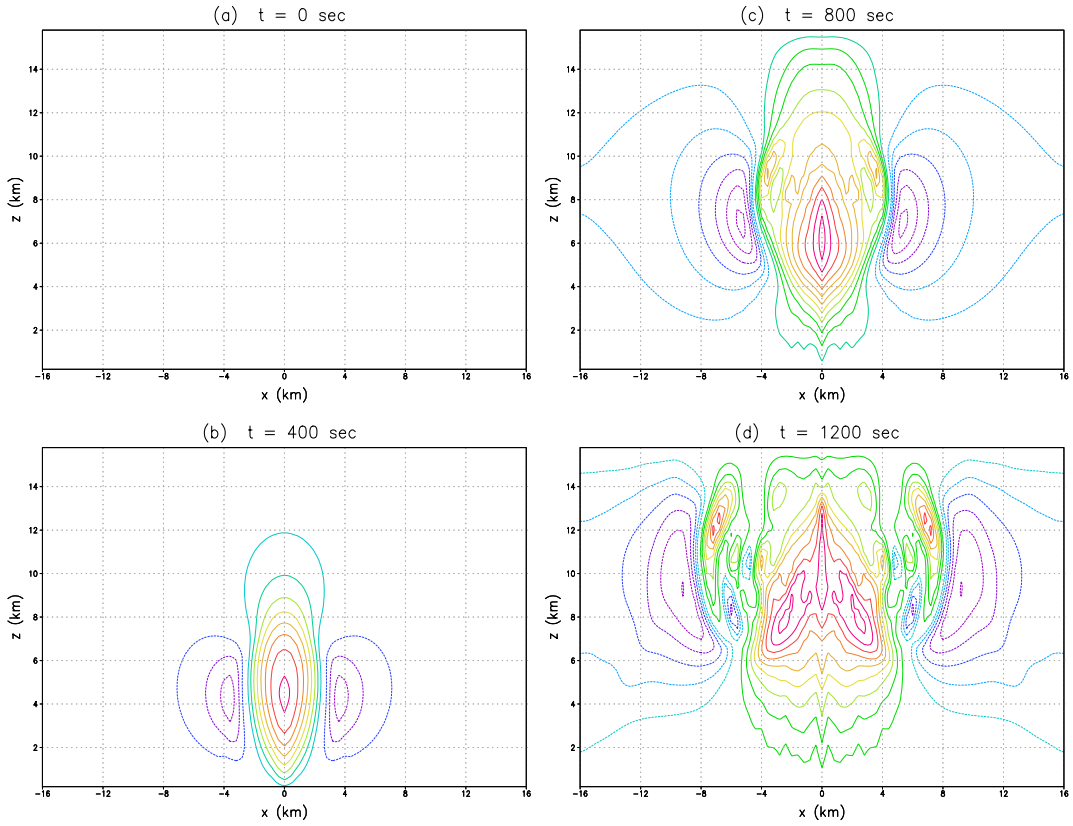

Figure 14.3: Vertical velocity field at initial and three subsequent times. Contour interval 2 m s^{-1}; zero contour suppressed.

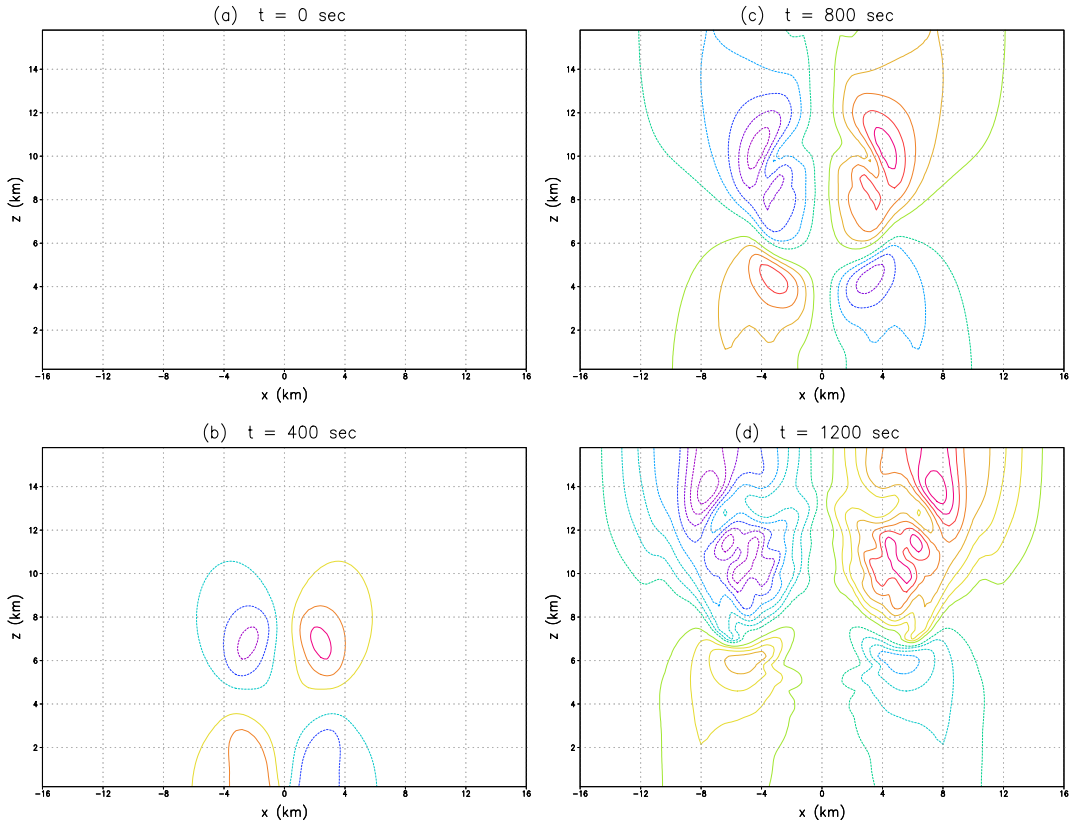

Figure 14.4: Horizontal velocity field at initial and three subsequent times. Contour interval 2.5 m s^{-1}; zero contour suppressed.

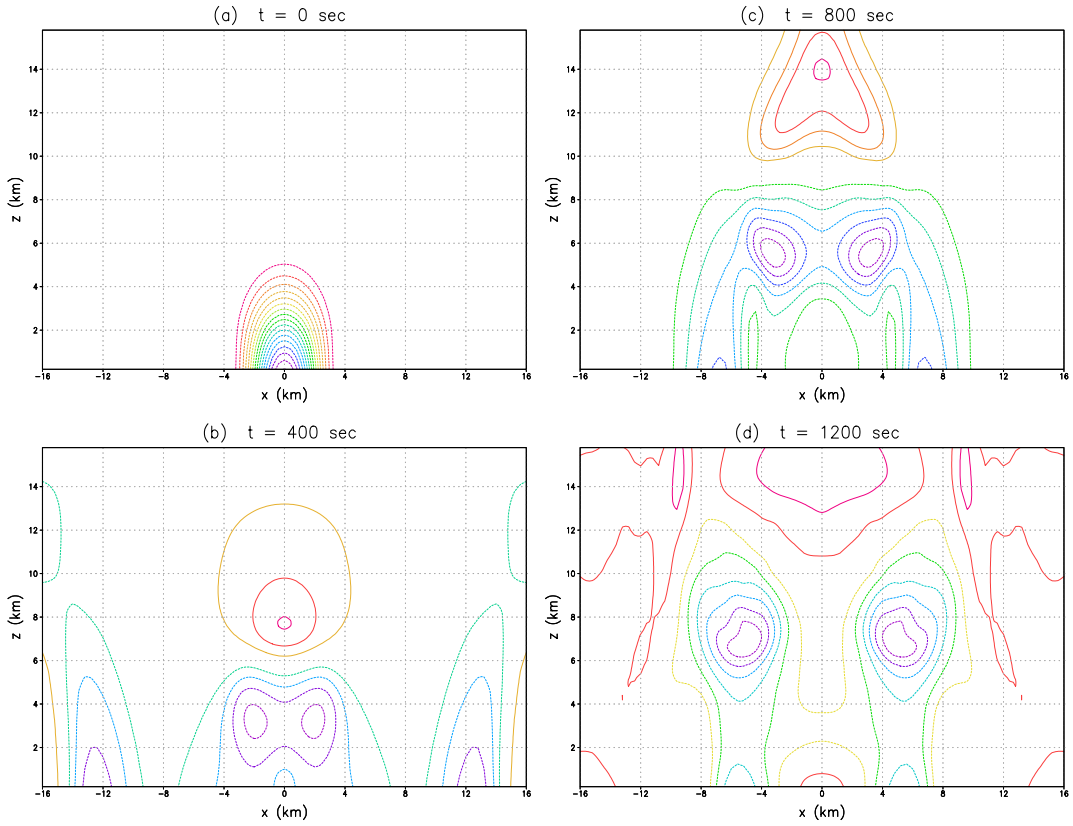

Figure 14.5: Dimensional pressure perturbation field at initial and three subsequent times. Contour interval 0.25 mb; zero contour suppressed.

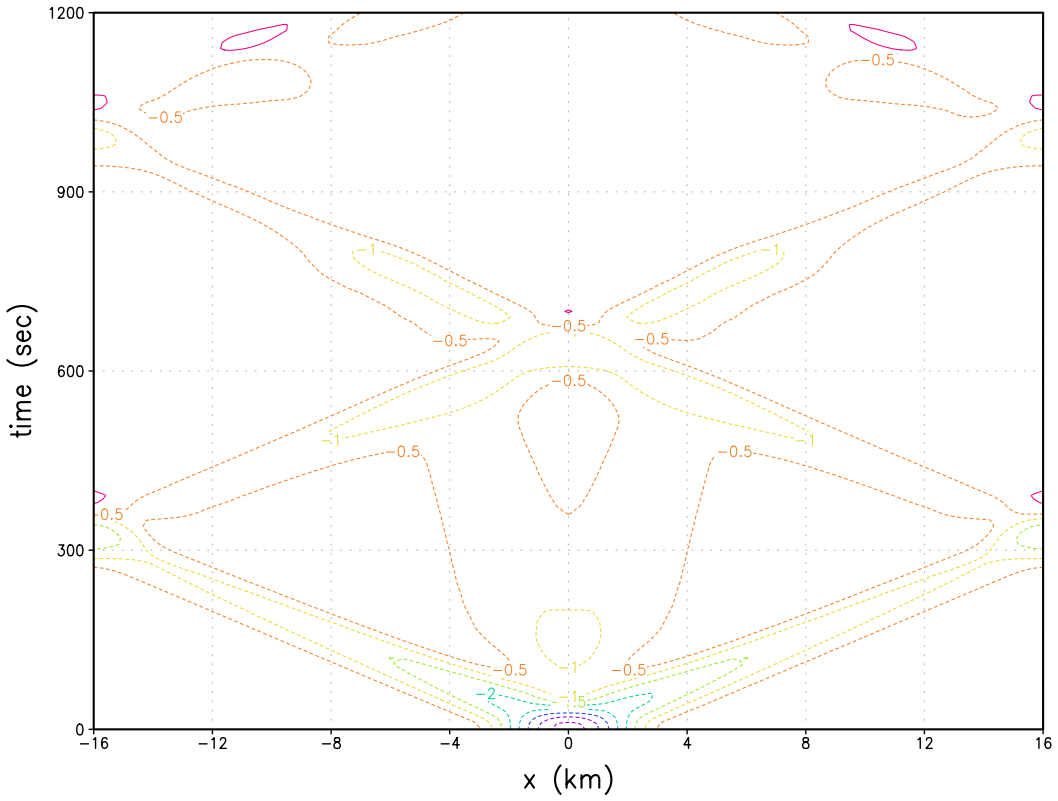

Figure 14.6: Hovmöller diagram of dimensional pressure perturbation field at the lowest scalar model level. Contour interval 0.5 mb; zero contour suppressed.

Chapter 15

Model Task #6: Next steps

15.1 Final project

The final modeling project will entail a written report, which can take the form of a PowerPoint presentation. The report should hit these main points:

- The problem to be simulated, including background relevant to the problem;
- The modifications to the model that had to be done to perform the simulation;
- A description of the basic results;
- The conclusions — what was learned from doing the project.

I don't expect amazing, new, earthshaking results here. Just a demonstration that effort has been expended and something was learned — about the physical problem simulated and/or about modeling in general.

As an example, you can augment Model Task #5 in a number of ways, such as by comparing your simulation results to those from laboratory and theoretical analyses (I can point out where those results may be found). Or, you can approach the problem from a pure modeling perspective and design experiments to address basic questions such as:

- What is the effect of changing the temporal and spatial resolution on the results?
- What differences are found if different numerical schemes are employed? (Such as second- vs. fourth-order leapfrog, Runge-Kutta, etc..)

- What errors are introduced by artificially slowing down the sound waves?
- In 2D, do a decomposition of the pressure field into buoyancy and dynamic pressure components by solving the anelastic pressure equation. Or make your model fully anelastic and compare results with compressible simulations.
- Make your model 3D and compare results between 2D and 3D. Why are 3D motions typically so much stronger?
- Add a passive tracer to your model and see how it gets pushed around by a thermal rising through its midst. (This involves adding another, simple equation to the model.) Where does the tracer go? For a tracer initialized outside of the original thermal perturbation, how much of the tracer winds up within the thermal itself?
- By making your thermal negatively buoyant, you can study how a cold air pool spreads along the ground, a sort of "dam break" problem. At sufficiently fine resolution, you can capture the development of Kelvin-Helmholtz waves along the interface between the chilled and ambient air. What resolution do you need to successfully capture these waves? Which is more important, vertical or horizontal resolution? What does diffusion do to the waves? Implement a passive tracer in your model, and perhaps initialize it outside the heat sink. How much tracer does the cold pool eventually entrain? Is that also a function of resolution?

One of those options, or a suitable alternative, is sufficient for a final modeling project. Suitable alternatives can include, and are not limited to, adding heat or momentum sources to the model, surface friction and/or surface heat fluxes, adding moisture and microphysics, and implementing non-flat lower boundaries to mimic flow over mountains or into canyons. This is certainly not an exhaustive listing. The rest of this chapter discusses how specific additions and improvements can be made to the Model Task #5 model.

15.2 Open lateral boundary conditions

15.2.1 Theoretical basis

Thus far, our model has employed periodic lateral boundary conditions (BCs), primarily for simplicity. We have already seen, however, that this represents a powerful constraint on the circulations that can develop within the domain. To alleviate this, we can try to "open

up" the domain, creating potential inflow and outflow boundaries. Oliger and Sundstrom's (1978) analysis demonstrated that for inflow boundaries, $p-1$ BCs are needed, where p is the number of equations being solved, whereas at outflow boundaries, only one boundary condition is required. The variable that obviously always requires a BC is u, as in the 2D model it is the only variable sited directly on the boundary when the grid is staggered.

Whether a domain side is an inflow or outflow boundary, however, is feature-dependent. Suppose there is a mean flow of u = -10 m s^{-1} across grid point $i = NX$. For advective phenomena, that is an inflow boundary, as the wind is directed into the domain from the outside. However, suppose a gravity wave with an eastward phase speed of 30 m s^{-1} is approaching that side. For that wave, the domain edge at $i = NX$ is an *outflow* boundary. Furthermore, it is possible that the wave will reflect upon reaching the edge of the domain, and subsequently progress inward, which is an undesirable result.

Below, we describe Klemp and Wilhelmson's (1978; "KW") approach to implementing open BCs. The idea is that a wave that might be approaching a domain side is identified and the velocity at the boundary is adjusted to try to let the wave pass by with minimal reflection by essentially "advecting" it out of the domain. Let c^* be the intrinsic phase speed of a wave moving independently of the flow, so $u \pm c^*$ is the wave's flow speed relative to the model domain. Specifically, $u + c^*$ is the approach speed for the eastern domain edge. This could represent an eastbound wave ($c^* > 0$) being hastened ($u > 0$) or slowed ($u < 0$ but also $|u| < c^*$) in its motion, or a westbound wave ($c^* < 0$) being pushed by a strong westerly current. Unless $u + c^* > 0$ on the domain's east end, however, that is not an outflow boundary, at least for that wave. Similarly, $u - c^* < 0$ is the wave's approach speed for the western boundary.

If a particular domain side is an inflow boundary for that wave, the local time tendency for u is set to zero. We do not actually know what resides outside our finite domain, so it is assumed that u there, whatever value it happens to have, is not changing. For the remaining model variables, a zero-gradient BC is applied. For a wave-relative outflow boundary, KW replaced the normal velocity's equation with a simplified version of the form

$$\frac{\partial u}{\partial t} = -\left(u \pm c^*\right) \frac{\partial u}{\partial x}, \tag{15.1}$$

where the plus (minus) sign is applied at the eastern (western) domain edge. Note the pressure gradient and vertical advection terms are absent; all we are doing is advecting a particular wave feature across the boundary[1].

[1]If Coriolis accelerations are incorporated into the model, these are retained in the KW boundary condition. See Klemp and Wilhelmson (1978, p. 1077).

But... *which* wave? Many may exist in a particular simulation and be approaching a given boundary, including sound waves and gravity modes, thereby representing a very wide spectrum of wave speeds. The KW approach involves the specification of a single value for c^*, and they adopted the intrinsic phase speed of the fastest-moving low-frequency gravity wave, which has phase speed of $\frac{Nz_T}{\pi}$, where the denominator represents trigonometric π, N is the Brunt-Väisälä frequency, and z_T is either the height of the domain top or the model tropopause. Referring to (2.38), we can see this represents a wave of infinite horizontal extent and a vertical wavelength of twice z_T.

With a tropospheric value of $N = 0.01$ s^{-1}, a z_T of 10 km (shallow even for a midlatitude warm-season tropopause) yields $c^* \approx 30$ m s^{-1}, which is what KW used in their 3D control experiment. KW's analysis showed that this BC will not reflect a wave having precisely the specified value of c^*. Yet, even if it is perfect for that one feature, that means that the potentially many other waves having different intrinsic speeds will reflect, at least to some degree. KW also tested values larger and smaller than this value (see their Fig. 10), and their analysis and simulation results suggested that overestimating c^* is less harmful than underestimating it.

It is noted here that other variants of this open BC exist. For example, Wilhelmson and Chen (1982) estimated separate, time-dependent c^* values just inside each lateral boundary. Durran and Klemp (1983) did something similar, but averaged vertically so c^* did not vary within a given model column.

15.2.2 Implementation

When the leapfrog scheme is employed, the original KW boundary condition can be coded in a manner like the following, where `cstar` is c^*. The BC is being applied to the u equation for `i = 2` and `nx`, which are the zonal velocity's boundary points. Note that the gradient of u is being computed at time $n - 1$. This is necessary to maintain stability.

```
  dtx=d2t/dx
  do k=2,nz-1
     up(2 ,k)=um(2 ,k)
1      -amin1((u(2 ,k)-cstar),0.)*(um(3,k)-um(2,k))*dtx
     up(nx,k)=um(nx,k)
1      -amax1((u(nx,k)+cstar),0.)*(um(nx,k)-um(nx-1,k))*dtx
   do i=3,nx-1
      [code for non-boundary points]
   enddo  ! i
  enddo   ! k
```

15.3 Solving the anelastic pressure equation

The anelastic continuity and pressure equations were introduced in Chapter 6, and an example of solving a simple version of the elliptic pressure equation was demonstrated. Here, we discuss how to solve the full pressure equation in a 2D framework. This can be implemented either as part of an anelastic model, or within a compressible model to provide an anelastic version of the pressure field that can be separated into components such as dynamic and buoyancy pressure. In the latter case, the anelastic pressure perturbation would not be the same as that prognosed in the model, and would not be formally used in the model integration.

In Chapter 6, we derived the anelastic pressure equation as

$$\frac{\partial^2 \pi'}{\partial x^2} + \frac{1}{\bar{\rho} c_{pd} \bar{\theta}_v} \frac{\partial}{\partial z} \bar{\rho} c_{pd} \bar{\theta}_v \frac{\partial \pi'}{\partial z} = -\frac{1}{\bar{\rho} c_{pd} \bar{\theta}_v} \left[\frac{\partial \bar{\rho} ADV(u)}{\partial x} + \frac{\partial \bar{\rho} ADV(w)}{\partial z} \right] + \frac{1}{\bar{\rho} c_{pd} \bar{\theta}_v} \frac{\partial \bar{\rho} B}{\partial z}. \quad (15.2)$$

Again, ADV also includes all of the terms in the u and w equations (such as diffusion, Coriolis and friction) apart from pressure gradients and buoyancy. The elliptic pressure equation can be solved either iteratively or using a direct method. Recall that the simplified equation resulted in a sparse, tridiagonal matrix (6.11). The above equation is more complex, but still yields a matrix that is very structured and sparse; it becomes "block-tridiagonal". We will employ a direct solver that exploits its ordered, sparse nature.

15.3.1 Subroutine `BLKTRI`

In two dimensions, we can use subroutine `BLKTRI`, from the NCAR scientific software library called `FISHPAK`. `BLKTRI` is designed to solve equations of the form

```
am(i)*pi(i-1,k) + an(k)*pi(i,k-1) + (bm(i)+bn(k))*pi(i,k) +
cm(i)*pi(i+1,k) + cn(k)*pi(i,k+1)  = f(i,k)
```

The coefficient arrays `am, an, bm, bn, cm, cn` are one-dimensional, representing either model width or depth, and essentially hold the non-zero values along the diagonal and off-diagonals of the block-tridiagonal matrix. They are dimensioned either `nx-2` or `nz-2` because the solver will ignore our domain's fake points. The 2D array `f(i,k)` holds the discretized version of the right hand side if (15.2).

We start by discretizing the LHS of (15.2), one term at a time. The second derivative with respect to x is

$$\frac{\pi'^n_{i+1,k} - 2\pi'^n_{i,k} + \pi'^n_{i-1,k}}{\Delta x^2},$$

By inspection, it is clear that the coefficients being applied to $\pi'^n_{i-1,k}$ and $\pi'^n_{i+1,k}$ (i.e., `am` and `cm`) are both $1/\Delta x^2$, and that the coefficient for `bm` is $-2/\Delta x^2$ .This means that, nominally, we are making these associations:

```
rdx2 = 1./(dx*dx)
do i=2,nx-1
 am(i-1) = rdx2
 cm(i-1) = rdx2
 bm(i-1) = -am(i-1)-cm(i-1)
enddo
```

I wrote "nominally" because we will subsequently consider if our boundary conditions will need to alter these assignments. At present, there is no need to have the values of `am, bm, cm` vary along the horizontal grid. That will change if, say, the horizontal grid stretching is implemented, which would make `dx` a function of x. The assignments are being made to point `i-1` because `BLKTRI` will be ignoring our fake row of points, shifting everything one point to the left.

The vertical derivative on the LHS of (15.2) is more complicated and will vary in the vertical direction even before boundary conditions are considered. Our discretization is

$$\frac{1}{\bar\rho_{u,k} c_{pd} \theta_{v,k}} c_{pd} \left(\bar\rho_{w,k+1} \frac{\left(\bar\theta_{v,k+1} + \bar\theta_{v,k}\right)}{2} \frac{\left[\pi'^n_{i,k+1} - \pi'^n_{i,k}\right]}{\Delta z} - \bar\rho_{w,k} \frac{\left(\bar\theta_{v,k} + \bar\theta_{v,k-1}\right)}{2} \frac{\left[\pi'^n_{i,k} - \pi'^n_{i,k-1}\right]}{\Delta z} \right) / \Delta z.$$

Thus, we are making these nominal assignments:

```
rdz2 = 1./(dz*dz)
do k=2,nz-1
 coef = 1./(rhou(k)*cpd*tbv(k))
 an(k-1) = coef*(rhow( k )*0.5*(tbv(k+1)+tbv(k)))*rdz2
 cn(k-1) = coef*(rhow(k+1)*0.5*(tbv(k)+tbv(k-1)))*rdz2
 bn(k-1) = -an(k-1)-cn(k-1)
enddo
```

We are shifting one point downward owing to the fake level at $k = 1$. So far, we have been presuming that all model levels are separated by `dz`. In practice, vertically stretched grids are often employed, meaning the grid interval between levels `k-1` and `k` and between `k` and `k+1` may not be the same. A stretched grid could be accommodated by defining `dz` and `rdz` as one-dimensional functions of height. This would require rewriting the code above.

Horizontal boundaries

At this point, we need to be concerned with how the boundaries are handled. Recall from the simple example in Chapter 6 that all known values are collected on the right hand side of the equation, i.e., into `f(i,k)`. If the horizontal boundaries are periodic, no changes to `am, bm, cm` are needed, because none of the values of π' are known before all of them are determined. If we open up the lateral boundaries, however, we will apply a zero-gradient BC to the pressure perturbation. This means

$$\frac{\pi'^n_{2,k} - \pi'^n_{1,k}}{\Delta x} = 0,$$

at `i = 1`, and at `i = nx-1`,

$$\frac{\pi'^n_{nx,k} - \pi'^n_{nx-1,k}}{\Delta x} = 0.$$

In other words, `am = rdx2` for all grid points, except when `i = 2`, because the zero-gradient BC forces $\pi'^n_{1,k}$ and $\pi'^n_{2,k}$ to be equal. Similarly, `cm = rdx2` for all grid points except `i = nx-1`, where it vanishes, and `bm` is `-2*rdx2` for all points except at both ends, where it is `-1.*rdx2`. This can be accommodated by appending the following code after the do loop in which `am, bm, cm` are defined (and the `iper` flag having been set):

```
if(iper.ne.1)then  ! lateral BCs are not periodic
 am(1)=0.
 bm(1)=-cm(1)
 cm(nx-2)=0.
 bm(nx-2)=-am(nx-2)
endif
```

Upper and lower boundaries

Our model is confined between rigid plates. The absence of flow across those plates implies that w, $\frac{\partial w}{\partial t}$ and $\frac{dw}{dt}$ are all zero there. Thus, our vertical equation of motion becomes

$$-c_{pd}\bar{\theta}_v \frac{\partial \pi'}{\partial z} = -g\frac{\theta'}{\bar{\theta}}.$$

Rather than forcing the gradient to be zero, we are demanding that the vertical pressure acceleration and the buoyancy cancel each other. It appears more complicated, but it causes essentially similar revisions to be made to the coefficient vectors:

```
 an(1)=0.
 bn(1)=-cn(1)
 cn(nz-2)=0.
 bn(nz-2)=-an(nz-2)
```

Initializing BLKTRI

After constructing the coefficient vectors, the solver is initialized by setting `iflg = 0` and calling the subroutine:

```
      iflg = 0
      np = 1
      mp = 1
      if(iper.eq.1) mp = 0  ! periodic horizontal BCs
      call blktri(iflg,np,nz-2,an,bn,cn,mp,nx-2,am,bm,cm,nx-2,rhs,
     1 ier,wa)
      print *, ier,wa(1)
      if(ier.ne.0) stop 'blktri_problem'
```

In the above, `mp, np` are 1 unless the horizontal and vertical directions are periodic. The array `rhs(nx-2,nz-2)` is a proxy for the RHS, and is ignored for the initialization. Just provide the routine with a 2D array of the required dimensions. `wa` is a vector provided for internal calculations. There is a complex formula for computing the required length but, in practice, just give it a large dimension like 8000. The length actually required is computed and stored in `wa(1)`, which can be inspected. The variable `ier` holds the error code. Unless it is zero, there is a problem.

Computing the right hand side of (15.2)

The code example below presumes that periodicity in x is being assumed, and will have to be modified for open boundaries. First, we construct `advu`, representing the RHS of the u equation other than the pressure gradient term, and enforce the BCs:

```
c compute ADV(u)
      do k=2,nz-1
       do i=2,nx-1
         advu(i,k)=-.25*((u(i+1,k)+u(i,k))**2
     1                  -(u(i-1,k)+u(i,k))**2)/dx
     2             -.25*(rhow(k+1)*(w(i,k+1)+w(i-1,k+1))
     2                             *(u(i,k+1)+u(i,k))
     3                  -rhow( k )*(w(i,k  )+w(i-1,k  ))
     3                             *(u(i,k-1)+u(i,k)))/
     4                (dz*rhou(k))
        enddo
       enddo

c BCs for ADV(u)
c zero gradient top and bottom
```

```
      do i=2,nx-1
       advu(i,1)=advu(i,2)
       advu(i,nz)=advu(i,nz-1)
      enddo
c now k=1,nz has been done
c periodic lateral boundaries
      do k=1,nz
       advu(1,k)=advu(nx-1,k)
       advu(nx,k)=advu(2,k)
      enddo
```

Now compute the buoyancy term `buoy`, and `advw`, representing the RHS of the w equation other than the pressure gradient and buoyancy, and impose our boundary conditions:

```
       do k=3,nz-1
        do i=2,nx-1
         buoy(i,k)=g*0.5*(th(i,k)/tb(k) + th(i,k-1)/tb(k-1))
         advw(i,k)=-.25*((u(i+1,k)+u(i+1,k-1))
     1                  *(w(i+1,k)+w(i,k))
     2                  -(u(i  ,k)+u(i  ,k-1))
     2                  *(w(i-1,k)+w(i,k)))/dx
     3              -.25*(rhou( k )*(w(i,k+1)+w(i,k))**2
     4                  -rhou(k-1)*(w(i,k-1)+w(i,k))**2)/(rhow(k)*dz)
        enddo
       enddo

c BCs for ADV(w)
c w is zero at k=2 and nz. also zero at k=1, which should not be referenced.
      do i=2,nx-1
       advw(i,2)=0.
       advw(i,1)=0.
       advw(i,nz)=0.
      enddo
c now k=1,nz has been done
c periodic lateral boundaries
      do k=1,nz
       advw(1,k)=advw(nx-1,k)
       advw(nx,k)=advw(2,k)
      enddo
```

We are not bothering to compute `advw` at `k = 3` and `nz` because we are making sure those vanish.

