

POTENTIAL VORTICITY INVERSION IN TERRAIN-FOLLOWING COORDINATES  
WITH APPLICATIONS TO MORPHOLOGICAL DATA ASSIMILATION

by

Steven G. Decker

A dissertation submitted in partial fulfillment of  
the requirements for the degree of

Doctor of Philosophy  
(Atmospheric and Oceanic Sciences)

at the  
UNIVERSITY OF WISCONSIN-MADISON  
2006

UMI Number: 3234707

### INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

**UMI<sup>®</sup>**

---

UMI Microform 3234707

Copyright 2006 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

**A dissertation entitled**

**Potential Vorticity Inversion in Terrain-Following  
Coordinates with Applications to Morphological Data Assimilation**

submitted to the Graduate School of the  
University of Wisconsin-Madison  
in partial fulfillment of the requirements for the  
degree of Doctor of Philosophy

**by**





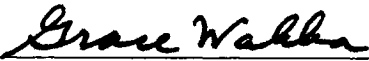
**Steven G. Decker**

**Date of Final Oral Examination: 07/26/06**


**Month & Year Degree to be Awarded: August 2006**

\*\*\*\*\*

Approval Signatures of Dissertation Committee

Signature, Dean of Graduate School



## Abstract

Forecasts generated by numerical weather prediction models continue to improve, but they are far from perfect. Forecast errors can be separated into those caused by shortcomings in the model (e.g., discretization and parameterization errors) and those caused by an imperfect estimate of the state of the atmosphere, oceans, and land surface (i.e., the initial conditions) given to the model. The goal of data assimilation is to eliminate the second class of errors to the greatest extent possible, given the observations at hand. Data assimilation is often treated as a statistical problem: Given a set of observations valid at some time, a previous forecast also valid at that time, and information about their error characteristics, what is the initial condition that is most likely to be closest to reality?

Morphological data assimilation is a complementary approach that seeks to use visual information, satellite data in particular, to help define the state of the atmosphere. The connection between satellite imagery and initial conditions in the form a model expects is made through the use of potential vorticity and its inversion.

The results presented in this dissertation document the construction of methods necessary to carry out morphological data assimilation in a more advanced and presumably accurate way than previous attempts. In particular, warping, wind partitioning, and potential vorticity inversion techniques are discussed within the context of morphological data assimilation. The potential vorticity inversion procedure is the most involved, and its discussion takes up the bulk of the work. Although the inversion procedure is not currently robust, prospects for the future are discussed that may lead to further improvement.

## Acknowledgments

This dissertation would not have been possible without the support of colleagues, friends, and family. I would like to thank my advisor, Dr. Jon Martin, for his support and willingness to let me do my own thing. He also deserves credit for providing financial support for travels to Boulder and Los Angeles for assorted summer schools and workshops. In addition, I engaged in many useful discussions with my committee members. Dr. Michael Morgan exposed me for the first time to tangent linear and adjoint models. Dr. Greg Tripoli kept me honest, while Dr. Grace Wahba provided the big picture view. Last, but not least, Dr. Matt Hitchman (Cheerio!) provided extra insights as only he can.

Friends provided immeasurable support as well. Mandy Decker (Nelson) kept me sane throughout the process, and provided for a wedding along the way as well. My former grad student colleagues Jason Otkin, Sarah and Kris Bedka, Monica Harkey, Brian and Gail Good, Blaine Thomas, Howard Berger, and Gregg Gallina, among others, provided plenty of support and extracurricular outlets. Friends in the C-YA group and its successors were also tremendous. The bell and chancel choirs allowed me a chance to use a different part of my brain, and the Lakeview Chess Club kept the gears turning as well.

Finally, I must mention the unending support of my family. They helped me from afar, and always made sure I had a ready answer to those questions about how the dissertation was coming.

# Table of Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>ii</b>
<b>Table of Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Traditional approaches toward improvement of numerical weather prediction	1
1.1.2 Potential vorticity and its inversion . . . . .	2
1.1.3 Altering model initial conditions using satellite data . . . . .	4
1.1.3.1 Feature calibration and alignment . . . . .	4
1.1.3.2 Warping . . . . .	5
1.1.3.3 PV Modification . . . . .	5
1.2 Dissertation objective . . . . .	6
<b>2 Methods used</b>	<b>8</b>
2.1 Structure of the WRF model . . . . .	8
2.1.1 Streamfunction calculation . . . . .	9

2.2	Warping technique . . . . .	14
2.3	PV inversion . . . . .	24
2.4	Test Cases . . . . .	25
2.4.1	WRF idealized baroclinic wave . . . . .	26
2.4.2	North Pacific system . . . . .	26
2.4.3	Continental United States . . . . .	28
<b>3</b>	<b>The nonlinear imbalance operator and its adjoint</b>	<b>32</b>
3.1	Balance equation in WRF vertical coordinate . . . . .	32
3.1.1	Expansion of divergence equation . . . . .	32
3.1.2	Scale analysis . . . . .	37
3.1.3	Balance equation . . . . .	47
3.2	Discretization . . . . .	51
3.3	Tangent linear and adjoint . . . . .	58
<b>4</b>	<b>The PV operator and its adjoint</b>	<b>63</b>
4.1	Derivation of operator . . . . .	63
4.2	Tangent linear and adjoint . . . . .	66
<b>5</b>	<b>Putting the pieces together: PV inversion</b>	<b>71</b>
5.1	Review . . . . .	71
5.2	Procedure . . . . .	71
5.3	Results . . . . .	73
5.3.1	Trial 1 . . . . .	73
5.3.2	Trial 2 . . . . .	73
5.3.3	Trial 3 . . . . .	77
<b>6</b>	<b>Discussion</b>	<b>81</b>
<b>7</b>	<b>Conclusions</b>	<b>86</b>

<b>8</b>	<b>Appendix</b>	<b>87</b>
8.1	File parameters . . . . .	87
8.2	Date utilities . . . . .	88
8.3	Meteorological diagnostics . . . . .	90
8.4	GEMPAK utilities . . . . .	106
8.5	Bookkeeping utilities . . . . .	109
8.6	Mathematical utilities . . . . .	109
8.7	Floating point kinds . . . . .	112
8.8	Simple constants . . . . .	112
8.9	User input and netCDF utilities . . . . .	112
8.10	Wind partitioning and PV inversion utilities . . . . .	128
8.11	Output definitions . . . . .	175
8.12	Main program . . . . .	226
	<b>References</b>	<b>229</b>



# List of Figures

2.1	Schematic of the WRF-ARW grid. (a) The horizontal grid. The circle indicates the collection of grid points assigned index $(i,j)$ , and the inner square represents grid box $(i,j)$ . (b) The vertical grid. . . . .	10
2.2	The initial data for the mass-weighted wind at around 500 hPa. The area displayed has been zoomed into an area of interest. Wind barbs represent magnitudes of $50 \text{ m s}^{-1}$ for each flag, $10 \text{ m s}^{-1}$ for each full barb, and $5 \text{ m s}^{-1}$ for each half barb. . . .	12
2.3	Results of partitioning the wind field in Fig. 2.2 in terms of (a) streamfunction (contour interval $5 \times 10^6 \text{ m}^2 \text{ s}^{-1}$ ), (b) velocity potential [contour interval $5 \times 10^5 \text{ m}^2 \text{ s}^{-1}$ , positive (negative) values solid (dashed)], (c) nondivergent wind, and (d) irrotational wind. The barb convention is as in Fig. 2.2. . . . .	13
2.4	Potential vorticity in the 315–325 K layer (contour interval 1 PVU, $1 \text{ PVU} = 10 \text{ m}^2 \text{ K kg}^{-1} \text{ s}^{-1}$ ) as derived from the GFS model initialized at 12 UTC 8 Nov 2005. . . .	15
2.5	Distribution of specific humidity near the tropopause as derived from satellite imagery at 12 UTC 8 Nov 2005. Warmer (cooler) colors indicate greater (lesser) moisture content. . . . .	16
2.6	A schematic of the warping algorithm. From Beier and Neely (1992) Figure 3. . . .	17

2.7	500-hPa geopotential heights (pink, contoured every 6 dam) analyzed at 12 UTC 7 Jun 2004, along with control line pairs. Lines in dark orange identify features in the original geopotential height distribution, and corresponding lines in blue identify the locations to which those features should be moved. Yellow-orange lines indicate identical line pairs used in an attempt to localize the warp. . . . .	19
2.8	Warped 500-hPa geopotential heights (pink, contoured every 6 dam) valid at 12 UTC 7 Jun 2004, showing final control lines in dark orange and yellow-orange. . .	20
2.9	500-hPa geopotential height perturbation [contour interval 3 dam, positive (negative) values solid (dashed)] introduced by the warping algorithm described in the text. .	21
2.10	Warping a PV field using the revised weighting function. (a) An overlay of Fig. 2.4 on top of Fig. 2.5. (b) Same as (a) except after the warp has been applied. (c) The PV perturbations introduced by the warp [contour interval $2 \times 10^{-7}$ PVU, positive (negative) values solid (dashed)]. . . . .	23
2.11	500-hPa geopotential heights (contour interval 6 dam, solid) and absolute vorticity (contour interval $5 \times 10^{-5} \text{ s}^{-1}$ , starting at $15 \times 10^{-5} \text{ s}^{-1}$ , dashed) at (a) 12 h, (b) 33 h, (c) 54 h, (d) 75 h, (e) 96 h, and (f) 117 h into the WRF model's "em_b_wave" idealized simulation. The northern and southern portions of the domain are excluded to focus on the baroclinic wave. . . . .	27
2.12	Test Case 2 at 1200 UTC 5 Apr 2006. (a) 500-hPa geopotential heights (contour interval 6 dam, solid) and absolute vorticity (contour interval $4 \times 10^{-5} \text{ s}^{-1}$ , starting at $12 \times 10^{-5} \text{ s}^{-1}$ , dashed). Vorticity maxima are marked by X. (b) 1000-hPa geopotential heights (contour interval 3 dam, solid) and temperatures (contour interval $2^\circ\text{C}$ , dashed). . . . .	29
2.13	As for Fig. 2.12, but at 0000 UTC 6 Apr 2006. . . . .	30
2.14	As for Fig. 2.12, but at 1200 UTC 6 Apr 2006. . . . .	31
3.1	Plot of $M$ 12 hours into the respective simulations for (a) Test Case 2 (contoured every 0.05 units) and (b) Test Case 3 (contoured every 0.5 units). . . . .	39

3.2	Terms from the balance equation for Test Case 1 at hour 117. (a) A1 [contour interval $10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. (b) Same as (a) but for A2. (c) Same as (a) but for A3. . . . .	41
3.3	Same as Fig. 3.2 except for (a) A4 (contour interval $2 \times 10^{-11} \text{ s}^{-2}$ ) and (b) A5 (contour interval $6 \times 10^{-11} \text{ s}^{-2}$ ). . . . .	43
3.4	Same as Fig. 3.2 except for (a) B1 (contour interval $4 \times 10^{-9} \text{ s}^{-2}$ ), (b) B2 (contour interval $10^{-10} \text{ s}^{-2}$ ), and (c) B3 (contour interval $2 \times 10^{-9} \text{ s}^{-2}$ ). . . . .	44
3.5	Same as Fig. 3.2 except for (a) C (contour interval $6 \times 10^{-9} \text{ s}^{-2}$ ) and (b) E (contour interval $6 \times 10^{-9} \text{ s}^{-2}$ ). . . . .	46
3.6	Examination of the left-hand side of the balance equation for Test Case 1 at hour 117. (a) The sum of the left-hand side of Eq. (3.14) [contour interval $6 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. (b) Same as (a) but including terms B1, B2, C, and D only. (c) The graphical difference of panels (a) and (b) [contour interval $3 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. . . . .	48
3.7	Same as Fig. 3.6 but for Test Case 2 at hour 12, and with (a-b) contour interval $4 \times 10^{-9} \text{ s}^{-2}$ , and (c) contour interval $2 \times 10^{-9} \text{ s}^{-2}$ . . . . .	49
3.8	Determination of imbalance for Test Case 1. (a) $l_1 - l_2 - l_3$ [contour interval $6 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)], (b) same as (a) but for $r$ , and (c) nonlinear imbalance ( $l_1 - l_2 - l_3 - r$ ) [contour interval $3 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. . . . .	56
3.9	Same as Fig. 3.8 but for Test Case 2. . . . .	57
3.10	Checking the tangent linear version of the imbalance operator for Test Case 2. (a) Result of imbalance operator at $\eta = 0.525$ after adding the perturbations described in the text [contour interval $3 \times 10^{-9} \text{ s}^{-2}$ , with positive (negative) values solid (dashed)]. (b) Same as panel (a) but for $\eta = 0.575$ . (c) Same as (a) but showing the difference on the imbalance operator between perturbed and nonperturbed input. (d) Same as (c) but for $\eta = 0.575$ . (e) Same as (a) but showing output from the tangent linear version of the imbalance operator. (f) Same as (e) but for $\eta = 0.575$ . . . . .	61

4.1	Potential temperature (contour interval 4 K) from (a) model output from Test Case 1, (b) model output from Test Case 2, (c) Eq. (4.5) applied to Test Case 1, and (d) Eq. (4.5) applied to Test Case 2. . . . .	65
4.2	(a) Potential vorticity calculated from Eq. (4.3) for Test Case 2 (contour interval 0.5 PVU). (b) Same as (a) but calculated from Eq. (4.6). (c) The graphical difference between (a) and (b) (contour interval 0.05 PVU, zero contour omitted). . . . .	67
4.3	Same as Fig. 3.10 except showing the tangent linear version of the potential vorticity operator [contour interval 0.25 PVU, positive (negative) values solid (dashed), except (b) contour interval 0.25 PVU, and (d) and (f) contour interval 0.1 PVU]. . . . .	69
5.1	A trivial case of PV inversion, as described in the text. (a) Diagnosed geopotential heights at $\eta = 0.525$ (contour interval 6 dam). (b) Diagnosed streamfunction at $\eta = 0.525$ (contour interval $5 \times 10^6 \text{ m}^2 \text{ s}^{-1}$ ). (c) Same as (a) except showing the result of the inversion. (d) Same as (b) except showing the result of the inversion. . . . .	74
5.2	Reduction in the cost function during the inversion process for the second trial. All values are in $\text{m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^{-2}$ . The line labeled “A” (“B”) represents that part of the cost function due to the PV (imbalance) operator. The line labeled “C” represents the total cost function. . . . .	75
5.3	A more complicated case of PV inversion. Panels (a) and (b) are the same as in Fig. 5.1. (c) and (d) Same as Figs. 5.1c–d but from the second trial. . . . .	76
5.4	Same as Fig. 5.2, but for the third trial. . . . .	78
5.5	Same as Fig. 5.3 but for the third trial. . . . .	79
6.1	The results of Trial 3 with a thirtyfold increase in $\epsilon$ . (a) Same as Fig. 5.5c. (b) Same as Fig. 5.5d. (c) Same as Fig. 5.4. . . . .	82

# List of Tables

3.1	The average absolute magnitudes of various terms in the balance equation. . . . .	45
-----	---	----

# Chapter 1

## Introduction

### 1.1 Background

#### 1.1.1 Traditional approaches toward improvement of numerical weather prediction

As early as a decade ago, Reynolds et al. (1994) suggested that errors in the analysis rather than errors in the model formulation were primarily responsible for forecast errors in the extratropics over the short-to-medium range. In the intervening decade, computer power has continued to increase, leading to the production of numerical forecasts with higher resolution, better physics, and more advanced data assimilation schemes. Despite these advances, large forecast errors still occur on occasion. Buizza and Chessa (2002), Zhang et al. (2002), and Zupanski et al. (2002) all discuss the “Surprise” Snowstorm of 2000, while McMurdie and Mass (2004) document recent bad forecasts affecting the Pacific Northwest. Closer to home, Alberta Clippers have recently produced unforecasted snowfalls both in Madison (12 February 2003) and near the Twin Cities (8 March 2004), while neither the Storm Prediction Center (SPC) nor local National Weather Service (NWS) forecast offices anticipated the tornado outbreak across Illinois on 20 April 2004. Clearly there is room for improvement.

A number of approaches have been taken in an effort to improve numerical weather forecast

fidelity. I will focus on improvements to the initial conditions, since that avenue should have the greatest impact on the 1–5-day range, which is the time-frame on which I wish to focus. The three main assimilation schemes in present use, 3DVar (Three dimensional variational), 4DVar (Four dimensional variational, time being the extra dimension), and the Ensemble Kalman Filter (EnKF), continue to be improved. Work on 3DVar currently focuses on improving the specification of the background error covariance matrix (e.g., Purser and Parrish 2003), and a 3DVar system has recently been introduced for the Weather Research and Forecasting (WRF) modeling system. 4DVar relies on adjoint models, which are constantly being upgraded to keep up with their related forward models' development. Currently, no adjoint model for the WRF is available, making 4DVar unfeasible with that model. EnKF requires on the order of 100 ensembles, which is very computationally intensive at present. Aside from that, the main issue preventing EnKF from becoming widely adopted is the occasional catastrophic failure it suffers known as filter divergence. Filter divergence occurs when model background error covariances become so small that all observations are essentially ignored. As a result, the model forecast loses touch with reality, rendering itself worthless.

### 1.1.2 Potential vorticity and its inversion

Over the last two decades, it has become evident that a good way to encapsulate the dynamics of the atmospheric flow is through consideration of the potential vorticity (PV). This is because PV has two particularly useful properties, conservation and invertibility (Hoskins et al. 1985). The latter property has proven especially important to many studies that have aimed to improve or otherwise alter model initial conditions. This is because the invertibility principle allows for the recovery of a number of other dynamical variables given a domain-wide PV distribution, a balance constraint, and appropriate boundary conditions on the domain. An omega equation may be used to recover vertical motion as well. The most well-known PV inversion technique is that of Davis and Emanuel (1991) (DE), which invokes the Charney nonlinear balance along with an approximate form of the definition of Ertel PV, using the Exner function as the vertical coordinate. Numerous studies have used the DE method; a notable recent example is Martin and Otkin (2004).

QGPV inversion is also quite popular, both in real-world (e.g., Hakim et al. 1996) and idealized (e.g., Kim and Morgan 2002) situations. QGPV inversion is advantageous because the equations are linear, but the results are less accurate than full Ertel PV inversion. Accordingly, studies aiming to improve model forecasts through direct intervention have tended to shy away from it.

Other inversion procedures are apparently less frequently used. Raymond (1992) derived a nonlinear balance equation in height coordinates and used two different scaling arguments to formulate approximate definitions of PV. The resulting elliptic equations were solved using a multigrid method (Fulton et al. 1986). Of particular note is that a low-pass filter was occasionally applied to keep the solution from oscillating. These methods apparently have not yet been applied to real-data cases, however.

Arbogast and Joly (1998) formulated PV inversion as a minimization problem, first in a two-dimensional context, and then (Mallet et al. 1999) in a real-data case involving an inversion from output of a numerical model in sigma coordinates, albeit using QGPV. Rather than solving elliptic equations, this method uses a cost function composed of two terms, one measuring the inaccuracy of the PV equation, the other the imbalance of the flow. A quasi-Newton method minimizes the cost function and requires the adjoints of the PV and nonlinear balance operators to do so. A particularly nice advantage to this approach is that the PV need not be strictly positive to guarantee a solution. On the other hand, the method may not find the cost function's global minimum.