Pressure decomposition and recomposition

Now we differentiate `advu` and `advw`, as required by (15.2). This is made easier because we already took care of the BCs.

```
c set up for pressure decomposition
      do k=2,nz-1
       a1=1./(rhou(k)*dz)
       a2=1./(cp*tb(k))
       do i=2,nx-1
        byctrm(i-1,k-1)=(buoy(i,k+1)*rhow(k+1)-buoy(i,k)*rhow(k))*a1*a2
        dyntrm(i-1,k-1)=((advu(i+1,k)-advu(i,k))/dx
     1                  +(advw(i,k+1)*rhow(k+1)-advw(i,k)*rhow(k))*a1)*a2
        alltrm(i-1,k-1)=dyntrm(i-1,k-1)+byctrm(i-1,k-1)
       enddo
      enddo
```

In Chapter 6, it was noted that a nice property of the elliptic equation (15.2) is that we can separate the forcing (RHS) into parts, and solve the equation for the part of the pressure field responding to that forcing. In the above, we have written the RHS anticipating we will want to separate buoyancy from dynamic pressure. Thus, `bycterm` is a 2D array storing the vertical gradient of the buoyancy term in (15.2). Solving the elliptic equation for it alone yields buoyancy pressure. Similarly, `dynterm` represents the derivatives of `advu` and `advw` and yields dynamic pressure[2]. We do not actually need to combine `dynterm` and `bycterm` as `alltrm` and solve the equation a third time, as we could just combine the results from the first two calls, but the provided code does this anyway, providing a sanity check.

```
      iflg = 1 ! BLKTRI has already been initialized
      np = 1   ! vertical dimension is not periodic
      mp = 0   ! horizontal domain is periodic
      call blktri(iflg,np,nz-2,an,bn,cn,mp,nx-2,am,bm,cm,nx-2,byctrm,ier,wa)
      call blktri(iflg,np,nz-2,an,bn,cn,mp,nx-2,am,bm,cm,nx-2,dyntrm,ier,wa)
      call blktri(iflg,np,nz-2,an,bn,cn,mp,nx-2,am,bm,cm,nx-2,alltrm,ier,wa)

      do k=2,nz-1
       do i=2,nx-1
        pbyc(i,k)=byctrm(i-1,k-1)-byctrm(1,1) ! cosmetic adjustment
        pdyn(i,k)=dyntrm(i-1,k-1)-dyntrm(1,1) ! cosmetic adjustment
        ptot(i,k)=alltrm(i-1,k-1)-alltrm(1,1) ! cosmetic adjustment
       enddo
      enddo
```

We should augment the code above to check the value of `ier` for each call; this was removed to enhance readability. On return from `BLKTRI`, the forcing arrays contain the pressure components, having been overwritten in the routine; these need to be shifted back to account for our fake points. Because we are solving the equation with periodic and/or Neumann BCs, however, there is no way of obtaining *unique* values of the pressure perturbation. This does

[2]As shown by Rotunno and Klemp (1982), the dynamic pressure can be further decomposed into its linear and nonlinear parts; this would require some rewriting of the above code.

not matter very much because we formulated our equations to require only pressure gradients. (This was discussed in Chapter 3 as one of the advantages of π' over dimensional p'.)

This leaves us, however, with pressure values that can *float*. The provided code addresses this annoyance by subtracting the corner value from each pressure component. One could select a different corner point, depending on the application, or do something more sophisticated, such as adjusting the mean domain value to a specified constant, such as zero. Owing to the way our model is formulated, however, this is completely cosmetic[3].

15.3.2 An example application

Figure 15.1 shows fields at 900 sec from Model Task # 5's thermal simulation. Note the pressure perturbation field consists of high pressure above the thermal, low pressure on the thermal's flanks, and sound waves (above $x = \pm 12$ km) with maximum amplitude at the surface. As discussed in the previous chapter, those waves resulted from our providing a hydrostatically balanced initial pressure perturbation field. In Fig. 15.2 the anelastic version of the pressure field (panel b) and its decomposition (panels c, d) into dynamic (p'_d) and buoyancy pressure (p'_b) components are shown for comparison with the compressible model's pressure prediction (repeated from the previous figure as panel a). It is immediately seen that the prognosed and diagnosed p' fields are not the same, and the most striking difference is the absence of the sound waves in the anelastic field.

Even though the anelastic and compressible pressure fields are not identical, we can still utilize the former's decomposition into dynamic and buoyancy pressures to understand what is going on. The dynamic pressure field consists of low pressure on the flanks and high pressure above and below the thermal (the below-thermal high being too small for the contour interval selected). As shown in Rotunno and Klemp (1982), high dynamic pressure is produced by both convergence *and* divergence[4], while low p'_d results from rotation. The buoyancy pressure pattern consists of high perturbation pressure above the thermal with low pressure beneath. This can be understood from a simplified, 1D version of (15.2):

$$\frac{\partial^2 \pi'}{\partial z^2} = \frac{\partial B}{\partial z}.$$

For wave-like features (describable with sines and cosines), the second derivative of a field is proportional to the field itself, with the sign reversed (e.g., $d^2 \sin x / dx^2 \propto -\sin x$). As

[3]With the version of BLKTRI provided, your Fortran compiler may issue a warning about the call to subroutine PPADD. The code appears to work, so this warning can be ignored.

[4]This requires the assumption that the pressure field is sinusoidal, which is not always true.

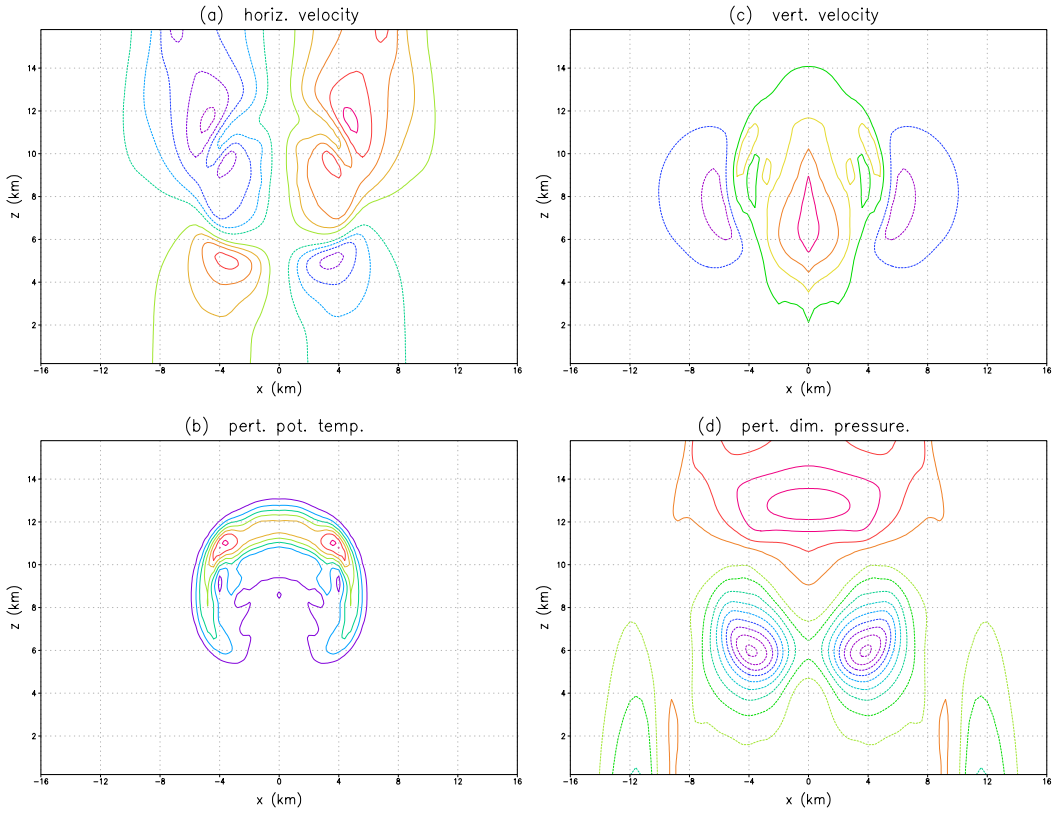

Figure 15.1: Fields at $t = 900$ sec for the Model Task # 5 thermal simulation: (a) horizontal velocity (3 m s^{-1} contours); (b) vertical velocity (5 m s^{-1} contours); (c) perturbation potential temperature (0.5 K contours); and (d) perturbation dimensional pressure (0.2 mb contours). Zero contours suppressed.

a consequence, we see that high buoyancy pressure would be expected where dB/dz is negative – above the thermal – and low pressure where buoyancy increases with height – below the thermal. The effect of buoyancy forcing on the vertical velocity could be examined by combining the vertical buoyancy pressure gradient with the buoyancy field itself (cf., Rotunno and Klemp 1982).

15.4 Adding a mean horizontal wind or shear

We wrote our potential temperature equation to prognose θ' (as `thp, th, thm`), perturbations from a horizontally homogeneous base state described by $\bar{\theta}$. Our water vapor prediction equation will also be written in terms of perturbations. This is traditional, likely because

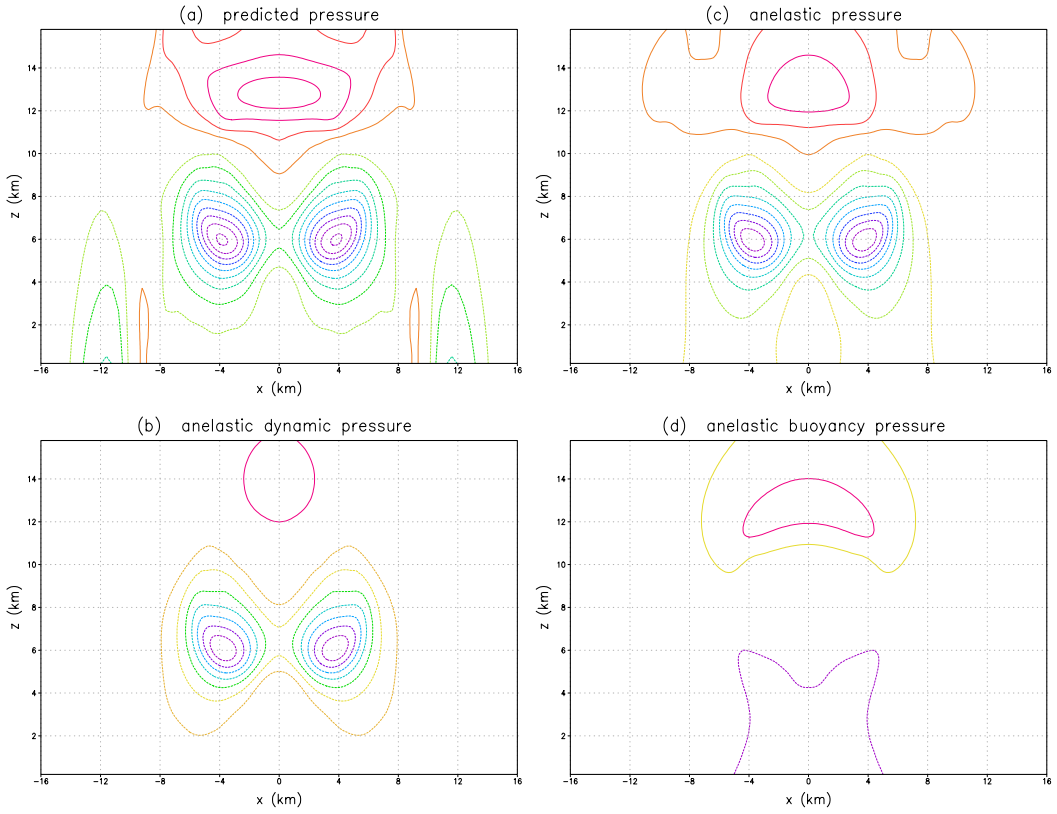

Figure 15.2: Dimensional pressure component fields at $t = 900$ sec for the Model Task # 5 thermal simulation (0.2 mb contours): (a) pressure predicted by the compressible model (same as Fig. 15.1d); (b) anelastic pressure field; (c) anelastic dynamic pressure component; and (d) anelastic buoyancy pressure component. Zero contours suppressed.

temperature and vapor perturbations are explicitly needed in the w equation's buoyancy term. As a consequence, when we code the vertical advection, we need a separate term to handle the advection of the mean state. As artificial diffusion designed to control nonlinear instability (see below) should be applied only to perturbations, however, no special handling of the vertical diffusion term will be needed.

Adding a mean (possibly sheared) horizontal wind to the model involves placing nonzero entries in `ub`. Although not required, it is traditional to handle u as the full field, representing mean plus perturbations. Under this approach, which is presumed in the examples that follow, `u` and `um` are set equal to `ub` everywhere during the initialization. However, we will need to subtract `ub` from `u` in the vertical diffusion term, owing to the possibility of mean state vertical shear.

15.5 Diffusion and time smoothing

We have seen repeatedly that models handle the shortest wavelengths worst. They are subjected to the largest phase errors and grow the fastest when the model goes unstable (Chapter 5), and aliasing of these waves was implicated in nonlinear computational instability (Chapter 7). Adding a diffusion term to the model is straightforward, as long as the term is applied at time level $n-1$ when the leapfrog scheme is used. If the purpose of the diffusion is solely to restrain computational instability, and not to represent a physically realistic mixing process, then it is standard to apply the smoothing only to perturbations from the basic state. This is not an issue with prognostic variables `th, pi` and `qv`, as they already represent perturbation quantities, nor with `w` for which the mean state is zero. However, if `u` was coded to represent the full horizontal velocity field, diffiusion should only be applied to the perturbations, i.e., `u(i,k)-ub(k)`. This is illustrated in the sample code below (derived from Model Task #5).

```
      dtx=d2t/dx
      dtz=d2t/dz
      rdx2=1./(dx*dx)
      rdz2=1./(dz*dz)
c do loop for U
c loop over unique points: i=2,nx-1 and k=2,nz-1
c model predicting full u but applying diffusion only to u'
       do k=2,nz-1
        do i=2,nx-1
         up(i,k)=um(i,k)-.25*dtx*((u(i+1,k)+u(i,k))**2
     1                           -(u(i-1,k)+u(i,k))**2)
```

```
     2                  -.25*dtz*(rhow(k+1)*(w(i,k+1)+w(i-1,k+1))
     2                                   *(u(i,k+1)+u(i  ,k))
     3                           -rhow( k )*(w(i,k  )+w(i-1,k  ))
     3                                   *(u(i,k  )+u(i  ,k-1)))/rhou(k)
     4                  -dtx*cp*tbv(k)*(pi(i,k)-pi(i-1,k))
     5    +dkx*d2t*rdx2*(um(i+1,k)-2.*um(i,k)+um(i-1,k))
     6    +dkz*d2t*rdz2*(um(i,k+1)-2.*um(i,k)+um(i,k-1)
     6                  -ub(  k+1)+2.*ub( k )-ub(  k-1))
      enddo
     enddo
```

This code sample specifies constant mixing coefficients, `dkx` and `dkz`, applied to the horizontal and vertical directions, respectively. The values can be selected based on their amplification factors for $2\Delta x$ and $2\Delta z$ waves (see Chapter 7). Again, if the mixing is purely for *numerical reasons*, these should be the smallest possible values that accomplish the required task. Higher order diffusers might also be considered, especially for the horizontal direction, to attain greater scale selectivity.

The above was explicit spatial smoothing. Recall that time smoothing has been used to control the leapfrog scheme's computational mode. Chapter 5 discussed two filters, the Robert-Asselin and a recent modification of that scheme. The code below suggests how this could be handled in the time stepping loop. Before the displayed code segment, the forecast for `up` has been completed, and BCs applied, but we have not yet swapped values from `u` to `um` and from `up` to `u`. The still-current time value `u` is being smoothed based on the forecast, original present value, and immediate past value, where `ts` is the smoothing coefficient. The operation is applied to all grid points, real and fake, because BCs have already been applied.

```
c time filter before setting for new time step
      do i=1,nx
       do k=1,nz
        u (i,k)=u (i,k)+ts*(up (i,k)-2.*u (i,k)+um (i,k))
        w (i,k)=w (i,k)+ts*(wp (i,k)-2.*w (i,k)+wm (i,k))
        th(i,k)=th(i,k)+ts*(thp(i,k)-2.*th(i,k)+thm(i,k))
        pi(i,k)=pi(i,k)+ts*(pip(i,k)-2.*pi(i,k)+pim(i,k))
       enddo
      enddo
```

15.6 Adding a heat source

A heat source can be implemented by supplying a forcing term for the potential temperature perturbation equation. The source can be held steady with time, or made to oscillate.

As examples, Nicholls et al. (1991) and Mapes (1993) considered the response of a stable environment to maintained heat sources representing diabatic forcing generated by convective storms. They gave their heat sources vertical profiles that resembled what is observed in nature and simulated by cloud models, and examined how the gravity waves that were excited by activating the heating modified the environment surrounding the source.

Specifically, they considered heating profiles that were describable as sine (or half-sine) functions in the vertical. As with the thermal used in Model Tasks #3 and 5, the source is confined to a limited area. The code below places into `hsrc` a source with amplitude `ampl` and horizontal radius `radx` at grid point `icnt`. A single half-sine of depth `H` comparable to the troposphere depth can represent the gravest mode of deep convective heating. A second mode, described by a full sine wave but with half the depth of the gravest component and half its amplitude is optionally included, if `xmode2 = 1`. The combination of these two waves can produce the "top heavy" heating profile considered in both Nicholls et al. (1991) and Mapes (1993), and made non-negative in the latter.

```
 do k=2,nz-1
  argz=((dz*(float(k)-1.5)-znaught)/radz)**2
  vert=(trigpi/H)*(dz*(float(k)-1.5))
  do i=2,nx-1
   argx=(dx*(i-icnt)/radx)**2
   rad=sqrt(argz+argx)
   if(rad.le.1.)then
    hsrc(i,k)=0.5*ampl*(cos(trigpi*rad)+1)
1                        *(sin(vert)-0.5*xmode2*sin(2*vert))
    hsrc(i,k)=amax1(hsrc(i,k),0.)
   endif
  enddo
 enddo
```

Add this code to the right-hand sides of the θ' equation. As usual, `trigpi` is trigonometric π, which can be obtained as `4.0*atan(1.0)`. Don't forget to multiply by $2\Delta t$.

In a stable atmosphere, the deeper heating mode should provoke a gravity wave of depth `2*radz`, propagating away fromt he source at phase speed

$$c = \pm \frac{NH}{\pi},$$

where N is the Brunt-Väisälä frequency and the denominator is trigonometric π. The second mode will move at half that speed, and thus travel through the gravest mode's wake. The source could also be made unsteady, characteristic of multicellular storms, and be made to tilt at some angle to the vertical, which is representative of mature squall lines. More

complex heat sources, including explicit diabatic cooling to represent evaporation, could also be crafted to mimic squall lines, as was considered by Pandya and Durran (1996).

Bretherton (1988) found that the atmosphere's response to a maintained heat source is complicated. While the vertical motion forced reaches a steady state, the horizontal velocity perturbations in the vicinity of the source grow logarithmically with time. Adding the Coriolis force to the model suppresses this logarithmic growth, however. (This can be done in 2D, and requires adding a term to the u equation as well as a v equation.) Can you demonstrate this occurs?

15.7 Adding a momentum source

A momentum source can be added to the equations to simulate steady or unsteady obstacles or forcings by defining a streamfunction ψ that enters into the u and w equations in this manner

$$\frac{\partial u}{\partial t} \leftarrow -\frac{1}{\bar{\rho}}\frac{\partial \psi}{\partial z}$$

$$\frac{\partial w}{\partial t} \leftarrow \frac{1}{\bar{\rho}}\frac{\partial \psi}{\partial x}.$$

The streamfunction can be confined in areal extent (much like our initial thermal), and made either steady or unsteady with time. As an example, Fovell et al. (1992) used an unsteady momentum source placed in the midtroposphere to excite stratospheric gravity waves. Changing the width of the source and frequency of the oscillation affects the wavelength and tilt of the resulting waves, as does adding in various vertically sheared flows.

Fovell (2005) used a series of steady momentum sources placed in the boundary layer to mimic horizontal convective rolls, which consist of a series of updrafts and downdrafts. By altering the flow passing over the drafts, vertically propagating gravity waves having various tilting angles can be generated. By shifting the streamfunction closer to the surface, and playing with the magnitude of the source, a "mountain" consisting of a series of hills or a single peak can be created. There will still be some flow through the virtual mountain – it is not actually impermeable, especially if the wind is strong – but in many ways this can provide a simple and straightforward way of including topographic-like features in the numerical model.

The code below implements Fovell (2005) forced rolls, being more straightforward than the method used in Fovell et al. (1992). `psik` and `psim` are the horizontal and vertical

wavenumbers. The source is centered at height `znaught` (set to 1 km in Fovell 2005) and horizontal grid point `icnt`. `ampl` is the source amplitude, taken to be 0.27 in that paper, and the horizontal and vertical wavelengths were 7000 and 3000 m, respectively. This code sets up the streamfunction, which is placed in 2D array `strfcn`.

```
argu1=psik*dx*(float(i-icnt)+.5)
argu2=psim*(dz*(k-1)-znaught)
argu=argu1+argu2
if(argu2.gt.-0.5*trigpi.and.argu2.lt.0.5*trigpi)then
 strfcn(i,k)=ampl*sin(argu1)*cos(argu2)
else
 strfcn(i,k)=0.
endif
```

Note that the streamfunction is being defined at a point that is $0.5\Delta z$ above and below u and $0.5\Delta x$ to the left and right of w, which makes sense given how it will be used to create forcing for these velocity components. Add this code to the right-hand sides of the u and w equations. Don't forget to multiply by $2\Delta t$.

15.8 Add a surface heat flux to the lower boundary

By adding the heat flux to only part of the domain, you can simulate the sea-breeze circulation (or an urban heat island). The heated portion of the domain represents the land surface during the day, while the unheated part represents the sea. Heated air over the land rises, and the relatively cooler air originating over the "sea" pushes inland. Adding a mean u flow directed "offshore" to the model restrains the inland progress of the sea-breeze front, but also can make it much stronger and sharper. An "onshore" flow speeds the front's propagation but makes it much more diffuse. Can you show how the sea-breeze frontal lifting varies with the intensity and direction of the mean wind?

By adding the heat flux to the entire domain, combined with an initial perturbation, you can simulate the development of Rayleigh convection, represented by roll-like circulations in 2D. If you implement a uniform heat flux, however, you should not provoke any motions, especially if your domain is periodic. (Why?) As a consequence, you will need to superimpose some variation on the surface flux. Will adding random noise to the heat flux create coherent, non-random circulations?

The code below implements a source term for the θ' equation, but only to the lowest real temperature level `k = 2`, and only if the potential temperature there ($\bar{\theta} + \theta'$) exceeds a

specified, but fixed ground temperature, `tground`. If the air is cooler than the ground surface, there is an upward heat flux. It is not uncommon either to augment the near-surface wind by a "convective velocity" value to mimic turbulent heat transfer and/or to employ a minimum wind speed in the flux equation so the source does not become too small when winds are calm. The code sample provided implements both ideas. Note the temperature difference `tdif` is computed at time $n-1$.

```
! tb is base state potential temperature (K)
! tground is a specified, fixed ground surface temperature
! cdh is a nondimensional heat transfer coefficient (usually something like 7E-3)
! addlfx implements random variation around a mean of 1.0
! apply ONLY at lowest real level
         if(ishflux.eq.1.and.k.eq.2)then
          tdif=amax1(tground(i)-(thm(i,k)+tb(k)),0.)
          avgu=0.5*abs(u(i+1,k)+u(i,k))
          avgu=amax1(avgu,2.0)  ! enforce a minimum wind speed
          wnetc=2.*sqrt(tdif)      ! "convective velocity" adjustment
          vel=sqrt(avgu*avgu+wnetc*wnetc)
          thp(i,k)=thp(i,k)+d2t*cdh*vel*addflx(i)*tdif/dz
         endif
```

15.9 Add a near-surface heat sink to the model

A simple way to mimic a thunderstorm cold pool or cold front is to put a heat sink into the model, an area within which the air will be either constantly cooled, or "nudged" to a selected, fixed negative buoyancy. (This is similar to the previous idea but implemented above the ground instead of at it.) The chilled air produced in the sink will commence spreading along the model surface.

Can you relate the pool's propagation speed to the density excess within the cold air? As it spreads, the cold air "underruns" the less dense air outside the pool, forcing it to rise. Can you relate the strength of this forced lifting to factors such as the pool's propagation speed and the stability of the environment? (The latter would obviously require making the base state non-isentropic.) What happens if you also impose a mean horizontal wind? What happens if that mean wind is also sheared?

15.10 Implementing surface drag (friction)

Surface drag is often implemented using a bulk aerodynamic formula involving nondimensional momentum drag coefficient C_{DM}:

$$\bar{\rho} C_{DM} |\vec{V}| u,$$

applied to the horizontal wind at its lowest real level above the ground. Values of C_{DM} such as 0.003 and 0.001 have been used over land and water surfaces, respectively. $|\vec{V}|$ is the wind speed at the first horizontal wind level, including the vertical velocity interpolated to there. A minimum velocity may also be included in $|\vec{V}|$, so the drag does not vanish when the winds are nearly calm. You need to think about whether you want to apply drag to the full u field, or just u'. Applying the drag to the entire wind field will cause the flow to spin down near the ground, which may be undesirable. After all, $\bar{u}$ is considered to be a time-independent base state so, in a sense, it is already supposed to incorporate the mean effect of friction near the surface.

15.11 Adding moisture and microphysics to the model

Simulate a simple, isolated cumulus cloud. What kind of life cycle does it undergo? How much rain falls out of the cloud? How fast does the cloud top rise and how long does it survive? Can you make a new convective cell grow after the old one dies out?

Adding moisture and even "warm rain" microphysics to the model is quite complicated, and the following step-by-step strategy is suggested. A lot of things have to be added, more than you might guess, and a lot of adjustments and refinements have to be made. However, you don't have to go all the way to a full-blown cloud model with rain and perhaps even ice species to get something worthy of using in an experiment. I am presuming you are starting with Model Task #5, the dry thermal in a neutral environment.

1. *Make the environment stable, although still dry.* A good choice is reverting to your MT1/MT2 sounding. The thermal will not rise as far, of course, and now you will excite gravity waves.

2. *Add spatial diffusion to the prognostic equations,* to suppress nonlinear instability. Terms like $K_x \frac{\partial^2 u}{\partial x^2}$ and $K_z \frac{\partial^2 u'}{\partial z^2}$.

- K_x and K_z are mixing coefficients, units squared meters per second. Optimal values depend on the problem and the resolution. For mesoscale applications, try 100 and 10 for K_x and K_z as a first guess.
- This diffusion is artificial in nature, and should be applied only to *perturbations from the mean state* (e.g., $u' = u - \bar{u}$). This will only matter after $\bar{u}$ varies with height, however.
- How does this influence your thermal? Adjust and tune the coefficients.

3. *Add temporal diffusion (Robert-Asselin filter) to the prognostic equations,* to control the computational mode (Sec. 5.2.3).

   - A good place to apply this is after you have finished computing advection, spatial diffusion and have applied the boundary conditions, but before you set for the next time step.
   - Try to keep the filter coefficient, ϵ, small (.01 or less).
   - Also add domain statistics that might help reveal computational mode activity: Domain maximum vertical and horizontal velocity, domain total kinetic energy, domain maximum and minimum temperature and pressure perturbations, etc..

4. *Add water vapor prognostic equation to the model,* treating it as a passive tracer at first.

   - Craft your equation in terms of q_v' and mimic the θ' equation. You will need a term for vertical advection of $\bar{q}_v$, in advective form. Add spatial diffusion, applied to q_v' (you will need it).
   - Re-enable $\bar{q}_v$ from MT1/MT2. Consider also adding q_v' to your initial thermal, by adjusting the RH in the thermal to 100%, say. (Otherwise the thermal represents an area with lower RH.)
   - Advect your water vapor field. Note negative *total* mixing ratios (i.e., $\bar{q}_v + q_v'$) will occur, just as negative values were produced in the wake of the cone in MT4.
   - Adjust the negative *total* mixing ratios in some fashion. Do this after advection and spatial diffusion and before applying the boundary conditions. Simply zeroing them will add a spurious moisture source to the model. Using better adjustments is fairly simple and straightforward. Implementing positive-definite advection schemes will be more challenging.
   - Do temporal diffusion and apply boundary conditions.

5. *Make water vapor active,* by implementing a saturation adjustment, as in MT2, but in a "cloudless cloud model" or pseudoadiabatic model.

   - Add q_v' to the buoyancy term of the w equation. Buoyancy is $g\left[\frac{\theta'}{\theta} + 0.61q_v'\right]$. See Sec. 8.2.2.
   - Following advection, spatial diffusion and boundary condition treatments, but *before* time filtering, check your time $n+1$ fields for supersaturation. Apply saturation adjustment where RH $> 100\%$ to adjust q_v' at supersaturated grid points back to 100% RH. This is a sink term for vapor, and a source term for potential temperature. (You are not doing anything with this condensed vapor; it is simply ignored, as if its fallspeed were infinite.)
   - The adjustment is instantaneous, and so is NOT multiplied by the time step. The supersaturation did emerge over the last time interval, but it is adjusted instantaneously.
   - When saturation adjustment is finished, *reapply boundary conditions*, do the time filtering, and then set for the next time step.
   - Diabatic heating owing to microphysics tends to force the model at small time and space scales. Watch out for linear and nonlinear instability, and excitement of the computational mode. Adjust time step, K_x, K_z and ϵ as needed. Watch out for strange goings-on in your model stratosphere, where the vertical static stability is high.