Although they do not perform inversion, Schneider et al. (2003) develop PV equations analogous to Maxwell's equations in electrostatics. In this formulation, PV and its flux determine the rest of the dynamical variables, rather than PV and a balance constraint. The study of Vallis et al. (1997) is intriguing since they were able to invert the PV of a small-scale flow arising during analysis of convectively generated balanced motion in a large-eddy simulation, a situation for which DE is unlikely to converge. This study used a linearized form of PV and gradient-wind balance in height coordinates.

Applications of PV inversion extend beyond morphological data assimilation or the analysis of flow fields. For example, in a study by Roebber et al. (2002), the sensitivity of the 3 May 1999



tornado outbreak over Oklahoma to a particular potential vorticity anomaly off the California coast was assessed. The PV anomaly had been hypothesized to be important to the development of the outbreak, and this was tested by changing the magnitude of the anomaly, inverting the various resulting PV distributions, and rerunning numerical forecasts using the inverted PV for the initial conditions.

### **1.1.3 Altering model initial conditions using satellite data**

#### **1.1.3.1 Feature calibration and alignment**

It is a common complaint that satellite data are routinely ignored during the data assimilation process at the national centers, particularly the National Centers for Environmental Prediction (NCEP) (Derber 2003, personal communication). A number of studies have attempted to overcome this through nontraditional use of satellite data to produce presumably better model initial states. The techniques used in these studies could all be called forms of morphological data assimilation. Hoffman et al. (1995) took an early step in this direction. They formulated a minimization problem by which both displacement and amplitude errors in the forecast could be determined by comparing a model-derived field with a corresponding satellite-derived product. Specifically, they compared Special Sensor Microwave/Imager (SSM/I) precipitable water data with a precipitable water field derived from a European Centre for Medium-Range Weather Forecasts (ECMWF) model analysis. Although they have not used this information to rerun any forecasts in any of the papers listed in this paragraph, they state forecast reruns as a possible application. Another application of their approach in the study of model error involves, say, comparing a 24-hour forecast with the verifying analysis. The resulting displacement and amplitude fields may provide a more meaningful depiction of model error than a simple RMS error or skill score. This technique, known as feature calibration and alignment, has been extended subsequently by Hoffman and Grassotti (1996), Grassotti et al. (1999), and Nehrkorn et al. (2003). Specifically, Hoffman and Grassotti introduced a double sine series representation for each component of the distortion, Grassotti et al. employed sensitivity tests to determine optimal values for various arbitrary parameters needed by the technique, and

Nehrkorn et al. expanded the domain to the entire Northern Hemisphere while eliminating the subjectivity of many of the parameters without the need for tuning.

### **1.1.3.2 Warping**

Alexander et al. (1998) took a different approach in their study using SSM/I data. They used the technique of warping to alter the initial conditions of Fifth-Generation NCAR/Penn State Mesoscale Model (MM5) forecasts, using integrated water vapor as the field of comparison between the model initial state and the satellite imagery.

In warping, one compares features between images, setting control points or lines that the analyst believes correspond to matching features. Wolberg (1998) provides an overview of warping techniques in the context of morphing. Morphing, which is used to produce animations, consists of three steps, feature specification, warp generation, and transition control. Warping, on the other hand, consists of only the first two steps, since, in warping applications, one cares not about how the warped field moves from the initial to altered state, but only about what that altered state is.

Alexander et al. (1998) report only the slightest of improvements in their reruns, and there are at least two reasons for this. First, they use a relatively simple warping technique, which they call third-order warping. In this technique, the displacements defining the transformation between initial and altered states are assumed to be two-dimensional polynomials of degree six. Thus, each control point affects the warp at every part of the grid. In contrast, the technique of Beier and Neely (1992), which uses control lines, limits the influence of each control line to the neighborhood of that line, resulting in more degrees of freedom than a simple polynomial can provide. In this way, features can be aligned more precisely. Another factor limiting the effectiveness of Alexander et al.'s approach is that they only change the water vapor field; in their reforecasts the dynamical fields remain as they were in the control run.

### **1.1.3.3 PV Modification**

With the exception of Mallet et al. (1999), to my knowledge all studies to date involving PV modification of initial conditions have employed either DE's technique or the QGPV inversion of

Hakim et al. (1996). Huo et al. (1998) used the piecewise inversion technique of DE to spread the effect of buoy and ship observations over the Gulf of Mexico throughout the lower troposphere as a revised initial condition for a simulation of the March 1993 superstorm. The model's forecast was significantly improved.

Demirtas and Thorpe (1999) compared PV on an isentropic surface intersecting the tropopause with water vapor imagery from Meteosat. They then altered the PV on a gridpoint-by-gridpoint basis until gradients in the PV matched those seen in the satellite imagery. Upon inverting the modified PV using the same general procedure as DE, they fed the resulting balanced fields into their model's assimilation system as "bogus obs." They noted marked improvement in the track and intensity of a cyclone affecting the United Kingdom. In a follow-up study, however, Swarbrick (2001) found that use of this technique on a regular basis produced little improvement. It should be noted, however, that in Swarbrick's study the regular assimilation scheme was turned off; in effect, altering the PV *was* the assimilation scheme. Thus, the comparison was not truly a fair one.

## 1.2 Dissertation objective

The objective of the research presented here has been to synthesize and extend the techniques described above in an attempt to reduce initial condition error in NWP. Specifically, I have endeavored to improve the initial conditions of WRF model runs by using the warping technique on the near-tropopause potential vorticity field. Inversion of this altered PV field then produces the necessary state variables for initialization into the WRF model. This research most directly builds on the warping technique of Beier and Neely (1992), the idea of Alexander et al. (1998) to use warping to alter initial conditions, the PV modification approach of Demirtas and Thorpe (1999) and Swarbrick (2001), and the PV inversion techniques of Davis and Emanuel (1991) and Arbogast and Joly (1998).

To document how these techniques have been extended, Chapter 2 provides an overview of the methods used while carrying out the research, including brief descriptions of the WRF model, the warping technique, the PV inversion technique, and the test cases used during development of the

techniques. The PV inversion technique is by far the most complicated, so Chapters 3–5 document the pieces that go into that technique in detail. Chapter 3 covers the nonlinear imbalance operator, including its derivation as well as the derivation of its tangent linear and adjoint cousins. Chapter 4 describes the PV operator and its tangent linear and adjoint versions, and Chapter 5 presents the preliminary results of putting these pieces together in an attempt to invert PV.

Unfortunately, the PV inversion technique is not yet robust, meaning that it often produces unphysical results. Chapter 6 discusses the implications of this, and offers prospects for the future, while Chapter 7 concludes the main body of the dissertation. Since a complete presentation of the procedures used to invert PV would be too lengthy, an appendix is included that contains the PV-inversion Fortran code in its entirety, including brief annotations.

## Chapter 2

# Methods used

The goal of this work is to be able to modify the initial conditions of a numerical weather prediction (NWP) model through the warping and inversion of that model's initial PV field. This requires three main ingredients: an understanding of the NWP model's formulation, a warping technique, and a PV inversion technique. In addition, test cases need to be used to ensure the accuracy of the various techniques. The salient features of these ingredients are described in this chapter, to be expanded upon in the chapters that follow.

### 2.1 Structure of the WRF model

The NWP model chosen for use is the Weather Research and Forecasting (WRF) model using the ARW (Advanced Research WRF) core (Skamarock et al. 2005). This model was chosen for the following reasons:

- It is designed to be used by both the research and operational communities. Thus, research done using the WRF model may be more easily transferred to operations.
- It is superior to the widely used MM5 model by just about any benchmark, and is now considered “research ready.”
- It can be run on readily available machines at reasonable speed.

The WRF model is formulated in a terrain-following coordinate that is closely related to traditional sigma coordinates. In the horizontal plane, Arakawa C staggering is used, while Charney-Phillips staggering is employed in the vertical plane. Figure 2.1 shows how the basic variables of the model are distributed amongst the gridboxes and also indicates the indexing convention used in this paper.

Recently, the Nonhydrostatic Mesoscale Model (NMM) core of the WRF model has been released, and it is this core that is now being used operationally at NCEP. Since the NMM core uses different coordinates and grid staggering schemes, the techniques developed herein will not transfer to the NMM core without additional work.

### 2.1.1 Streamfunction calculation

The PV inversion process uses the streamfunction of the mass-weighted flow as one of two dependent variables (geopotential being the other). Values of streamfunction must be specified along the lateral boundaries of the domain, and interior values of streamfunction serve as a first guess. The technique used to compute the streamfunction is derived from the algorithm of Bijlsma et al. (1986). Their algorithm uses a horizontal staggering similar to the WRF, except an additional half grid box is added to the north and east sides of the domain such that the northern (eastern) edge of the domain contains values for the x- (y-) component of the wind.

If we were using a global domain, the Helmholtz theorem tells us that any vector field can be decomposed into two unambiguous components, one that is nondivergent (defined by the streamfunction), and the other that is irrotational (defined by the velocity potential). However, on a limited-area domain, such as the WRF, there is a third component, commonly called the harmonic component (e.g., Loughe et al. 1995), that is both nondivergent and irrotational. There are an infinite number of ways to assign the harmonic component, in whole or in part, to the nondivergent and irrotational wind components. The method described here assigns all of the harmonic component to the nondivergent component, since the inverted PV should capture as much of the total flow as possible.

The method works as follows. First, let us identify the mass-weighted wind as  $\vec{\mathfrak{M}}$ . (This wind

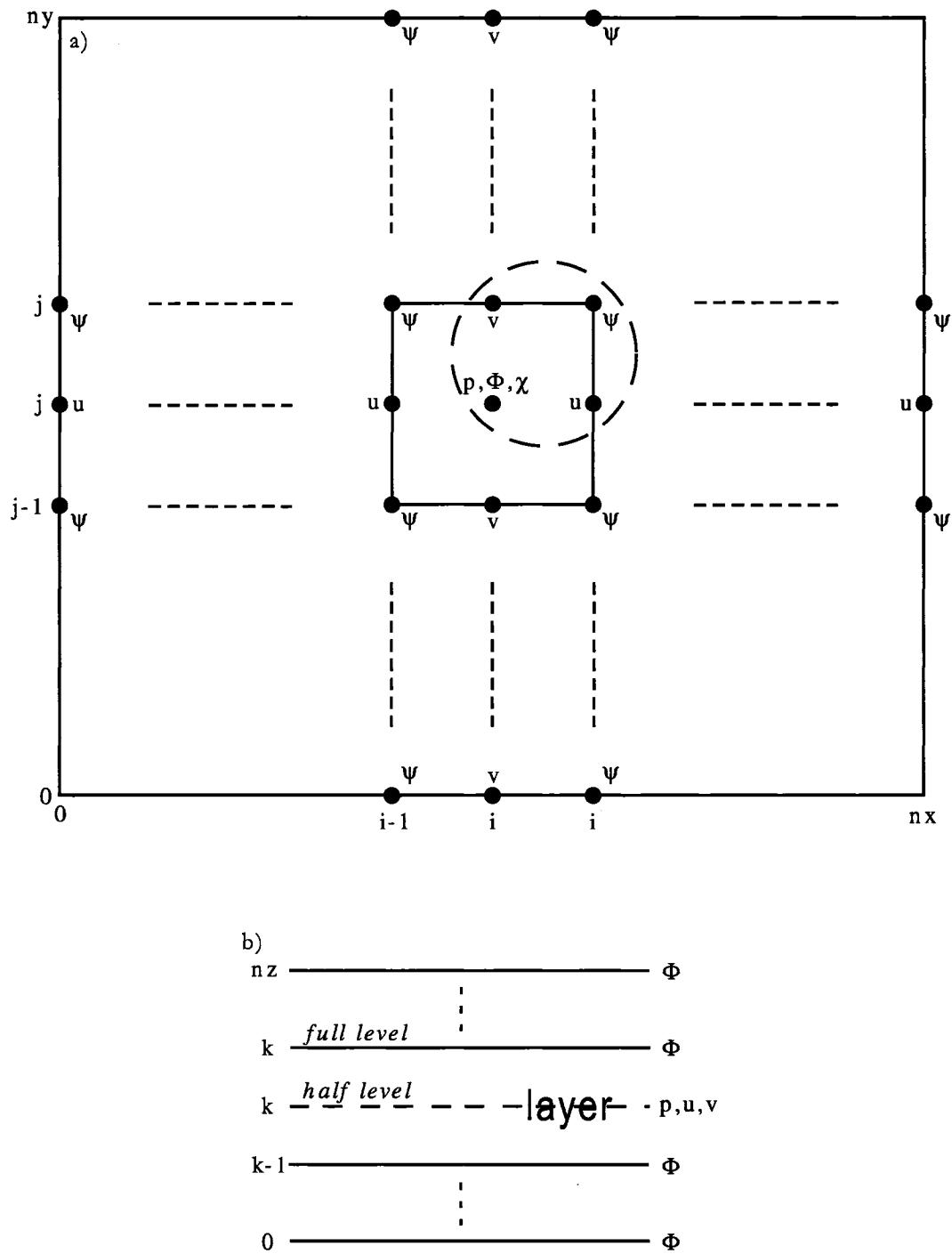


Figure 2.1: Schematic of the WRF-ARW grid. (a) The horizontal grid. The circle indicates the collection of grid points assigned index  $(i,j)$ , and the inner square represents grid box  $(i,j)$ . (b) The vertical grid.

is more explicitly defined later.) This wind can be partitioned as

$$\vec{\mathfrak{M}} = \vec{\mathfrak{M}}_\psi + \vec{\mathfrak{M}}_\chi = \hat{k} \times \nabla\psi + \nabla\chi, \quad (2.1)$$

where  $\psi$  is the streamfunction and  $\chi$  is the velocity potential. From Eq. (2.1), the Laplacian of the streamfunction (velocity potential) can be equated to the vorticity (divergence) of the mass-weighted wind, and the following boundary conditions may be derived:

$$\hat{s} \cdot \vec{\mathfrak{M}} = \frac{\partial\psi}{\partial n} + \frac{\partial\chi}{\partial s} \quad (2.2)$$

$$\hat{n} \cdot \vec{\mathfrak{M}} = -\frac{\partial\psi}{\partial s} + \frac{\partial\chi}{\partial n}, \quad (2.3)$$

where  $\hat{s}$  and  $\hat{n}$  are unit vectors in the directions tangential and normal to the boundary, respectively, and  $s$  and  $n$  are distances along those directions. The discrete forms for the relationships between  $\psi$ ,  $\chi$ , the vorticity  $\zeta$ , and the divergence  $\mathfrak{D}$  are:

$$\frac{(m_{i,j}^\psi)^2}{(\Delta x)^2} (\psi_{i-1,j} + \psi_{i+1,j} + \psi_{i,j-1} + \psi_{i,j+1} - 4\psi_{i,j}) = \zeta_{i,j} \quad (2.4)$$

$$\frac{(m_{i,j})^2}{(\Delta x)^2} (\chi_{i-1,j} + \chi_{i+1,j} + \chi_{i,j-1} + \chi_{i,j+1} - 4\chi_{i,j}) = \mathfrak{D}_{i,j}, \quad (2.5)$$

where  $m^\psi$  represents map factor values at the vorticity points, and  $\Delta x$  is the nominal grid spacing. Using these equations to solve for  $\psi$  and  $\chi$  on the interior of the domain requires knowledge of  $\psi$  and  $\chi$  along the boundaries, that is, when  $i = 0$ ,  $i = nx$ ,  $j = 0$ , or  $j = nx$ . (Refer to Fig. 2.1 for a depiction of the grid discretization.) However, the values of  $\vec{\mathfrak{M}}$  are known along the boundaries, so the discrete versions of Eqs. (2.2) and (2.3) may be used to determine  $\psi$  and  $\chi$  along the boundaries.

Putting it all together, then, we begin by setting  $\psi$  and  $\chi$  to zero everywhere and then solve Eq. (2.2) to determine values for  $\psi$  around the boundaries, keeping  $\psi_{0,1} = 0$  always. This is followed by solving Eq. (2.4) for interior values of  $\psi$  using successive over-relaxation for one iteration only. Boundary values of  $\chi$  are maintained at zero so that all of the harmonic component is assigned to the nondivergent flow. Eq. (2.5) is then solved (again for only one iteration) for interior values of  $\chi$ . This



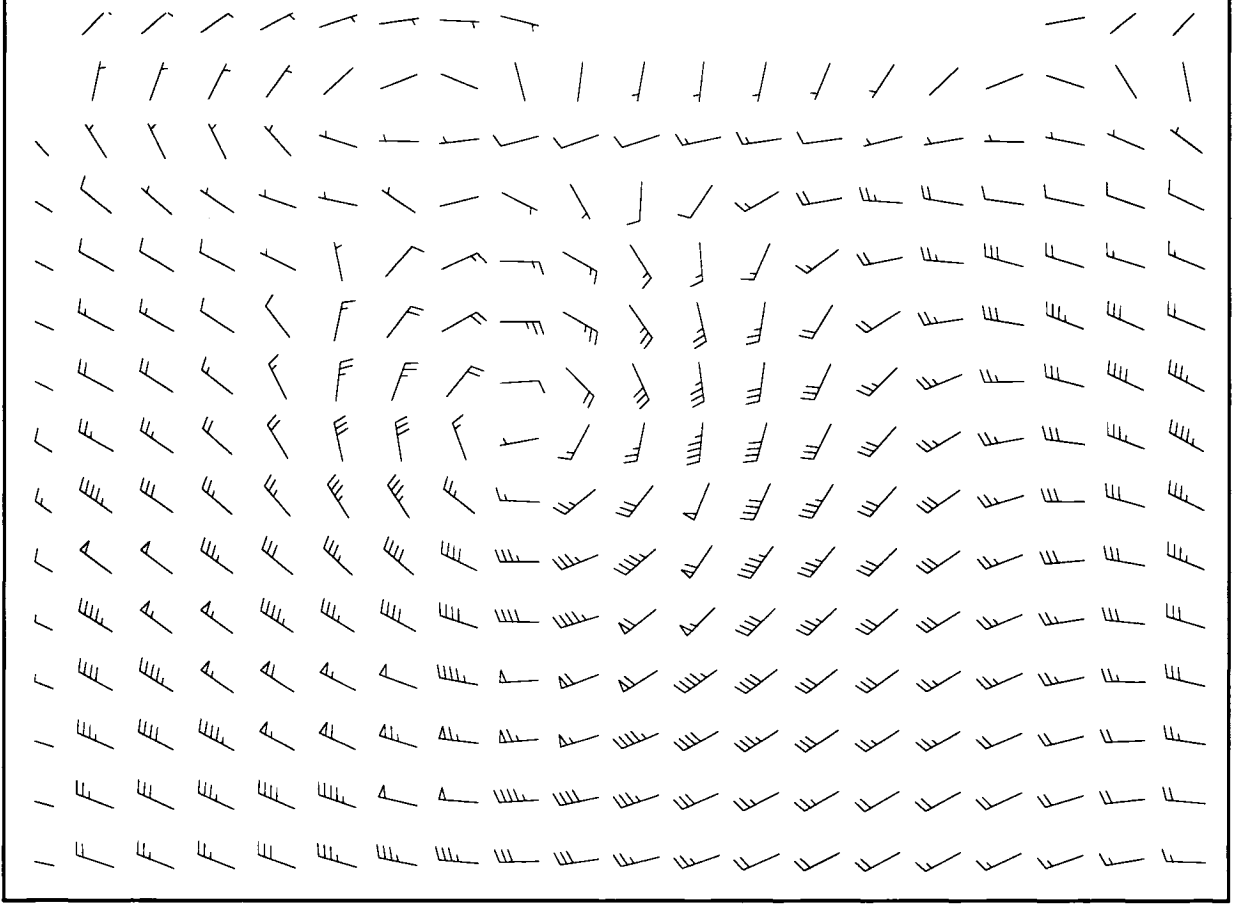


Figure 2.2: The initial data for the mass-weighted wind at around 500 hPa. The area displayed has been zoomed into an area of interest. Wind barbs represent magnitudes of  $50 \text{ m s}^{-1}$  for each flag,  $10 \text{ m s}^{-1}$  for each full barb, and  $5 \text{ m s}^{-1}$  for each half barb.

ends one step of the procedure, which may be repeated (by solving Eq. (2.2) again) as many times as necessary. The solution is deemed converged when the metric  $\sqrt{\sum (\mathfrak{M}'_1 - \mathfrak{M}_1)^2 + \sum (\mathfrak{M}'_2 - \mathfrak{M}_2)^2}$  is less than  $10^{-4} \text{ m s}^{-1}$ , where the primed quantities represent the wind components reconstituted from the fields of  $\psi$  and  $\chi$  according to Eq. (2.1). The process of updating the boundary conditions as  $\psi$  and  $\chi$  are being solved for leads to good convergence properties (Bijlsma et al. 1986).