6. *Add cloud water, q_c, to the model,* making a rainless cloud model.

   - Advect cloud water. Deal with negative mixing ratios. They should not exist.
   - Add q_c to the buoyancy term. Now buoyancy is $g\left[\frac{\theta'}{\theta} + 0.61q_v' - q_c\right]$. Cloud droplets represent a drag on the air, and so contribute to negative buoyancy.
   - Now the saturation adjustment has two roles: to create new cloud water mass at supersaturated grid points, and to remove cloud water if present where RH $<$ 100%. So, look not only for grid points where RH $> 100\%$ but also for subsaturated points at which $q_c > 0$.
   - In MT2, the mixing ratio adjustment was labeled C. Now positive C is a sink for vapor, a source for θ' and a source for q_c. Negative C is a source for vapor, and a sink for θ' and q_c. $|C|$ can exceed available cloud water, so do not let that happen. All of these adjustments are instantaneous.

- As you implement cloud water, reconsider your time steps and diffusion coefficients, and watch out for $2\Delta t$ noise.

7. *Add rain water, q_r, to the model,* representing "warm rain" (Kessler-type) microphysics (refer to Chapter 8).

   - Advect rain water and adjust the negative mixing ratios.
   - Unlike cloud droplets, raindrops have a non-negligible fall velocity relative to still air. Add a terminal velocity (V_T) term to vertical advection in the q_r equation, creating $(w - V_T)$ (see Sec. 8.1.2). In practice, V_T depends on drop size (mass), and drop size is (usually) presumed to depend on q_r itself; the more rain mass in a grid volume, the more large drops are presumed to exist. See Sec. 8.1.1 for details. It may suffice here, however, to impose a constant V_T to get something going. A 6 m s^{-1} fall speed is probably a good start.
   - Raindrops also represent a drag source, so buoyancy is now $g\left[\frac{\theta'}{\bar{\theta}} + 0.61q'_v - q_c - q_r\right]$.
   - After advection and spatial diffusion are accomplished, but before the saturation adjustment and time filtering, it is time to perform *microphysics*:
     - Warm rain microphysics consists of "autoconversion" (self-aggregation) of cloud droplets into raindrops, the accretion of cloud droplets by falling raindrops, and the evaporation of raindrops in subsaturated air.
     - Kessler imagined that cloud droplets would evolve into raindrops at some rate, as long as droplets were sufficiently numerous. He implemented this as $k_1(q_c - q_{c0})$, where k_1 is a conversion rate (typically 0.001 s^{-1}) and q_{c0} is a cloud water threshold (often 0.001 kg$_w$ kg$_a^{-1}$). This is typically conceded now as a poor way of handling this process.
     - Once created, rain mass also grows owing to accretion, as the drops collided with more slowly falling (relative to still air) cloud particles. Kessler distilled the geometric "sweep-out" problem into this simple term: $k_2 q_c \left[\bar{\rho} q_r\right]^{7/8}$, with k_2 the accretion rate of 2.2 m$^{7/8}$ kg$_w^{7/8}$ s^{-1} .
     - Raindrops also evaporate in subsaturated air. This process is not instantaneous and the equation is complex (see Sec. 8.1.4).
   - Details regarding the implementation of microphysics:
     - Microphysics represents processes that occurred over the previous time interval. As they reside on the right hand side, these terms are multplied by `d2t` when the leapfrog scheme is employed.

- There is not uniform agreement on which time level to base microphysical rates on. As an example, the accretion of cloud by rain depends on both q_c and q_r. Since the rates are applied to the interval between times $n-1$ and $n+1$, it is reasonable to assume that you use the cloud and rain values at time level n. Some cloud models (e.g., ARPS) base microphysics rates like autoconversion and accretion on time $n+1$ values, at least when Kessler microphysics is employed. This means that doing autoconversion before accretion changes the accretion rate, as the latter is based on q_c^{n+1} and q_r^{n+1}, which were just altered. The sample code that follows adopts this approach, but I would be cautious about this.
- The sample code below does frequent checks for negative mixing ratios, more than necessary. If you took care to adjust negative values that were caused by advection prior to doing computing microphysics, and take care not to transfer more mass from a particular species than actually exists, you should not need to be so vigilant.
- Some models compute raindrop sedimentation (fallspeed) separate from advection, with the result that an updraft may push drops some vertical distance, only to be shifted downward again when the fallspeed is finally applied. It seems to make more sense to combine the fallspeed with the vertical velocity (i.e., $w - \hat{V_T}$), as was written in Sec. 8.2, and do sedimentation as part of advection.

- As always, reconsider your time steps and diffusion coefficients, and watch out for $2\Delta t$ noise. Microphysics can and will add noise to the model, and very likely excite the leapfrog computational mode.

To reiterate, here is the order in which the model should operate in the time-stepping loop.

(a) Compute advection and spatial diffusion for all real points, for all prognostic variables, creating forecasts at time $n+1$. This includes rainwater sedimentation.

(b) Take care of negative mixing ratios in the time $n+1$ quantities. For water vapor, which we handle as q'_v, we are making sure the total vapor mixing ratio, $q'_v + \bar{q}_v$ is non-negative.

(c) Take care of the boundary conditions.

(d) Perform microphysical calculations, updating time $n+1$ quantities. This includes raindrop evaporation. These rates are multiplied by `d2t`.

(e) Perform saturation adjustment, which removes supersaturation (creating q_c) or evaporates cloud droplets present in subsaturated air. These adjustments are instantaneous, and **not** multiplied by `d2t`.

(f) Update the boundary condition again, on fields potentially modified by microphysics and saturation.

(g) Apply the time filter to suppress the computational mode.

(h) Set for the new time step.

Sample code for Kessler-type microphysics. This mimics code from the ARPS model.

```
! ------------------------------------------------------------------------------
! Kessler (warm rain) microphysics
! ------------------------------------------------------------------------------
! compute autoconversion of cloud water to rain, accretion of cloud by rain
!  and evaporation of rain in subsaturated air

! ------------------------------------------------------------------------------
! this logic follows that used in the OU ARPS model
! -- rates are multiplied by d2t because leapfrog scheme is used
! -- rates are based on values of qc, qr, qv at time n+1.  This is arguable, and can
! --   cause results to be potentially dependent on order in which terms are computed
! -- checks are made to ensure mixing ratios qc, qr, and qv+qb do not become negative
! -- negative mixing ratios are zeroed out, creating a spurious source.  Not optimal.
! ------------------------------------------------------------------------------

! things defined prior to computing microphysics

       d2t = dt + dt                                  ! after 1st time step
       xki = cpd/rd                                   ! cpd/rd
       psl = 1000.e2                                  ! reference pressure (Pa)

! -- base state information
       pbar = pb(k)                                   ! mean dimensional pressure (Pa)
       pibar = pib(k)                                 ! mean nondimensional pressure
       thetabar = tb(k)                               ! mean potential temp (K)
       qbar = qb(k)                                   ! mean vapor mixing ratio (kg/kg)
       xlf = 2.5e6                                    ! latent heat of vaporization

! ------------------------------------------------------------------------------
! autoconversion of qc to qr (Sec. 8.1)
! ------------------------------------------------------------------------------

       autort = 0.001                                ! autocon rate, 1/sec
       autotr = 0.001                                ! autocon threshold, kg/kg

       qcplus = max(0.0, qcp(i,k) )                  ! not using negative mixing ratios
       ar = autort *( qcplus - autotr)               ! qc at n+1 used to comp. autocon
       ar = max(0.0, ar)                             ! no autocon is qc < threshold
```

```
        arcrdt = min( ar*d2t, qcplus )             ! mult autocon rate by d2t, and
                                                   ! do not exceed available qc
        qcp(i,k) = qcp(i,k) - arcrdt               ! subtract result from qc
        qrp(i,k) = qrp(i,k) + arcrdt               ! add result to qr

! ------------------------------------------------------------------------------
! accretion of qc by qr (Eq. 8.14)
! ------------------------------------------------------------------------------

        acrt = 2.2                                 ! accretion rate, (m/kg)^(7/8)/s

        qcplus = max(0.0, qcp(i,k) )               ! not using negative mixing ratios
                                                   ! (shouldn't be needed here)
        qrplus = max(0.0, qrp(i,k) )               ! not using negative mixing ratios

        cr = acrt*qcplus*(rhou(k)*qrplus)**0.875   ! accretion rate
        arcrdt = min( cr*d2t, qcplus )             ! mult accretion rate by d2t, and
                                                   ! do not exceed available qc

        qcp(i,k) = qcp(i,k) - arcrdt               ! subtract result from qc
        qrp(i,k) = qrp(i,k) + arcrdt               ! add result to qr

! ------------------------------------------------------------------------------
! evaporation of rainwater (Eq. 8.15)
! ------------------------------------------------------------------------------

        qrplus = max(0.0, qrp(i,k) )               ! not using negative mixing ratios
                                                   ! (shouldn't be needed here)
        qvplus = max(0.0, qvp(i,k)+qbar )          ! not using negative mixing ratios


! -- saturation mixing ratio at time n+1 (see MT2)
        pc=380./(pibar**xki*psl)                   ! coefficient for qvs eqn.
        pth = thp(i,k) + thetabar                  ! full theta, time n+1

        qvs=pc*exp(17.27*(pibar*pth-273.)/(pibar*pth-36.)) ! Tetens' for qvs at n+1

! -- evaporation
        coef = 1.6 + 30.39*( rhou(k)*qrplus )**0.2046  ! ventilation coef.
        deficit = amax1((1.0 - qvplus/qvs(i,j,k)),0.)  ! saturation deficit (RH < 100%)

        er = coef*deficit*((rhou(k)*qrplus)**0.525 ) & ! evaporation rate, kg/kg/s, >0
     &          /( (2.03e4 + 9.584e6/(pbar*qvs) ) &
     &         *rhou(k) )

        erdt = min( qrplus, max(0.0, er*d2t) )         ! do not exceed available qr

        qrp(i,k) = qrp(i,k) - erdt                     ! subtract from qr
        qvp(i,k) = qvp(i,k) + erdt                     ! add to qv

        thp(i,k) = thp(i,k) &                          ! latent cooling
     &        - xlf*erdt/( cpd*pibar )
! ------------------------------------------------------------------------------
```

Part III

Adjoint models

Chapter 16

Theory and construction

In the course of a modeling study, one may wish to gauge the effect the model's initial conditions (ICs) and/or parameter settings (PSs) might have on the model forecast. One may be interested in assessing either how a *single alteration at the initial time* eventually influences *everything at the final time*. Alternatively, one may be more concerned with how a *single aspect of the final forecast was influenced by everything at the initial time*. These complementary but incommensurable strategies represent the simplest foci for the modeling experiment.

For the first focus, the obvious solution is to run the model repeatedly, spanning every possible combination of ICs and PSs. This could well be a formidibly expensive, or even outright impossible, task. Some efficiency might be achieved by replacing the original, nonlinear numerical model with a *tangent linear model* (TLM). As will be seen, the TLM is constructed from the original model using Taylor series truncated to first order, creating a model that forecasts perturbations from the original model solution while constraining them to share the original model's temporal trajectory.

First, a control run is created by the integration of the full nonlinear numerical model. Then the TLM is initialized, not with the control run's initial fields and parameter values, but rather with small variations or perturbations for a subset of those values. Finally, the TLM is integrated to forecast how those perturbations from this control state would evolve. Owing to its truncated Taylor underpinning, it is clear the perturbations must start off and remain small, lest the neglected higher order terms (involving products of these perturbations) become non-negligible.

In the second focus, one wishes to trace some aspect of the control run's final forecast

backward in time in order to identify that aspect's precursors, those specifically being the variables and the locales which have combined to most influence the magnitude and even appearance of the aspect. As an example, say the forecast aspect of interest is the surface central pressure of a cyclone predicted by the control run to reside at a particular location and to have a certain value. That final central pressure has certainly been influenced by the atmospheric state at earlier times, and might also have been different had alternate values for various model parameters been selected. However, which fields, at which locations and times, and which parameters, wielded the most influence on how the simulated cyclone came to be?

To investigate this, it is necessary to run a new model – the *adjoint model* – backward in time, commencing with the final forecast one wishes to examine. This new model is created, in effect, by transposing the TLM. In this situation, the TLM is but an intermediate step in the construction of the adjoint model. Indeed, the chief use of the TLM appears to be in validating the correctness of the adjoint model.

Unlike the TLM, however, the adjoint model does not propagate the original model's variables, or even perturbations from those variables. Instead, the model deals with *sensitivity* and is "initialized" at the final time with sensitivity concentrated in the field(s) and locale(s) of particular interest. In the present example, the adjoint would start off with some nonzero sensitivity in the model variable representing surface pressure at the cyclone center since that is the field and the locale we are focusing on. All other model variables would initially hold zeroes. The adjoint model then propagates this sensitivity backwards in time, during which it spreads among spatial locales and other model variables. In this way, the model attempts to identify the fields, locales and parameters that were most influential on determining the forecasted central pressure, through distributions and concentrations of sensitivity.

Put another way, I like to think of the adjoint model as propagating a *dynamically active tracer* or tracker backwards in time. This is a considerably enhanced extension of the concept of passive tracers. One often uses passive tracers when one wishes to identify the origin of air that is found to reside at a particular place and time. If you saved the wind data as the model ran forwards, then you can initialize a passive tracer at the locale of interest at the final time and run the "wind tape" (as it were) backwards, advecting the tracer upstream as time rewinds. Owing to its absolutely passive nature, this tracer does not influence the flow field which bears it – and this makes sense because in a very real sense the "future" has already occurred and should not be altered by this operation. The absolutely passive tracer is dynamically inert as well because it is simply being shunted by the wind.

But now suppose you are not as concerned with the origin of the air at the location of interest as with the origin of the *dynamics* that conspired to determine atmospheric conditions – the winds and more – there. Unlike the passive tracer model, the adjoint model is composed of a set of sensitivity trackers, one for every prognostic field and parameter of interest in the model. Even though the model may be initialized (say) at the final time with nonzero values in only one of the tracked fields, at only one of the domain's grid points, this tracker will spread to other tracked fields (and locales) as time rewinds owing to the coupled nature and structure of the equations. As far as the forward model is concerned, the adjoint tracer is still passive, and running the adjoint model itself does not change the already known future. However, the model can – within its inherent limitations – identify how that future came to be.

16.1 The Tangent Linear Model - Introduction

Here is a simple model equation consisting of a single prognostic variable u, a function of [2D or 3D] space and time, and a single model parameter α, the value of which is set at the initial time:

$$\frac{\partial u}{\partial t} = F_u(u, \alpha). \tag{16.1}$$

Integration of (16.1) in time starting from initial values yields the forecast solution $u(x, z, t)$. Let one such solution be termed the control run, which will be designated with the subscript "C" in this introductory discussion. The initial condition at time t=0 is $u_C(x, z, 0)$ and this simulation's parameter value is α_C. The forecast equation in this case will be revised slightly to read:

$$\frac{\partial u_C}{\partial t} = F_u(u_C, \alpha_C). \tag{16.2}$$

The initial values of u_C and α_C completely determine the subsequent evolution of the u field.

Now, let another, alternative simulation, to be designated with the subscript "A", also be run, yielding $u_A(x, z, t)$. This run may start with a different initial condition $u_A(x, z, 0)$ and/or a different value for the model parameter α_A. For this simulation, (16.1) becomes

$$\frac{\partial u_A}{\partial t} = F_u(u_A, \alpha_A). \tag{16.3}$$

The alternative solution may be viewed as a variation on the control run. Let the difference between the alternative and control runs' model parameter values be termed α'', so we have

$$\alpha'' = \alpha_A - \alpha_C. \tag{16.4}$$

At any given point in time and space, the difference between the two forecasts for u will be called u'', and defined as:

$$u''(x,z,t) = u_A(x,z,t) - u_C(x,z,t). \tag{16.5}$$

The obvious way to handle this situation is to simply run the nonlinear model twice, or for however many instances are needed to examine a parameter space consisting of initial u values and model parameter α values. As noted above, this could be a formidable task. If – and this is a big if – the perturbations from the control run are and remain small, then a good approximation to the alternative run(s) could be obtained from a tangent linear version of the original nonlinear model. Instead of predicting u_A and subtracting it from u_C to obtain u'', we are directly prognosing u'' using an approximated model. This model, the TLM, may be more efficient to run.

There appear to be two ways of defining the TLM. One is straightforward and utilizes first-order truncated Taylor series; the other is involved mathematically but appears to wind up with the same TLM formulation. The former is examined below; the latter in Appendix B. For simple models, the Taylor approach can be applied directly to the model differential equations, generating the TLM model in analytic form. The TLM differential equation is then discretized and integrated. For more interesting and commonly occurring situations, however, it seems one is stuck with starting with the model *difference* equations; i.e., the model code. Since we are concerned with assessing model sensitivity, and the "model" subsumes its coding, we should be more generally concerned with creating the TLM of the *finite difference* rather than the original differential equations anyway.

16.2 Construction of the TLM

16.2.1 TLM formulation of a differential equation

A function of a single variable, like $f(u)$, may be expanded by Taylor series as follows:

$$f(u_A) - f(u_C) = (u_A - u_C)\frac{\partial f}{\partial u}|_C + h.o.t.,$$

where *h.o.t.* stands for the higher order terms. At first glance, it appears this expression is directly extensible to functions of two or more variables in the following fashion:

$$f(u_A,\alpha_A) - f(u_C,\alpha_C) = (u_A - u_C)\frac{\partial f}{\partial u}|_C + (\alpha_A - \alpha_C)\frac{\partial f}{\partial \alpha}|_C + h.o.t.. \tag{16.6}$$

It turns out, however, that this is true only if the two partial derivatives on the right hand side (RHS) exist and are continuous at the reference point (u_C, α_C)[1]. This may need to be called into question later, and may be the salient difference between this approach and the more complex procedure discussed in Appendix B.

Assuming for the present that (16.6) is applicable, it may be applied to the RHS of the model equation, specifically expanding the RHS of (16.3) about the state defined by (16.2). In this case, we get:

$$F_u(u_A, \alpha_A) - F_u(u_C, \alpha_C) = (u_A - u_C)\frac{\partial F_u}{\partial u}|_C + (\alpha_A - \alpha_C)\frac{\partial F_u}{\partial \alpha}|_C + h.o.t.. \tag{16.7}$$

Henceforth, the higher order terms will be neglected. Using (16.2) and (16.3), it is seen the left hand side (LHS) of (16.7) is simply

$$\frac{\partial u_A}{\partial t} - \frac{\partial u_C}{\partial t} = \frac{\partial u''}{\partial t},$$

so that to a first order approximation (owing to the neglected terms), the equation forecasting the difference between the two simulations takes the form

$$\frac{\partial u''}{\partial t} \approx u''\frac{\partial F_u}{\partial u}|_C + \alpha''\frac{\partial F_u}{\partial \alpha}|_C. \tag{16.8}$$

Definitions (16.4) and (16.5) were used to obtain the above expression. This is the TLM for the perturbations, which were again designated u''. The derivatives depend only on the temporally evolving control run, and so perturbation forecasts possess the same evolving temporal trajectory as the control run simulation. The TLM is predicated on the existence of these derivatives and the negligibility of the missing higher order (perturbation product) terms. The latter assumption can be particularly specious in some, if not many, applications.

16.2.2 A simple example

The following example is cast in one-dimensional space for simplicity. First, we consider a tendency equation with one forcing term, consisting of Rayleigh friction. This is a simple damping function. The differential equation is:

$$\frac{\partial u}{\partial t} = F_u = -\alpha u, \tag{16.9}$$

where α represents the damping rate.

[1]Condon and Odishaw, *Handbook of Physics*, 1958, p. 1-38

To apply (16.8), we need the partial derivatives of F_u with respect the two independent variables, u and α, evaluated for the control run. It is seen that $\frac{\partial F_u}{\partial u}|_C = -\alpha_C$ and $\frac{\partial F_u}{\partial \alpha}|_C = -u_C$. Thus, the TLM of the model differential equation (16.9) is

$$\frac{\partial u''}{\partial t} = -u'' a_C - \alpha'' u_C. \tag{16.10}$$

Although we're neglecting the approximation sign, the expression (16.10) is not precise. The perturbation product term,

$$u''\alpha'' \frac{\partial^2 F_u}{\partial u \partial \alpha} = -u''\alpha''$$

is missing and the TLM is accurate only as long as this term is negligible.

16.2.3 A partially discretized simple example

We are actually concerned with constructing the TLM of the model discretized *difference* equations rather than its differential equations. As an example, we revisit the Rayleigh friciton equation and discretize its RHS. We will have to make a decision regarding the LHS discretization before long, however. If the centered, three time level leapfrog scheme is used, the damping term has to be evaluated at the past time (time level n-1) for stability. Presuming this choice has been made, the mixed differential-difference equation for (16.9) is:

$$\frac{\partial u_i}{\partial t} = F_{u,i} = -\alpha u_i^{n-1}, \tag{16.11}$$

where i is the spatial index. This is the equation we wish to construct a TLM from.

Now Taylor expansions are applied. The equation (16.11) may be written as:

$$\frac{\partial u_i}{\partial t} = F_u(u_i^{n-1}, \alpha).$$

The truncated expansion of the RHS for the alternative run about the control run is:

$$F_u(u_{i,C}^{n-1} + u''^{n-1}_i, \alpha_C + \alpha'') - F_u(u_{i,C}^{n-1}, \alpha_C) \;\approx\; u''^{n-1}_i \frac{\partial F_u}{\partial u^{n-1}}|_C + \alpha'' \frac{\partial F_u}{\partial \alpha}|_C$$

However, the two partial derivatives on the above equation's RHS are:

$$\begin{aligned} \frac{\partial F_u}{\partial u^{n-1}}|_C &= \frac{\partial}{\partial u^{n-1}} \left[-\alpha u_i^{n-1}\right]|_C \\ &= -\alpha_C, \end{aligned}$$

and

$$\begin{aligned} \frac{\partial F_u}{\partial \alpha}|_C &= \frac{\partial}{\partial \alpha} \left[-\alpha u_i^{n-1}\right]|_C \\ &= -u_{i,C}^{n-1}, \end{aligned}$$

while the LHS is simply $\frac{\partial u''}{\partial t}$. Thus, the TLM model prognostic equation is:

$$\frac{\partial u''}{\partial t} = -\alpha_C u_i''^{n-1} - \alpha'' u_{i,C}^{n-1}.$$

16.2.4 TLM of a model equation with constant advective velocity

As already mentioned, analytic forms for the partial derivatives of F_u for simple models may be easily constructed. In more complex settings, one seems forced to discretize the model equations first. We have now reached this point.

Again considering only a single spatial dimension for simplicity, the PDE governing constant advection of a variable u may be written as:

$$\frac{\partial u}{\partial t} = F_u = -c_x \frac{\partial u}{\partial x}, \tag{16.12}$$

where c_x is the advection speed. Employing the second-order centered approximation for the RHS of (16.12), we create:

$$\begin{aligned} \frac{\partial u}{\partial t} &= -\frac{c_x}{2\Delta x}\left[u_{i+1}^n - u_{i-1}^n\right] && (16.13)\\ &= -\frac{c_x}{2\Delta x}u_{i+1}^n + \frac{c_x}{2\Delta x}u_{i-1}^n && (16.14)\\ &= F_u\left[c_x, u_{i+1}^n, u_{i-1}^n\right]. && (16.15) \end{aligned}$$

Equation (16.15) shows that the forcing F_u is a function of three independent variables, being the advection speed and the present time's values of u on either side of the current grid point i.

For the truncated Taylor expansion, we need the partial derivatives of the RHS with respect to each of the independent variables separately. I know of no way of doing this for the original differential equation, but differentiating the discretized version is very simple. Thus, starting with

$$\frac{\partial u''}{\partial t} \approx c_x'' \frac{\partial F_u}{\partial c_x}|_C + u''^n_{i+1}\frac{\partial F_u}{\partial u_{i+1}^n}|_C + u''^n_{i-1}\frac{\partial F_u}{\partial u_{i-1}^n}|_C. \tag{16.16}$$

we quickly wind up with

$$\frac{\partial u''}{\partial t} \approx -\frac{c_x''}{2\Delta x}\left[u_{i+1,C}^n - u_{i-1,C}^n\right] - \frac{c_{xC}}{2\Delta x}u''^n_{i+1} + \frac{c_{xC}}{2\Delta x}u''^n_{i-1}. \tag{16.17}$$

16.2.5 TLM of a model equation with nonlinear advective velocity

If we replace c_x in (16.12) with u, creating a nonlinear term, (16.12), (16.13) and (16.15) become

$$\begin{aligned}\frac{\partial u}{\partial t} &= -u\frac{\partial u}{\partial x} \\ &= -\frac{u_i^n}{2\Delta x}\left[u_{i+1}^n - u_{i-1}^n\right] \\ &= F_u\left[u_i^n, u_{i+1}^n, u_{i-1}^n\right],\end{aligned}$$

respectively, and our TLM is evaluated using:

$$\frac{\partial u''}{\partial t} \approx u''^n_i \frac{\partial F_u}{\partial u_i^n}|_C + u''^n_{i+1}\frac{\partial F_u}{\partial u_{i+1}^n}|_C + u''^n_{i-1}\frac{\partial F_u}{\partial u_{i-1}^n}|_C. \tag{16.18}$$

Combining all of the above examples, it is seen that the total number of independent variables requiring differentiation in the construction of the TLM is equal to the number of model parameters on the RHS of the equation plus – for *each* prognostic variable and *each* time level involved – the number of unique points contributing to the discretization of the spatial operators present on the RHS.

16.2.6 TLM of a simple system of nonlinear equations

We extend the foregoing to a system of coupled equations by considering a simple example consisting of two prognostic variables (u and v), two model parameters (g and α) and a single grid point in space. The parameters are not perturbed in this example. The model differential equations for this example are:

$$\frac{\partial u}{\partial t} = gv - \alpha u + uv \tag{16.19}$$

$$\frac{\partial v}{\partial t} = -gu - \alpha v - uv. \tag{16.20}$$

The right hand sides of the above equations will be referred to as F_u and F_v, respectively.

Since only the model variables are being subjected to perturbation, we have

$$\frac{\partial u''}{\partial t} = u''\frac{\partial F_u}{\partial u}|_C + v''\frac{\partial F_u}{\partial v}|_C \tag{16.21}$$

$$\frac{\partial v''}{\partial t} = u''\frac{\partial F_v}{\partial u}|_C + v''\frac{\partial F_v}{\partial v}|_C \tag{16.22}$$

for the TLM model equations. Upon evaluation of the required RHS derivatives, we come up with

$$\frac{\partial u''}{\partial t} = u''\left(-\alpha_C + v_C\right) + v''\left(g_C + u_C\right) \tag{16.23}$$

$$\frac{\partial v''}{\partial t} = u''\left(-g_C - v_C\right) + v''\left(-\alpha_C - u_C\right). \tag{16.24}$$

for the TLM. The preceeding can be written more compactly in matrix form. Specifically, we have

$$\begin{bmatrix} \frac{\partial u''}{\partial t} \\ \frac{\partial v''}{\partial t} \end{bmatrix} = \begin{bmatrix} -\alpha_C + v_C & g_C + u_C \\ -g_C - v_C & -\alpha_C - u_C \end{bmatrix} \begin{bmatrix} u'' \\ v'' \end{bmatrix}, \tag{16.25}$$

whereas the general form of (16.21)-(16.22) is

$$\begin{bmatrix} \frac{\partial u''}{\partial t} \\ \frac{\partial v''}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_u}{\partial u}|_C & \frac{\partial F_u}{\partial v}|_C \\ \frac{\partial F_v}{\partial u}|_C & \frac{\partial F_v}{\partial v}|_C \end{bmatrix} \begin{bmatrix} u'' \\ v'' \end{bmatrix}. \tag{16.26}$$

The matrix is identified as the Jacobian of partial derivatives, evaluated for the control run.

All that remains is to discretize the time derivatives on the LHS, supply initial conditions and to integrate the resulting model. If a two time level scheme is employed with time step Δt, we have

$$\begin{bmatrix} u'' \\ v'' \end{bmatrix}^{n+1} = \Delta t \begin{bmatrix} -\alpha_C + v_C^n + 1 & g_C + u_C^n \\ -g_C - v_C^n & -\alpha_C - u_C^n + 1 \end{bmatrix} \begin{bmatrix} u'' \\ v'' \end{bmatrix}^n, \tag{16.27}$$

where it is appreciated that the now slightly augmented matrix is a function of time but depends solely on the control run. Defining $\mathbf{x}''$ as $[u'', v'']^T$, this is more compactly written as

$$\mathbf{x}''^{n+1} = \mathbf{A}_n \mathbf{x}''^n, \tag{16.28}$$

where $\mathbf{A}_n$ has incorporated the time discretization Δt.

16.3 The adjoint model

16.3.1 Roadmap for adjoint formulation and discussion

Now we employ the TLM to construct the adjoint model. The adjoint is essentially created by transposing the TLM's Jacobian matrix and then running the resulting model backwards. As it stands, this statement may not be particularly enlightening. However, we start with this and (hopefully) clarify some concepts along the way. Part of the following discussion mirrors that presented in Errico and Vukicevic (1992, *MWR*; hereafter "EV").

Importantly, numerous aspects of the adjoint model are simplified, and proper coding of same much more easily checked, when *time integration schemes involving only two time levels* are employed. This raises a sticky situation, however: For advection, two time level schemes are usually either unacceptably flawed or computationally burdensome. On the other hand, adjoint models made from forward codes using more efficient three time level schemes are much more convoluted. Despite this, until further notice the following examples implicitly presume or explicitly use simple, two time level time differencing[2].