As a test, the process above was carried out using the initial field of  $\vec{\mathfrak{M}}$  shown in Fig. 2.2. The dominant features in this example are a closed low near the center of the domain and a ridge downstream. Figure 2.3 displays the results of this test. The closed low and downstream

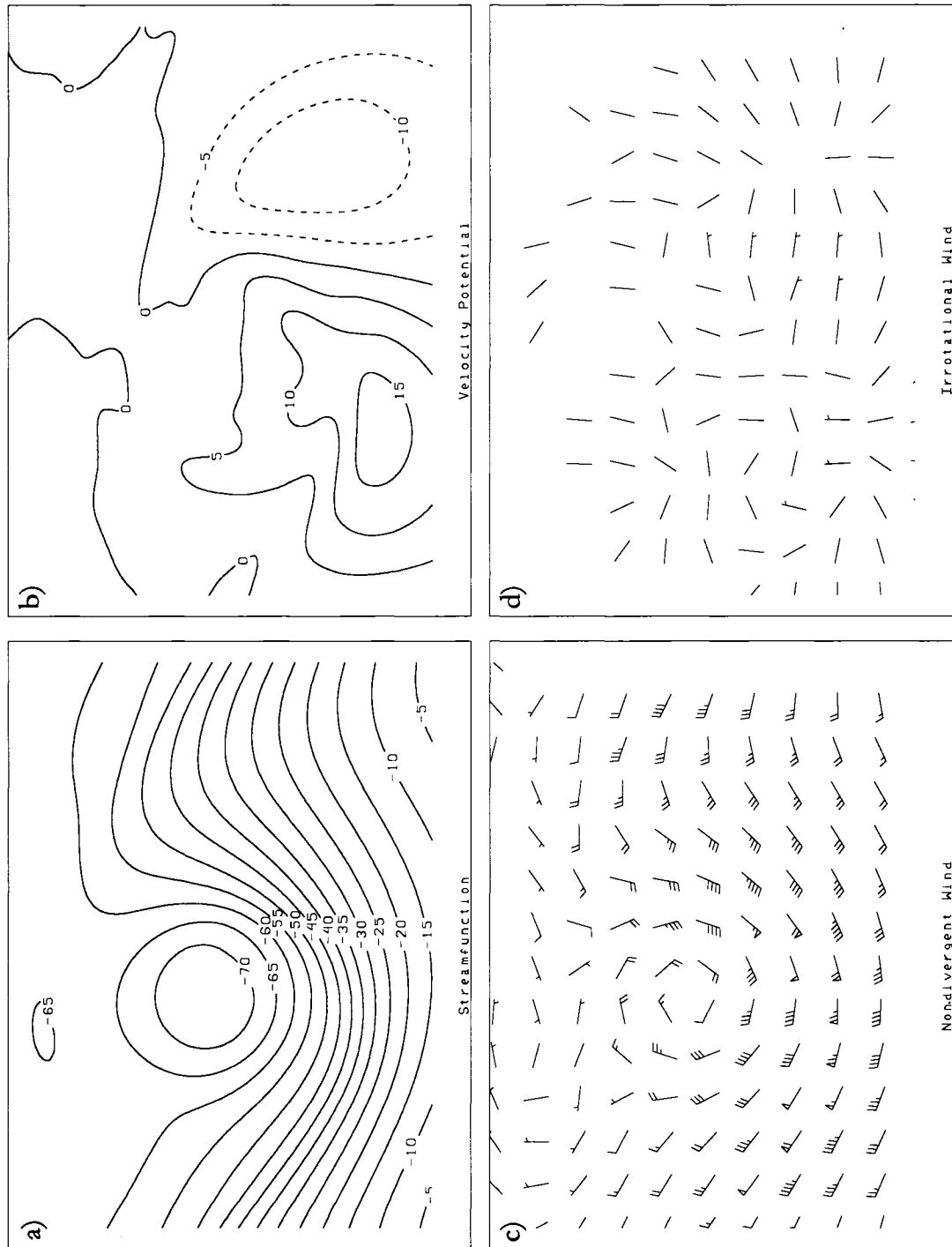


Figure 2.3: Results of partitioning the wind field in Fig. 2.2 in terms of (a) streamfunction (contour interval  $5 \times 10^6 \text{ m}^2 \text{ s}^{-1}$ ), (b) velocity potential [contour interval  $5 \times 10^5 \text{ m}^2 \text{ s}^{-1}$ , positive (negative) values solid (dashed)], (c) nondivergent wind, and (d) irrotational wind. The barb convention is as in Fig. 2.2.

ridge are captured very well by the streamfunction in Fig. 2.3a. The velocity potential (Fig. 2.3b) indicates divergent flow downstream of the trough and convergent flow further upstream. Since this example uses data from approximately the 500 hPa level, this makes sense synoptically. The bottom panels (Figs. 2.3c–d) clearly show that the nondivergent wind captures the bulk of the flow, with a small residual given by the irrotational wind. Again, this is not unexpected in typical large-scale midlatitude flow.

Calculations were performed for many different fields of  $\vec{\mathfrak{M}}$  (for instance, using different levels and/or times in one model run) that are not shown here. All produced good results. As a check, the velocity and divergence fields were derived from the reconstituted wind field and compared against their original values. The maximum errors at any given gridpoint rarely exceeded  $10^{-11} \text{ s}^{-1}$  for vorticity and  $10^{-19} \text{ s}^{-1}$  for divergence. As an additional check, the divergence of the nondivergent wind and the vorticity of the irrotational wind were both computed. The maximum absolute value of these quantities, which should be identically zero, rarely exceeded  $10^{-20} \text{ s}^{-1}$ .

## 2.2 Warping technique

The technique used to warp the PV distribution is based on that of Beier and Neely (1992). The first step in this technique is to define two images. Since we are warping PV, one of the images, known as the source image, will be an original distribution of PV in the form of a contour plot on an isentropic surface (or in a layer) that intersects the tropopause in the midlatitudes. The particular surface chosen will vary based on the season and situation, but typically a value around 315 K is used (e.g., Demirtas and Thorpe 1999). An example of such an image is given in Figure 2.4, which shows the near-tropopause PV distribution as analyzed by the Global Forecast System (GFS) model at 12 UTC 8 November 2005.

The second image to be defined, known as the destination image, contains information showing what the first image (i.e., the PV) should look like to some degree. There are a number of satellite images or products available that could be used to suggest what the PV distribution should look like, including the water vapor channel, ozone, and the specific humidity product of Wimmers and

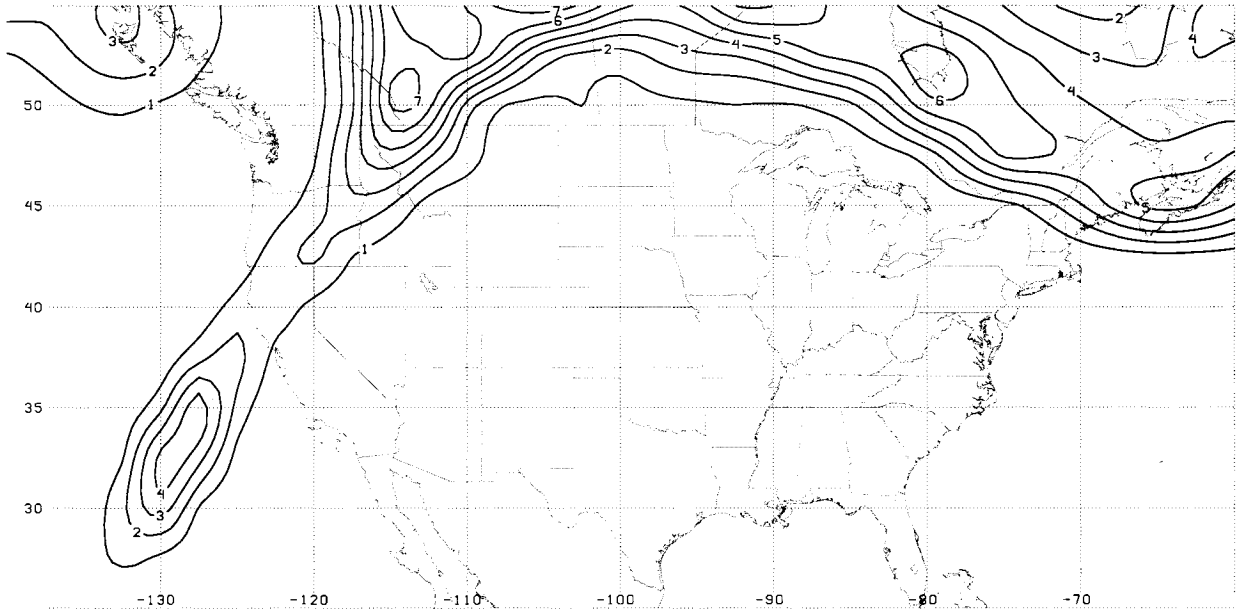


Figure 2.4: Potential vorticity in the 315–325 K layer (contour interval 1 PVU, 1 PVU =  $10 \text{ m}^2 \text{ K kg}^{-1} \text{ s}^{-1}$ ) as derived from the GFS model initialized at 12 UTC 8 Nov 2005.

Moody (2001). For instance, Davis et al. (1999) have shown that a correlation exists between PV and ozone anomalies<sup>1</sup>, particularly near the polar jet. This correlation exists because both ozone mixing ratio and PV have considerably larger magnitudes in the stratosphere relative to the troposphere (ozone action days excepted). In addition, they both act as passive tracers to a good degree. As a result, features and especially gradients in these fields are frequently co-located. As another option, upper-level specific humidity can be derived from (and is presumed to be an improvement upon) water vapor imagery. Upper-level specific humidity is similar to ozone in that it is of much different magnitude in the stratosphere than in the troposphere; the frigid temperatures at the tropopause effectively dehumidify the stratosphere. The difference is that specific humidity is inversely correlated to PV. In this work the specific humidity product of Wimmers and Moody (2001) is used due to its ready availability. The specific humidity product is a modification of the water vapor channel observed by geostationary satellites that aims to show the distribution of

<sup>1</sup>A correlation exists between the actual values of PV and ozone, but the correlation is strengthened when *anomalies* in PV and ozone are considered. This removes such effects as the climatological gradients in PV and ozone that exist between the poles and the equator.

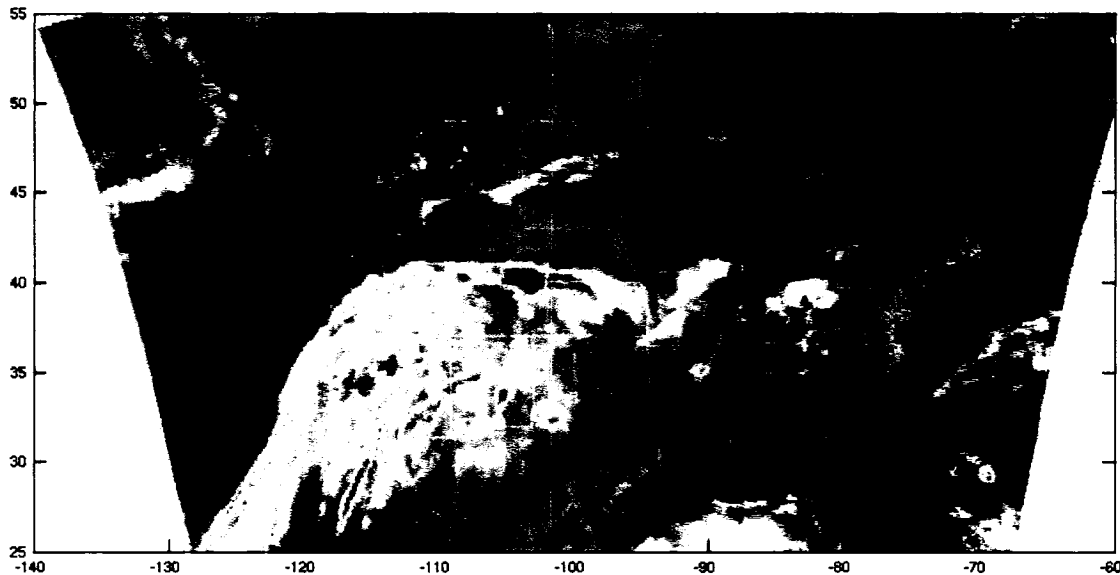


Figure 2.5: Distribution of specific humidity near the tropopause as derived from satellite imagery at 12 UTC 8 Nov 2005. Warmer (cooler) colors indicate greater (lesser) moisture content.

specific humidity in the upper troposphere and lower stratosphere (roughly from 200 to 500 hPa). Figure 2.5 shows an example of the specific humidity product valid at the same time as the PV distribution shown earlier. Notice that similar features appear in both figures.

Using the control line form of warping, we specify linear features that correspond in each image. In the case of PV and specific humidity, these features will typically be gradients. The features are identified as latitude/longitude pairs, and the warp takes place on a cylindrical equidistant grid. Figure 2.6 shows how the algorithm works in general. In this figure, two control lines have been specified,  $P_1Q_1$  and  $P_2Q_2$  in the destination image, and the corresponding  $P'_1Q'_1$  and  $P'_2Q'_2$  in the source image. This indicates that, for whatever reason, line 1 (and the feature it specifies) should be rotated counterclockwise and shortened, and line 2 should be rotated clockwise and displaced upward. The algorithm then proceeds as follows. For every point  $X$  in the destination image, perpendiculars are dropped to every control line. In the figure, these segments are denoted  $v_1$  and  $v_2$ . Additionally, the relative distance between  $Q$  and  $P$  along each control line where the

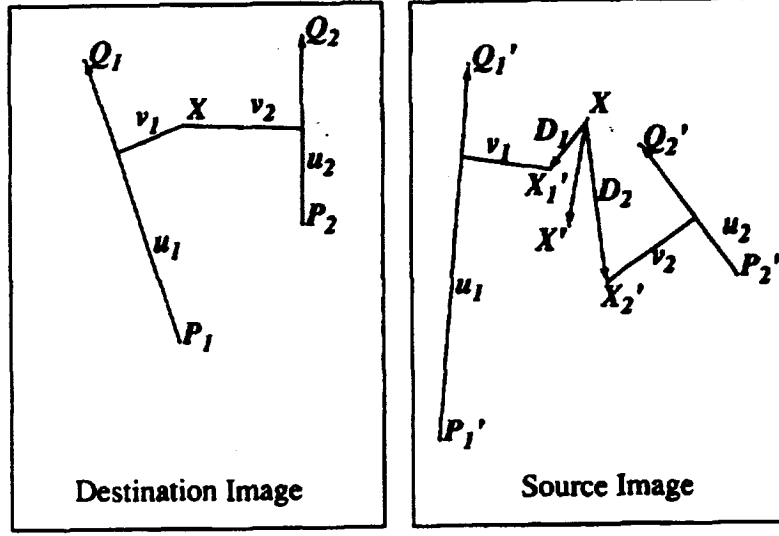


Figure 2.6: A schematic of the warping algorithm. From Beier and Neely (1992) Figure 3.

perpendicular intersects is measured and denoted by  $u_1$  and  $u_2$ . Then, for each control line, a new value for  $X$  is computed using the values of  $Q'$ ,  $P'$ ,  $u$ , and  $v$ . With two pairs of control lines, as shown in the figure, two new values for  $X$ ,  $X'_1$  and  $X'_2$ , are determined. Since these points are in general not coincident, a weighted average of all of the  $X'_n$  points must be undertaken to arrive at a single point  $X'$ . In the original algorithm, the weights are computed using the formula

$$w = \left( \frac{l^p}{a + d} \right)^b, \quad (2.6)$$

where  $l$  is the line's length,  $d$  the distance from  $X$  to the line,  $a$ ,  $b$ , and  $p$  are tunable parameters, and lengths are determined in terms of degrees latitude/longitude (equivalent to pixels on a cylindrical equidistant grid). The tunable parameters effect the weighting function as follows. The first parameter,  $a$ , is given in the same units as distance and determines how much of a factor distance to a control line is in determining the weight. A value of 0 for  $a$  maximizes the effect distance has on the weight, whereas  $a \gg \max d$  eliminates distance to the line as a factor. The parameter  $b$  accentuates the weight in the case where the point is close to a long line, while penalizing points that are far from a short line. A value of 0 for  $b$  assigns all lines equal weight; as  $b$  increases from

0, the accentuation becomes more and more prominent. Finally, the parameter  $p$  determines the degree to which the length of a control line effects its weight. A value of 0 for  $p$  eliminates line length as a factor, and increasing values increase the degree to which line length is important. Once  $X'$  has been determined, location  $X$  in the destination image is assigned the value at location  $X'$  in the source image.

Let us now examine the behavior of the Beier and Neely (1992) algorithm as applied to warping meteorological variables on a limited-area domain. In this example, geopotential heights are used with no rationale for the choices of control lines other than to show the effects of the algorithm. Figure 2.7 shows the 500-hPa geopotential heights analyzed at 12 UTC 7 June 2004 along with sets of control lines. Two primary features have been identified for warping. The first is the cutoff low over Oregon, and two line pairs have been associated with it. These line pairs have been set in an attempt to change the orientation of the low from the original east-west orientation to one that is more north-south. This is accomplished by moving the height gradient on the west side of the low eastward while bringing the southern portion of the height gradient further to the south. The second feature is the trough/ridge couplet over Manitoba and Ontario. In this case, the height field is rotated cyclonically in an attempt to increase the amplitude of the trough/ridge couplet. The six long lines placed in a hexagonal pattern around the region of interest are identical line pairs, which are used in an attempt to prevent the warp from changing the geopotential height distribution far afield.