The structure of this discussion is as follows: First, we recapituate the presentation of the fully discrete TLM in matrix form, as this will facilitate the derivations to follow. Then, the concept of the *forecast aspect* will be introduced, at its simplest and most straightforward representing one thing about the final control model forecast you'd like to know more about the evolution of. The adjoint variables will appear at this point, representing the sensitivities you wish to track backwards in time.

16.3.2 The TLM in matrix form

Let $\mathbf{x}$ be a vector representing all temporally varying independent variables, or degrees of freedom, in the model. In the discretized model it includes all of the prognostic fields (wind components, temperature, humidity variables, etc.) at all of the model grid points. Thus, if there are M variables and I total grid points, the length of $\mathbf{x}$ is $L = M \times I$. Then, let $\mathbf{x}''$ be the perturbations from a control run as forecast by a TLM.

In the simplest possible model, one discretized with a forward time scheme (requiring only two time levels at any instant of time) and having no model parameters that are varied from the control run configuration, we can express the prediction of the state of $\mathbf{x}''$ at time level $n+1$ based on the state at time n as:

$$\mathbf{x}''^{n+1} = \mathbf{A}_n \mathbf{x}''^{n}, \tag{16.29}$$

just as in (16.28). Here $\mathbf{A}_n$ is the $L \times L$ time-dependent Jacobian matrix evaluated using the control run information at time n.

In (16.26), the Jacobian was a simple 2×2 matrix, because there were only two degrees of freedom. In a model with a substantial set of prognostic variables, and particularly a lot

[2]As a compromise, I have made forward and backward models using the leapfrog and Euler backwards schemes. The latter is a two time level scheme, but requires twice the computational burden of the leapfrog since it involves both predictive and corrective calculations for each time step. Still, it generates virtually identical results (for the simulations I've made thusfar) and its two time level construction facilitates accuracy assessment.

of grid points, $\mathbf{A}$ can be a very huge matrix. Within its structure, it allows the possibility that every perturbation forecast in $\mathbf{x}^{n+1}$ can potentially be influenced by every perturbation in the model at the present time. This would be the case if $\mathbf{A}_n$ were dense, with few or no zeroes. In compressible models, however, the forecasts at time $n+1$ and grid point i depends only on previous values located in the immediate vicinity of i. Thus, for large domains $\mathbf{A}$ is a very, very sparse matrix.

Now, since $\mathbf{x}''^n$ itself depended upon $\mathbf{A}_{n-1}$ and $\mathbf{x}''^{n-1}$ in a manner analogous to (16.29), so (16.29) could also be written as:

$$\mathbf{x}''^{n+1} = \mathbf{A}_n \mathbf{A}_{n-1} \mathbf{x}''^{n-1}. \tag{16.30}$$

The state at time $n-1$, in turn, relied upon the still earlier time $n-2$ and so on, right on back to the initial condition at time step $n = 0$. Thus, we could also write (16.29) as

$$\mathbf{x}''^{n+1} = \mathbf{P}_{n+1} \mathbf{x}''^{0}, \tag{16.31}$$

where $\mathbf{P}_{n+1} = \mathbf{A}_n \mathbf{A}_{n-1} \cdots \mathbf{A}_0$ and is called the *transition matrix*. The upshot is that *in a deterministic model such as this, the model state at any given subsequent time can be considered to be a transformation applied to the state at the initial time.*

This has a further important implication. As noted above, the sparseness of the matrix $\mathbf{A}_n$ at any given time n reflects the fact that a perturbation forecast at time $n+1$ can be influenced only by a small number of perturbations present at time n. However, the multiplication of $\mathbf{A}_n$ with $\mathbf{A}_{n-1}$ in (16.30), which relates the perturbations at times $n+1$ and $n-1$, would logically produce a fuller and more complex matrix. Now consider a temporal integration proceeding from the initial condition at time 0 to final forecast time N which would entail the sequential multiplication of every individual, sparse matrix $\mathbf{A}$ spanning the integration period. By (16.31), this matrix is identified as $\mathbf{P}_N$, and it would likely be very full. The implication is that a given forecast at the final time is directly linked to a wide array of points and fields in the initial state, and is thus correspondingly sensitive to alterations in those initial conditions. It will be seen the adjoint model uses this matrix (or, more precisely, its transpose), to quantify this sensitivity and dependence.

16.3.3 The forecast aspect

At this point, we need to introduce the *forecast aspect*, which quite literally represents some aspect of our forecast about which we wish to examine the sensitivity. This aspect, which

will be designated J, can be a complex function of many model fields, spatial locations and points in time, so $J = J(\mathbf{x})$. One is not limited to directly prognosed fields; derived fields (e.g., vorticity and relative humidity, say) are also fair game. Dramatic simplification of the presentation, however, results from defining J at a single time. This restriction will henceforth be adopted.

In the simplest and most tractable application, J might be be a function of a single field (such as pressure), at a single location (such as at the surface, at a single grid point in the center of a cyclone, perhaps) at a single time (such as the final forecast time N). We might then wonder how this particular aspect of interest came to be, how it was influenced or even controlled by the atmospheric state existing at earlier times. That is, we desire to learn which fields and which locations at earlier times most profoundly determined how surface pressure at the grid point of interest evolved.

Consider a J based on the control run at the final forecast time N which we will call $\tilde{J}_N$, some function of $\tilde{\mathbf{x}}_N$. If we changed this run's initial conditions, we might conceivably alter what the forecast aspect comes to be at that final forecast time. This would be an alternative run, having its own forecast aspect J_N^*. Or, say the control run's surface pressure forecast is known to be wrong, and we're interested in how that error came to be. That is, we wish to track the error backwards in time, to other prognostic fields and spatial points. In this case, our J_N^* would be the observed pressure at the aspect point.

In either scenario, we can define $dJ_N \equiv J_N^* - \tilde{J}_N$, representing either the aspect difference between two simulations or between control simulation and reality. As J is a function of $\mathbf{x}_N$, we can approximate dJ_N as ΔJ_N, defined using a truncated Taylor series expansion about the control run state. Keeping in mind that the TLM variables ${x''}_l^N$ approximate the difference between the control and alternative fields, we can write the truncated Taylor expansion as:

$$\Delta J_N = \sum_{l=1}^{L} {x''}_l^N \frac{\partial \tilde{J}}{\partial \tilde{x}_l^N}. \tag{16.32}$$

(Recall that l ranges over all M fields and I locales, as $L = M \times I$.)

Thus, whether a particular field located at a particular point (i.e., x_l^N) can affect the forecast aspect (i.e., contribute to ΔJ_N) depends on two things: the *existence of variation from its control run value there* (i.e., ${x''}_l^N \neq 0$) and, more importantly, the *presence of any sensitivity to that perturbation there* (that is, $\frac{\partial \tilde{J}}{\partial \tilde{x}_l^N} \neq 0$). Naturally, if the sensitivity is zero, no perturbation, of any size, can influence the final forecast at the grid point of interest... at least as far as the linearized TLM and adjoint models are concerned. In contrast, if the sensitivity

is huge then even a small variation could exert a very substantial effect.

The perturbation values ${x''}_l^n$ at any time n can be obtained by an integration of the TLM forward from perturbed initial conditions. The corresponding sensitivities, $\frac{\partial \tilde{J}}{\partial \tilde{x}_l^n}$, will be provided by the adjoint model. As a shorthand, we will adopt the notation

$$\frac{\partial \tilde{J}}{\partial \tilde{x}_l^n} = \hat{x}_l^n, \tag{16.33}$$

which is interpreted as follows: $\hat{x}_l^n$ is the sensitivity of aspect J to model variable x_l at time n. So, as an example, say that J is the central pressure of the cyclone at final time N. One of entries of $\hat{x}_l^n$ represents the vertical velocity field at some grid point at time n. The magnitude of the adjoint variable corresponding to that particular field and locale represents the adjoint model's assessment of how sensitive that final central pressure is to the vertical velocity field at that time and place. If J has units of millibars, that particular adjoint variable would be dimensioned millibars per meter per second.

Now it must be realized that ΔJ_N (or, more properly, dJ_N) is *known information*, at least in this application. We've chosen our J and can be presumed to have both the control and alternative datasets for time N at hand. Moreover, the right hand side of (16.32), applied at the final time N, is also known. The TLM has already been integrated, yielding ${x''}_l^N$ for all l (fields and grid points combined).

More importantly, *the sensitivities involved in* (16.32) *are trivial*, at least as far as the final forecast time is concerned. For a forecast aspect also consisting of a single field at a single grid point, the only way that any final time perturbation could instantaneously influence the forecast aspect is if it were in the field that defines the aspect (e.g., pressure) and located directly at the forecast aspect point. Thus, in this example, the only nonzero sensitivity at final time N is that owing to the aspect field at point itself. Indeed, that nonzero value is 1, because the partial derivative of J depends only on itself there.

The key is to obtain the sensitivities representing *earlier times*. Owing to the coupled nature of the equations, we can reasonably expect that as we move backwards in time, more and more fields and locales could be contributing some influence towards the final forecast aspect. It is the backward integration of the adjoint model that will yield these sensitivities. How is the adjoint model constructed? This is the topic of the next subsection.

16.3.4 Construction of the adjoint

In this subsection, the following strategy is employed: First, we define the adjoint model in a practical way, using a shortcut rather than a mathematically rigorous procedure. Then the basis of the shortcut is examined. Finally, a neat manner, based on the forecast aspect sensitivity already considered, in which the fidelity of the adjoint model may be assessed is presented.

The adjoint to (16.29) is practically constructed in the following way. First, the matrix $\mathbf{A}_n$ is transposed and transported (not inverted) across the equivalence sign. Then the TLM perturbation variable vector $\mathbf{x}''$ on either side is replaced with the adjoint sensitivity vector, denoted $\hat{\mathbf{x}}$. That means there is an adjoint version of every degree of freedom in the TLM. The adjoint variables in $\hat{\mathbf{x}}$ are "initialized" at the control run's *final* time, and are integrated *backward* in time, perhaps as far as that run's initial time. In other words, the TLM forecast representation (16.29) has been transformed into:

$$\hat{\mathbf{x}}^n = \mathbf{A}_n^T \hat{\mathbf{x}}^{n+1}, \tag{16.34}$$

where the superscript "T" denotes transposition.

Note (16.34) does not really immediately follow from (16.29), not only because $\hat{\mathbf{x}}$ is not $\mathbf{x}''$, but also due to the fact that $\mathbf{A}^T \neq \mathbf{A}^{-1}$ (except perhaps in very simple cases). Thus, we are not actually running the TLM model backwards. Instead, we are using the TLM formulation of the original nonlinear model to create a *new* model – the adjoint model – that is designed to be run backwards. Indeed, instead of using $\mathbf{A}_n$ to push us forward from time n to $n+1$, as we did in the forward integration of the TLM, we are now using the transpose of that same matrix to draw our (adjoint) model backwards over that same time interval, from time $n+1$ to n. This is further illustrated in Fig. 1.

Recall that the adjoint variable $\hat{\mathbf{x}}^n$ at any given time n is simply the sensitivity of the forecast aspect at that time, according to (16.33). Thus, the adjoint model takes the trivial sensitivities defined at the control run's final time and prognoses them backwards in time. Note also that if we run the adjoint model backwards over the entire length spanned by the forward model, then we have employed – in reverse succession – every matrix $\mathbf{A}_n$ that resulted from the forward integration. Thus, we have (and can exploit) the same transition matrix formed in the forward procedure.

This is also illustrated in the following operation. Starting with (16.31), the equation which related the TLM states at the initial and final times via the transition matrix $\mathbf{P}_N$, and using

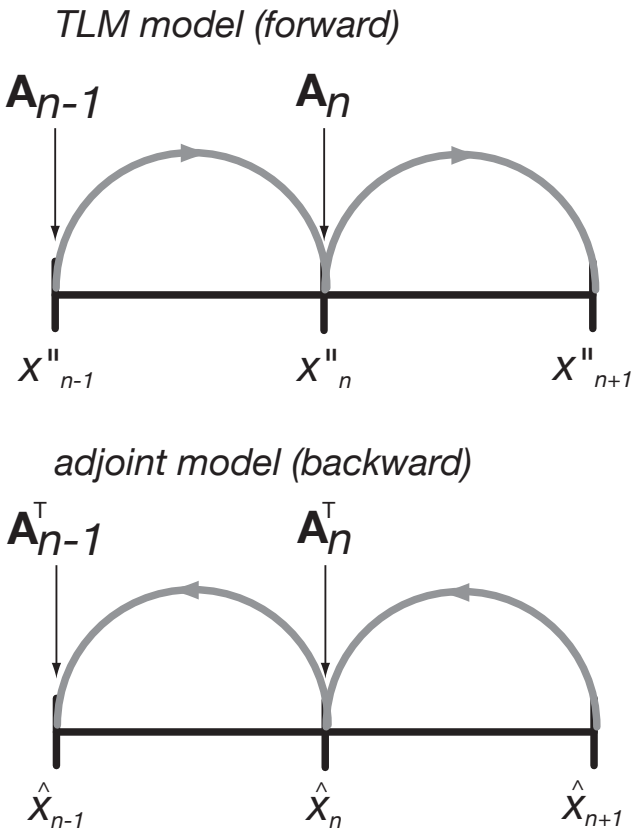

Figure 16.1: TLM model forward time stepping and adjoint model backward time stepping

the same strategy employed for (16.34), we create

$$\hat{\mathbf{x}}^0 = \mathbf{P}_N^T \mathbf{x}^N. \tag{16.35}$$

Noting this property of matrix multiplications

$$(abc)^T = c^T b^T a^T,$$

it is seen that

$$\mathbf{P}_N^T = \mathbf{A}_0^T \cdots \mathbf{A}_{N-2}^T \mathbf{A}_{N-1}^T.$$

Thus, the same control-run dependent mappings that stepped the TLM forward in time are used in reverse sequence to draw the adjoint model backward from the final time.

The basis of the adjoint model lies in what is termed the "adjoint property", which may be simply stated as:

$$(\mathbf{a}, \mathbf{L}\mathbf{b}) = \left(\mathbf{L}^T \mathbf{a}, \mathbf{b}\right). \tag{16.36}$$

The parentheses indicate matrix multiplication (the inner product), the vectors $\mathbf{a}$ and $\mathbf{b}$ are dimensioned $L \times 1$ and the matrix $\mathbf{L}$ is $L \times L$. The adjoint property essentially says this: A

multiplication of the vector $\mathbf{a}$ by the vector resulting from the operation $\mathbf{Lb}$ yields the same result as the operation which multiplies $\mathbf{L}^T\mathbf{a}$ by $\mathbf{b}$. In the situations we will encounter, $\mathbf{L^T}$ is the transpose of $\mathbf{L}$ but in more general situations it is that matrix's *adjoint*.

This property permits us to directly relate ΔJ_N with ΔJ_0. Using definition (16.33) along with the inner product notation, we can rewrite (16.32) as

$$\Delta J_N = \left(\hat{\mathbf{x}}^N, \mathbf{x}''^N\right). \tag{16.37}$$

Again, keep in mind that ΔJ_N is known and trivial. Still, since the final TLM model state is directly related to the initial condition via the transition matrix $\mathbf{P}_N$, we can use (16.31) to rewrite the above equation as

$$\Delta J_N = \left(\hat{\mathbf{x}}^N, \mathbf{P}_N\mathbf{x}''^0\right). \tag{16.38}$$

Now the adjoint property (16.36) can be used to move the transition matrix's position within the inner product:

$$\Delta J_N = \left(\mathbf{P}_N^T\hat{\mathbf{x}}^N, \mathbf{x}''^0\right). \tag{16.39}$$

By (16.35), though, this means we actually have:

$$\Delta J_N = \left(\hat{\mathbf{x}}^0, \mathbf{x}''^0\right) \equiv \Delta J_0. \tag{16.40}$$

Given perturbations in the initial condition ($\mathbf{x}''^0$), their effect on the final forecast aspect is determined by the sensitivities ($\hat{\mathbf{x}}^0$) at that time. Those sensitivities were drawn back starting at the final time via integration of the adjoint model (16.34).

To recap, the summed product of the control run perturbations and adjoint sensitivities at the final time (which was trivial) is equal to the summed product of the control run perturbations and adjoint sensitivities at the initial time. This is even more general than it might appear, since we can define the transition matrix as valid for any time interval bounded by n and N where n need not be zero (the initial time). Thus, we can generalize (16.39) for any time n by writing

$$\Delta J = \left(\hat{\mathbf{x}}^n, \mathbf{x}''^n\right). \tag{16.41}$$

In the above expression, the subscript on ΔJ is dropped because its temporal invariance has now been established. This holds true whenever two time level discretization is employed. Since ΔJ was trivial at the final time N, it necessarily follows that it is trivial at any and every other time. This seems useless, but actually this characteristic provides a very useful

gauge of the fidelity of the adjoint model. If one runs the TLM model forward and saves those fields at every time step, then ΔJ values may be calculated at every step during the adjoint model's reverse integration. *These values should not vary, at least to within roundoff error*[3]. Failure to preserve the temporal invariance of ΔJ indicates an inconsistency between the TLM and adjoint modes exists.

16.3.5 An example

It is not always necessary to discretize the model's time derivatives prior to forming the adjoint model (at least when we confine ourselves to simple two time level schemes, that is). To simplify the discussion, the notation will be changed a bit when this alternative solution path is adopted. The example below comes from Lorenz and Emanuel (1998, *JAS*), though the notation has been altered for convenience.

There is one prognostic variable, ϕ, and the model domain consists of N gridpoints, indexed with the subscript i. The partially discretized model equation predicting ϕ_i is

$$\begin{aligned} \frac{d\phi_i}{dt} &= (\phi^n_{i+1} - \phi^n_{i-2})\phi^n_{i-1} - \phi^n_i + G & (16.42) \\ &= F\left[\phi^n_i, \phi^n_{i+1}, \phi^n_{i-1}, \phi^n_{i-2}, G\right], & (16.43) \end{aligned}$$

where G represents a model forcing that is held constant with time and not varied from its control run value. The perturbation value ϕ'' is evaluated via truncated Taylor series about the control run values (indicated by the "C" subscripts and by Φ) as:

$$\begin{aligned} \frac{d\phi''_i}{dt} &= \phi''^n_i \frac{\partial F}{\partial \phi^n_i}|_C + \phi''^n_{i+1}\frac{\partial F}{\partial \phi^n_{i+1}}|_C + \phi''^n_{i-1}\frac{\partial F}{\partial \phi^n_{i-1}}|_C + \phi''^n_{i-2}\frac{\partial F}{\partial \phi^n_{i-2}}|_C & (16.44) \\ &= -\phi''^n_i + \phi''^n_{i+1}\Phi^n_{i-1} + \phi''^n_{i-1}\left[\Phi^n_{i+1} - \Phi^n_{i-2}\right] - \phi''^n_{i-2}\Phi^n_{i-1}, & (16.45) \end{aligned}$$

the second expression coming from evaluating the required partial derivatives. This is the TLM. The control run values are indicated by the capital letters to simply the notation.

All of the RHS entries are computed at time n. The matrix version of the TLM for a very small domain consisting of five points would look like this:

$$\begin{bmatrix} d_t+1 & -\Phi_5 & 0 & \Phi_5 & -(\Phi_2-\Phi_4) \\ -(\Phi_3-\Phi_5) & d_t+1 & -\Phi_1 & 0 & \Phi_1 \\ \Phi_2 & -(\Phi_4-\Phi_1) & d_t+1 & -\Phi_2 & 0 \\ 0 & \Phi_3 & -(\Phi_5-\Phi_2) & d_t+1 & -\Phi_3 \\ -\Phi_4 & 0 & \Phi_4 & -(\Phi_1-\Phi_3) & d_t+1 \end{bmatrix} \begin{bmatrix} \phi''_1 \\ \phi''_2 \\ \phi''_3 \\ \phi''_4 \\ \phi''_5 \end{bmatrix} \quad (16.46)$$

[3]Running double precision on all reals on a Sun workstation, we have gained seventeen digits of accuracy even for prolonged integrations.

In the above, d_t denotes the time derivative and all variables that appear are at time n. Specification of boundary conditions was required; we presumed periodicity such that indices 0 and 5 represented the same point.

Now we take the transpose of the mapping matrix and replace ϕ'' with $\hat{\phi}$. Thus, the first column in the matrix above becomes the first row. For the point $i = 1$ this yields:

$$\left[\frac{d}{dt} + 1\right] \hat{\phi}_1 - [\Phi_3 - \Phi_5]\, \hat{\phi}_2 + \Phi_2 \hat{\phi}_3 - \Phi_4 \hat{\phi}_5 = 0. \tag{16.47}$$

This is the adjoint equation for gridpoint $i = 1$. Actually, all of the gridpoints would yield a structurally similar equation, especially owing to the presumed periodicity. Generalizing and rearranging this, we see the adjoint model is

$$\frac{d\hat{\phi}_i}{dt} = -\hat{\phi}_i^n + (\Phi_{i+2}^n - \Phi_{i-1}^n)\hat{\phi}_{i+1}^n - \Phi_{i+1}^n \hat{\phi}_{i+2}^n + \Phi_{i-2}^n \hat{\phi}_{i-1}^n. \tag{16.48}$$

It is noted this is Lorenz and Emanuel's equation (8) (with a typographical error repaired).

16.3.6 An alternative strategy

As stated earlier, for models with a large number of grid points and prognostic variables, the matrix $\mathbf{A}_n$ will be huge yet sparse. EV outline a strategy for constructing the adjoint equations by hand, essentially by doing the transposition locally. This strategy works as illustrated herein when two time level discretization is employed, and is similar to the path taken by automatic differentiators. We first examine the Lorenz-Emanuel example.

In the EV strategy, we first identify every independent variable in the model TLM's discretized RHS(s) that shows up at a given time step and grid point location. (The time derivative is not yet discretized.) This therefore includes every prognostic variable at every grid point and time level that appears on the TLM's RHS. Later, when more complex problems are considered, model parameters that will be varied from their control run values will also be identified. Adjoint variables will be created from all of these.

Then, as an intermediate step, each equation is converted into a set of adjoint equations. In (16.45), the LHS ${\phi''}_i$ becomes the multiplier $\hat{\phi}_i$ for the adjoint equations' RHS terms. The RHS perturbation variables each create their own prognostic adjoint equation. In this example, the TLM has only one variable at a given grid point, namely ${\phi''}_i$. This creates the adjoint term multiplier $\hat{\phi}_i^n$. The RHS independent variables are ${\phi''}_i^n, {\phi''}_{i+1}^n, {\phi''}_{i-1}^n$, and ${\phi''}_{i-2}^n$. At this intermediate step, these generate four prognostic adjoint equations

$$\frac{d\hat{\phi}_i}{dt}, \frac{d\hat{\phi}_{i+1}}{dt}, \frac{d\hat{\phi}_{i-1}}{dt}, \frac{d\hat{\phi}_{i-2}}{dt}.$$

Thus, for example, the ${\phi''}_{i-2}$ term in (16.45) yields the adjoint time derivative $\frac{d\hat{\phi}_{i-2}}{dt}$, the RHS of which is the coefficient of ${\phi''}_{i-2}$ (namely Φ^n_{i-1}) multiplied by the adjoint version of the LHS (namely $\hat{\phi}_i$). When completed, this system of equations is produced:

$$\frac{d\hat{\phi}_i}{dt} \leftarrow -\hat{\phi}^n_i \tag{16.49}$$

$$\frac{d\hat{\phi}_{i+1}}{dt} \leftarrow \Phi^n_{i-1}\hat{\phi}^n_i \tag{16.50}$$

$$\frac{d\hat{\phi}_{i-1}}{dt} \leftarrow (\Phi^n_{i+1} - \Phi^n_{i-2})\hat{\phi}^n_i \tag{16.51}$$

$$\frac{d\hat{\phi}_{i-2}}{dt} \leftarrow -\Phi^n_{i-1}\hat{\phi}^n_i \tag{16.52}$$

Note in this simple example every adjoint multiplier on the RHS is $\hat{\phi}^n_i$. This is because these four adjoint equations came from a single TLM equation (with but one LHS, of course). The backarrow is employed to indicate that the RHS term is added to an LHS equation that may also have other terms. That would be more obvious in an example with more than one equation, having common terms on the RHS.

I've noticed that – at least under some conditions – these four equations thus created can be condensed into the single equation (16.48). This is done by taking the three equations applicable to grid points other than point i and mapping them to that point. So, (16.52), for example, becomes

$$\frac{d\hat{\phi}_i}{dt} \leftarrow -\Phi^n_{i+1}\hat{\phi}^n_{i+2}.$$

Each index i was increased by two wherever it appeared. After this mapping, all four equations have the same LHS and the RHS terms can thus be aggregated. This yields the adjoint equation previously presented in (16.48), without the intermediate steps of constructing and transposing the matrix $\mathbf{A}_n$. (Since I've not read this anywhere, keep a look out for places where this strategy might fail.)

Once this remapping is completed, the adjoint model difference equations look rather like the original equations they were constucted from. This makes applying the boundary conditions (BCs) straightforward and easy, at least when the BCs are zero-gradient, periodic or solution-specified. This remapping strategy makes proper implementation of BCs for diffusion terms particularly easy. It is noted that this strategy does not appear to work when open radiation-type boundary conditions are employed. (In fact, I have not gotten an open BC to work, and it is possible it simply cannot work properly in reverse.)

As the final step, the adjoint's LHS has to be discretized. The same scheme that was used in the nonlinear (and TLM) model is used, but simply run backwards in time. EV emphasize that proper ordering in the adjoint model must be respected, and that for the reverse model operations are executed in reverse. Thus, if the forward model performs the sequential operations A $\rightarrow$ B $\rightarrow$ C, then the adjoint model executes the adjoint of those operations in reverse sequence, i.e., from C $\rightarrow$ B $\rightarrow$ A.

It is noted that automatic differentiation software tools effectively skip this final step because they differentiate the entire model difference code at once, including the temporal discretization. This doesn't make a difference when a simple two time level scheme is employed. Using a three time level scheme (such as the leapfrog scheme) is much more complicated; see the appendix of Talagrand and Courtier (1987, *QJRMS*) for a demonstration. The leapfrog scheme is discussed in Sec. 9.3.9 below.

Actually, since the adjoint model never references the TLM perturbations at all (only the control run information appears on the RHS), it is again emphasized that the *TLM itself never has to be integrated forward at all*, unless one is performing the temporal consistency check on ΔJ. The TLM was just used as an intermediate but necessary step in the construction of the adjoint model. Typically, one desires to know the sensitivies at any given time. In this situation, the perturbations applied at that time represent whatever it takes to bring about desired result at the final time.

16.3.7 A more complex 1D example

Here is an equation with constant advection (at speed c_x) and Rayleigh frictional damping (at rate α), in differential and semi-discretized forms:

$$\frac{\partial u}{\partial t} = -c_x \frac{\partial u}{\partial x} - \alpha u, \tag{16.53}$$

$$= -\frac{c_x}{2\Delta x}\left[u_{i+1}^n - u_{i-1}^n\right] - \alpha u_i^n. \tag{16.54}$$

In this example, the time derivative would again have to be discretized with the two time level scheme because the sponge term is evaluated at time n. Of course, employing the simple forward time scheme yields an unstable combination anyway, but serves the purposes of demonstration.

The TLM version of the foregoing equation (using upper case symbols U, A and C_x to denote

the control run values) is:

$$\frac{\partial u''}{\partial t} = -\frac{C_x}{2\Delta x}\left[u''^n_{i+1} - u''^n_{i-1}\right] - \frac{c''_x}{2\Delta x}\left[U^n_{i+1} - U^n_{i-1}\right] - \mathrm{A}u''^n_i - \alpha'' U^n_i. \tag{16.55}$$

The RHS contains three time-dependent independent perturbation variables (u''^n_{i+1}, u''^n_{i-1}, and u''^n_i) as well as two temporally fixed model perturbation parameters (c''_x and α''). This will result in *five* time-dependent adjoint variables.

Using the EV strategy, these five equations are:

$$\frac{\partial \hat{u}_{i+1}}{\partial t} \leftarrow -\frac{C_x}{2\Delta x}\hat{u_i}^n \tag{16.56}$$

$$\frac{\partial \hat{u}_{i-1}}{\partial t} \leftarrow \frac{C_x}{2\Delta x}\hat{u_i}^n \tag{16.57}$$

$$\frac{\partial \hat{u}_i}{\partial t} \leftarrow -\mathrm{A}\hat{u_i}^n \tag{16.58}$$

$$\hat{c}_{xi} \leftarrow -\frac{1}{2\Delta x}\left[U^n_{i+1} - U^n_{i-1}\right]\hat{u_i}^n \tag{16.59}$$

$$\hat{\alpha}_i \leftarrow -U^{n+1}_i \hat{u_i}^n. \tag{16.60}$$

Note that while the parameter perturbations are temporally fixed, *their adjoint versions are time-dependent*. Since these parameters do not have an associated prognostic equation, the way (16.59) and (16.60) are handled is different, as outlined below.

As before, equations (16.56)-(16.58) can be combined into a single equation:

$$\frac{\partial \hat{u}_i}{\partial t} = \frac{C_x}{2\Delta x}\left[\hat{u_{i+1}}^n - \hat{u_{i-1}}^n\right] - \mathrm{A}\hat{u_i}^n. \tag{16.61}$$

The $\leftarrow$ sign has been replaced since we're done building this equation.

16.3.8 Integrating the adjoint; the forecast aspect

To integrate the adjoint backwards in time, we first discretize the time derivatives using the same (two time level) temporal discretization scheme employed in the forward direction by the original nonlinear model (and the TLM) and then initialize the adjoint variables at the "final" time. As the model parameters are not governed by prognostic equations, their adjoints are simply summed across time. That is, (16.59) and (16.60) are replaced with

$$\hat{c}_{xi} = \hat{c}_{xi} - \frac{\Delta t}{2\Delta x}\left[U^n_{i+1} - U^n_{i-1}\right]\hat{u_i}^n \tag{16.62}$$

$$\hat{\alpha}_i = \hat{\alpha}_i - \Delta t U^{n+1}_i \hat{u_i}^n. \tag{16.63}$$

The old values of $\hat{c_x}$ and $\hat{\alpha}$ at each grid point are updated each time step.

Again, the details of the adjoint initialization at the final time depends upon the sensitivity one wishes to assess. No matter how J is formulated in the present example, ΔJ is constructed as

$$\Delta J = \sum_i \left[u''^n_i \frac{\partial \tilde{J}}{\partial \tilde{u}^n_i} + c''_x \frac{\partial \tilde{J}}{\partial \tilde{c}^n_x} + \alpha'' \frac{\partial \tilde{J}}{\partial \tilde{\alpha}^n} \right]. \tag{16.64}$$

Because the partial derivatives on the RHS are, in fact, the adjoint variables by definition, we can also write (16.64) as:

$$\Delta J = \sum_i \left[u''^n_i \hat{u}^n_i + c''_x \hat{c}^n_{x,i} + \alpha'' \hat{\alpha}^n{}_i. \right] \tag{16.65}$$

Since we've seen that ΔJ is trivial for a given specified J, it again follows that ΔJ provides nothing more than that (very useful) model check. *If this check is not needed, then the TLM need not be integrated at all.*

16.3.9 Using and coding the leapfrog scheme

Now we'll touch upon using the leapfrog (LF) scheme in the forward and reverse models. As a three time level scheme, its adjoint will be more complicated. For the forward model, it is presumed that the LF scheme is started off from time $t = 0$ with an "Euler" step with stepsize Δt. This forward time, center space scheme is unstable but is only used for this first step. Subsequent steps use the standard LF scheme and are of length $2\Delta t$, "leaping" from time levels $n - 1$ to $n + 1$. The final step generates the forecast for time step N.

For this example, we'll look at the TLM for just the advective part of (16.53), ignoring any parameter variation for simplicity. Also, we'll presume laterally periodic BCs are used and have been applied when and where necessary. Thus, the equation we start with is:

$$\frac{\partial u''}{\partial t} = -C_x \frac{\partial u''}{\partial x}, \tag{16.66}$$

where C_x is the control run's value for the advection speed parameter.

The initial condition (IC) for the forward model is placed in u''^0_i for any or all i. The first Euler step, forecasting u''^1_i, is coded as:

$$u''^1_i = u''^0_i - \frac{C_x \Delta t}{2\Delta x} \left[u''^0_{i+1} - u''^0_{i-1} \right]. \tag{16.67}$$

For time steps $n = 1$,N-1, the standard LF is coded as:

$$u''^{n+1}_i = u''^{n-1}_i - \frac{C_x 2\Delta t}{2\Delta x} \left[u''^n_{i+1} - u''^n_{i-1} \right]. \tag{16.68}$$

Time step $N-1$ concludes with the finished forecast representing time N.

To recap, the forward model code is structured as follows: First, the IC is specified. Then the forecast for time $n=1$ is made using the Euler scheme. Then subsequent LF forecasts are made. In my manner of coding, after each LF forecast we ready ourselves for the next time step by shifting the value from u''^{n}_{i} into u''^{n-1}_{i} and then the value of u''^{n+1}_{i} into u''^{n}_{i}. The code might look like this:

```
c arrays used:
c    hup(i) represents TLM u'' field at time n+1
c    hu (i) represents TLM u'' field at time n
c    hum(i) represents TLM u'' field at time n-1
c    ** for forward model we proceed hum -> hu -> hup
c    CX is the advection speed
c    DT is the time step
c    DX is the grid spacing
c we're starting at time=0
      time=0
c first we put the initial condition into UM, which represents time 0
      hum(i) = IC     ! for all i
c forecast time is updated
      time = time + dt
c first time step uses the Euler scheme (for all i)
      hu (i) = hum(i) - dt/(2*dx)*CX*(hum(i+1)-hum(i-1))  ! line A

c now integrate subsequent steps using the LF scheme.
c we integrate from time steps 1 to N-1,
c   making forecasts for times 2 to N
      do n=1,N-1,1
c forecast time is updated
       time = time + dt
c forecasts for time n+1 (stored in hup) are made (for all i)
       hup(i) = hum(i) - 2*dt/(2*dx)*CX*(hu(i+1)-hu(i-1))   ! line B
c now we set for new time step by remapping times (for all i)
       hum(i) = hu (i)      ! line C
       hu (i) = hup(i)      ! line D
       hup(i) = 0.          ! not really needed
c go on to the next time step
      enddo
```

The adjoint model operates in reverse. The initial sensitivity is defined at time level $n=N$. Time steps $n=N-1$ to 1, going in reverse, use the adjoint of (16.68). The model concludes by employing the adjoint of the forward model's Euler step, (16.67). Since the forward model

remaps the TLM time level arrays *after* each standard LF application, the adjoint model will start every reverse time step doing the adjoint of the remapping *first*. Everything is proceeding in reverse.

This brings us to the problem with using a three time level scheme like the LF for the adjoint model. Note that (16.68) takes information from the two time levels n and n-1 and uses it to forge the forecast at time $n+1$. This is illustrated in the upper panel of Fig. 2. The adjoint, proceeding in reverse, unravels this (as it were). The adjoint of (16.68) will take information at time level $n+1$ and push it back into to both times n and $n-1$. Put another way, the the $n+1$ time level *was influenced by* two prior times in the forward model, so in the reverse model the $n+1$ time *influences* two earlier times. See Fig. 2's lower panel.

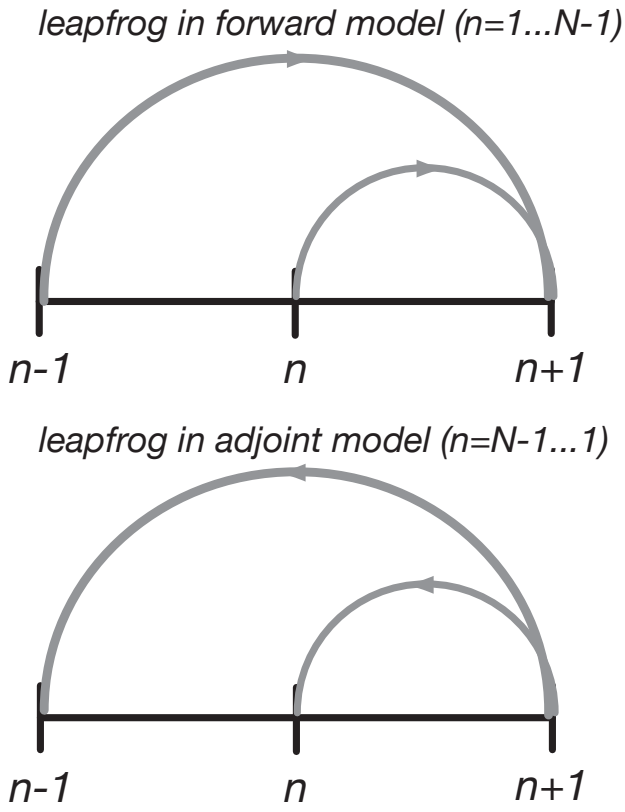

Figure 16.2: The leapfrog scheme in forward and reverse modes

Continuing on, the adjoint of (16.68) yields:

$$\hat{u}_i^{n-1} \leftarrow \hat{u}_i^{n+1} \tag{16.69}$$

$$\hat{u}_{i+1}^{n} \leftarrow -C_x \frac{2\Delta t}{2\Delta x} \hat{u}_i^{n+1} \tag{16.70}$$

$$\hat{u}_{i-1}^{n} \leftarrow +C_x \frac{2\Delta t}{2\Delta x} \hat{u}_i^{n+1}. \tag{16.71}$$

Note again that at this step, we're placing information into $\hat{u}$ at two different time levels. We follow this procedure for n=N-1...1 (going in reverse, of course), with the final application putting information into $\hat{u}_i^1$ and $\hat{u}_i^0$ for all i. The program will conclude by applying the adjoint of (16.67), which is:

$$\hat{u}_i^0 \leftarrow \hat{u}_i^1 \tag{16.72}$$

$$\hat{u}_{i+1}^0 \leftarrow -C_x \frac{\Delta t}{2\Delta x} \hat{u}_i^1 \tag{16.73}$$

$$\hat{u}_{i-1}^0 \leftarrow +C_x \frac{\Delta t}{2\Delta x} \hat{u}_i^1, \tag{16.74}$$

which finally combines all information into the final, concluding time level.

The coding example below utilizes the "line-by-line" strategy usually employed by automatic differentiators. These programs take a forward TLM model code line like

$$A'' = B'' + C''$$

and convert it into

$$\begin{aligned} \hat{B} &= \hat{B} + \hat{A} \\ \hat{C} &= \hat{C} + \hat{A} \\ \hat{A} &= 0. \end{aligned}$$

The adjoint IC is placed into $\hat{A}$ prior to running this code snippet. The ICs of $\hat{B}$ and $\hat{C}$ are zero.

```
c now the adjoint code starts
c arrays used:
c    dup(i) represents adjoint uhat field at time n+1
c    du (i) represents adjoint uhat field at time n
c    dum(i) represents adjoint uhat field at time n-1
c    ** for reverse model we proceed dup -> du -> dum

c the time counter should be set to the final time from the forward run
c we are presently at time step n = N, the final time
c put the initial sensitivity into the adjoint into DU, here
c   representing the adjoint model at final time N

      du(i) = IC  ! for all i

c we're ready to integrate backwards
```

```
      do n=N-1,1,-1   ! backwards from N-1 to 1

c update the forecast time
       time = time - dt

c for each standard LF step, the forward model set the new time step
c   mapping last, so for the adjoint we do it first -- for all i

       dup(i) = dup(i) + du(i)   ! these two lines are the adjoint
       du (i) = 0.               ! of line D from forward model

       du (i) = du(i) + dum(i)   ! these two lines are the adjoint
       dum(i) = 0.               ! of line C from forward model

c now we do the adjoint of the standard LF -- for all i
c the forward model MAKES time n+1 from values at times n and n-1
c therefore the reverse model uses time n+1 to PUT values into times
c   n and n-1

       dum( i ) = dum( i ) + dup(i)                ! these three lines
       du (i+1) = du (i+1) - 2*dt/(2*dx)*CX*dup(i) ! comprise adjoint of
       du (i-1) = du (i-1) + 2*dt/(2*dx)*CX*dup(i) ! forward model line B

c we're done for this reverse time step
      enddo

c the forward model did the Euler step first,
c  so the reverse model does it last
c note in reverse model, each time step put values into TWO time levels,
c the final time step finally ``stitches'' the two times back together

c update the forecast time -- we should be back to time = 0 now
      time = time - dt

c the adjoint of line A yields, for all i

      dum( i ) = dum( i ) + du ( i )
      dum(i+1) = dum(i+1) - dt/(2*dx)*CX*du ( i )
      dum(i-1) = dum(i-1) + dt/(2*dx)*CX*du ( i )

c and we're done, the adjoint sensitivity at the forward model's start
c  time is in array DUM

c end of model
```

The chief disadvantage of the leapfrog scheme is that at any intermediate time between

time levels N and 0 one needs to combine adjacent times to construct a complete adjoint sensitivity field. This is illustrated in Fig. 3 for the simple advection example we've been considering. The grid depicted shows time levels and grid points. The adjoint is initialized at time level N at a single grid point, designated i (i.e., $\hat{u}_i^N$). This is point indicated with the white dot and labeled "0". All other values at this time level contain zero values.

From (16.69)-(16.71), we see that as we step from time N to $N-1$, this IC causes information to be placed into three time/space locales: into $\hat{u}_{i+1}^{N-1}$ and $\hat{u}_{i-1}^{N-1}$, both at time level $N-1$, and into $\hat{u}_i^{N-2}$. Since this is the first adjoint operation, this is labeled "1". The light grey dots so labeled identify the appropriate positions on the figure grid.

Now we move to the next time step, $N-2$, in which (16.69)-(16.71) will be applied to each locale bearing the label "1". The time/space locales influenced in this operation are identified as the dark grey dots labeled "2". The time/space locale $(i, N-2)$ is being modified for the second time, and so it bears both labels. Then step $N-3$, representing the third operation, is applied to all locales bearing the "2" marking. The IC's influence has now spread to the time/space positions labeled "3".

The example ceases at this point, but two items must be addressed. First, note the staggering that is resulting from this operation. It is clear that in order to retrieve a full intermediate field, two adjacent time levels would have to be combined. Second, this temporal-spatial separation isn't finally cured or "congealed" until the very last LF operation, the Euler step. It is that operation that finally stitches the odd and even time level fields together. This illustrates a significant limitation that is not shared by adjoints employing two time level schemes.

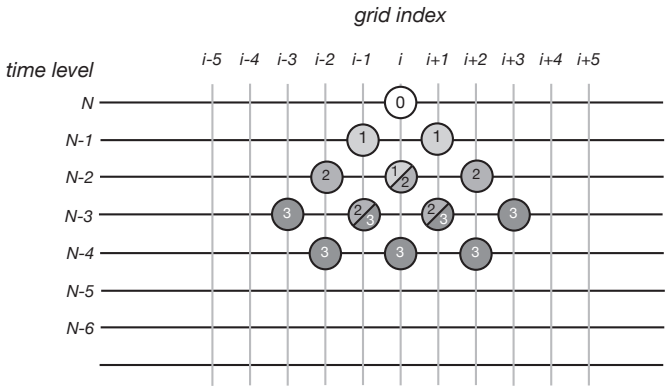

Figure 16.3: The adjoint leapfrog scheme, started with a single-point IC (point "0". Labels on other points show the steps that create and/or modify that particular time/space point.

Chapter 17

Example applications

In this chapter, we describe example applications of an adjoint model, specifically motivated by Fovell and Tan (2000, *QJRMS*; hereafter "FT00"). That study evaluated the performance of a simplfied, parameterized moisture model in simulating 2D squall-line type circulations. Their model was anelastic; herein, our forward model is based on the quasi-compressible equation set. Using the compressible equations makes the adjoint model construction less complicated.

The moisture parameterization obviates the explicit treatment of water substance and latent heating and cooling. In particular, condensation heating is made conditionally proportional to vertical velocity, and evaporation cooling is handled by a simple sponge-type term. Traditional "full physics" cloud models typically track at least three forms of water and possess microphysical interaction terms that are tricky to handle in tangent linear and adjoint models. The simple parameterized moisture (PM) framework greatly facilitates construction of the adjoint.

Naturally, the adjoint simulations will be of little to no value unless the PM model permits reasonably realistic simulations. Indeed, we have used the adjoint to track the source of deficiencies in the first-generation PM model and to suggest avenues of improvement.

17.1 The moisture parameterization

The PM framework (see Fig. 1) was described in detail in FT00. The chief effects of moisture are parameterized through the inclusion of terms representing vapor-condensation warming and liquid-water evaporation, labeled Q^+ and Q^-, respectively. These terms appear in the

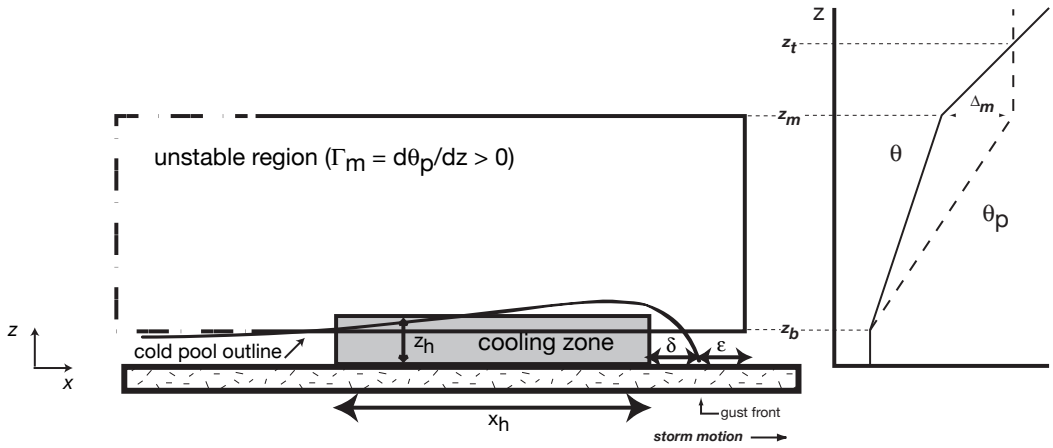

Figure 17.1: Schematic model illustrating PM model implementation.

model's perturbation potential temperature (θ') equation as

$$\frac{d\theta'}{dt} = Q^+ + Q^-. \tag{17.1}$$

This perturbation is defined relative to a temporally and horizontally invariant basic state.

The evaporation cooling term, Q^-, is treated as a shallow, lower-tropospheric cooling zone dimensioned $x_h \times z_h$ in which air is continuously relaxed to the preset potential temperature perturbation, θ'_c with a designated time scale τ_c. The sink's upstream edge is positioned a small distance δ behind the storm's gust front, subsequent to that feature becoming established. As per usual, the model domain is translated in such a way as to keep the gust front as stationary as possible, but in some cases domain-relative motion can still occur. The cooling zone's alignment relative to the front is examined every time step, and shifted if necessary.

In the PM model latent heat release is made proportional to ascent velocity ($w > 0$) by presuming air rises moist adiabatically within the "cloud". The slope of the moist adiabat in (θ, z) space is γ^*, a predetermined, nonnegative function of height alone, so Q^+ is handled as

$$Q^+ = \gamma^* \max(w, 0). \tag{17.2}$$

The model subdomain in which γ^* is nonzero is termed the "unstable region". As depicted in Fig. 1, this region resides between the specified height levels z_m and z_b. Further, the unstable region stretches laterally rearward to the domain's downstream boundary but its upstream extent is truncated a small distance ϵ ahead of the surface gust front's position.

FT00 reported this truncation had virtually no effect on their simulations. As in FT00, both δ and ϵ are taken to be 5 km.

17.2 Model implementation

17.2.1 Basic model design

The model domain is 550 km wide and 21.2 km deep. The horizontal and vertical grid spacings are $\Delta x = 1$ km and $\Delta z = 250$ m, respectively. In the quasi-compressible model, the sound speed is treated as a free parameter, and typically discounted somewhat to permit larger time steps. The sound wave speed is set at 100 m s^{-1}; simulations with larger values generated very similar results. The time step Δt is 0.5 s. Note that "time splitting", the approach that integrates acoustically active and inactive terms with different time steps, was not adopted. We have chosen straightforward over efficient design at this time.

The model grid is staggered, using the Arakawa "C" grid. See Fig. 2 for the specific grid arrangement and indexing convention employed. The upper and lower boundaries are rigid, free-slip plates; the horizontal domain is periodic. Using periodic boundaries simplifies adjoint construction but is not realistic for the kind of circulations under examination. To compensate for this, the domain has been made sufficiently wide to make the boundary assumptions unimportant over the lengths of our integrations.

The modeling system has three components: the nonlinear and tangent linear models, both integrated forward in time, and the adjoint model, operated in reverse (see next section). Two independently implemented versions of the system have been created, one employing the leapfrog method and the other using the Euler-backward scheme. Both schemes incorporate second order centered differencing for spatial derivatives. The leapfrog approach is fast and efficient but its three time level structure complicates implementation and fidelity assessment of the adjoint model, as well as interpretation of that model's intermediate results. These are the strengths of the two-time level Euler-backward version. The Euler-backward scheme, however, requires roughly twice the number of computations of the leapfrog approach and also tends to damp high frequency signals[1]. Because the model is compressible and not time split, however, the modes of interest are actually quite slowly varying. Even for prolonged integrations, the two versions yield virtually indistinguishable results, for both forward and reverse integrations.

[1]Haltiner and Williams (1980), p. 134-35.

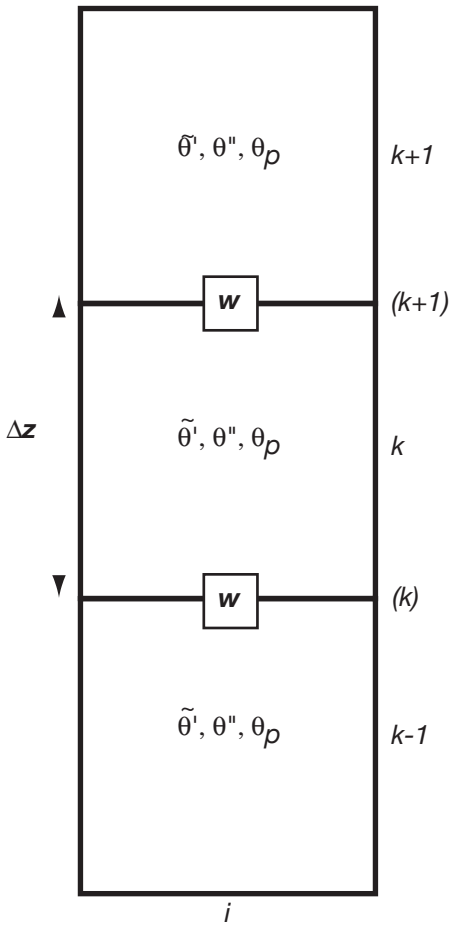

Figure 17.2: Staggered grid employed by the forward and adjoint models.

For the moisture parameterization, the moist adiabatic slope γ_* is defined at the w locations at the top and bottom of each grid box. In the nonlinear forward model, presuming a two time level scheme is employed, (17.2) is implemented at time n and spatial location i, k as:

$$Q^+_{i,k} = \frac{\Delta t}{2} \left[\max(w^n_{i,k+1}, 0)\gamma^*_{k+1} + \max(w^n_{i,k}, 0)\gamma^*_k \right]. \tag{17.3}$$

That is, the parameterized heating is calculated at the w locations and then averaged to the grid box center where θ' resides. Parameterized evaporation cooling is handled as

$$Q^-_{i,k} = -\left(\theta'^n_{i,k} - \theta'_c\right) \tau_c^{-1} \tag{17.4}$$

within the spatially confined cooling zone.

17.3 Recapitulation

The full nonlinear model may be employed to make two distinct simulations, termed the "control" and "alternative" runs. These may have have started with different initial conditions (ICs) and/or parameter settings. The tangent linear model (TLM) is a modified version of the fully nonlinear model that attempts to prognose and track the discrepancies between the two nonlinear model runs. This model is obtained by taking the full model's code and linearizing it about a state provided by the control simulation using truncated Taylor series. Thus, an approximation to the alternative run is obtained by combining the control and TLM solutions. The accuracy of the approximation depends upon the importance of the missing terms.

The adjoint model is subsequently obtained by transposing this TLM. In our work, the TLM represents the intermediate step in adjoint model construction, and is mainly used to verify that the adjoint is coded properly. Symbolically, $w^n_{i,k}$, $\tilde{w}^n_{i,k}$, $w''^n_{i,k}$ and $\hat{w}^n_{i,k}$ will represent w variables from the alternative, control, TLM and adjoint solutions, respectively. In this case, the variable identified is the vertical velocity at time n and spatial location i, k.

17.3.1 Implementing the parameterization in the TLM

The PM model's discretized latent heating term (17.3) is a simple function of vertical velocity

$$Q^+_{i,k} = F_+ \left(w^n_{i,k+1}, w^n_{i,k}\right)$$

making its TLM implementation quite simple. The TLM term is

$$Q''^{+}_{i,k} = w''^{n}_{i,k+1}\frac{\partial F_+}{\partial w^n_{i,k+1}}|_C + w''^{n}_{i,k}\frac{\partial F_+}{\partial w^n_{i,k}}|_C \tag{17.5}$$

$$= \frac{\Delta t}{2}\left[\tilde{\beta}^n_{i,k+1} w''^{n}_{i,k+1}\gamma^*_{k+1} + \tilde{\beta}^n_{i,k} w''^{n}_{i,k}\gamma^*_k\right], \tag{17.6}$$

presuming that grid location (i,k) resides in the spatially truncated unstable zone. In the first expression, the subscript C indicates the term is evaluated for the control run. In the second, the value of the indicator variable β depends upon whether ascent was present in the control run at the given point and time:

$$\tilde{\beta}^n_{i,k} = \begin{cases} 1 & \tilde{w}^n_{i,k} > 0; \\ 0 & otherwise. \end{cases}$$

The TLM version of (17.4) is simply

$$Q''^{-}_{i,k} = -\theta''^{n}_{i,k}\tau_c^{-1}, \tag{17.7}$$

presuming grid point (i,k) resides in the cooling zone[2]. The position of the cooling zone may shift with time if the control run's specified domain translation speed fails keep the gust front stationary. At a given time, this position is forced to be the same in the control, TLM and adjoint models.

17.3.2 Adjoint implementations of the parameterization

Now we turn to the adjoint formulation. Starting with (17.6) and (17.7) the adjoint was constructed by hand transposition in the following manner: First, the salient parts of the TLM prognostic equation for θ'', using a two time level scheme and in coded form, is:

$$\theta''^{n+1}_{i,k} = \theta''^{n}_{i,k} + \frac{\Delta t}{2}\left[\tilde{\beta}^n_{k+1} w''^{n}_{i,k}\gamma^*_{k+1} + \tilde{\beta}^n_{k} w''^{n}_{i,k}\gamma^*_k\right] - \Delta t\tau_c^{-1}\theta''^{n}_{i,k} \tag{17.8}$$

Next, the TLM variables are replaced with their adjoint counterparts (designated here by hats). This equation spawns the following three lines of code

$$\hat{w}^n_{i,k+1} \leftarrow \frac{\Delta t}{2}\tilde{\beta}^n_{i,k+1}\hat{\theta}^{n+1}_{i,k}\gamma^*_{k+1}$$

$$\hat{w}^n_{i,k} \leftarrow \frac{\Delta t}{2}\tilde{\beta}^n_{i,k}\hat{\theta}^{n+1}_{i,k}\gamma^*_{k}$$

$$\hat{\theta}^n_{i,k} \leftarrow -\Delta t\tau_c^{-1}\hat{\theta}^{n+1}_{i,k},$$

[2]If the leapfrog scheme is employed, the θ' and θ'' values used must represent the past time step, $n-1$, for stability.

where β retains its previous meaning. Since the adjoint model is being integrated backward in time, time level $n+1$ "preceeds" time level n. As in the forward model, these terms affect the adjoint fields only in their respective, designated subdomains.

17.3.3 Verification of the TLM and adjoint

The control/TLM combination may fail to accurately reproduce the alternative run not only owing to the TLM's inherent approximations but also due to design and coding errors. Effort was made to make sure the TLM was free of the latter error source. Then the verified TLM model was used to vet the correctness of the adjoint model's coding.

In the course of TLM model validation, the strictly adiabatic discretized model was examined first. In this configuration, inherent TLM error arises solely due to the neglect of terms beyond first order in the Taylor expansions. This truncation, however, affects only the advection terms, and the missing terms themselves are no higher than second order. Indeed, they are nothing other than easily identified TLM perturbation products.

We tested the adiabatic TLM model code by installing and temporarily reenabling these missing terms. The output of this "augmented" model should track the discrepancies between the control and alternative simulations to within roundoff error. This was found to be the case, thereby validating the adiabatic TLM code itself. Naturally, the perturbation product terms have to be excluded from the TLM since they cannot be written in matrix form, and thus are not transposable for use in the adjoint model.

The PM model's diabatic terms are first order but note (17.2) is not differentiable at $w = 0$. Thus, there is error beyond simple roundoff wherever and whenever the control and alternative simulation vertical velocities have different signs (as might happen if the updraft boundary were slightly shifted between two nonlinear simulations, for example). A formulation for (17.6) that is exact (to within roundoff error) is

$$\begin{aligned} Q''^{+}_{i,k} &= \frac{\Delta t}{2}\left[\langle \max(\tilde{w}^n_{i,k+1} + w''^n_{i,k+1}, 0) - \max(\tilde{w}^n_{i,k+1}, 0)\rangle \gamma^*_{k+1}\right. \\ &\quad \left. + \langle \max(\tilde{w}^n_{i,k} + w''^n_{i,k}, 0) - \max(\tilde{w}^n_{i,k}, 0)\rangle \gamma^*_k\right] \end{aligned} \tag{17.9}$$

but this is also not transposable. However, this formulation was used temporarily as a checking and assessment tool.

For adjoint simulations in general, a control simulation must first be made and archived. Those data are read back in reverse order during the adjoint integration. The adjoint in-

tegration commences with the definition of a forecast aspect J that represents something about the control run that one wishes to investigate further. For every prognostic field and parameter in the forward model there is a corresponding adjoint variable assessing its influence upon J. Thus, the adjoint variable $\hat{w}$ represents $\frac{\partial J}{\partial w}|_C$, etc..

For adjoint validation runs, TLM forecasts are made and archived as well. These data are used to calculate ΔJ, the sum of the products of the TLM perturbation and adjoint variables accumulated for all fields and perturbed parameters over all grid points, at each time step during the reverse integration. When a two time level scheme is employed, the ΔJ values so calculated should be independent of time (e.g., Talagrand and Courtier 1987). In our tests with periodic lateral boundaries, the Euler backward version of the model can preserve ΔJ to seventeen digits when double precision is used for all reals.

It was already noted that the leapfrog version generates simulations that are almost indistinguishable from those issuing from the Euler backward model, for both forward and reverse modes. For the leapfrog adjoint, however, inspection of intermediate results, for times falling between the initial and final times, requires that adjacent time values be combined to yield "whole" fields. This is a consequence of the leapfrog's three time-level structure.