Having set the control lines and provided an initial field of data (in this case, of geopotential heights), the warping algorithm is applied. The tunable parameters are set to  $a = 3$ ,  $b = 2$ , and  $p = \frac{1}{2}$ , although the results are relatively insensitive to the exact values chosen. The choices used here allow the control lines to reshape the PV in a region close to them (in a synoptic sense), while attempting to reduce their effect far afield. The result of the warp is shown in Figure 2.8. Notice that the cutoff low over Oregon now has a more north-south orientation, and the trough/ridge couplet north of the Great Lakes has been amplified, both of which we intended to accomplish via

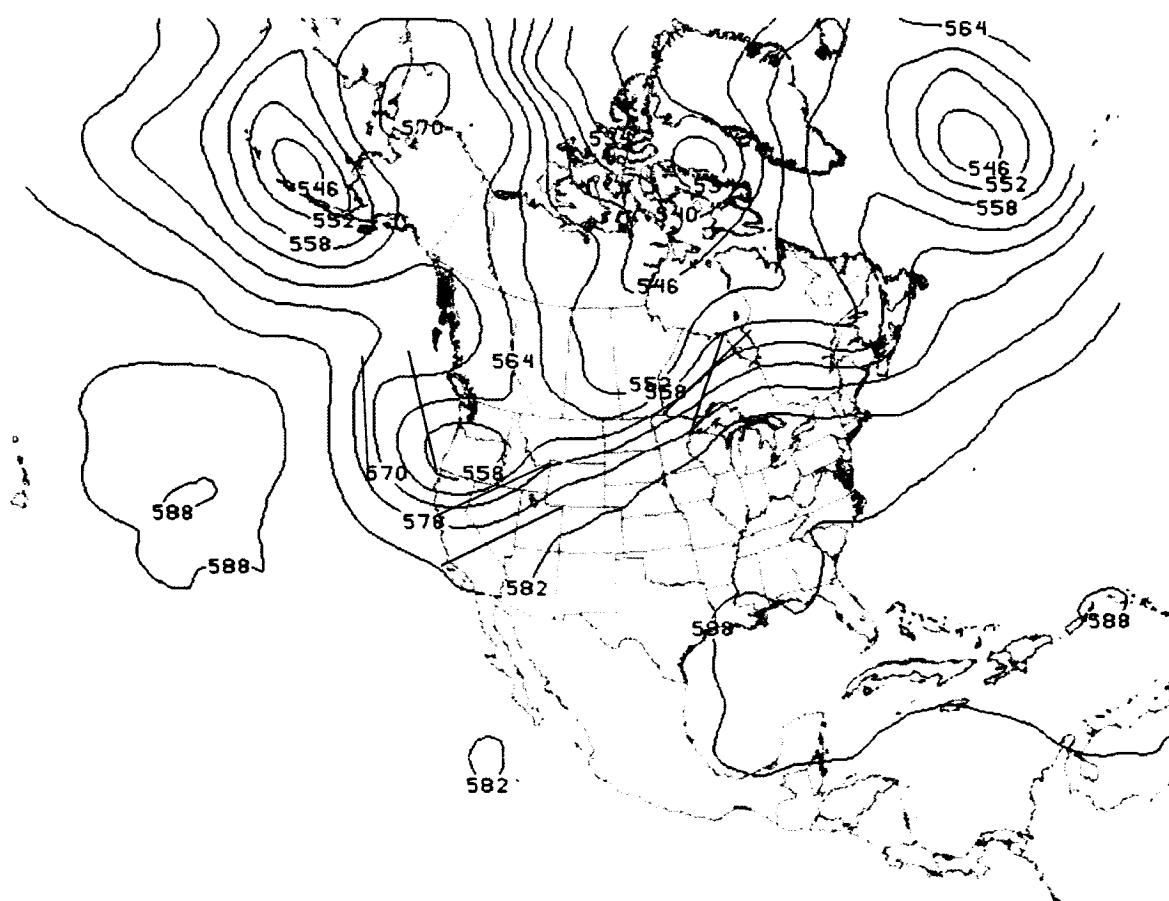


Figure 2.7: 500-hPa geopotential heights (pink, contoured every 6 dam) analyzed at 12 UTC 7 Jun 2004, along with control line pairs. Lines in dark orange identify features in the original geopotential height distribution, and corresponding lines in blue identify the locations to which those features should be moved. Yellow-orange lines indicate identical line pairs used in an attempt to localize the warp.



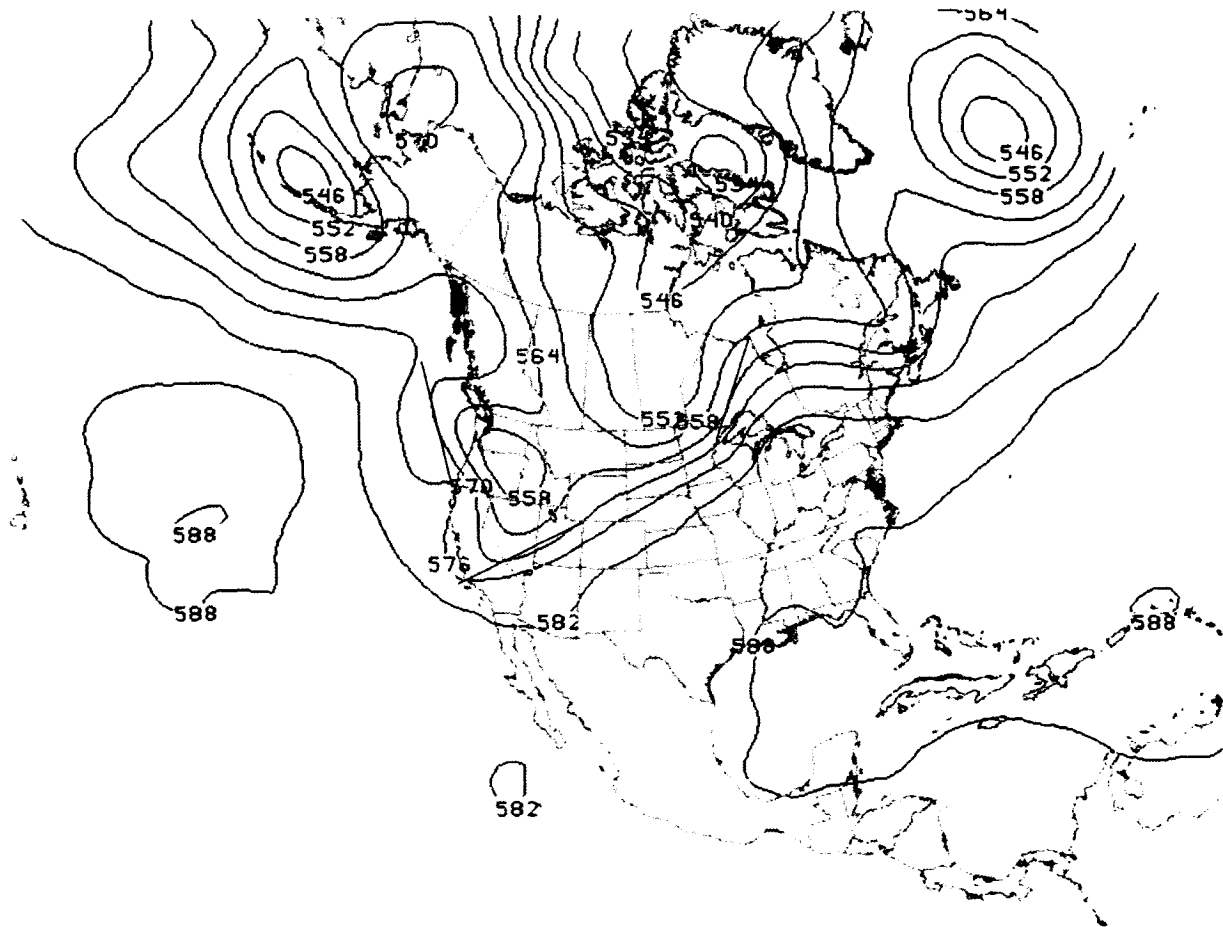


Figure 2.8: Warped 500-hPa geopotential heights (pink, contoured every 6 dam) valid at 12 UTC 7 Jun 2004, showing final control lines in dark orange and yellow-orange.

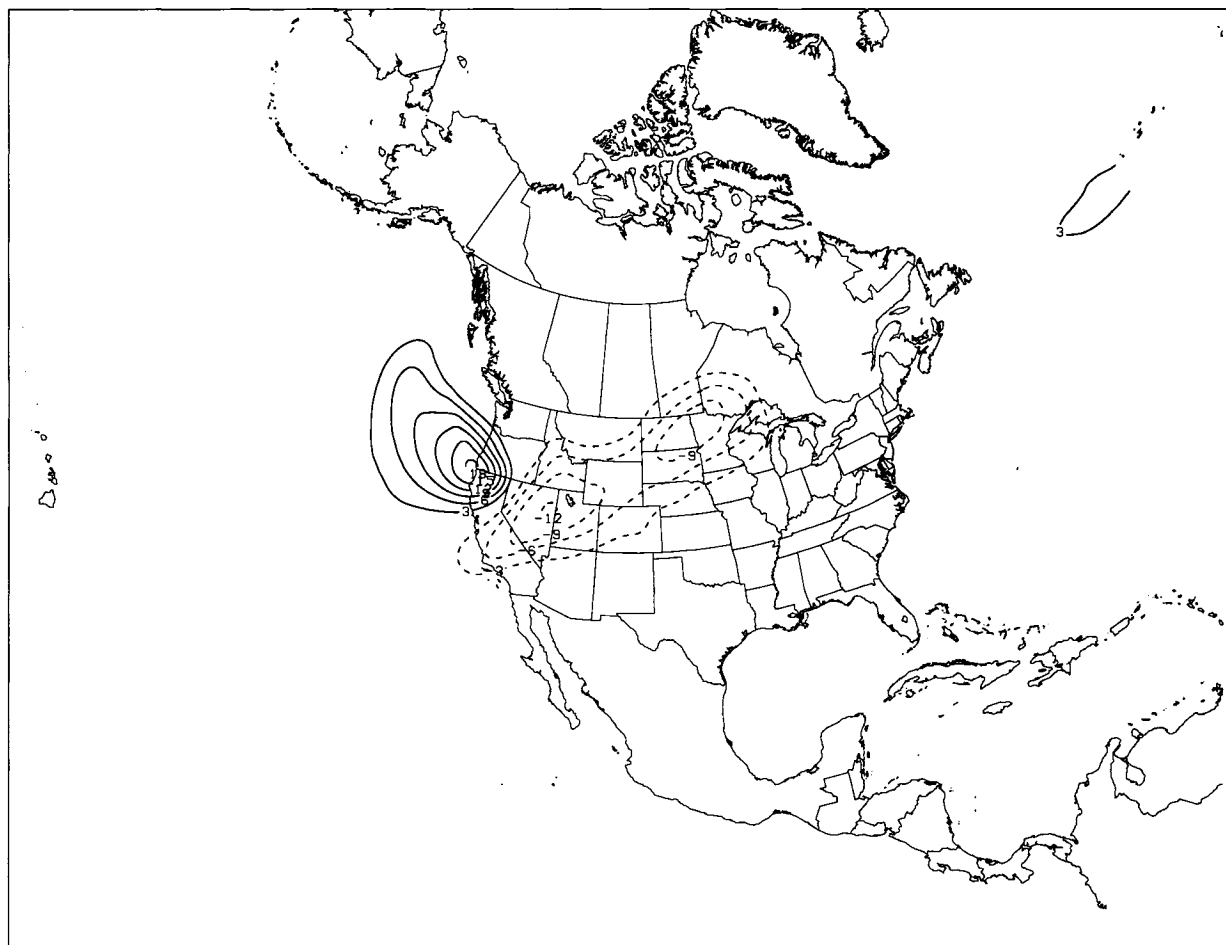


Figure 2.9: 500-hPa geopotential height perturbation [contour interval 3 dam, positive (negative) values solid (dashed)] introduced by the warping algorithm described in the text.

the warp<sup>2</sup>. However, closer inspection reveals that features quite distant from the control lines have also been warped slightly. In general, all features far afield have been rotated cyclonically to a small degree, with the exception of those features in very close proximity to the long, fixed control lines. For example, the cutoff low south of Iceland has been warped slightly to the north-northwest. This subtle shift in position is more apparent when the height perturbations associated with the warp are examined. Figure 2.9 shows these perturbations. Although the perturbations are concentrated in the areas intended, notice that a 3-dam contour is present east of Newfoundland, indicating that

<sup>2</sup>The gradient along the West Coast is not as smooth as it should be, probably because the control line was drawn too long. Since this section is primarily devoted to developing and testing the warping algorithm, an attempt to improve the warped shape has not been made.

heights were raised 3 dam in that area as a result of the warp. Clearly, such large perturbations far afield are undesirable.

To limit the warped field from changing in areas far from control lines, the weighting function in Eq. (2.6) was modified to

$$w = \begin{cases} \left(\frac{lp}{a+d}\right)^b & d < 5^\circ \\ 0 & d \geq 5^\circ \end{cases} \quad (2.7)$$

This modified weighting function was then used to warp the PV field previously shown in Fig. 2.4, and the results are displayed in Fig. 2.10. The top panel combines Figs. 2.4 and 2.5, making it easy to compare how the potential vorticity field matches up with the specific humidity field derived from satellite observations. In general, drier air (the cooler colors) matches higher PV values, as expected. However, there are some notable differences. In the Gulf of Alaska, for instance, a PV maximum is displaced to the east of a minimum in specific humidity. Further to the south, an axis of higher PV appears too straight relative to the curvature apparent in the nearly co-located axis of dry air. Turning our attention to eastern Canada, the PV axis is displaced to the north of the axis of dry air.

In an attempt to better match the PV distribution to the satellite observations, control lines were drawn (not shown), focusing on the PV maximum off the California coast and the axis across eastern Canada. The mismatch over the Gulf of Alaska was ignored because that region intersected the boundary of the domain, where any warp would be ill-defined. The revised warping procedure was applied, and Fig. 2.10b presents the results. Notice how the shape of the PV axis now better matches the structure visible in the specific humidity field over both the West Coast (in particular the California–Oregon border) and eastern Canada. Also note that, in contrast to the earlier warping test, contours far from the warped areas are unchanged. This is confirmed in Fig. 2.10c, which shows the perturbations made to the PV field due to the warp.

The distribution of these perturbations is much more complex than those used in studies such as Roebber et al. (2002) while being easier to perform compared to the process used in Demirtas and Thorpe (1999) and Swarbrick (2001), demonstrating the advantage of using a warp to alter

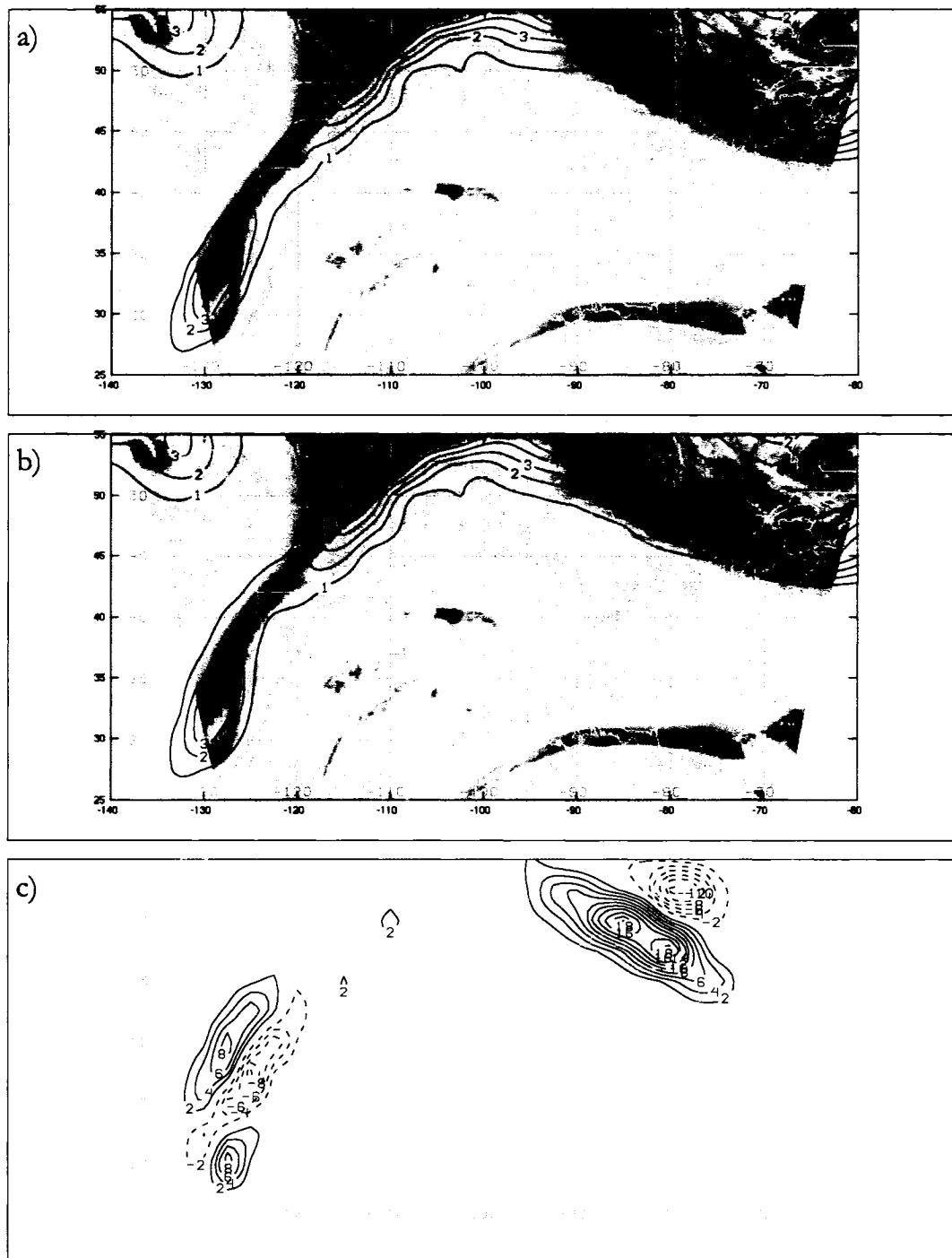


Figure 2.10: Warping a PV field using the revised weighting function. (a) An overlay of Fig. 2.4 on top of Fig. 2.5. (b) Same as (a) except after the warp has been applied. (c) The PV perturbations introduced by the warp [contour interval  $2 \times 10^{-7}$  PVU, positive (negative) values solid (dashed)].

fields of PV. One disadvantage of warping, however, is that it is difficult to change the magnitude of PV extrema. If the PV does not exceed, for example, 5 PVU, no amount of warping can produce a region where the PV is 6 PVU. For the application of morphological data assimilation, this is not a major concern, since information about PV magnitudes (as opposed to PV gradients) cannot easily be determined based on satellite observations alone. Nevertheless, an extension of this method to include the third dimension would be useful, especially for applications in which what-if experiments are being performed (as in Roebber et al. 2002), since the stratospheric PV reservoir could be tapped in that case to, say, double the value of PV at a certain level.