17.4 Control model simulation

We now examine a simulation made using the FT00 low CAPE sounding graphically depicted on Fig. 1. The sounding, which was drawn from an earlier study by Garner and Thorpe (1992, *QJRMS*), is divided into three layers distinguished by stability. The figure also illustrates the path taken by a parcel rising undiluted from the low-level mixed layer. This parcel's level of free convection is at height z_b, taken to be 1 km. The maximum buoyancy of this parcel (Δ_m= 3K) is realized at height level z_m, set at 7 km. Above this point, we presume the parcel has run out of vapor to condense. Its equilibrium level would reside at z_t = 9 km. The CAPE for this hypothetical parcel is 400 J kg^{-1}.

The initial horizontal wind profile (not shown) consisted of 9.75 m s^{-1} of wind speed change over the lowest 3 km, with zero shear farther aloft. In the shear layer, the vertical shear vector points eastward, so we will refer to the west and east sides of the storm as the "upshear" and "downshear" directions, respectively.

The cooling zone was 1.5 km deep and 45 km wide, with θ'_c=-4.5K and τ_c = 600 s. As already noted, the model domain is translated in an attempt to keep the cooling zone and

the cold pool it generates as stationary relative to the domain as possible. For this particular simulation, the domain speed was 11.5 m s^{-1} in the eastward (downshear) direction. As this translation rendered the model storm effectively stationary, this value also represents the storm speed.

The model was run for 9000 s, more than enough time for the storm to develop and attain a statistically steady structure, at least in the vicinity of the convection. Figures 3 and 4 present control model fields spanning most of this integration period. The former depicts vertical velocity w along with perturbation potential temperature θ'. Figure 4 depicts storm-relative horizontal velocity u and perturbation pressure p'. (Though these fields are from the control run, note we've dropped the tildes from these symbols.)

It is seen that the cold pool is already well formed by 2000 s (Fig. 3a). Lifting of air at the pool's downshear edge ($x \approx 270$ km) has provoked a small amount of parameterized heat release by this time. By 6000 s, the storm circulation evinces a mature organization which has become statistically steady in the vicinity of the principal storm updraft. That feature leans upshear with height, but is compact and relatively strong (Fig. 4b). Lower tropospheric flow is accelerated upward and rearward (upshear) over the cold pool, whereupon the flow splits into two principal branches, the front-to-rear (FTR) flow which continues rearward, and the forward anvil outflow which overturns and spreads upstream of the storm. Beneath the FTR flow, the rear inflow current is established, residing largely in the lower troposphere above the spreading cold pool.

The environment responds to the initiation of convective heating by generating compensating subsidence waves which propagate as gravity waves away from the main convecting region in both directions. We will henceforth focus on the eastward-propagating wave which moves through the storm's upstream environment. The leading edge of this wave is marked by low surface pressure (Fig. 4). Between this leading edge and the storm updraft, the horizontal airflow has been accelerated towards (away) from the convection in the lower (upper) troposphere. Uninterrupted positive buoyancy spans this region, concentrated in the middle troposphere. This is a combination of *in situ* subsidence-induced warming and convectively generated warming advected by the forward anvil outflow.

The downshear-progressing wave's propagation speed is about 10 m s^{-1} relative to the domain, making for a 21.5 m s^{-1} ground-relative motion. Following Nicholls et al. (1991), we see this phase speed's dependence upon H, the vertical dimension of the diabatic source it is responding to, in the expression

$$c = \bar{u} + \frac{NH}{\pi}, \tag{17.10}$$

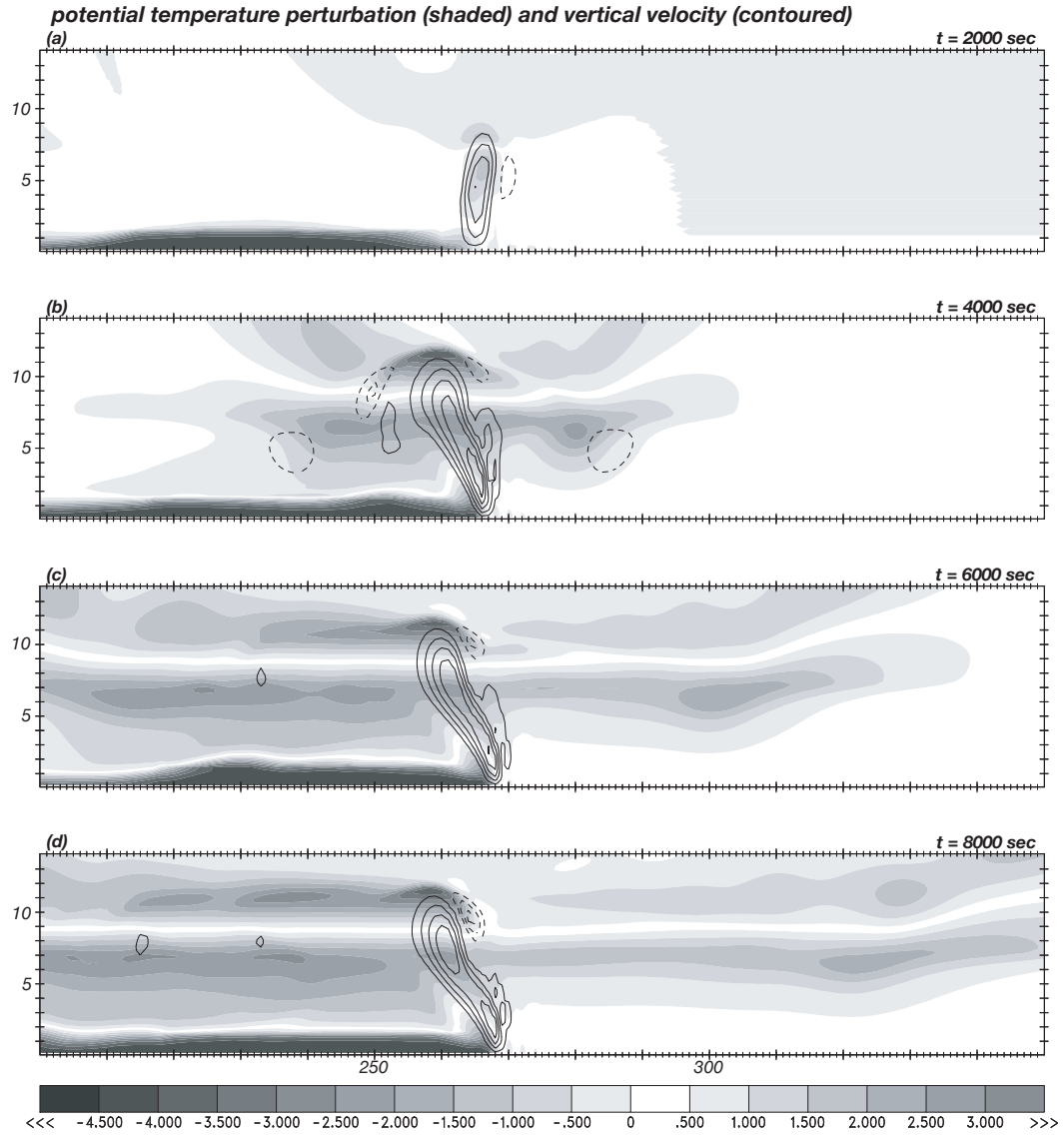

Figure 17.3: Potential temperature perturbation (shaded) and vertical velocity (contoured; interval 2 m s^{-1}) for the control run. Only a portion of the domain is shown.

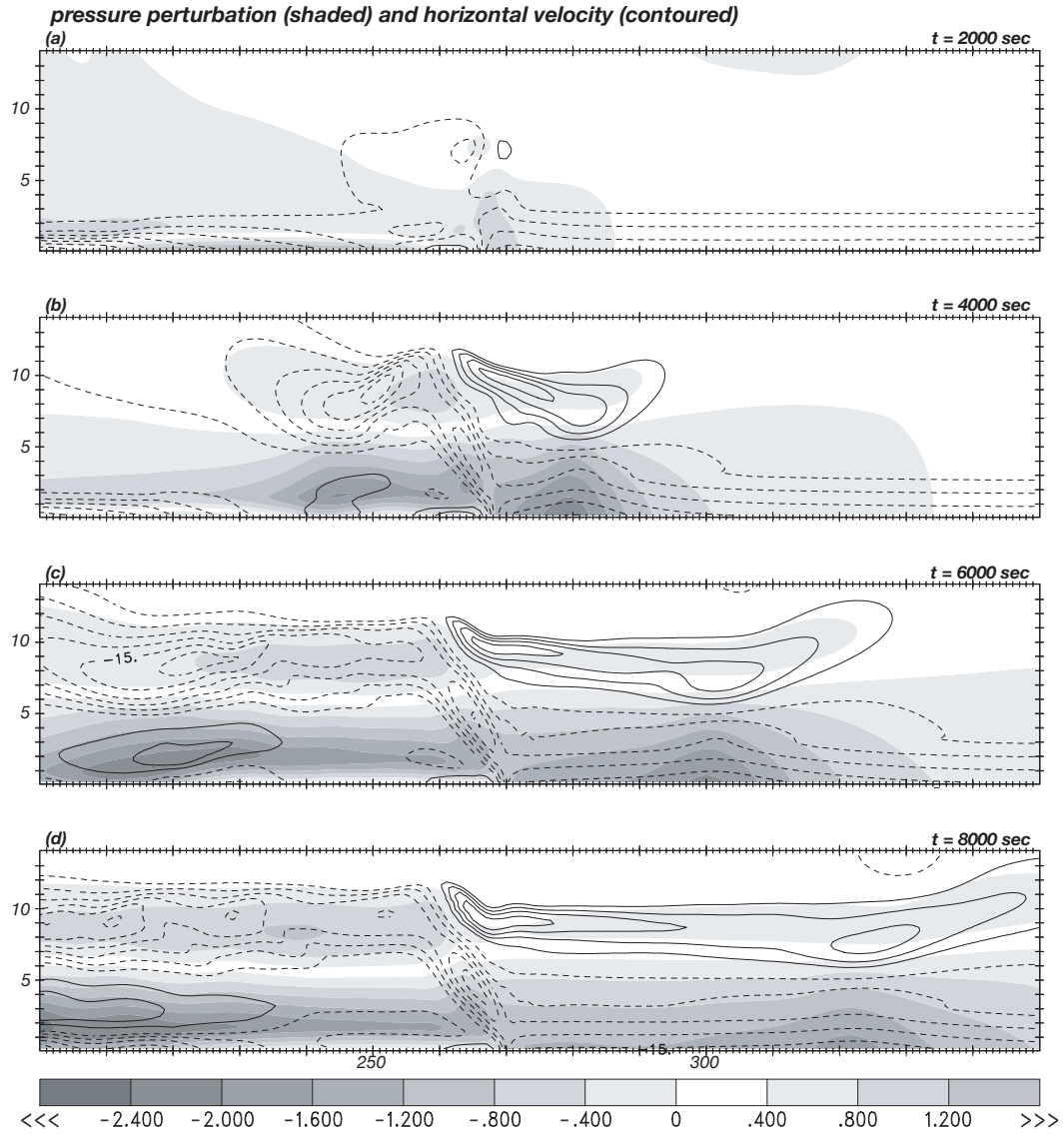

Figure 17.4: As in Fig. 3, but for perturbation pressure (shaded) and storm-relative horizontal velocity (contoured; interval 3 m s^{-1}).

where $\bar{u}$ is the mean wind (9.75 m s^{-1} above the shear layer) and N is the Brunt-Väisälä frequency (.004 in the intermediate stability layer). The expected $c = 21.5$ m s^{-1} results when H is taken to be 9 km. This H is the depth of the layer in which diabatically generated warming and cooling resides, in and beneath the main storm updraft (see Fig. 3).

17.5 Examination of forward anvil outflow strength with the adjoint model

The adjoint model has been used to examine the dynamical precursors of the enhanced upper tropospheric forward anvil outflow on the storm's downshear side. The question asked here is which fields and locations at earlier times had the most influence on bringing about and/or determining the strength of the westerly forward anvil outflow at 6000 sec? This question is addressed by a backward integration of the adjoint model, with a forecast aspect J defined as being the u field concentrated in the subsidence wave's leading edge. Figure 5, which provides a close-up view of the control run at 6000 sec, shows the aspect region is embedded within the westerly forward anvil outflow, centered at a local westerly wind maximum.

As J is a function of u alone, the adjoint started off with non-zero sensitivity only in $\hat{u}$, the horizontal velocity variable's adjoint. The initial $\hat{u}$ field in the aspect region was given a maximum value of 1 and was smoothly tapered to zero over a region having horizontal and vertical radii of 3000 amd 800 m, respectively, as depicted in Fig. 5. Using a smoothed initial condition helps reduce noise in the ensuing backward simulation.

In this case, a positive ΔJ value would represent a further enhancement of the westerly forward anvil outflow already present in the aspect region. Recall that ΔJ represents the product of adjoint sensitivities and control run perturbations, summed over all model fields, parameters and grid points. If at a given time the adjoint predicts positive sensitivity for some field at some location, this suggests that placing a positive perturbation with respect to the control run field there and then should result in subsequent enhancement of the forecast aspect metric. A positive enhancement can also be obtained by applying a negative perturbation to a field and location evincing negative sensitivity. In contrast, sensitivities and control run perturbations with opposite signs would tend to decrease the westerly flow the forecast aspect represents.

As an example, say the adjoint predicts positive sensitivity to the temperature perturbation θ' at grid point i, k. Let's further assume the control run had warming ($\theta'_{i,k} > 0$ there. The

adjoint is predicting that if one augmented the warming already present (i.e., specified a positive $\theta''_{i,k}$), this would lead to a larger value of J later. If the control run had cooling at that location instead, then *decreasing* the negative anomaly at that time would lead to an subsequent increase in J.

17.5.1 Adiabatic adjoint run

Two backwards runs were made with the forward anvil J, excluding and including the adjoint's diabatic terms. Both were integrated for 3500 sec. Excluding the moisture parameterization terms renders the adjoint model strictly adiabatic. This simulation will be examined first.

Figures 6 and 7 depict adjoint sensitivity fields (contoured) superimposed upon control run fields (shaded) at 4000 sec, 2000 sec into this backward integration[3]. It could be anticipated that since the forward anvil outflow and subsidence wave were thermally driven that by and large the sensitivity should become concentrated in the temperature field and backtrack towards the actively convecting region as time rewinds. Although the sensitivity was initialized in the $\hat{u}$ field, it was indeed very rapidly transferred into the temperature adjoint field. Please note that the contour intervals employed for $\hat{u}$, $\hat{p}$ and $\hat{w}$ in Figs. 6 and 7 are respectively 7, 25 and 50 times smaller than the interval employed for $\hat{\theta}$.

As expected, the bulk of the physically relevant sensitivity has stayed with the retrograding subsidence wave. The temperature adjoint field (Fig. 6a) concentrates positive sensitivity in the warmed air just behind its leading edge along with negative values located immediately above. The former says that one way to make the aspect region's westerly anvil outflow stronger at 6000 sec is to intensify the warm anomaly located above $x \approx 280$ km at 4000 sec. The latter indicates that the subsequent outflow could also be enhanced by cooling the neutrally buoyant air located above the warm anomaly.

These operations are sensible, both separately and jointly. Intensifying the already warm air at the subsidence wave's leading edge would make the horizontal buoyancy gradient across that boundary larger, increasing the local generation of positive horizontal vorticity. As depicted qualitatively on the figure, the circulatory tendency associated with this vorticity generation would tend to encourage ascent in the warm anomaly itself as well as westerly flow above and to the east of it. Cooling the air above the warm anomaly would have the

[3]Recall that an adjoint variable $\hat{\phi}$ represents $\frac{\partial J}{\partial \phi}$. As J is dimensioned m s^{-1}, then temperature adjoint $\hat{\theta}$ has units m s^{-1} K^{-1}, for example.

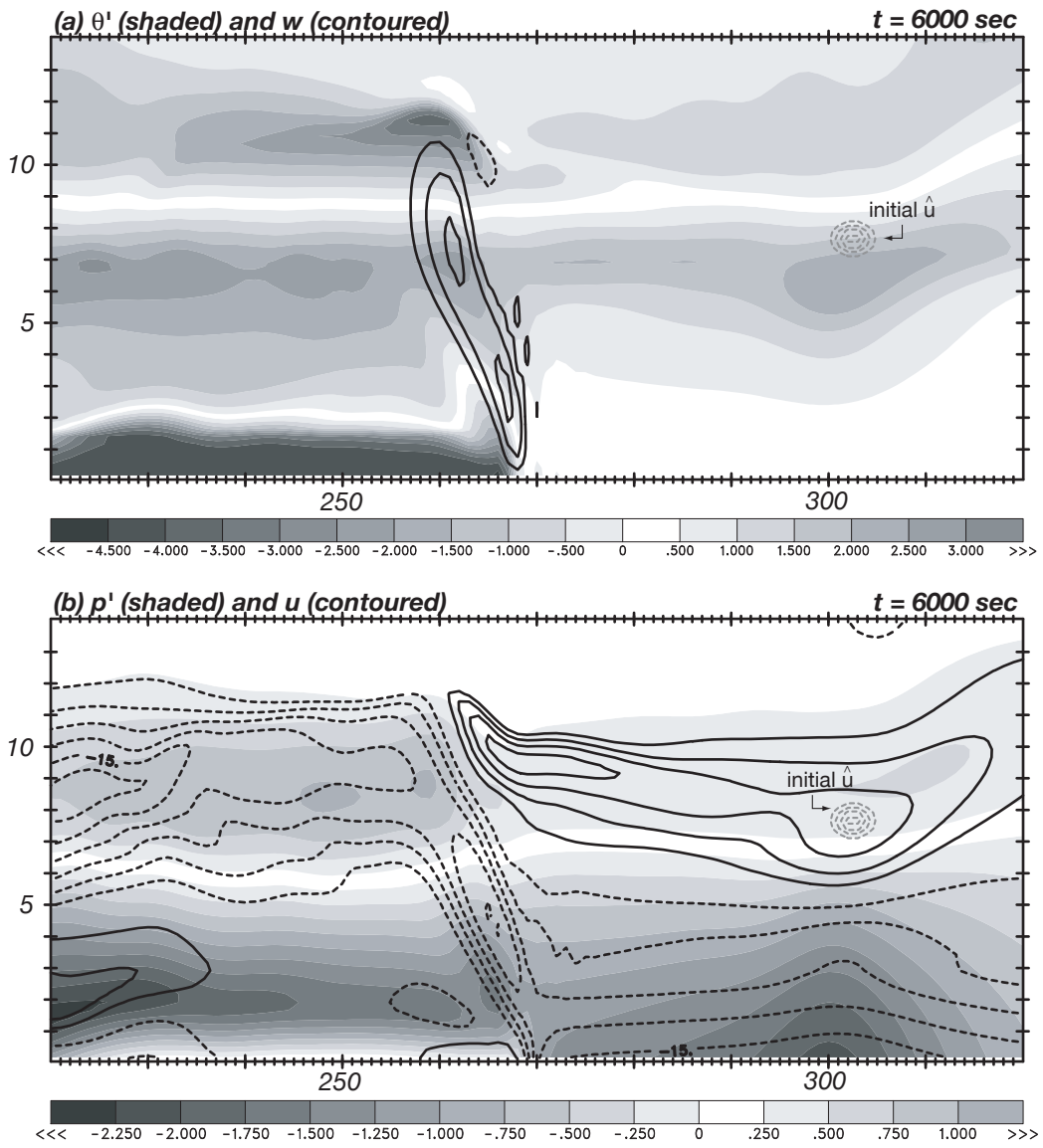

Figure 17.5: Close-up view of the control run at 6000 sec, showing location of initial adjoint $\hat{u}$ sensitivity.

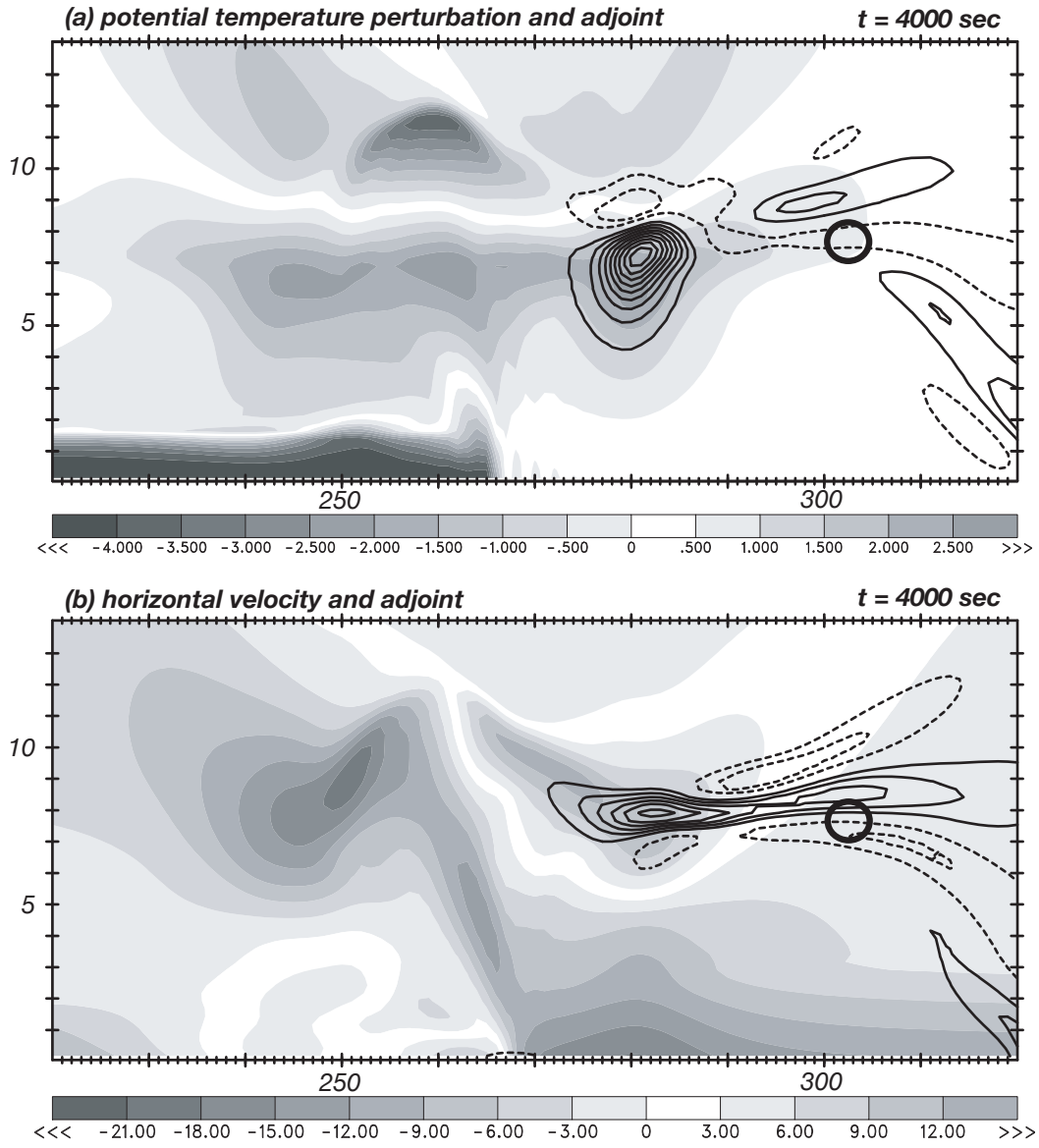

Figure 17.6: Control (shaded) and adjoint sensitivity (contoured) fields at 4000 sec for the subdomain depicted in Fig. 5. (a) potential temperature field (contour interval .05 m s^{-1} K^{-1}); (b) horizontal velocity field (contour interval .0075, nondimensional). Shown in panel (a) is the circulatory tendency implied by the adjoint sensitivity field.

complementary effect of encouraging subsidence within the anomaly and westerly flow below and to the east of its center. Both of these westerly enhancements at the present time and place would reasonably lead to subsequent enhanced westerly flow in the aspect region farther downstream.

Furthermore, note that the largest positive sensitivity resides about 1 km above the location of the warmest air in the control run at at 4000 sec. The adjoint model is indicating that shifting the present warm and cold anomalies closer together vertically – displacing the cooled (warmed) locale downward (upward) – would also serve to enhance the subsequent westerly outflow. This should indeed result because the buoyancy-induced outflows in this case would be concentrated into a still narrower layer, and would thus be intensified.

In the horizontal velocity field (Fig. 6b), principal center of positive sensitivity is seen to be located in the forward anvil outflow at the 8 km level above $x \approx 280$ km, where u values in the control run are positive (westerly). The adjoint model is saying that if we increase the westerly winds at the anvil's leading edge at 4000 sec, this will result in stronger westerlies at the forecast aspect region farther downstream 2000 sec later. This seems obvious.

For vertical velocity (Fig. 7a), the adjoint model is again indicating that increasing the strength of the subsidence wave at the present time would lead to stronger westerly outflow in the aspect region later. In the range $270 < x < 290$ km, there is positive (negative) sensitivity attached to areas presently experiencing upward (downward) motions. Comparison with Fig. 6a shows these adjoint sensitivities are in quadrature with the $\hat{\theta}$ field, and consistent with eastward gravity wave phase propagation. Note some sensitivity has already reached into the main convecting region. There are other features in the sensitivity field as well, but keep in mind the overall magnitudes of $\hat{w}$ are extremely small.

The principal signature in the $\hat{p}$ field (Fig. 7b) is negative sensitivity in the middle troposphere, above the present surface low center. Perhaps lowering the pressure in the indicated region would increase the horizontal pressure gradient force exerted on parcels located west of it. This would accelerate the westerly flow in the forward outflow behind the subsidence wave's leading edge at the time depicted. Another interpretation has the adjoint model suggesting that a deepening of the already present low pressure anomaly now would intensify the anvil outflow later. As the storm's total depth is constrained by the stable stratosphere, this might encourage the westerly flow to be squeezed into a shallower layer and thereby cause its intensification.

Figures 8 and 9 present the control run and adjoint sensitivity fields at 3000 sec. Again, the

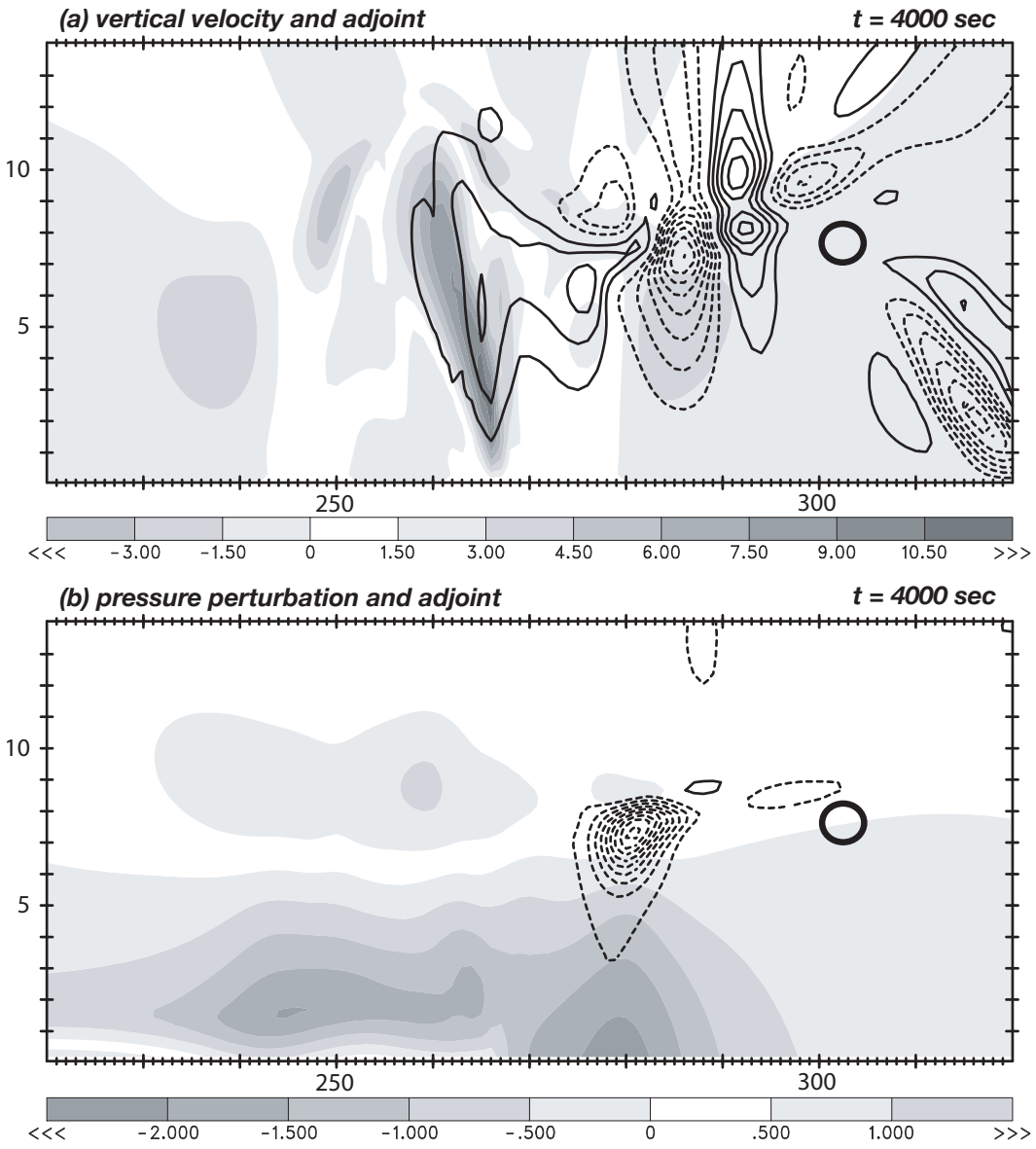

Figure 17.7: As in Fig. 6, but for: (a) vertical velocity field (contour interval .001, nondimensional); (b) perturbation pressure field (contour interval .002 m s^{-1} mb^{-1}).