## 2.3 PV inversion

Once a distribution of potential vorticity has been determined via warping, it must be inverted to provide new values for the WRF state variables (e.g., geopotential, wind, temperature). One might think that the popular DE method would be well suited for this purpose. This is problematic, however, for a variety of reasons. First, the DE method is carried out in Exner function (i.e., pressure) coordinates, whereas the WRF model uses a terrain-following vertical coordinate. Although interpolation to pressure coordinates is possible, it is not optimal since some loss in accuracy is inevitable, and the forecast domain will include mountainous terrain such as the Rocky Mountains, making the lower boundary perhaps difficult to handle. It should be noted, however, that some authors have used the DE method over highly variable terrain (e.g., Chang et al. (2000) over China) with no apparent problems. The author's personal experiences with using the DE method in areas of steep terrain suggest, on the other hand, that fields such as the geopotential heights derived from the inversion can oscillate wildly in an exaggerated diurnal cycle, as can be seen in Fig. 3.44 of Decker (2003). An additional problem with the DE method is that it frequently does not converge when the grid spacing is less than about 100 km. Since modern NWP is performed at grid spacings on the order of 10 km, this would result in additional interpolation, this time between small-scale and large-scale grids.

In an attempt to avoid these issues, the approach taken here is to extend the variational

approach of Arbogast and Joly (1998) to Ertel PV on the three-dimensional WRF domain. With this approach, a potential vorticity operator is defined such that, given a state vector  $\mathbf{X}$  containing values of streamfunction and geopotential at every gridpoint in the model domain for which they are defined (recall that, according to Fig. 2.1, streamfunction and geopotential are staggered relative to each other), the application of the PV operator on  $\mathbf{X}$  results in a new vector representing the value of the potential vorticity in each WRF grid box. A second operator, the nonlinear imbalance operator, is also defined that operates on  $\mathbf{X}$ . Application of this operator on  $\mathbf{X}$  results in a new vector representing the degree to which each grid box is in nonlinear balance. A cost function is then defined to be

$$J = \frac{1}{2} \langle \tilde{P}\mathbf{X} - \hat{P}, \tilde{P}\mathbf{X} - \hat{P} \rangle + \frac{1}{2} \varepsilon \langle \tilde{B}\mathbf{X}, \tilde{B}\mathbf{X} \rangle, \quad (2.8)$$

where  $\tilde{P}$  is the PV operator,  $\hat{P}$  is the given PV distribution (i.e., the PV to be inverted),  $\varepsilon$  is a constant to give each term in the cost function equal weighting, and  $\tilde{B}$  is the nonlinear balance operator. The inner product is taken to be a simple dot product. The gradient of  $J$  with respect to the state vector  $\mathbf{X}$  is then

$$\nabla J = P^* (\tilde{P}\mathbf{X} - \hat{P}) + \varepsilon B^* \tilde{B}\mathbf{X}, \quad (2.9)$$

where the asterisks denote the adjoints of the linearized operators. This gradient is then used to minimize  $J$  using the method of steepest descent, or, more generally, any quasi-Newton method. Further derivations and details regarding these operators and the minimization process are contained in subsequent chapters.

## 2.4 Test Cases

Throughout the development of the techniques discussed herein, multiple test cases, ranging in complexity from an idealized run to a CONUS-scale high-resolution run, have been utilized, and they are summarized in this section.

### 2.4.1 WRF idealized baroclinic wave

The simplest test case (hereafter called Test Case 1) is distributed with the WRF model under the name “em\_b\_wave.” It simulates a baroclinic wave growing on a baroclinically unstable jet on an  $f$ -plane. Periodic boundary conditions are used along the east and west boundaries, with symmetric boundary conditions along the north and south boundaries. 100-km grid spacing is used, with 41 grid boxes in the zonal direction, 81 grid boxes in the meridional direction, and 64 vertical layers. All map factors are identically one in this test case, so any errors in the implementation of the various operators with respect to the map factors will not be apparent. Furthermore, with no terrain in the model, each terrain-following coordinate closely approximates an isobaric surface.

Figure 2.11 shows how the baroclinic wave grows exponentially during the simulation. What is a hardly noticeable perturbation up to 33 h into the run (Figs. 2.11a–b) becomes a strong shortwave by hour 75 (Figs. 2.11c–d) before becoming an extremely potent closed low only 42 hours later (Figs. 2.11e–f).

Test Case 1 was used to provide input data for the wind field partitioning test described in Section 2.1.1.

### 2.4.2 North Pacific system

For a more complicated test case (hereafter called Test Case 2), a real-world domain was chosen over the North Pacific Ocean, centered on 40°N, 150°W. This area of the world is known for frequent cyclones, yet the absence of topography allows for further testing to be performed in a simplified environment. Again the grid spacing was set to 100 km, and a relatively small domain of 34×26 grid boxes was used, with 20 equally spaced vertical levels. This small domain reduces the amount of computation time needed during testing and implementation of the PV inversion technique.

The WRF model was initialized at 1200 UTC 5 Apr 2006, and a 24-hr forecast was produced. The initial time (Fig. 2.12) was characterized by a large-scale trough at 500 hPa along the northern portion of the domain. Within this larger trough, smaller shortwaves and absolute vorticity maxima

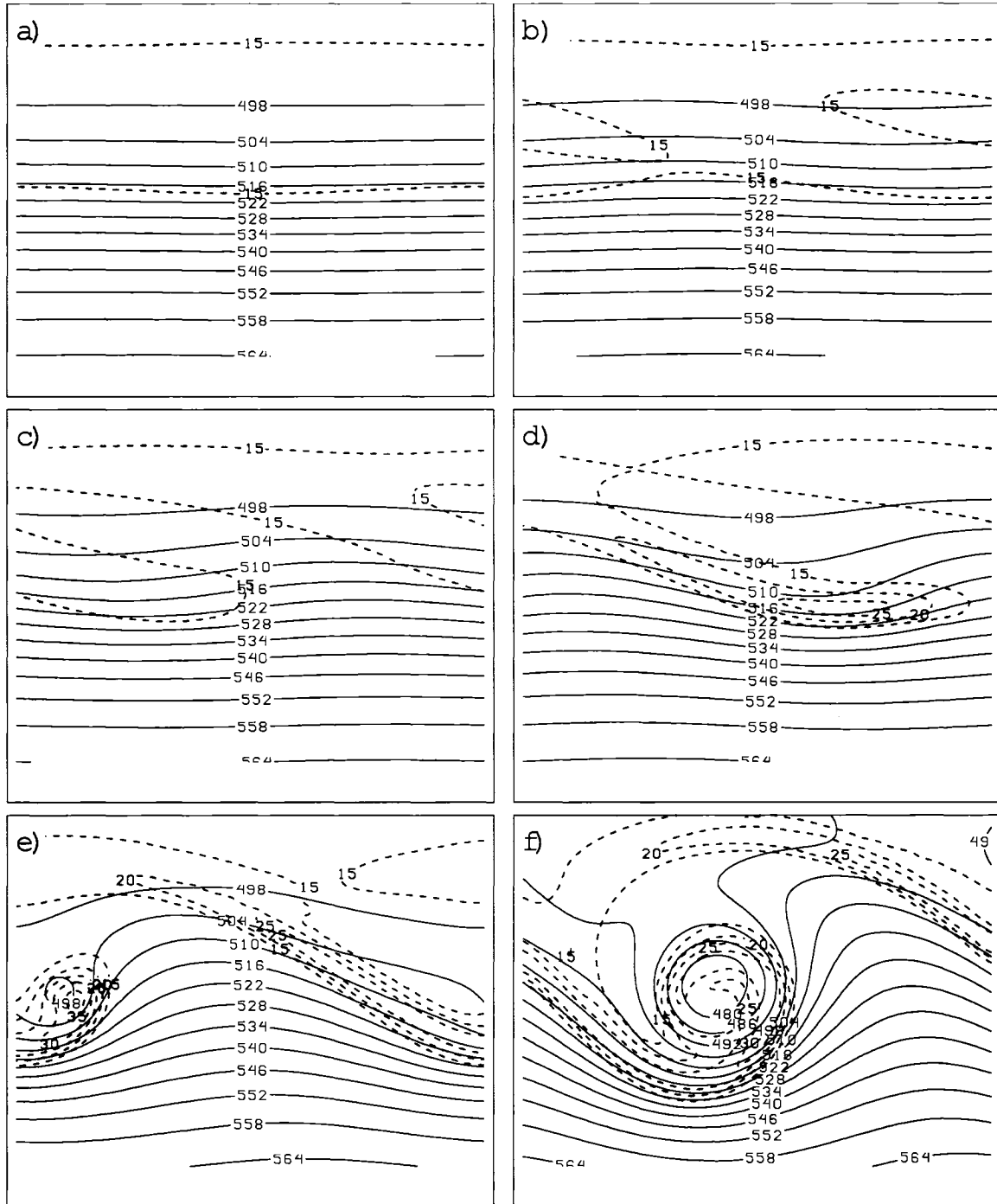


Figure 2.11: 500-hPa geopotential heights (contour interval 6 dam, solid) and absolute vorticity (contour interval  $5 \times 10^{-5} \text{ s}^{-1}$ , starting at  $15 \times 10^{-5} \text{ s}^{-1}$ , dashed) at (a) 12 h, (b) 33 h, (c) 54 h, (d) 75 h, (e) 96 h, and (f) 117 h into the WRF model's "em\_b\_wave" idealized simulation. The northern and southern portions of the domain are excluded to focus on the baroclinic wave.



were evident. Near the surface (Fig. 2.12b), a cyclone was located along the northern boundary directly below the 500-hPa height minimum. This fact, along with the lack of temperature contrast along the trough extending southeastward from the 1000-hPa height minimum, indicated that the cyclone was well-occluded. The triple point was located just north of  $40^{\circ}\text{N}$ ,  $140^{\circ}\text{W}$  (Fig. 2.12b), and its location just east of a 500-hPa vorticity maximum (Fig. 2.12a) suggested that a secondary development could occur in that area. Trailing southwest from the triple point was a cold front and broad baroclinic zone, marked by a height trough especially in the southwest.

Twelve hours into the forecast (Fig. 2.13), the westernmost vorticity maximum at 500-hPa started to become dominant (Fig. 2.13a), while at 1000 hPa (Fig. 2.13b), the primary cyclone filled as the secondary development almost became closed off at the peak of the warm sector. A long cold front continued to extend towards the southwest, and although the temperature gradient remained strong in the immediate vicinity of the front, the broad baroclinic zone to the north began to diffuse. All subsequent references to Test Case 2 will focus on this forecast time.

By the end of the forecast period (Fig. 2.14), the vorticity maxima associated with the 500-hPa low had become consolidated (Fig. 2.14a), and the secondary development had become the primary cyclone at 1000 hPa (Fig. 2.14b). The cold front continued to push toward the southernmost portions of the domain, and it retained a remarkably strong temperature gradient along its leading edge considering the oceanic nature of the domain.

### 2.4.3 Continental United States

A final test domain has been created that covers the continental United States with 24-km grid spacing. This domain consists of 210 grid boxes in the zonal direction, 136 grid boxes in the meridional direction, and 49 vertical layers. It is intended that the pronounced topography and relatively-high resolution of this domain will serve as a stringent test of the various techniques described below. One forecast (Test Case 3) has been run on this domain, initialized at 1200 UTC 24 Mar 2006. However, it has not yet been used extensively.

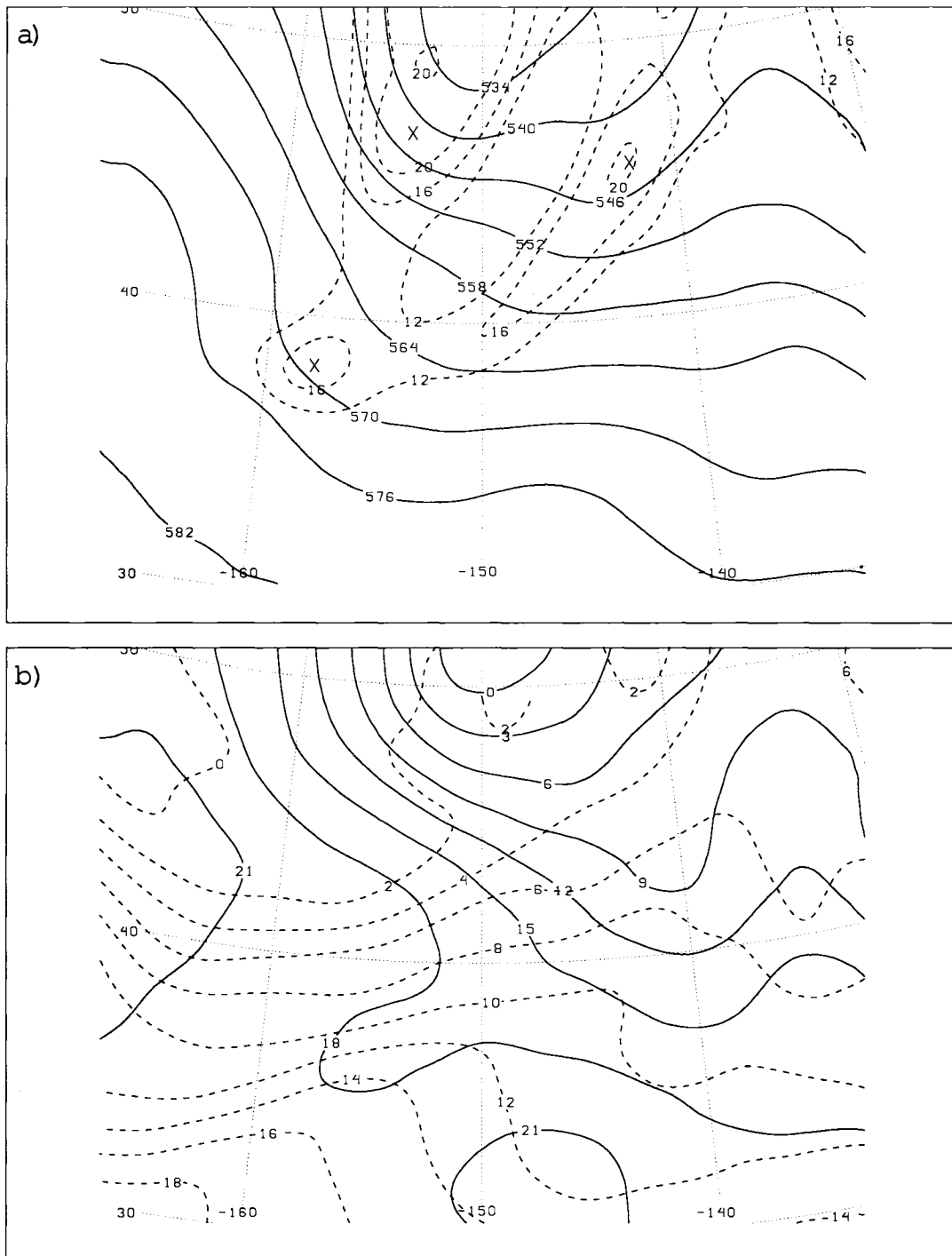


Figure 2.12: Test Case 2 at 1200 UTC 5 Apr 2006. (a) 500-hPa geopotential heights (contour interval 6 dam, solid) and absolute vorticity (contour interval  $4 \times 10^{-5} \text{ s}^{-1}$ , starting at  $12 \times 10^{-5} \text{ s}^{-1}$ , dashed). Vorticity maxima are marked by X. (b) 1000-hPa geopotential heights (contour interval 3 dam, solid) and temperatures (contour interval  $2^\circ\text{C}$ , dashed).

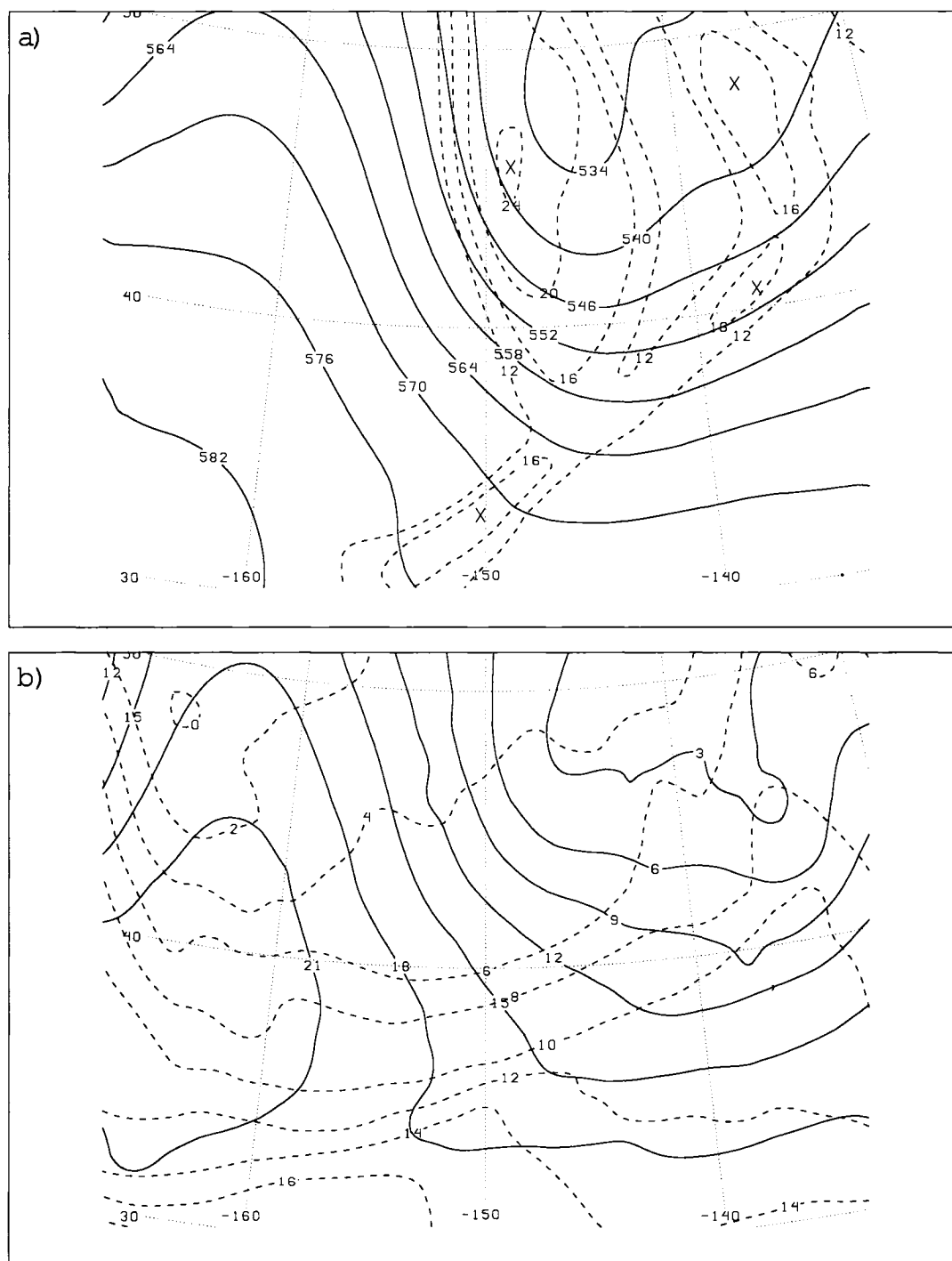


Figure 2.13: As for Fig. 2.12, but at 0000 UTC 6 Apr 2006.