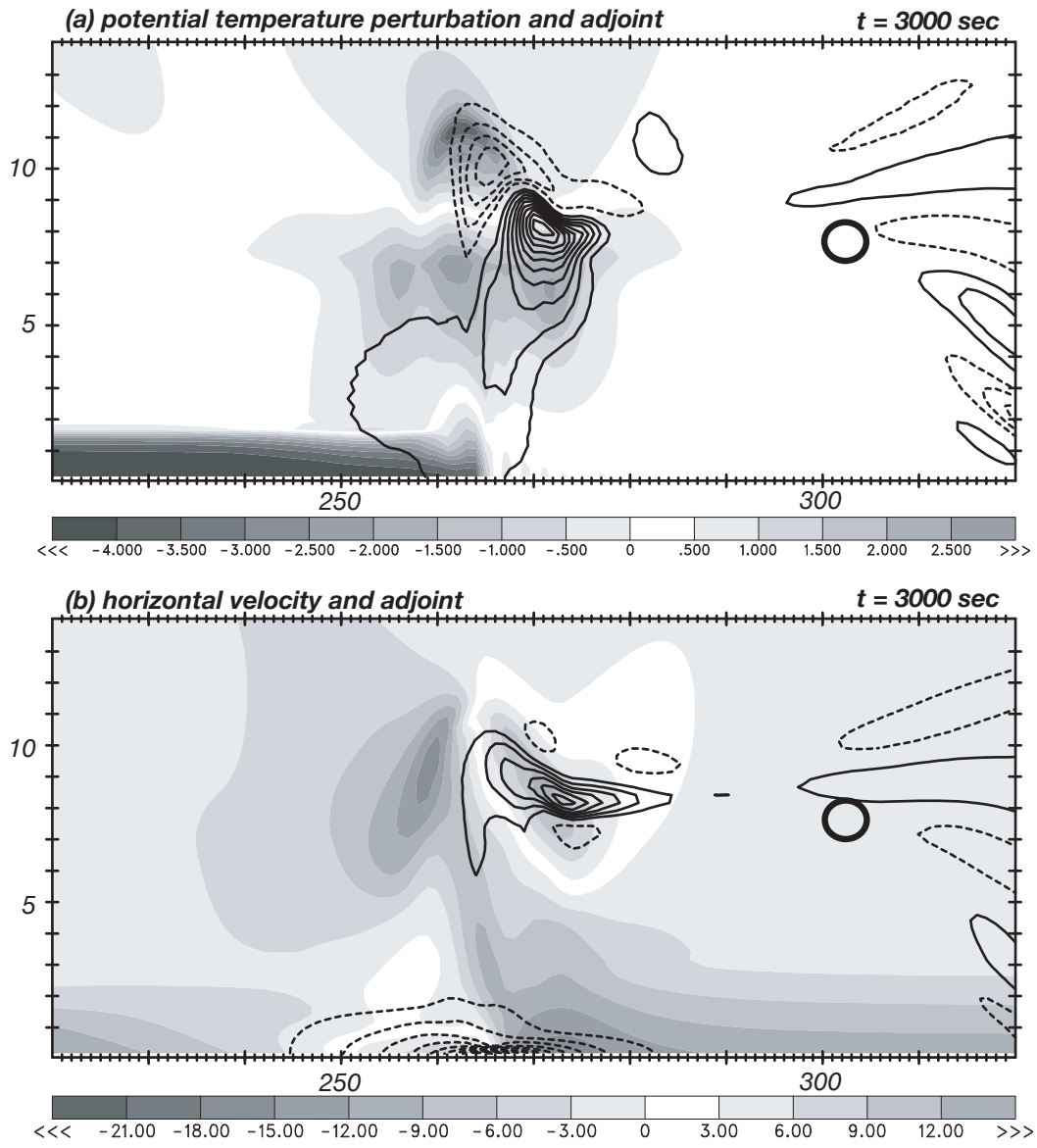

Figure 17.8: As in Fig. 6, but at 3000 sec. Temperature adjoint contour interval is .025 m s^{-1} K^{-1}; for the horizontal velocity adjoint it is .005 (nondimensional).

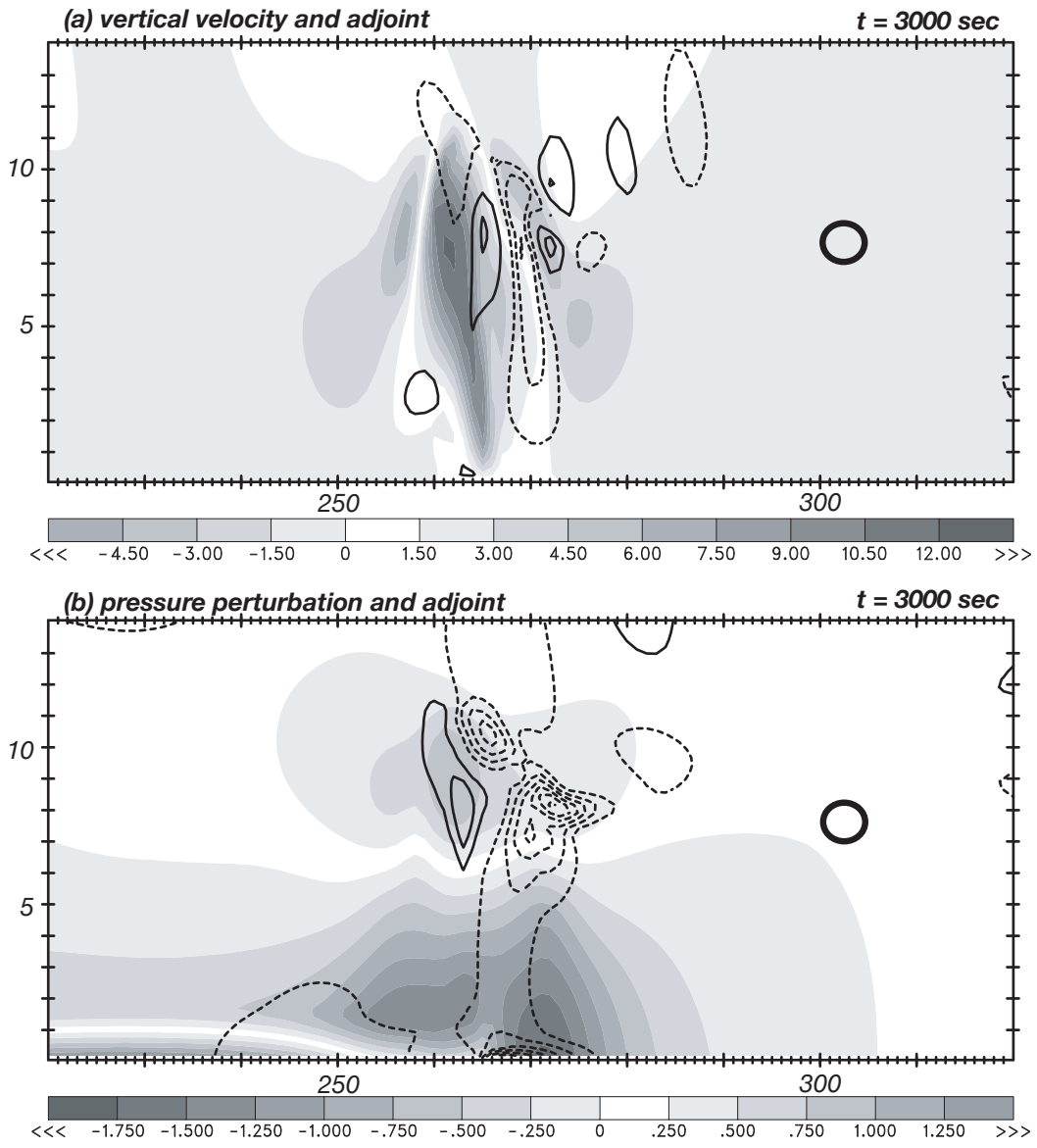

Figure 17.9: As in Fig. 7, but at 3000 sec. Vertical velocity adjoint contour interval is .002 (nondimensional); for the perturbation pressure adjoint it is .001 m s^{-1} mb^{-1})

largest sensitivities of all are in the temperature field (Fig. 8a), and are especially concentrated in the warming near the subsidence wave's present location. Also notable is the significant negative temperature sensitivity located farther aloft. The cooling that was present there in the control run resulted from parcels in the storm's main updraft (compare with Fig. 9a) overshooting their level of neutral buoyancy. Note that the forward anvil outflow issues from, and is strongest within, the area between these sensitivity centers (Fig. 8b). The adjoint model is again saying that anything that would serve to make these marked temperature anomalies stronger and closer together now would result in enhanced westerly flow in the aspect region later.

In addition to the expected positive $\hat{u}$ sensitivity in the forward anvil, an area of substantial negative sensitivity has appeared in the lower troposphere (Fig. 8b), straddling the gust front located at $x \approx 265$ km. Across this zone, the storm-relative control run flow switches from weak westerly behind the gust front to easterly ahead of it. The adjoint model is predicting that the subsequent forward anvil outflow would be intensified if that easterly flow were further enhanced (i.e., $u'' < 0$). Weakening the westerlies behind the front are expected to help, for reasons that are less immediately clear.

The $\hat{w}$ sensitivity in the main storm updraft (Fig. 9a) seems to be trying to shorten but also expand that feature laterally eastward. Either might help shift the cooling resulting from overshooting towards the east. The negative sensitivity above $x = 270$ km may be attempting to stretch the subsidence wave's downdraft westward. Taken together, these alterations would sharpen the horizontal w gradients across the main storm updraft. The larger horizontal vorticity this would represent would also be consistent with stronger westerly flow in the anvil region.

The pressure adjoint field (Fig. 9b) suggests a similar interpretation now as at 4000 sec. One cautionary note needs to be sounded, however. Significant sensitivity has now reached the convecting region, rendering the adiabatic restriction questionable. The diabatic adjoint run is examined next.

17.5.2 Diabatic adjoint run

Figures 10 and 11 present the sensitivity fields obtained at 3000 sec when the adjoint's diabatic terms were enabled. Inclusion of the cooling zone term in the backwards model had but a very small effect on the results. The parameterized warming term, active within the region indicated on Fig. 10a, exerted a much larger impact.

Outside of the unstable region, the results for potential temperature adjoint (Fig. 10a) qualitatively resemble those from the adiabatic simulation. Enhancing the already present warm and cold anomalies located above the unstable region would again subsequently drive stronger westerly anvil outflow. The diabatic model, however, puts the largest sensitivity on the east side of the main storm updraft, just inside the unstable region. This warming is colocated with positive $\hat{w}$ sensitivity (Fig. 11a), suggesting that enhancing the updraft here (resulting in diabatic heating) in the forward model would help increase J. This appears tantamount to increasing the width of the main storm updraft.

Taken together, the $\hat{u}$ and $\hat{w}$ adjoints suggest inducing a clockwise circulation anchored around $z = 6$ km above $x = 271$ km. This circulation would indeed encourage westerly flow into the forward anvil. As noted above, the adjoint's diabatic heating term drives the ascending branch of the circulation. Beyond the unstable region's east edge, the $\hat{w}$ and $\hat{\theta}$ sensitivities are in quadrature, likely suggesting eastward gravity wave propagation. The point of largest negative pressure sensitivity resides near the center of this circulation.

Overall, the importance of the diabatic terms in the adjoint becomes clear. Since the westerly outflow is associated with the subsidence wave, anything that acts to strengthen the latter encourages the former. As the wave itself was triggered by the convection, wave intensification proceeds from convective enhancement. This means more storm updraft and more diabatic heating within it. The adjoint model's suggestion of a wider storm updraft perhaps stands in part for an increase in the vertical mass flux. As more mass is transported upward, more may be directed into the forward anvil, leading to a strengthening of the upper tropospheric westerlies so long as the thickness of the outflow layer is not increased. Recall the sensitivities above and beyond the unstable region appear to be encouraging a narrowing of that layer.

17.6 Examination of the lower tropospheric storm inflow with the adjoint model

17.6.1 Background

FT00's study was primarily concerned with assessing the realism of the PM model, as revealed through comparisons with traditional, or full physics, cloud model simulations. They judged the PM model storms' tendency to substantially and permanently enhance the upstream lower tropospheric inflow as unrealistic. This alteration, which commenced during

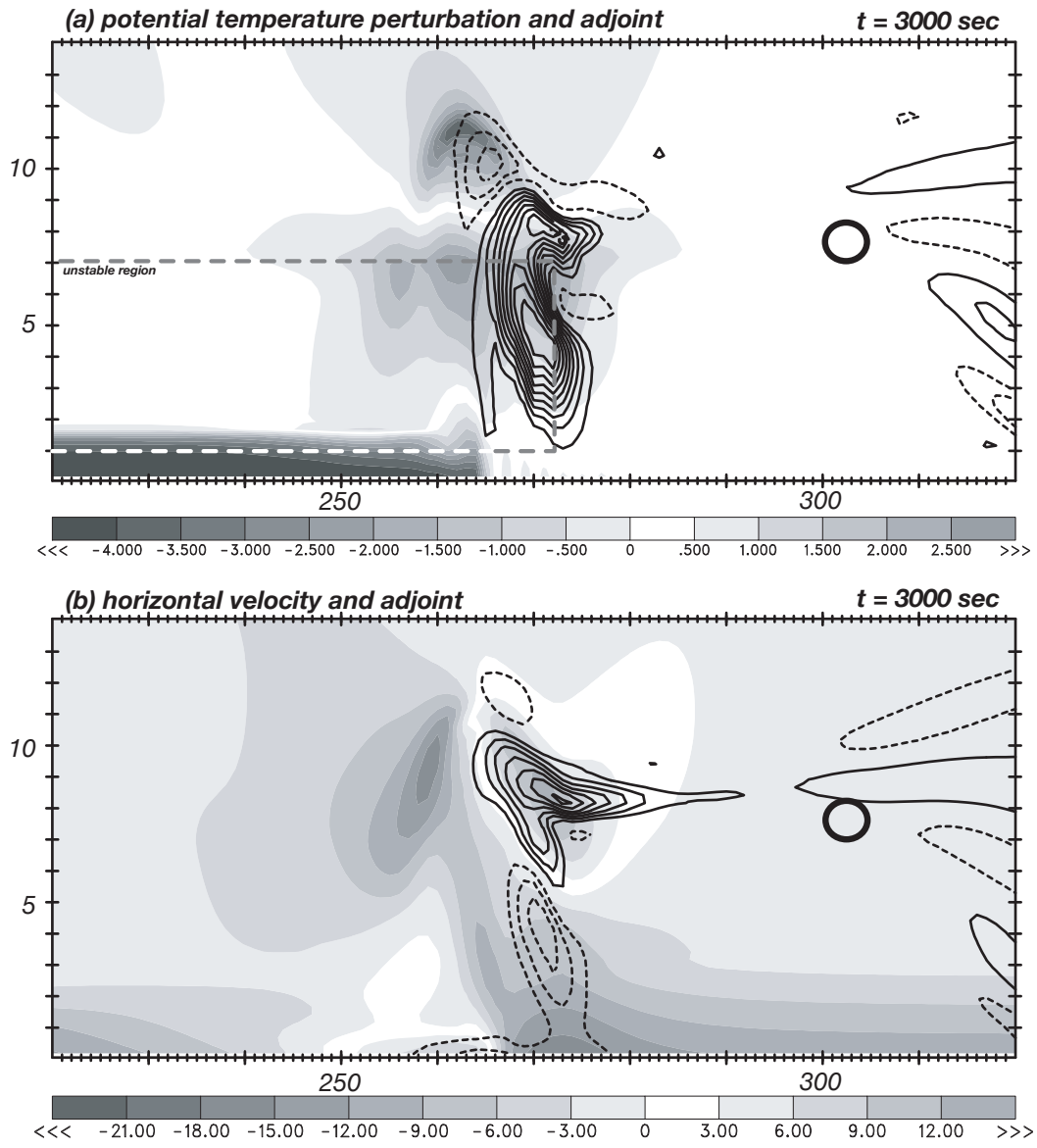

Figure 17.10: As in Fig. 8, but for the diabatic adjoint run. Temperature adjoint contour interval is .03 m s^{-1} K^{-1}; for the horizontal velocity adjoint it is .005 (nondimensional).

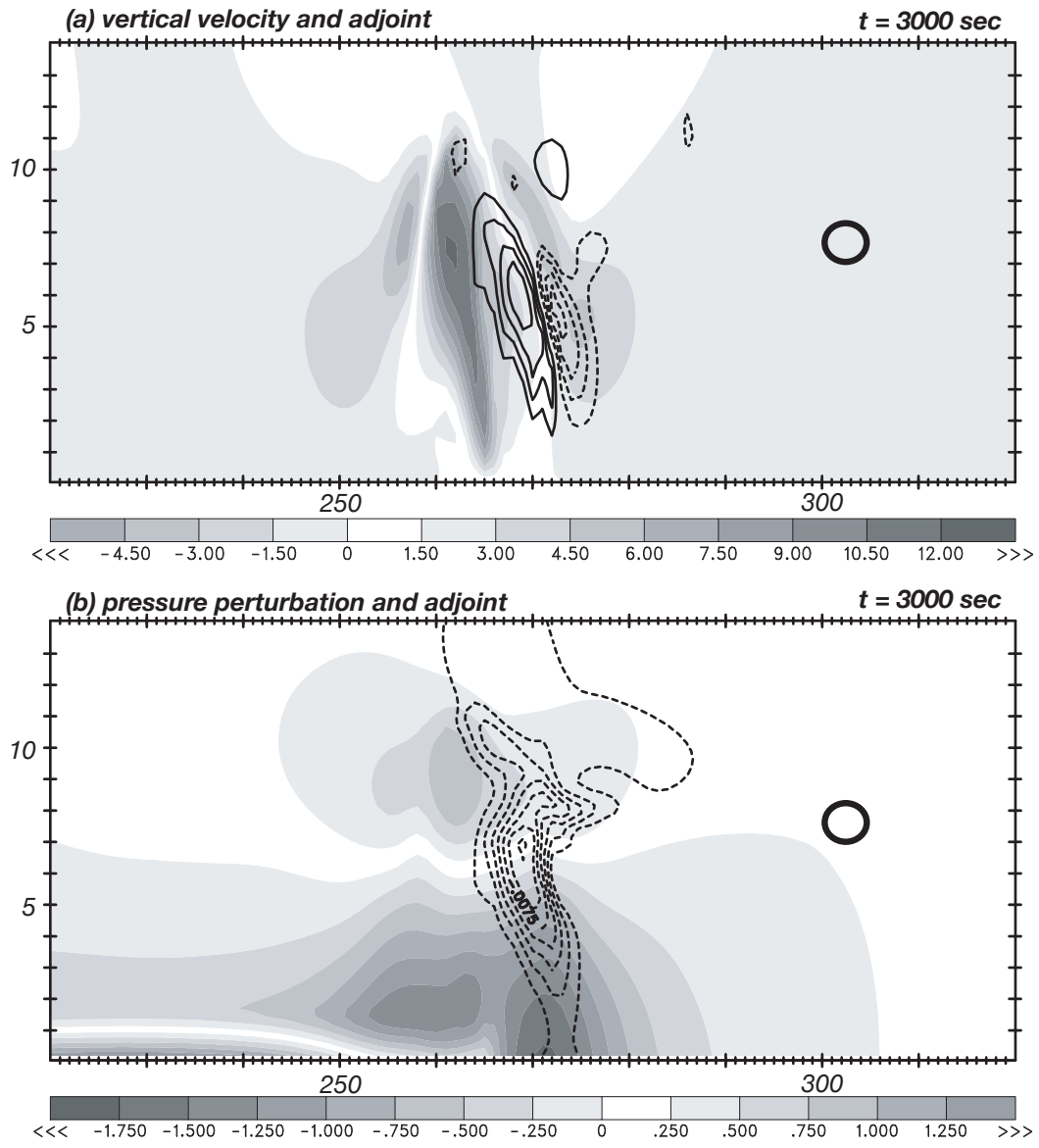

Figure 17.11: As in Fig. 9, but for the diabatic adjoint run. Vertical velocity adjoint contour interval is .003 (nondimensional); for the perturbation pressure adjoint it is .0015 m s^{-1} mb^{-1})

the storms' organizational period and appeared in PM simulations made with both low and moderate CAPE soundings, seemed excessive when comparison was made to model storms created using full physics cloud models.

Figure 12, adapted from FT00, shows mature phase profiles of ground-relative horizontal wind taken 20 km ahead of the surface gust front position for three simulations which shared the same initial wind profile (shown in solid black). The benchmark run (grey dashed line) was created with the ARPS[4] full physics cloud model and Fovell and Ogura's (1988, *JAS*; hereafter "FO") moderate CAPE sounding. The other two simulations employed PM models initialized with a modified sounding ("FO-MOD") based on the FO environment. The potential buoyancy a lower tropospheric parcel would experience upon undiluted ascent was discounted in this sounding to reflect the PM model's formal lack of water loading and dry air entrainment. The PM simulations gauge the effect of a model add-on FT00 termed a "convective sponge", described below. The "basic model" run did not include this extra term.

In all three simulations, the model storms have generated forward anvil outflows in the upper troposphere. Mass continuity dictates some compensating inflow must exist at another level; in the full physics simulation, it is confined to a relatively shallow midtropospheric layer, centered at about 5 km. Left largely untouched relative to the initial state was the storm's primary inflow source, which is drawn from the lowest 3 km.

In the basic PM model simulation, however, storm-relative inflow was enhanced at all levels beneath the forward anvil, with the maximum increase located very close to the ground. This is a significant difference. The full physics model storm's augmented inflow comes in the form of dry, middle tropospheric air which might be expected to weaken the convective intensity. In the PM model storm, in contrast, the maximum mass flux enhancement occurs where the air's potential buoyancy is largest.

Further, the magnitude of the enhanced inflow appeared large, and (again owing to mass continuity) this was likely due to the very intense forward anvil outflow the PM model storm supported. FT00 suspected that the apparently excessive forcing driving the PM storm's anvil outflow reflected a fundamental deficiency in the parameterization. The trailing region of a storm is typically warm in the middle to upper troposphere, resulting in the establishment of general high pressure poised above the warmed layer. FT00 judged this warming was excessively large in the typical PM model simulation, making for a rather pronounced cross-storm horizontal pressure gradient. This drove the intense westerly winds

[4]The University of Oklahoma's Advanced Regional Prediction System.

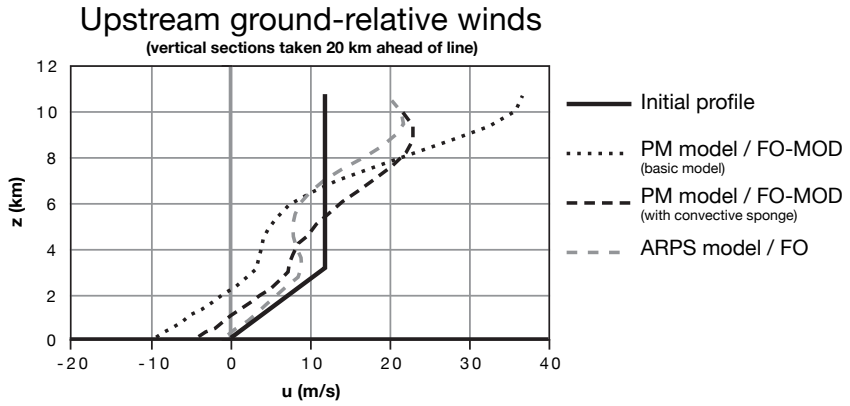

Figure 17.12: Instantaneous mature phase vertical profiles of ground-relative horizontal wind, taken at a location 20 km upstream of the surface gust front position. Also shown is the initial, undisturbed wind profile. Profiles were taken from simulations from the full physics ARPS model and also from PM model runs made without and with a "convective sponge" term. The FO-MOD sounding was derived from Fovell and Ogura's (1988; "FO") initial environment, used in the ARPS run. After Fovell and Tan (2000).

which comprised the storm's forward anvil outflow. . . and indirectly brought about the rather dramatically enhanced relative inflow below.

They then reasoned that the trailing region could become too warm because PM model storm's constituent convective cells never became starved of moisture as they propagated through this portion of the storm. In full physics models, the cell updrafts soon cease generating significant latent heating as they translate rearward. In the PM framework, however, such heating continues as long as the updraft persists, and of course this heating serves to maintain the updraft. In this manner, the warming can become uncharacteristically large.

To address this, FT00 fashioned a "convective sponge", an addition to the model θ' equation which relaxed positive temperature perturbations in the unstable region back to zero over a relatively long time scale. This kludge was an attempt to suppress the trailing region warming without unduly impacting the parameterized heat release in the main storm updraft. As suggested by Fig. 12, the sponge did work to reduce the strength of the forward anvil outflow, and so the compensating lower tropospheric inflow was correspondingly weaker in the convectively sponged simulation.

17.6.2 Further analysis

Subsequent intercomparison between PM and full physics model outputs suggested that the discrepancies resided in the *vertical structure*, rather than simply in the magnitude, of the compensating inflow. The salient difference is the location at which the enhanced inflow occurs. Again, in the PM model, the maximum easterly induced flow resides near the model surface and thus served to promote the inflow of parcels with large potential buoyancy. The traditional cloud model (apparently invariably) places the inflow enhancement in the dry middle troposphere. The convective sponge kludge didn't really work because it still didn't concentrate the easterly flow into the "correct" layer.

Our understanding of the model inconsistencies was aided by inspection of ground-relative perturbation horizontal velocity (u'_{grel}) fields in place of the storm-relative plots typically presented. The u'_{grel} field is obtained by removing both the initial sheared wind profile and the domain translation speed from the full horizontal velocity field the model prognoses. The storm-relative flow fields, in retaining the background shear, tends to obscure where and why the inflow enhancement is occurring.

On Figure 13, the u'_{grel} is superimposed upon the potential temperature perturbation field for times in the control run ranging out to 9000 sec. Focusing on the storm's upstream side, it is seen that as the subsidence wave spreads away from the convection, its wake is marked by westerly outflow (easterly inflow) in the upper (lower) troposphere. For most of the extent between this wave and the convection, the maximum easterly inflow perturbation is located close to the ground.

Now we examine Fig. 14, which presents rather typical results from a full physics multicellular model storm. Enhanced inflow is indeed present throughout the lower and middle troposphere on the full physics storm's upstream side. Far from the storm, the maximum inflow even resides near the surface as in the PM model simulation. Closer to the storm's leading edge, however, the enhancement has shifted to the middle troposphere, leaving the ground-relative perturbations near the ground to be very small. A vertical profile taken through this zone would look qualitatively similar to the full physics result presented in Fig. 12.

What does it take to shift the enhanced inflow layer into the middle troposphere, at least in the vicinity of the convection? To address this question, two diabatic adjoint model simulations were started at 9000 sec, both employing forecast aspects consisting of the horizontal velocity field within a spatially confined zone residing in the enhanced upstream inflow. In

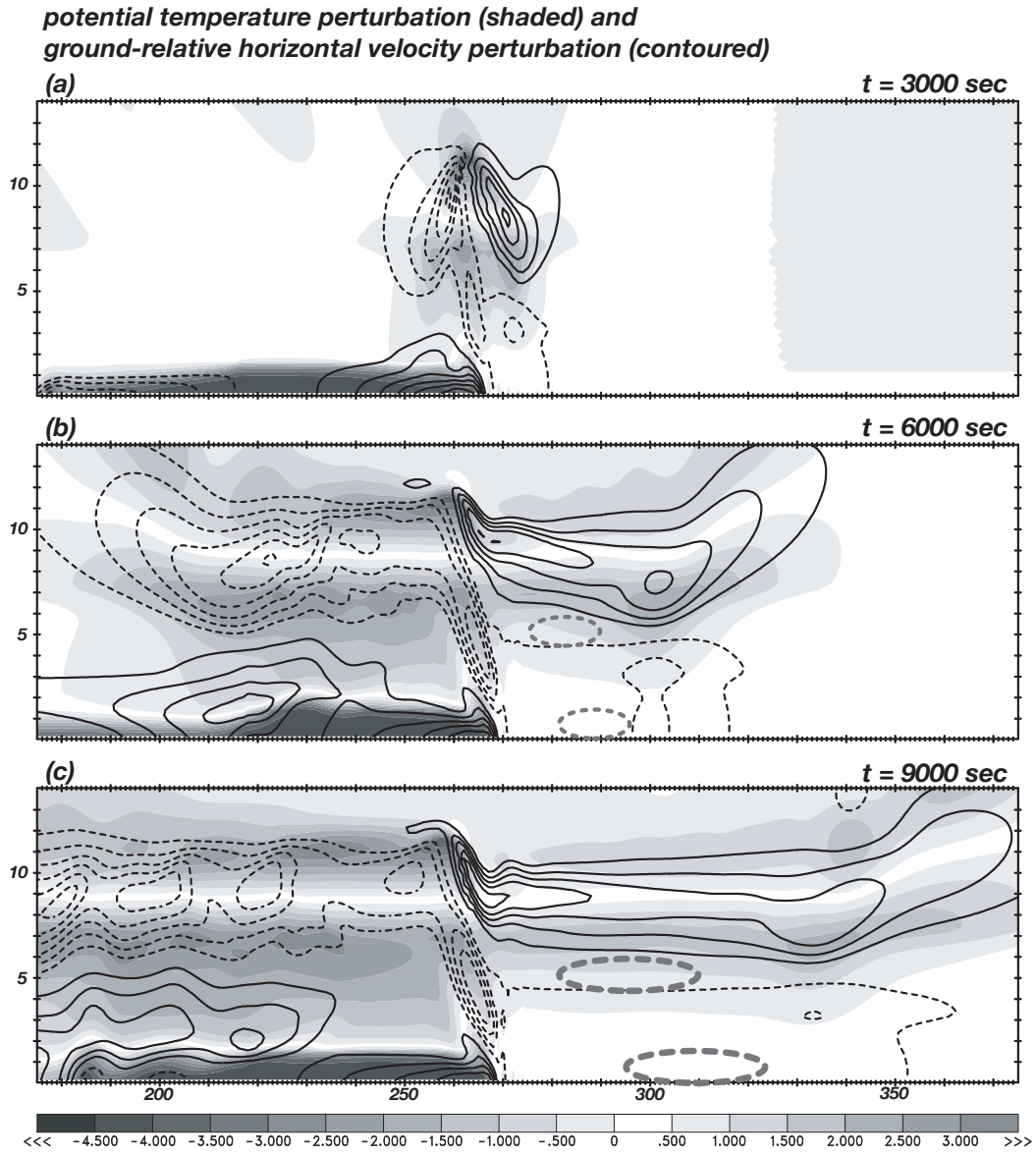

Figure 17.13: Potential temperature perturbation (shaded) and ground-relative horizontal velocity perturbation (contoured; interval 3 m s^{-1}) for the control run. For the latter, both the initial sheared wind profile and the domain translation speed have been removed from the full u field. The locations of the forecast aspects employed are also indicated (see text).