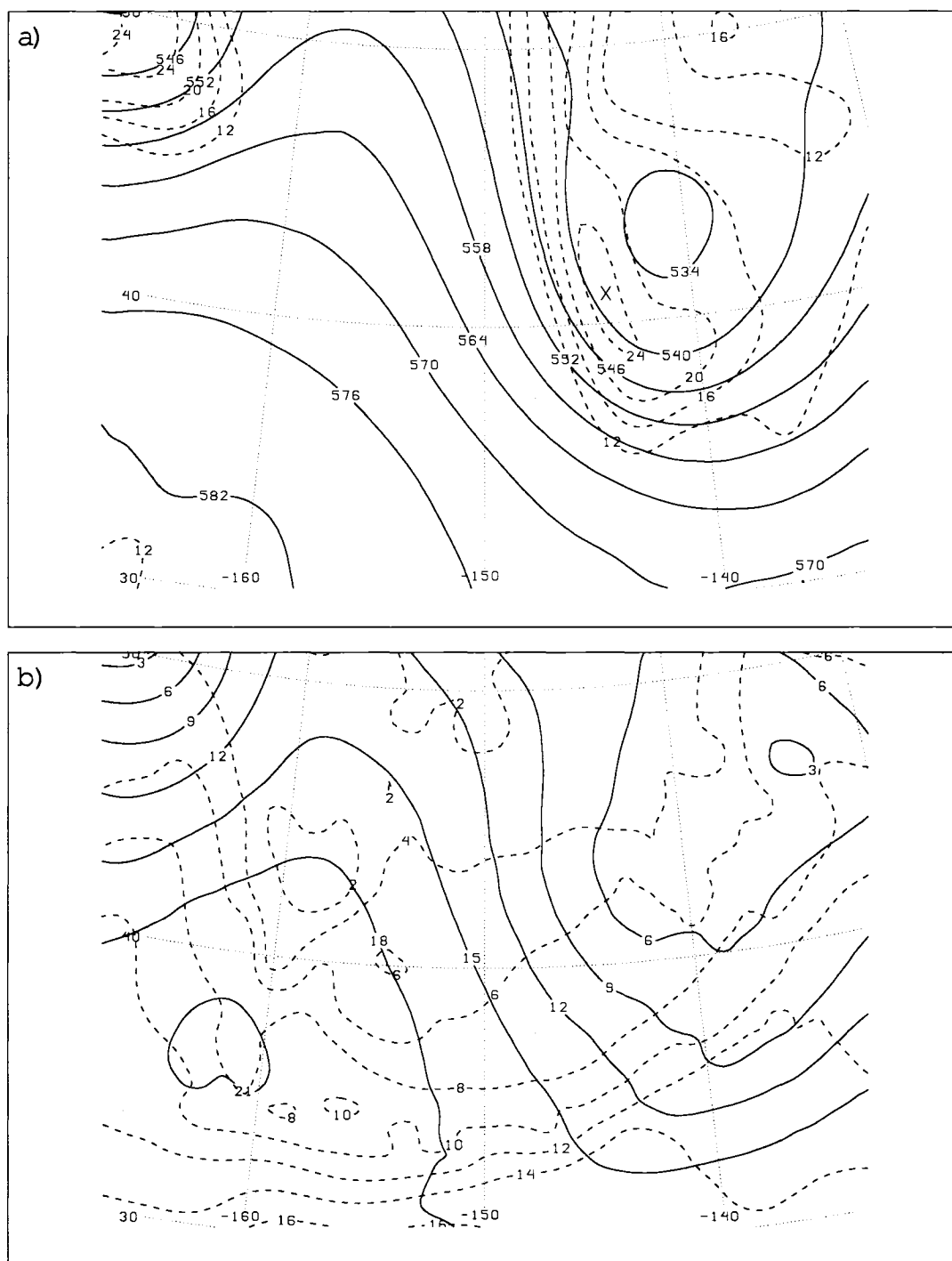


Figure 2.14: As for Fig. 2.12, but at 1200 UTC 6 Apr 2006.

## Chapter 3

# The nonlinear imbalance operator and its adjoint

### 3.1 Balance equation in WRF vertical coordinate

#### 3.1.1 Expansion of divergence equation

The WRF vertical coordinate is defined as

$$\eta \equiv \frac{p_d - p_t}{p_s - p_t}, \quad (3.1)$$

where  $p_d$  is the dry hydrostatic pressure, and  $p_s$  and  $p_t$  refer to values of the dry hydrostatic pressure at the surface and model top, respectively. The dry hydrostatic pressure is a measure of the weight of the dry air in an atmospheric column above a given point. In particular,  $\mu \equiv p_s - p_t$  is directly proportional to the weight of the dry air in an atmospheric column above a surface grid box.

Using this vertical coordinate and setting the vertical velocity to zero, the horizontal momentum

equations can be written as (Skamarock et al. 2005)

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{\mu u}{m} \right) + m \frac{\partial}{\partial x} \left( \frac{\mu u u}{m} \right) + m \frac{\partial}{\partial y} \left( \frac{\mu v u}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{\mu \dot{\eta} u}{m} \right) + \mu \alpha \chi \frac{\partial p}{\partial x} + \chi \frac{\partial p}{\partial \eta} \frac{\partial \Phi}{\partial x} \\ = \frac{\mu}{m} \left( f + u \frac{\partial m}{\partial y} - v \frac{\partial m}{\partial x} \right) v \end{aligned} \quad (3.2a)$$

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{\mu v}{m} \right) + m \frac{\partial}{\partial x} \left( \frac{\mu u v}{m} \right) + m \frac{\partial}{\partial y} \left( \frac{\mu v v}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{\mu \dot{\eta} v}{m} \right) + \mu \alpha \chi \frac{\partial p}{\partial y} + \chi \frac{\partial p}{\partial \eta} \frac{\partial \Phi}{\partial y} \\ = -\frac{\mu}{m} \left( f + u \frac{\partial m}{\partial y} - v \frac{\partial m}{\partial x} \right) u. \end{aligned} \quad (3.2b)$$

In these equations,  $u$  and  $v$  are the zonal and meridional winds, respectively, on the computational grid,  $\dot{\eta} \equiv \frac{d\eta}{dt}$ ,  $\alpha$  is the specific volume of dry air,  $p$  is the total pressure,  $\Phi$  is the geopotential, and  $x$  and  $y$  are the computational coordinates. The map-scale factors are given by  $m \equiv \frac{\Delta x}{\text{Earth Distance}}$ , where  $\Delta x$  is the constant horizontal grid spacing. Finally,  $\chi \equiv (1 + q_v + q_c + q_r + q_i + \dots)^{-1}$ , where  $q_{v,c,r,i}$  are the mixing ratios for water vapor, cloud, rain, ice, etc.

The use of indicial notation (i.e., the Einstein summation convention) greatly simplifies the derivation. See Kundu (1990) for details regarding this notation. Eqs. (3.2a) and (3.2b) can be combined into one equation using indicial notation, resulting in

$$\frac{\partial}{\partial t} \left( \frac{\mu u_i}{m} \right) + m \frac{\partial}{\partial x_j} \left( \frac{\mu u_j u_i}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{\mu \dot{\eta} u_i}{m} \right) + \mu \alpha \chi \frac{\partial p}{\partial x_i} + \chi \frac{\partial p}{\partial \eta} \frac{\partial \Phi}{\partial x_i} = \frac{\mu}{m} \left( f + \epsilon_{kl3} u_k \frac{\partial m}{\partial x_l} \right) \epsilon_{ij3} u_j, \quad (3.3)$$

where  $\epsilon_{ijk}$  is the permutation symbol, the indices can take values of 1 or 2,  $u_1 \equiv u$ ,  $u_2 \equiv v$ ,  $x_1 \equiv x$ , and  $x_2 \equiv y$ . We can clean this equation up in a few ways. First, we'll normalize the equation, and then we'll eliminate  $\alpha$  and  $p$ . We'll also make use of the continuity equation, which can be written as

$$\frac{\partial \mu}{\partial t} + m^2 \frac{\partial}{\partial x_j} \left( \frac{\mu u_j}{m} \right) + m \frac{\partial}{\partial \eta} \left( \frac{\mu \dot{\eta}}{m} \right) = 0. \quad (3.4)$$

As a first step, we divide both equations by the constant  $p_0 - p_t$ , and multiply Eq. (3.4) by  $u_i$ .

The resulting equations are

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{Mu_i}{m} \right) + m \frac{\partial}{\partial x_j} \left( \frac{Mu_j u_i}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{M \dot{\eta} u_i}{m} \right) + M \alpha \chi \frac{\partial p}{\partial x_i} + \chi \frac{M}{\mu} \frac{\partial p}{\partial \eta} \frac{\partial \Phi}{\partial x_i} \\ = \frac{M}{m} \left( f + \epsilon_{kl3} u_k \frac{\partial m}{\partial x_l} \right) \epsilon_{ij3} u_j \end{aligned} \quad (3.5)$$

and

$$u_i \frac{\partial M}{\partial t} + m^2 u_i \frac{\partial}{\partial x_j} \left( \frac{Mu_j}{m} \right) + m u_i \frac{\partial}{\partial \eta} \left( \frac{M \dot{\eta}}{m} \right) = 0, \quad (3.6)$$

where  $M \equiv \frac{p_s - p_t}{p_0 - p_t} = \frac{\mu}{p_0 - p_t}$  and  $p_0$  is a reference pressure (say, 1000 hPa).

Now the hydrostatic equation can be written as

$$\frac{\partial \Phi}{\partial \eta} = -\alpha \mu. \quad (3.7)$$

In addition, using Eq. (3.1) we have the identities  $\frac{\partial p}{\partial \eta} = \frac{\partial(p_d + p_v)}{\partial \eta} = \mu + \frac{\partial p_v}{\partial \eta}$  and  $\frac{\partial p}{\partial x_i} = \eta \frac{\partial \mu}{\partial x_i} + \frac{\partial p_v}{\partial x_i}$ , where  $p_v$  is the vapor pressure. Substituting these identities and Eq. (3.7) into the LHS of Eq. (3.5) and simplifying results in

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{Mu_i}{m} \right) + m \frac{\partial}{\partial x_j} \left( \frac{Mu_j u_i}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{M \dot{\eta} u_i}{m} \right) \\ + M \left( -\frac{1}{\mu} \frac{\partial \Phi}{\partial \eta} \right) \chi \left( \eta \frac{\partial \mu}{\partial x_i} + \frac{\partial p_v}{\partial x_i} \right) + \chi \frac{M}{\mu} \left( \mu + \frac{\partial p_v}{\partial \eta} \right) \frac{\partial \Phi}{\partial x_i} = \frac{M}{m} \left( f + \epsilon_{kl3} u_k \frac{\partial m}{\partial x_l} \right) \epsilon_{ij3} u_j \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{Mu_i}{m} \right) + m \frac{\partial}{\partial x_j} \left( \frac{Mu_j u_i}{m} \right) + \frac{\partial}{\partial \eta} \left( \frac{M \dot{\eta} u_i}{m} \right) \\ + \chi \left[ \left( 1 + \frac{1}{\mu} \frac{\partial p_v}{\partial \eta} \right) M \frac{\partial \Phi}{\partial x_i} - \left( \eta \frac{\partial M}{\partial x_i} + \frac{M}{\mu} \frac{\partial p_v}{\partial x_i} \right) \frac{\partial \Phi}{\partial \eta} \right] = \frac{M}{m} \left( f + \epsilon_{kl3} u_k \frac{\partial m}{\partial x_l} \right) \epsilon_{ij3} u_j. \end{aligned} \quad (3.8)$$

In obtaining the balance equation, we'll first consider a Cartesian frame ( $m = 1$ ) for simplicity. Later, we will transform our result to the WRF grid. With  $m = 1$ , Eqs. (3.8) and (3.6) simplify to

$$\frac{\partial}{\partial t} (Mu_i) + \frac{\partial}{\partial x_j} (Mu_j u_i) + \frac{\partial}{\partial \eta} (M \dot{\eta} u_i) - M f \epsilon_{ij3} u_j = -P_i \quad (3.9)$$

and

$$u_i \frac{\partial M}{\partial t} + u_i \frac{\partial}{\partial x_j} (M u_j) + u_i \frac{\partial}{\partial \eta} (M \dot{\eta}) = 0, \quad (3.10)$$

respectively, where  $P_i = \chi \left[ \left( 1 + \frac{1}{\mu} \frac{\partial p_v}{\partial \eta} \right) M \frac{\partial \Phi}{\partial x_i} - \left( \eta \frac{\partial M}{\partial x_i} + \frac{M}{\mu} \frac{\partial p_v}{\partial x_i} \right) \frac{\partial \Phi}{\partial \eta} \right]$ . Note that in the dry case,  $P_i = M \frac{\partial \Phi}{\partial x_i} - \eta \frac{\partial \Phi}{\partial \eta} \frac{\partial M}{\partial x_i}$ . The balance equation is derived by subtracting the divergence of Eq. (3.10) from the divergence of Eq. (3.9) and defining  $\mathfrak{M}_i = M u_i$ ,  $\mathfrak{D} = \nabla \cdot \vec{\mathfrak{M}}$ , and  $\zeta = \hat{k} \cdot \nabla \times \vec{\mathfrak{M}}$ . The result is

$$\begin{aligned} \frac{\partial}{\partial x_i} \left( \frac{\partial \mathfrak{M}_i}{\partial t} \right) - \frac{\partial}{\partial x_i} \left( \frac{M u_i}{M} \frac{\partial M}{\partial t} \right) + \frac{\partial}{\partial x_i} \left[ \frac{\partial}{\partial x_j} (\mathfrak{M}_j u_i) \right] - \frac{\partial}{\partial x_i} \left( u_i \frac{\partial \mathfrak{M}_j}{\partial x_j} \right) \\ + \frac{\partial}{\partial x_i} \left[ \frac{\partial}{\partial \eta} (\mathfrak{M}_i \dot{\eta}) \right] - \frac{\partial}{\partial x_i} \left[ u_i \frac{\partial}{\partial \eta} (M \dot{\eta}) \right] - \frac{\partial}{\partial x_i} (f \epsilon_{ij3} \mathfrak{M}_j) = - \frac{\partial P_i}{\partial x_i}. \end{aligned} \quad (3.11)$$

This equation can be simplified as follows:

$$\begin{aligned} \frac{\partial}{\partial t} \left( \frac{\partial \mathfrak{M}_i}{\partial x_i} \right) - \frac{\partial}{\partial x_i} \left( \mathfrak{M}_i \frac{\partial}{\partial t} \ln M \right) + \frac{\partial}{\partial x_i} \left[ \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} + u_i \frac{\partial \mathfrak{M}_j}{\partial x_j} \right] - \frac{\partial}{\partial x_i} \left( u_i \frac{\partial \mathfrak{M}_j}{\partial x_j} \right) \\ + \frac{\partial}{\partial x_i} \left( \dot{\eta} \frac{\partial \mathfrak{M}_i}{\partial \eta} + \mathfrak{M}_i \frac{\partial \dot{\eta}}{\partial \eta} \right) - \frac{\partial}{\partial x_i} \left( \mathfrak{M}_i \frac{\partial \dot{\eta}}{\partial \eta} \right) - f \epsilon_{ij3} \frac{\partial \mathfrak{M}_j}{\partial x_i} - \mathfrak{M}_j \frac{\partial}{\partial x_i} (f \epsilon_{ij3}) = - \frac{\partial P_i}{\partial x_i} \end{aligned}$$

leads to

$$\begin{aligned} \frac{\partial \mathfrak{D}}{\partial t} - \frac{\partial \mathfrak{M}_i}{\partial x_i} \frac{\partial}{\partial t} \ln M - \mathfrak{M}_i \frac{\partial}{\partial x_i} \left( \frac{\partial}{\partial t} \ln M \right) + \frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right) + \frac{\partial}{\partial x_i} \left( \dot{\eta} \frac{\partial \mathfrak{M}_i}{\partial \eta} \right) \\ - f \epsilon_{3ij} \frac{\partial \mathfrak{M}_j}{\partial x_i} + \epsilon_{3ij} \mathfrak{M}_i \frac{\partial f}{\partial x_j} = - \frac{\partial P_i}{\partial x_i}, \end{aligned}$$

which can be simplified to

$$\begin{aligned} \frac{\partial \mathfrak{D}}{\partial t} - \mathfrak{D} \frac{\partial}{\partial t} \ln M - \vec{\mathfrak{M}} \cdot \nabla \frac{\partial}{\partial t} \ln M + \frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right) + \frac{\partial \dot{\eta}}{\partial x_i} \frac{\partial \mathfrak{M}_i}{\partial \eta} + \dot{\eta} \frac{\partial}{\partial \eta} \left( \frac{\partial \mathfrak{M}_i}{\partial x_i} \right) \\ - f \zeta + \hat{k} \cdot \vec{\mathfrak{M}} \times \nabla f = - \frac{\partial P_i}{\partial x_i}, \end{aligned}$$



which finally results in

$$\begin{aligned} \frac{\partial \mathfrak{D}}{\partial t} - \mathfrak{D} \frac{\partial}{\partial t} \ln M - \vec{\mathfrak{M}} \cdot \nabla \frac{\partial}{\partial t} \ln M + \frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right) + \nabla \dot{\eta} \cdot \frac{\partial \vec{\mathfrak{M}}}{\partial \eta} + \dot{\eta} \frac{\partial \mathfrak{D}}{\partial \eta} \\ - f\zeta + \hat{k} \cdot \vec{\mathfrak{M}} \times \nabla f = - \frac{\partial P_i}{\partial x_i}. \end{aligned} \quad (3.12)$$

The term  $\frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right)$  can be expanded as

$$\frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right) = \frac{\partial \mathfrak{M}_j}{\partial x_i} \frac{\partial u_i}{\partial x_j} + \mathfrak{M}_j \frac{\partial^2 u_i}{\partial x_i \partial x_j} + 0_2 = \frac{\partial \mathfrak{M}_j}{\partial x_i} \frac{\partial u_i}{\partial x_j} + M u_i \frac{\partial^2 u_j}{\partial x_j \partial x_i} + 0_2, \quad (3.13)$$

where indices have been swapped in the second term on the right-hand side, and

$$\begin{aligned} 0_2 = u_i \frac{\partial u_j}{\partial x_i} \frac{\partial M}{\partial x_j} - u_i \frac{\partial u_j}{\partial x_i} \frac{\partial M}{\partial x_j} + u_i \frac{\partial u_j}{\partial x_j} \frac{\partial M}{\partial x_i} - u_i \frac{\partial u_j}{\partial x_j} \frac{\partial M}{\partial x_i} + \frac{u_i u_j}{M} \frac{\partial M}{\partial x_i} \frac{\partial M}{\partial x_j} - \frac{u_i u_j}{M} \frac{\partial M}{\partial x_i} \frac{\partial M}{\partial x_j} \\ + u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j} - u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j}. \end{aligned}$$