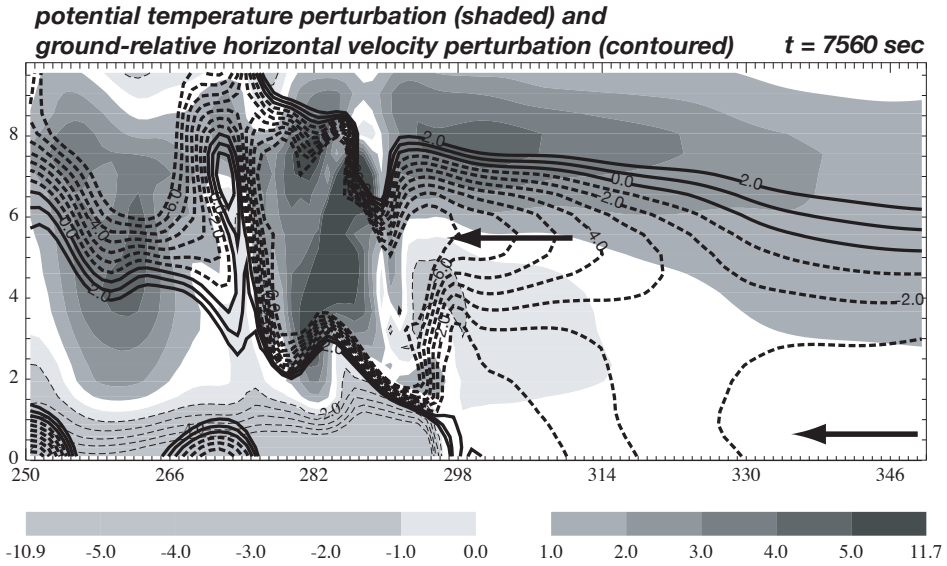

Figure 17.14: Potential temperature perturbation (shaded) and ground-relative horizontal velocity perturbation (contoured; interval 1 m s^{-1} between -7 and 2 m s^{-1}) for a typical multicellular case simulated by a full physics model (the ARPS model).

the first simulation, the aspect is placed in the lower troposphere; the second shifts the aspect to the middle troposphere. The initial aspect locations are shown in Fig. 13c. These starting positions were suggested by Fig. 14.

For the lower tropospheric aspect, the question being asked here is what would it take to *decrease* the strength of the easterly winds present at this level at this time? This is tantamount to inquiring after the mechanism that would permit the layer of easterly winds – which are needed owing to mass continuity anyway – to shift upward away from the lower troposphere. Decreasing the easterly flow there corresponds to the encouragement of a westerly perturbation. Recall that a positive ΔJ would indicate an enhancement of the westerly wind, so we are looking for combinations of TLM perturbations and adjoint sensitivities that would result in positive products.

It was again found that the sensitivity very quickly shifted into the temperature field which subsequently dominated the sensitivity results for both simulations. Figure 15a shows the temperature adjoint field at 5500 sec resulting from the lower tropospheric aspect. Negative temperature sensitivity is seen concentrated in the middle troposphere, just upstream of the storm's leading edge. Thus, the adjoint model is saying that actively *cooling* the control run storm at this location and time would subsequently result in weaker enhanced inflow in the upstream lower troposphere, since the combination of a negative thermal alteration

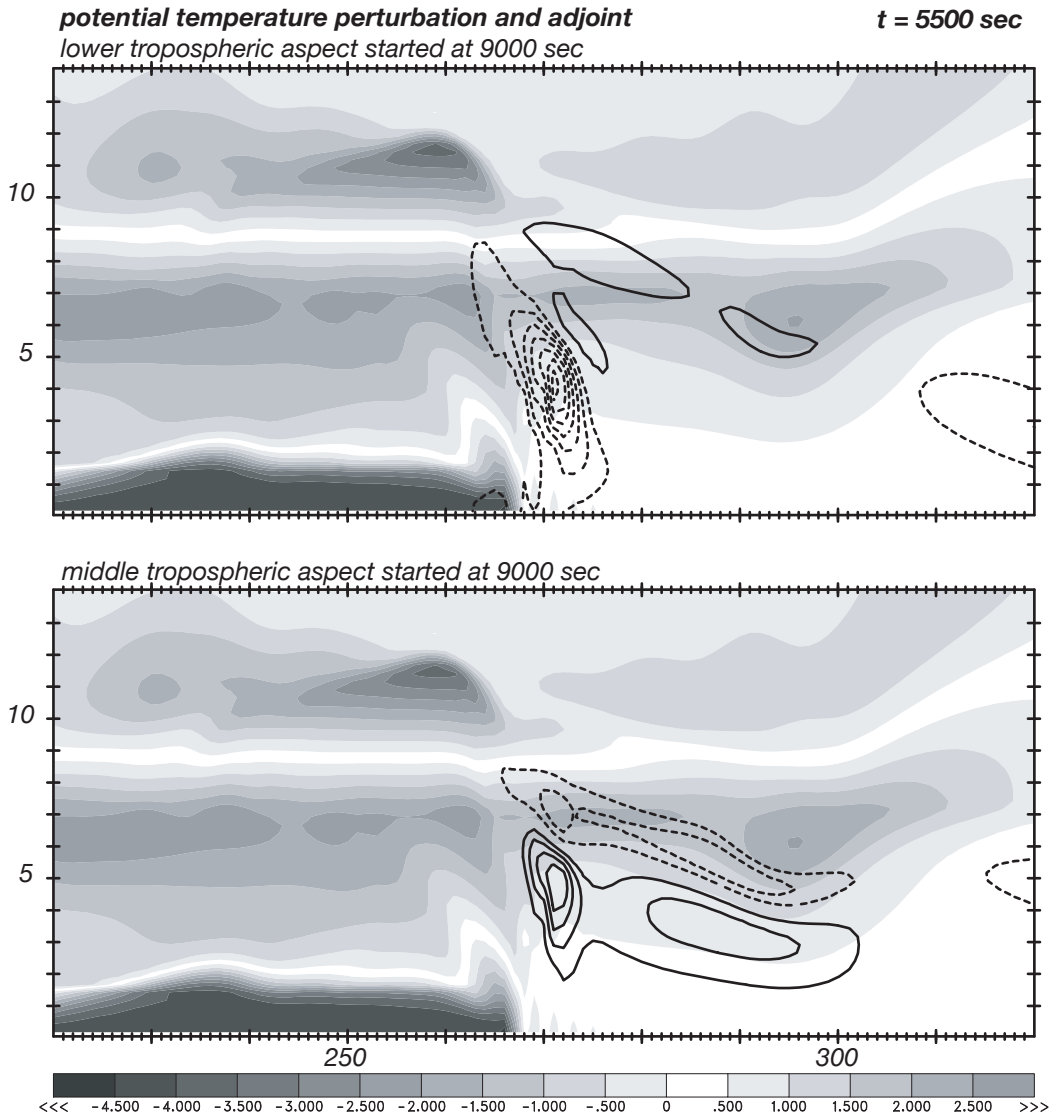

Figure 17.15: Potential temperature perturbation field from the control run (shaded) at 5500 sec and the adjoint sensitivity (contoured) fields resulting from the (a) lower tropospheric and (b) middle tropospheric forecast aspects identified in Fig. 13c. In (a), the contour interval is .2 m s^{-1} K^{-1}; in (b), it is .15 m s^{-1} K^{-1}.

(cooling) with negative sensitivity yields a positive ΔJ. The shaded field reveals there is some warming present within the sensitive region. Weakening this warming, or even instituting local cooling here, would bring about the desired effect.

For the middle tropospheric aspect, we are interested in *increasing* the easterly flow since that is consistent with elevating the enhanced inflow layer. Thus, we now look for TLM perturbations and adjoint sensitivities that result in negative contributions to ΔJ. The temperature sensitivity field at 5500 sec for this aspect is depicted in Fig. 15b. This aspect puts positive sensitivity in the same place where negative sensitivity resided for the lower tropospheric aspect. The two simulations are actually telling the same story: instituting cooling in this area at this time would encourage easterly midtropospheric flow even as it discourages such flow in the lower troposphere. In other words, the enhanced inflow layer would be elevated from near the surface, where it resides in the control run, to the middle troposphere, where the full physics model says it belongs.

Reinspection of Fig. 14 reveals there is indeed local cooling present at the leading edge of the full physics model storm, something that is absent from its PM counterpart (Fig. 13). Analysis revealed that this cooling resulted from the lifting of stable air in the vicinity of the main storm updraft. This air of middle tropospheric origin was compelled to rise upon encountering the storm updraft. Although the two-dimensionality of these simulations undoubtedly exaggerates this effect somewhat, the same phenomenon has been found to be quite pronounced in three-dimensional model storms as well (not shown). Further analysis indicated that the upstream modification could be interpreted, at least in part, as a gravity wave response to heating – and cooling – occurring in and around the main storm updraft (see, for example, Nicholls et al. 1991, in *JAS*).

The PM model is quite capable of handling the gravity wave response; what is missing is the cooling. Thus, the flaw in the PM framework is that all ascending air in the unstable region is presumed to be saturated thereby always yielding condensational warming. In actuality, the gently rising air at the periphery of convective updrafts may represent stable air that is and remains subsaturated on ascent, as demonstrated in Fig. 16. The left panel shows the updraft emanating from the gust front located at $x = 289$ km is capped by chilled air. The right panel contrasts this θ' field with the equivalent potential temperature (θ_e) distribution. θ_e is a nearly conserved quantity (apart from mixing), and thus provides an indicator of the air's level of origin. Most of the chilled air possesses small values of θ_e, indicating a midtropospheric source.

Although the cooling resulting from this stable midtropospheric air's displacement is small

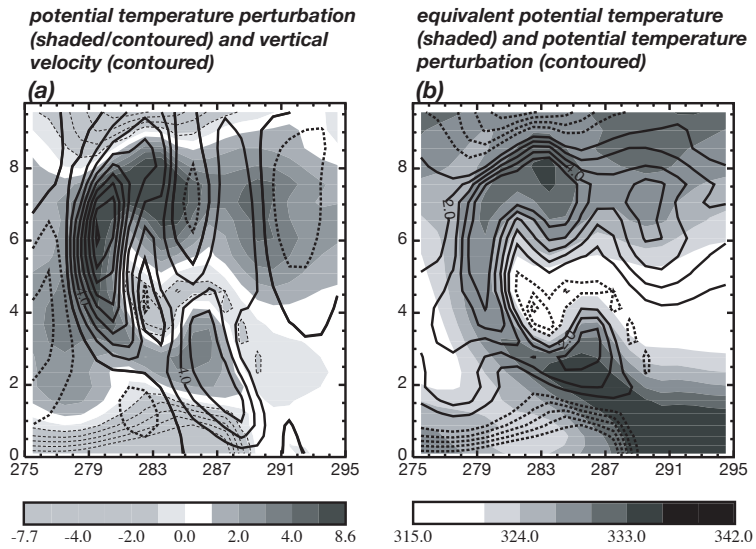

Figure 17.16: Close up view of storm's leading edge in a full physics ARPS model simulation. (a) potential temperature perturbation (shaded/contoured) and vertical velocity (contoured, interval 2 m s^{-1}). (b) equivalent potential temperature (shaded) and potential temperature perturbation (contoured, interval 1 K).

in magnitude and localized, it turns out to have a rather dramatic effect on the storm inflow structure. A simple, if crude, "fix" to capture this cooling might be to presume that only updrafts stronger than some small value $w_0 > 0$ should result in warming. That is, we replace (17.2) with

$$Q^+ = \gamma^* \max(w, w_0). \tag{17.11}$$

In this way, the weak ascent ($0 < w < w_0$) on the updraft periphery will be strictly dry adiabatic. Figure 17 demonstrates the impact of this alteration on simulations made using FT00's anelastic PM model and the moderate CAPE FO-MOD sounding. The upper panel shows θ' and u'_{grel} for an "unfixed" control run; the fields bear substantial resemblance to those of the low CAPE control run we have been analyzing. For the lower panel, the w_0 threshold was 1.5 m s^{-1}. Note the elevated zone of weak cooling which has appeared just upstream of the gust front and the impact this has had on the location of the enhanced inflow. The result, while still imperfect, is very much improved. It is noted that low CAPE environment was not amenable to this fix; even very small positive values of w_0 caused those storms to perish.

It is telling that neither adjoint simulation placed sensitivity in the model storm's trailing region, at least by the time depicted in Fig. 15. To see whether any sensitivity would appear there for an earlier time, the adjoint integrations were extended back to 3500 sec. Owing

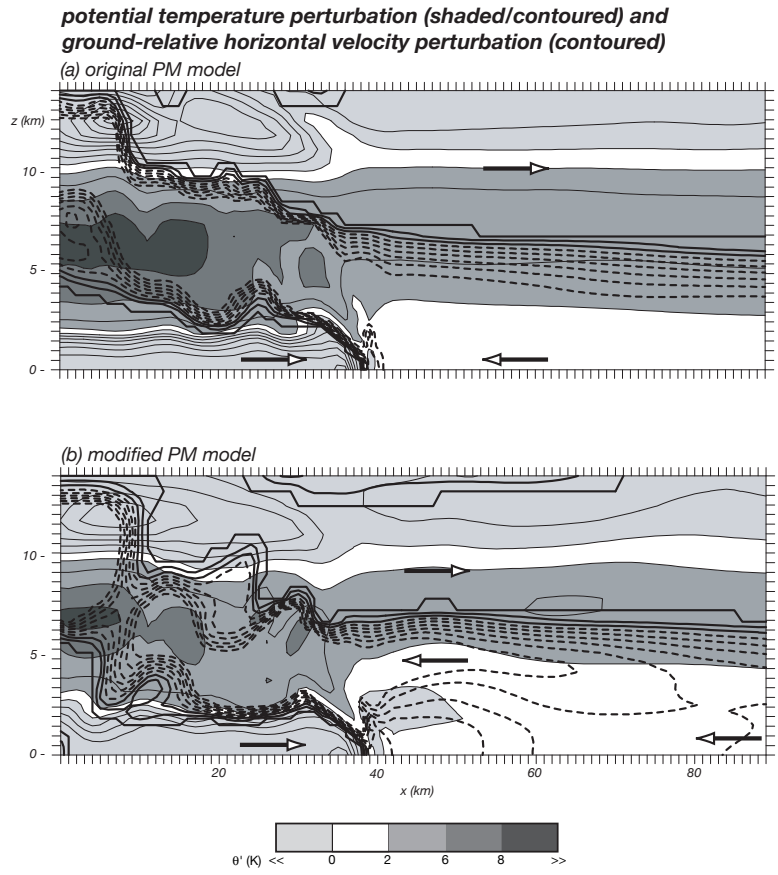

Figure 17.17: Potential temperature and ground-relative horizontal velocity perturbation fields from two PM model simulations, without and with local cooling in the vicinity of the storm leading edge. For u'_{grel}, the contour interval is 1 m s^{-1} but only contours between -7 and 2 m s^{-1} are drawn. These simulations were made using FT00's anelastic PM with the FO-MOD sounding.

to their inherent approximations and limitations, adjoint simulations probably should not be carried on too long. Thus, we started these sensitivity runs at 6000 sec, using the initial forecast aspects depicted on Fig. 13b. We are essentially posing the same questions as before.

The temperature sensitivities resulting from these two aspects for 3500 sec are shown in Fig. 18. In the vicinity of the storm's leading edge, the fields are rather comparable to those shown in Fig. 15. There is little sensitivity westward of the leading edge, and nothing of the type or at the location we were initially expecting to uncover[5]. In this application, the adjoint model proved to be a useful tool in disproving one hypothesis, suggesting and proving another, and leading to an improvement of both the PPM model framework and our understanding of convective storms.

[5]The sensitivity farther upstream is from transients associated with the abrupt model startup and appear to be dynamically unimportant.

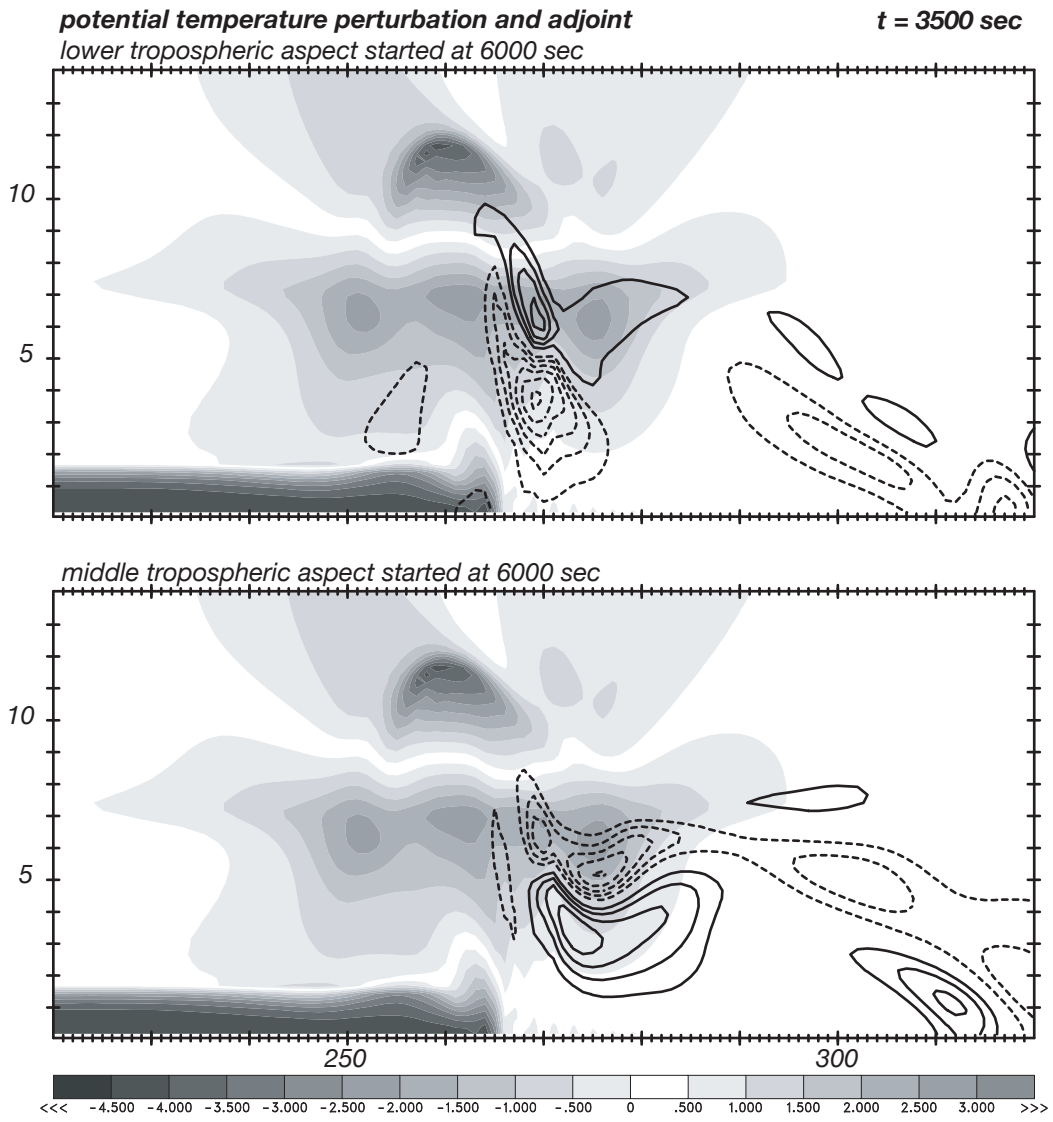

Figure 17.18: As in Fig. 15, but for 3500 sec. These adjoint runs were started at 6000 sec, using the (a) lower and (b) middle tropospheric forecast aspects depicted on Fig. 13b. In (a), the contour interval is .15 m s^{-1} K^{-1}; in (b), it is .1 m s^{-1} K^{-1}.

Part IV

Appendices

Appendix A

Justification of (5.8) and (5.9)

For the leapfrog scheme, the change function had two roots:

$$\begin{aligned} \lambda_+ &= \sqrt{1-a^2} - ia \\ \lambda_- &= -\sqrt{1-a^2} - ia. \end{aligned}$$

For the negative $(-)$ root, the polar form is

$$\lambda_- = -|\lambda| e^{i\theta_-},$$

with θ_- defined as in (5.7) using the $-$ root components of λ:

$$\theta_- = \arctan \frac{a}{\sqrt{1-a^2}} = \arcsin(a).$$

For the leapfrog scheme, $|\lambda| = 1$. Thus, we obtain:

$$\begin{aligned} \lambda_- &= -e^{i\theta_-} \\ &= -\cos\theta_- - i\sin\theta_- \\ &= -\cos[\arcsin(a)] - i\sin(\arcsin(a)) \\ &= -\sqrt{1-a^2} - ia, \end{aligned}$$

yielding the required expression. The latter expression was obtained because

$$\cos[\arcsin(x)] = \sqrt{1-x^2}$$

holds. Since $e^{i\pi} = -1$, we can also write:

$$\lambda_- = e^{i(\theta_- + \pi)}.$$

For the positive root, the polar form must be

$$\lambda_+ = |\lambda| e^{i\theta_+}$$

with

$$\begin{aligned} \theta_+ &= \arctan \frac{-a}{\sqrt{1-a^2}} \\ &= \arcsin(-a) \\ &= -\arcsin(a). \end{aligned}$$

Note that $\theta_- = -\theta_+$, and again $|\lambda| = 1$. Therefore,

$$\begin{aligned} \lambda_+ &= e^{i\theta_+} \\ &= \cos\theta_+ + i\sin\theta_+ \\ &= \cos[-\arcsin(a)] - i\sin(\arcsin(a)) \\ &= \sqrt{1-a^2} - ia, \end{aligned}$$

which is what we wanted. The latter expression employed

$$\cos[-\arcsin(x)] = \cos[\arcsin(x)] = \sqrt{1-x^2}.$$

Since $\theta_- = -\theta_+$, we can write both expressions in terms of θ_-, and drop the subscript on θ. This is how the equations were presented in (5.8) and (5.9).

Appendix B

Formulation of the TLM with Gateaux differentiation

B.1 Explanation

The more mathematically involved approach utilizes Gateaux (G-) differentiation, a general method of which Taylor series expansion is a special case. In this approach, we first rewrite the model equation (16.1) as

$$N(u, \alpha) = \frac{\partial u}{\partial t} - F_u(u, \alpha) = 0. \tag{B.1}$$

At this point, all we've done is to define a function N which is zero because of the property of the model equation. For the control run, the function is $N(u_C(x, z, t), \alpha_C)$, while for the TLM version of the alternative simulation it is $N(u_C(x, z, t) + u''(x, z, t), \alpha_C + \alpha'')$. The Gateaux expansion of the latter about the former yields

$$N(u_C + u'', \alpha_C + \alpha'') - N(u_C, \alpha_C) = VN + h.o.t., \tag{B.2}$$

where again the higher order terms will be neglected. The expression VN on the RHS is specified by:

$$VN = \frac{d}{d\epsilon} N(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'')|_{\epsilon=0} = 0, \tag{B.3}$$

where ϵ is a small number. *In this particular application*, VN=0 because both $N(u_C, \alpha_C)$ and $N(u_C + u'', \alpha_C + \alpha'')$ are zero separately by definition in (B.1).

The above came from the problem of finding the extremum of a functional, which in this case is $N(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'')$, via the Euler-Lagrange approach[1]. To proceed, we first replace

[1] Chan Man Fong et al., *Advanced Mathematics for Applied and Pure Sciences*, 1997, p. 748.

$N(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'')$ with the model equation on the RHS of (B.1), yielding

$$VN = \frac{d}{d\epsilon}\left[\frac{\partial(u_C + \epsilon u'')}{\partial t} - F_u(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'')\right]|_{\epsilon=0} = 0.$$

Now the bracketed term is differentiated with respect to ϵ. In the F_u term, the simplfying substitutions $u_1 = u_C + \epsilon u''$ and $\alpha_1 = \alpha_C + \epsilon\alpha''$ are used in the sequence below:

$$\begin{aligned} VN &= \left[\frac{d}{d\epsilon}\frac{\partial(u_C + \epsilon u'')}{\partial t} - \frac{d}{d\epsilon}F_u(u_1, \alpha_1)\right]|_{\epsilon=0} \\ &= \left[\frac{\partial u''}{\partial t} - \frac{\partial F_u}{\partial u_1}\frac{\partial u_1}{\partial \epsilon} - \frac{\partial F_u}{\partial \alpha_1}\frac{\partial \alpha_1}{\partial \epsilon}\right]|_{\epsilon=0} \\ &= \left[\frac{\partial u''}{\partial t} - u''\frac{\partial F_u}{\partial u_1} - \alpha''\frac{\partial F_u}{\partial \alpha_1}\right]|_{\epsilon=0}. \end{aligned}$$

Finally, the expression is evaluated at $\epsilon = 0$, at which point u_1 becomes u_C and α_1 becomes α_C, yielding:

$$VN = \frac{\partial u''}{\partial t} - u''\frac{\partial F_u}{\partial u}|_C - \alpha''\frac{\partial F_u}{\partial \alpha}|_C = 0.$$

Recognizing that $VN = 0$ in this case, we can rearrange the expression and see it is the same TLM that was (naively?) constructed via truncated Taylor expansions:

$$\frac{\partial u''}{\partial t} = u''\frac{\partial F_u}{\partial u}|_C + \alpha''\frac{\partial F_u}{\partial \alpha}|_C \tag{B.4}$$

B.2 A simple example

As an demonstration, the foregoing is applied to the differential equation embodied in (16.9). This starts off with:

$$N(u, \alpha) = \frac{\partial u}{\partial t} + \alpha u = 0,$$

where here $F_u(u, \alpha) = -\alpha u$. Note that $\frac{\partial F_u}{\partial u} = -\alpha$ and $\frac{\partial F_u}{\partial \alpha} = -u$. For the control run, $u = u_C$ and $\alpha = \alpha_C$, so the partial derivatives evaluated for the control run are

$$\frac{\partial F_u}{\partial u}|_C = -\alpha_C, \frac{\partial F_u}{\partial \alpha_C}|_C = -u_C.$$

Now we define

$$N(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'') = \frac{\partial(u_C + \epsilon u'')}{\partial t} + (\alpha_C + \epsilon\alpha'')(u_C + \epsilon u'') = 0.$$

The G-differential VN in this case is

$$VN = \frac{d}{d\epsilon}\left[N(u_C + \epsilon u'', \alpha_C + \epsilon\alpha'')\right]|_{\epsilon=0} = 0.$$

Substitution of our example model equation into the above and differentiation with respect to ϵ yields:

$$VN = \frac{d}{d\epsilon}\left[\frac{\partial(u_C + \epsilon u'')}{\partial t} + (\alpha_C + \epsilon\alpha'')(u_C + \epsilon u'')\right]|_{\epsilon=0} = 0,$$

then

$$VN = \left[\frac{\partial u''}{\partial t} + \frac{d}{d\epsilon}\left[\alpha_C u_C + \epsilon\alpha_C u'' + \epsilon\alpha'' u_C + \epsilon^2 u''\alpha''\right]\right]|_{\epsilon=0} = 0,$$

and finally, after dropping perturbation products (the approximation sign is implied), we end up with

$$VN = \left[\frac{\partial u''}{\partial t} + \alpha_C u'' + \alpha'' u_C + 2\epsilon u''\alpha''\right]|_{\epsilon=0} = 0.$$

The last term in the bracketed expression disappears when the expression is evaluated at $\epsilon = 0$. After this is done, we see we have

$$\frac{\partial u''}{\partial t} + \alpha_C u'' + \alpha'' u_C = 0.$$

Note that from the foregoing, $\alpha_C u'' = -u''\frac{\partial F_u}{\partial u}|_C$ and $\alpha'' u_C = -\alpha''\frac{\partial F_u}{\partial \alpha}|_C$, so what we actually have is (after rearrangement):

$$\frac{\partial u''}{\partial t} = u''\frac{\partial F_u}{\partial u}|_C + \alpha''\frac{\partial F_u}{\partial \alpha}|_C,$$

the same TLM that Taylor series would have produced.