Thus, Eq. (3.13) can be rewritten as

$$\begin{aligned} \frac{\partial}{\partial x_i} \left( \mathfrak{M}_j \frac{\partial u_i}{\partial x_j} \right) &= \frac{\partial u_i}{\partial x_j} \frac{\partial \mathfrak{M}_j}{\partial x_i} + \frac{u_i u_j}{M} \frac{\partial M}{\partial x_i} \frac{\partial M}{\partial x_j} - u_i \frac{\partial u_j}{\partial x_i} \frac{\partial M}{\partial x_j} - u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j} + u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j} \\ &\quad + u_i \frac{\partial M}{\partial x_j} \frac{\partial u_j}{\partial x_i} + u_i \frac{\partial M}{\partial x_i} \frac{\partial u_j}{\partial x_j} + M u_i \frac{\partial^2 u_j}{\partial x_i \partial x_j} - \frac{u_i u_j}{M} \frac{\partial M}{\partial x_i} \frac{\partial M}{\partial x_j} - u_i \frac{\partial M}{\partial x_i} \frac{\partial u_j}{\partial x_j} \\ &= \nabla \vec{V} : \nabla \vec{\mathfrak{M}} - M u_i \left( -\frac{1}{M^2} \frac{\partial M}{\partial x_i} u_j + \frac{1}{M} \frac{\partial u_j}{\partial x_i} \right) \frac{\partial M}{\partial x_j} - u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j} \\ &\quad + u_i \frac{\partial}{\partial x_i} \left( \frac{\partial M}{\partial x_j} u_j + M \frac{\partial u_j}{\partial x_j} \right) - \left( \frac{\partial M}{\partial x_j} u_j + M \frac{\partial u_j}{\partial x_j} \right) \frac{u_i}{M} \frac{\partial M}{\partial x_i} \\ &= \frac{\partial \vec{V}}{\partial x} \cdot \nabla \mathfrak{M}_1 + \frac{\partial \vec{V}}{\partial y} \cdot \nabla \mathfrak{M}_2 - M u_i \frac{\partial}{\partial x_i} \left( \frac{u_j}{M} \right) \frac{\partial M}{\partial x_j} - u_i u_j \frac{\partial^2 M}{\partial x_i \partial x_j} \\ &\quad + u_i \frac{\partial}{\partial x_i} \left( \frac{\partial \mathfrak{M}_j}{\partial x_j} \right) - \frac{\partial \mathfrak{M}_j}{\partial x_j} u_i \frac{\partial}{\partial x_i} \ln M \\ &= \frac{\partial \vec{V}}{\partial x} \cdot \nabla \mathfrak{M}_1 + \frac{\partial \vec{V}}{\partial y} \cdot \nabla \mathfrak{M}_2 - \mathfrak{M}_i \frac{\partial}{\partial x_i} \left( \frac{u_j}{M} \frac{\partial M}{\partial x_j} \right) + u_i \frac{\partial \mathfrak{D}}{\partial x_i} - \mathfrak{D} u_i \frac{\partial}{\partial x_i} \ln M \\ &= \frac{\partial \vec{V}}{\partial x} \cdot \nabla \mathfrak{M}_1 + \frac{\partial \vec{V}}{\partial y} \cdot \nabla \mathfrak{M}_2 - \vec{\mathfrak{M}} \cdot \nabla \left( \vec{V} \cdot \nabla \ln M \right) + \vec{V} \cdot \nabla \mathfrak{D} - \mathfrak{D} \vec{V} \cdot \nabla \ln M. \end{aligned}$$

Collecting everything together, Eq. (3.12) can be written as

$$A1 + A2 + A3 + A4 + A5 + B1 + B2 + B3 + C + D = E, \quad (3.14)$$

where

$$\begin{aligned} A1 &\equiv \frac{\partial \mathcal{D}}{\partial t} \\ A2 &\equiv \dot{\eta} \frac{\partial \mathcal{D}}{\partial \eta} \\ A3 &\equiv \nabla \dot{\eta} \cdot \frac{\partial \vec{\mathfrak{M}}}{\partial \eta} \\ A4 &\equiv -\mathcal{D} \frac{\partial}{\partial t} \ln M \\ A5 &\equiv -\vec{\mathfrak{M}} \cdot \nabla \frac{\partial}{\partial t} \ln M \\ B1 &\equiv \frac{\partial \vec{V}}{\partial x} \cdot \nabla \mathfrak{M}_1 + \frac{\partial \vec{V}}{\partial y} \cdot \nabla \mathfrak{M}_2 \\ B2 &\equiv -\vec{\mathfrak{M}} \cdot \nabla \left( \vec{V} \cdot \nabla \ln M \right) \\ B3 &\equiv \vec{V} \cdot \nabla \mathcal{D} - \mathcal{D} \vec{V} \cdot \nabla \ln M \\ C &\equiv -f\zeta \\ D &\equiv \hat{k} \cdot \vec{\mathfrak{M}} \times \nabla f \\ E &\equiv -\nabla \cdot \left\{ \chi \left[ \left( 1 + \frac{1}{\mu} \frac{\partial p_v}{\partial \eta} \right) M \nabla \Phi - \left( \eta \nabla M + \frac{M}{\mu} \nabla p_v \right) \frac{\partial \Phi}{\partial \eta} \right] \right\}. \end{aligned}$$

### 3.1.2 Scale analysis

Many of the terms in Eq. (3.14) can be neglected based on scale analysis, following Sundqvist (1975). To begin, we will find a scaling  $\dot{H}$  for  $\dot{\eta}$  that is valid over level ground. The starting point is the scaling over level ground for  $\omega \equiv \frac{Dp}{Dt}$ , the pressure velocity, that Sundqvist (1975) gives:

$$\frac{\omega}{p} \sim 10^{-6} \text{s}^{-1}. \quad (3.15)$$

The pressure velocity can be related to  $\dot{\eta}$  as follows:

$$\begin{aligned}
 \dot{\eta} &\equiv \frac{D\eta}{Dt} = \frac{D}{Dt} \left( \frac{p_d - p_t}{\mu} \right) = \frac{\mu \frac{Dp_d}{Dt} - (p_d - p_t) \frac{D\mu}{Dt}}{\mu^2} = \frac{\omega_d}{\mu} - \frac{p_d - p_t}{\mu^2} \left( \frac{\partial \mu}{\partial t} + \vec{V} \cdot \nabla \mu \right) \\
 &= \frac{\omega_d}{\mu} - \frac{p_d - p_t}{\mu M (p_0 - p_t)} \left[ (p_0 - p_t) \frac{\partial M}{\partial t} + (p_0 - p_t) \vec{V} \cdot \nabla M \right] \\
 &= \frac{p_d - p_t}{\mu} \left( \frac{\omega_d}{p_d - p_t} - \frac{1}{M} \vec{V} \cdot \nabla M - \frac{1}{M} \frac{\partial M}{\partial t} \right) \\
 &= \eta \left( \frac{\omega_d}{p_d - p_t} - \frac{1}{M} \vec{V} \cdot \nabla M - \frac{1}{M} \frac{\partial M}{\partial t} \right). \tag{3.16}
 \end{aligned}$$

By definition,  $0 \leq \eta \leq 1$ . In addition,  $\frac{\omega_d}{p_d - p_t} \sim \frac{\omega}{p_d - p_t} \sim \frac{\omega}{p_d} \sim \frac{\omega}{p} \sim 10^{-6} \text{ s}^{-1}$  using Eq. (3.15). As shown in Figure 3.1a,  $M \approx 1$  and varies on the order  $\epsilon = 10^{-2}$  when topography is not an issue. Thus,  $\frac{\partial M}{\partial t} \sim \frac{1}{M} \frac{\partial M}{\partial t} \sim \frac{\epsilon}{\tau} = \epsilon \frac{C}{S} = f\epsilon \text{Ro}$ , where the time scale  $\tau$  is given by  $\tau = \frac{S}{C}$ ,  $S$  is the horizontal scale,  $C$  is the velocity scale, and the Rossby number,  $\text{Ro}$ , is defined as  $\text{Ro} = \frac{C}{fS}$ . Using values of  $10^{-4} \text{ s}^{-1}$  for the Coriolis parameter and  $10^{-1}$  for the Rossby number,  $\frac{\partial M}{\partial t} \sim \frac{1}{M} \frac{\partial M}{\partial t} \sim 10^{-7} \text{ s}^{-1}$ . Finally,  $\frac{1}{M} \vec{V} \cdot \nabla M \sim \frac{C\epsilon}{S} = f\epsilon \text{Ro} \sim 10^{-7} \text{ s}^{-1}$  as well. Given these scalings, Eq. (3.16) implies that  $\dot{H} \sim 10^{-6} \text{ s}^{-1}$ .

Let us now reconsider the continuity equation, Eq. (3.10), divided by  $u_i$ :

$$\frac{\partial M}{\partial t} + \mathfrak{D} + \frac{\partial}{\partial \eta} (M\dot{\eta}) = 0.$$

The previous analysis has already shown that  $\frac{\partial M}{\partial t} \sim 10^{-7} \text{ s}^{-1}$ . Furthermore, the third term,  $\frac{\partial}{\partial \eta} (M\dot{\eta})$ , is on the order of  $(1)(1)(10^{-6} \text{ s}^{-1}) \sim 10^{-6} \text{ s}^{-1}$  since  $\eta$  lies in the range between zero and one. Thus, we know that  $\mathfrak{D} \sim 10^{-6} \text{ s}^{-1}$ .

All of the scalings made so far are equally valid whether topography is present or not. However, in performing a scale analysis on Eq. (3.14), topography must be taken into account. Therefore, let us assume that pressure variations crossing mountainous terrain (i.e., fluctuations of  $M$ ) will be on the order of 10% ( $\epsilon_m \sim 10^{-1}$ ), equivalent to about one kilometer of altitude. Figure 3.1b, which provides an example from Test Case 3, verifies that this is the case. Topography may also impose a horizontal scale on the flow that is on the order of the scale of the topography itself. Since our

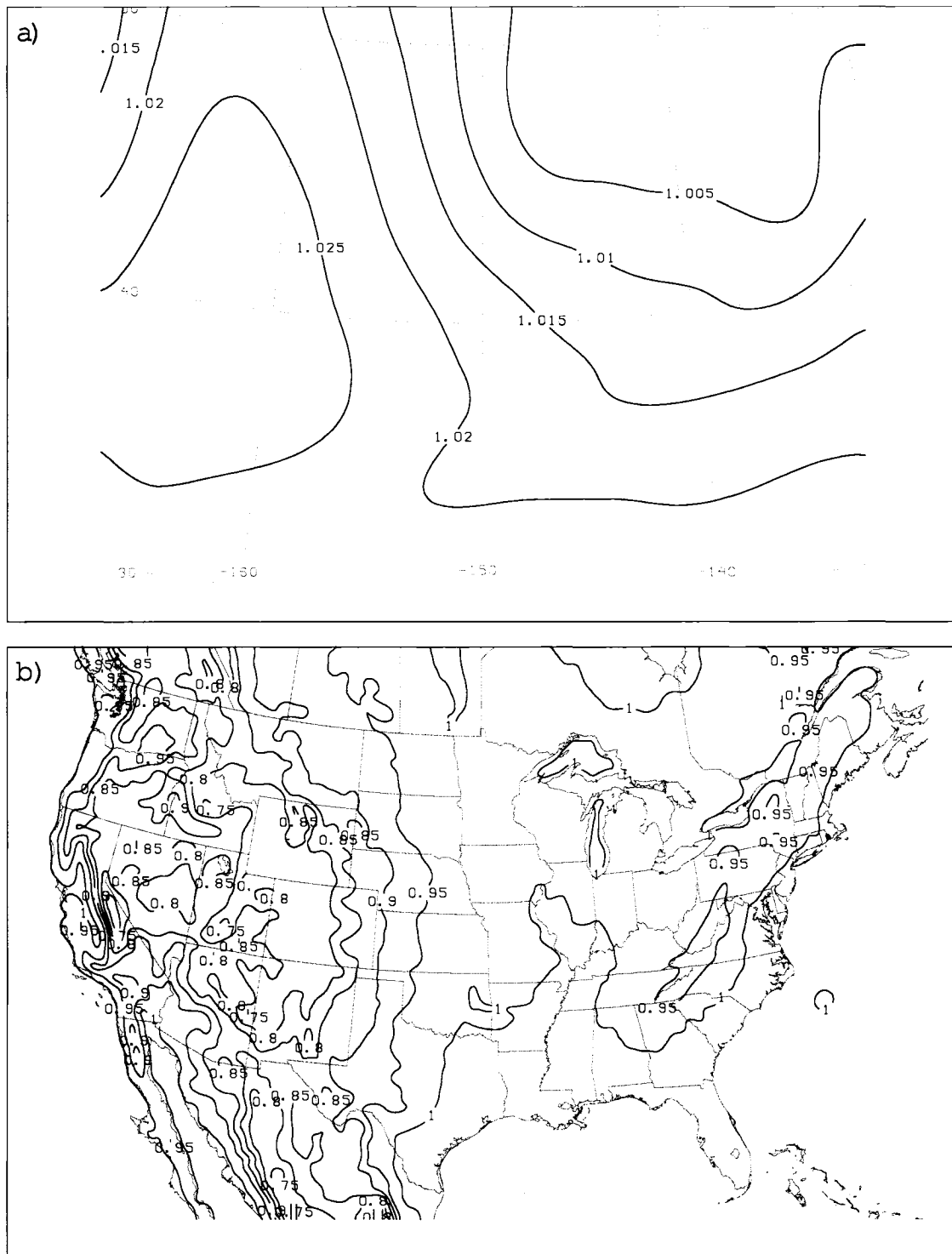


Figure 3.1: Plot of  $M$  12 hours into the respective simulations for (a) Test Case 2 (contoured every 0.05 units) and (b) Test Case 3 (contoured every 0.5 units).

model will ultimately have a grid spacing on the order of 10 km, this imposed horizontal scale may also be on the order of 10 km, or about  $\frac{1}{100}$  times the synoptic horizontal scale  $S$ . Thus, we use  $Ro \sim 10$  in the scaling that follows.

Considering each term, we have:

$$\begin{aligned}
A1 &\sim \frac{10^{-6} s^{-1}}{\tau} \sim fRo (10^{-6} s^{-1}) \sim (10^{-4} s^{-1}) (10) (10^{-6} s^{-1}) \sim 10^{-9} s^{-2} \\
A2 &\sim \dot{H} (10^{-6} s^{-1}) \sim (10^{-6} s^{-1}) (10^{-6} s^{-1}) \sim 10^{-12} s^{-2} \\
A3 &\sim \frac{\dot{H}}{S} C \sim \dot{H} fRo \sim (10^{-6} s^{-1}) (10^{-4} s^{-1}) (10) \sim 10^{-9} s^{-2} \\
A4 &\sim (10^{-6} s^{-1}) (10^{-7} s^{-1}) \sim 10^{-13} s^{-2} \\
A5 &\sim \frac{C}{S} (10^{-7} s^{-1}) \sim fRo (10^{-7} s^{-1}) \sim (10^{-4} s^{-1}) (10) (10^{-7} s^{-1}) \sim 10^{-10} s^{-2} \\
B1 &\sim \frac{C}{S} \cdot \frac{C}{S} \sim f^2 Ro^2 \sim (10^{-8} s^{-2}) (100) \sim 10^{-6} s^{-2} \\
B2 &\sim \frac{C}{S} \cdot \frac{C}{S} \epsilon_m \sim f^2 Ro^2 \epsilon_m \sim (10^{-8} s^{-2}) (100) (10^{-1}) \sim 10^{-7} s^{-2} \\
B3 &\sim \frac{C}{S} (10^{-6} s^{-1}) + (10^{-6} s^{-1}) \frac{C}{S} \epsilon_m \sim \frac{C}{S} (10^{-6} s^{-1}) (1 + \epsilon_m) \sim fRo (10^{-6} s^{-1}) \sim 10^{-9} s^{-2} \\
C &\sim f \frac{C}{S} \sim f^2 Ro \sim (10^{-8} s^{-2}) (10) \sim 10^{-7} s^{-2} \\
D &\sim \frac{C}{S} f \sim 10^{-7} s^{-2} \\
E &\sim 10^{-6} s^{-2} \quad (\text{so that the LHS is balanced})
\end{aligned}$$

As a check to see how accurate the above scalings are, each term has been plotted for Test Case 1 at hour 117 for  $\eta = 0.5129$ , which is roughly 500 hPa to correspond with Fig. 2.11f. For each plot, two metrics were computed, the maximum absolute value of the term at any grid point (to be referred to as the maximum value for short), and the average absolute value of the term over all of the grid points (to be referred to as the average value for short).

Terms A1–3 are displayed in Fig. 3.2. The first term (Fig. 3.2a), the time derivative of mass-weighted divergence, is fairly noisy, with the highest magnitudes in the vicinity of the cutoff low and along the connected vorticity strip. The average value of this term was found to be  $4.115 \times 10^{-10} s^{-2}$ , with a maximum value of  $6.799 \times 10^{-9} s^{-2}$ . It should be noted that, because output from Test

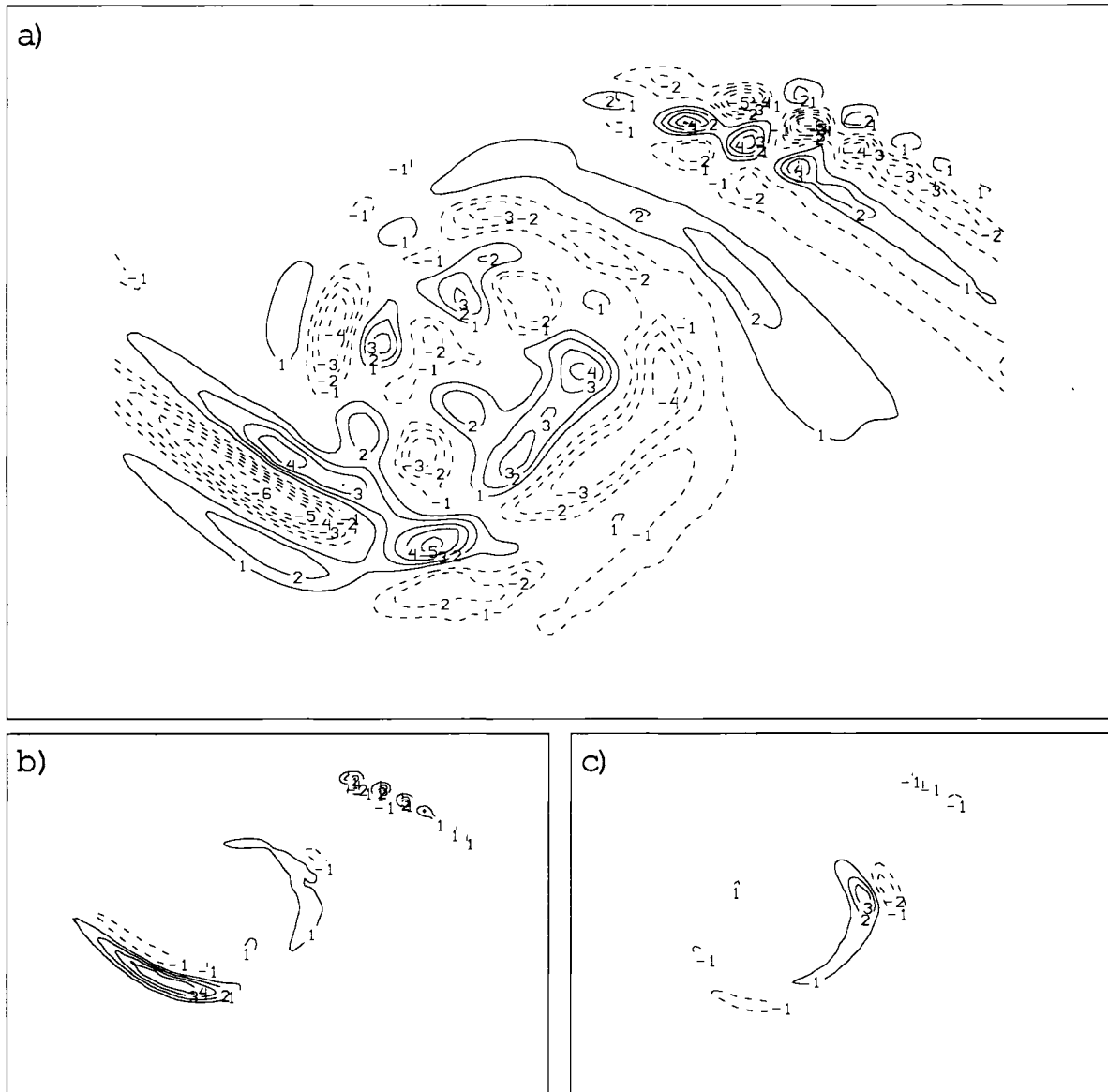


Figure 3.2: Terms from the balance equation for Test Case 1 at hour 117. (a) A1 [contour interval  $10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. (b) Same as (a) but for A2. (c) Same as (a) but for A3.

Case 1 was only available every hour, the true magnitude of this term (using the model's time step instead) is likely higher. The second term (Fig. 3.2b), related to vertical advection of divergence, is generally quite small, except near the vorticity strip. Its average value was  $9.858 \times 10^{-11} \text{ s}^{-2}$ , with a maximum value of  $4.929 \times 10^{-9} \text{ s}^{-2}$ . The third term (Fig. 3.2c), involving vertical shear, was quite similar in shape to Term A2. Its average value was  $1.067 \times 10^{-10} \text{ s}^{-2}$ , with a maximum value of  $3.494 \times 10^{-9} \text{ s}^{-2}$ . All three of these terms agree quite well with the scale analysis.

Terms A4–5 are displayed in Fig. 3.3. Both terms are difficult to interpret physically, but both terms are also small. Term A4 (Fig. 3.3a) varies on a relatively small scale, whereas Term A5 (Fig. 3.3b) has a quite large scale. Both terms have greatest magnitudes in the vicinity of the low, but Term A4 tends to be focused around the edges of the low (where the height gradient is largest), while Term A5 has a maximum generally co-located with the low. Term A4 has an average value of  $3.180 \times 10^{-12} \text{ s}^{-2}$  at this level, with a maximum value of  $1.168 \times 10^{-10} \text{ s}^{-2}$ . Term A5 has an average value of  $2.624 \times 10^{-11} \text{ s}^{-2}$ , with a maximum value of  $4.546 \times 10^{-10} \text{ s}^{-2}$ . Term A5 is in line with our scale analysis, but Term A4 is not as small as the scale analysis suggested.

The B terms, which generally represent curvature in the flow (an analog to the centripetal force in the gradient wind balance), are shown in Fig. 3.4. Term B1 (Fig. 3.4a) is of relatively large scale, and its comma shape is again reminiscent of the vorticity distribution. This makes sense given the connection between this term and curvature, and the connection between curvature and vorticity. The average value of Term B1 at this level is  $7.990 \times 10^{-10} \text{ s}^{-2}$ , and its maximum value is  $1.849 \times 10^{-8} \text{ s}^{-2}$ . This is much smaller than the magnitude estimated above, but becomes closer to our estimate if we realize that this domain has no topography. In that case,  $Ro = 0.1$ , and the scaling becomes  $10^{-8} \text{ s}^{-2}$ . Term B2 (Fig. 3.4b) is negligible compared to the other B terms, with an average value of  $1.200 \times 10^{-11} \text{ s}^{-2}$ , and a maximum value of  $6.615 \times 10^{-10} \text{ s}^{-2}$ . For the same reason as for B1, these values are much less than expected based on the scale analysis. Term B3 (Fig. 3.4c) is the noisiest of the three, with numerous dipole structures in close proximity. The magnitude of this term is close to that of the scale analysis, with an average value of  $2.365 \times 10^{-10} \text{ s}^{-2}$ , and a maximum value of  $7.165 \times 10^{-9} \text{ s}^{-2}$ .

Terms C and D are Coriolis terms. Since Test Case 1 is carried out on an f-plane, Term D

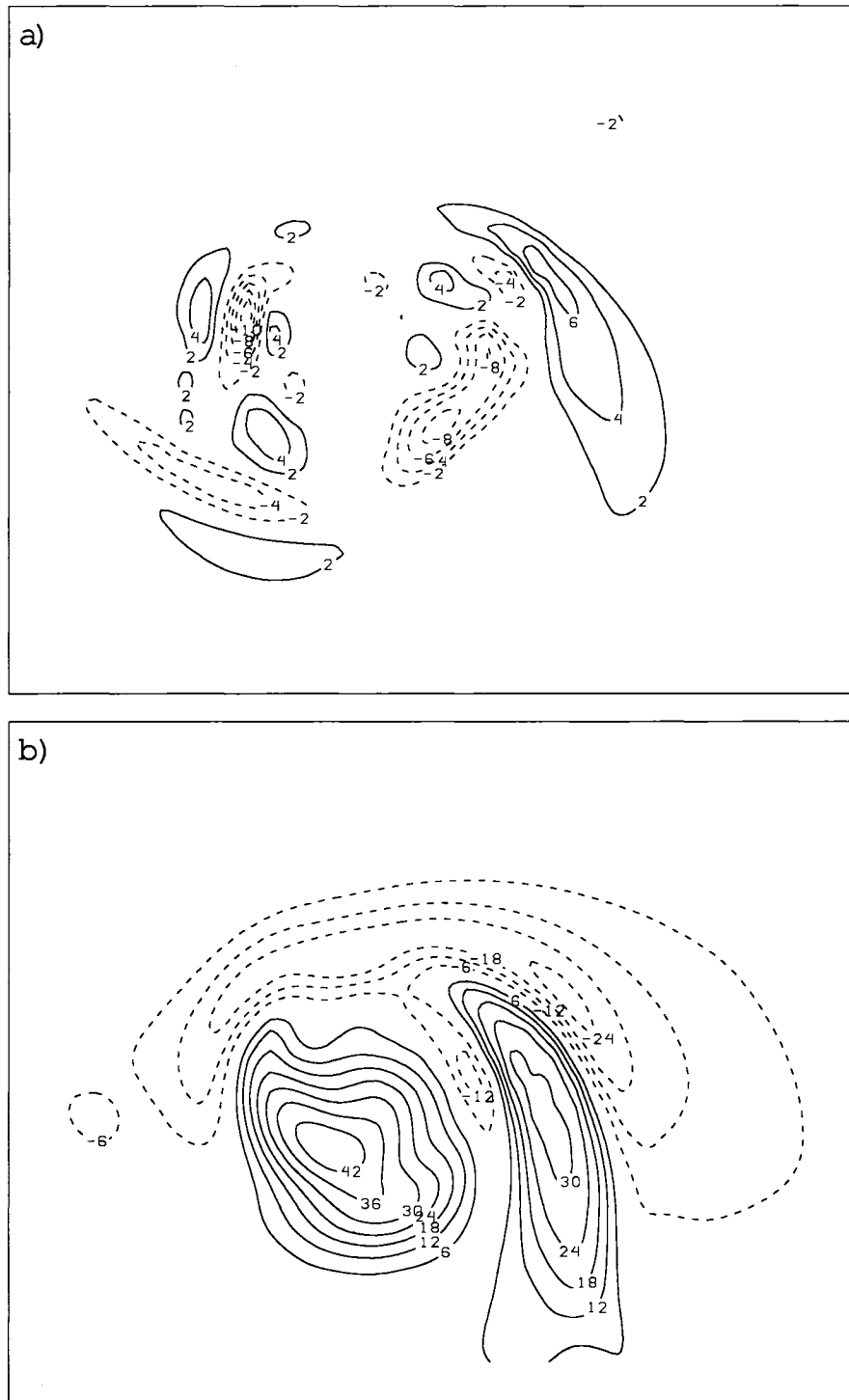


Figure 3.3: Same as Fig. 3.2 except for (a) A4 (contour interval  $2 \times 10^{-11} \text{ s}^{-2}$ ) and (b) A5 (contour interval  $6 \times 10^{-11} \text{ s}^{-2}$ ).



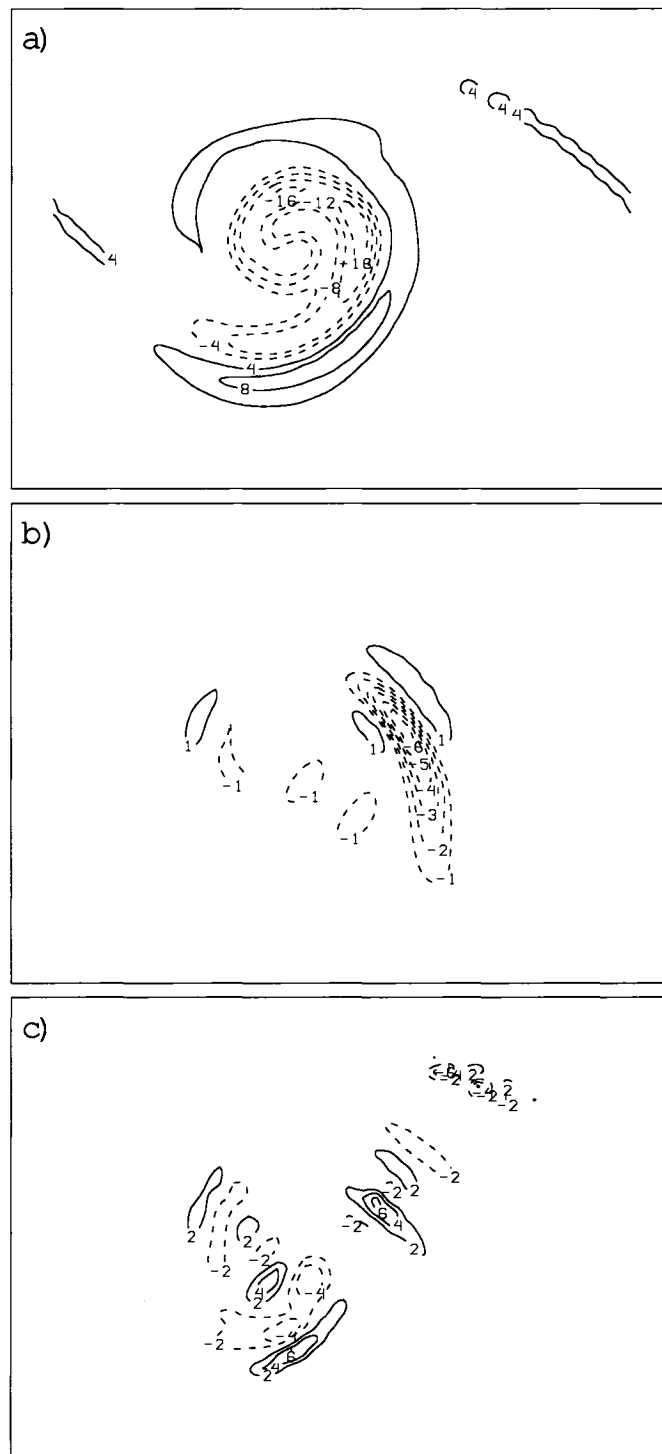


Figure 3.4: Same as Fig. 3.2 except for (a) B1 (contour interval  $4 \times 10^{-9} \text{s}^{-2}$ ), (b) B2 (contour interval  $10^{-10} \text{s}^{-2}$ ), and (c) B3 (contour interval  $2 \times 10^{-9} \text{s}^{-2}$ ).

Term	Test Case 1	Test Case 2	Test Case 3
A1	$4.12 \times 10^{-10} \text{ s}^{-2}$	$1.35 \times 10^{-9} \text{ s}^{-2}$	$3.39 \times 10^{-9} \text{ s}^{-2}$
A2	$9.89 \times 10^{-11} \text{ s}^{-2}$	$1.87 \times 10^{-10} \text{ s}^{-2}$	$1.09 \times 10^{-9} \text{ s}^{-2}$
A3	$1.07 \times 10^{-10} \text{ s}^{-2}$	$2.07 \times 10^{-10} \text{ s}^{-2}$	$2.19 \times 10^{-9} \text{ s}^{-2}$
A4	$3.18 \times 10^{-12} \text{ s}^{-2}$	$1.28 \times 10^{-12} \text{ s}^{-2}$	$4.02 \times 10^{-12} \text{ s}^{-2}$
A5	$2.62 \times 10^{-11} \text{ s}^{-2}$	$9.69 \times 10^{-12} \text{ s}^{-2}$	$1.34 \times 10^{-11} \text{ s}^{-2}$
B1	$7.99 \times 10^{-10} \text{ s}^{-2}$	$1.34 \times 10^{-9} \text{ s}^{-2}$	$1.42 \times 10^{-9} \text{ s}^{-2}$
B2	$1.20 \times 10^{-11} \text{ s}^{-2}$	$1.32 \times 10^{-11} \text{ s}^{-2}$	$1.39 \times 10^{-9} \text{ s}^{-2}$
B3	$2.37 \times 10^{-10} \text{ s}^{-2}$	$1.76 \times 10^{-9} \text{ s}^{-2}$	$6.05 \times 10^{-9} \text{ s}^{-2}$
C	$1.95 \times 10^{-9} \text{ s}^{-2}$	$5.35 \times 10^{-9} \text{ s}^{-2}$	$2.66 \times 10^{-9} \text{ s}^{-2}$
D	$0 \text{ s}^{-2}$	$3.61 \times 10^{-10} \text{ s}^{-2}$	$2.18 \times 10^{-10} \text{ s}^{-2}$
E	$2.34 \times 10^{-9} \text{ s}^{-2}$	$4.56 \times 10^{-9} \text{ s}^{-2}$	$1.15 \times 10^{-8} \text{ s}^{-2}$

Table 3.1: The average absolute magnitudes of various terms in the balance equation.

is identically zero, and Term C closely matches the distribution of vorticity (Fig. 3.5a). Term C has a maximum value of  $1.914 \times 10^{-8} \text{ s}^{-2}$ , and an average value of  $1.949 \times 10^{-9} \text{ s}^{-2}$ , both of which are again less than what the scale analysis suggests. Term E is essentially the convergence of the pressure gradient force, and thus tends to be positive near troughs and negative near ridges (Fig. 3.5b). Term E is also less than anticipated due to the lack of topography, with an average value of  $2.339 \times 10^{-9} \text{ s}^{-2}$ , and a maximum value of  $3.145 \times 10^{-8} \text{ s}^{-2}$ .

Similar checks were performed for the other test cases, and the results are summarized in Table 3.1. Notice that many of the terms that had magnitudes smaller than those suggested by the scale analysis do increase in magnitude as the topography increases and grid spacing decreases. However, the resultant magnitudes are still overestimated by the scale analysis, suggesting that the adjustments made for the presence of topography were too much. Nevertheless, the general patterns in the scale analysis do hold up overall. It is worthwhile to note that the terms deemed small based on scale analysis were generally noisier than the presumably larger terms. If small wavelengths had been filtered out of the numerical solutions (and the WRF, being nonhydrostatic, contains small-scale features such as sound waves), the disparity in magnitude between terms would likely have been accentuated. Thus, the scalings presented above will be used to help simplify the balance equation.

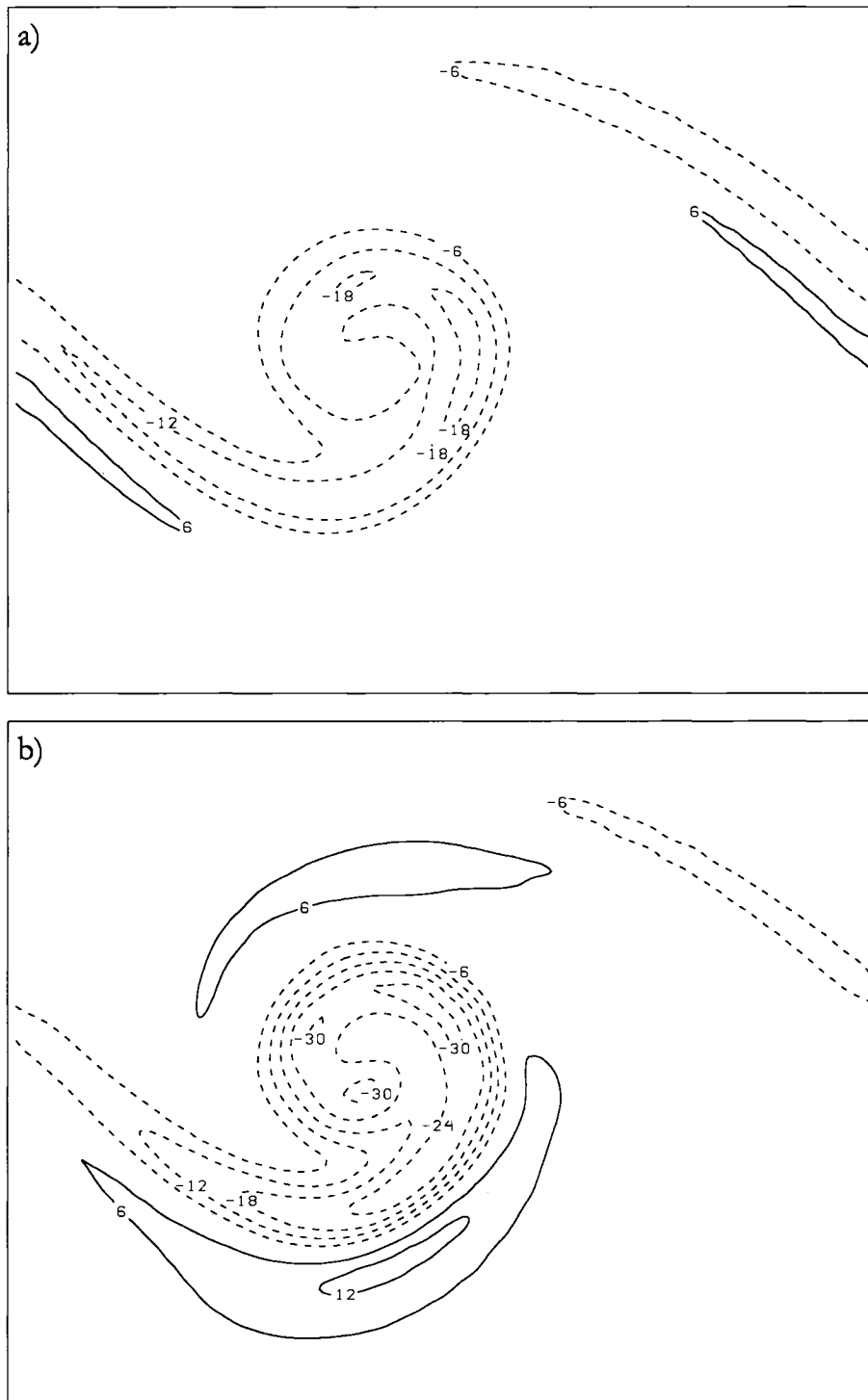


Figure 3.5: Same as Fig. 3.2 except for (a)  $C$  (contour interval  $6 \times 10^{-9} \text{ s}^{-2}$ ) and (b)  $E$  (contour interval  $6 \times 10^{-9} \text{ s}^{-2}$ ).

### 3.1.3 Balance equation

If we neglect all terms at least three orders of magnitude smaller than  $E$  (according to the original scale analysis), Eq. (3.14) reduces to

$$\begin{aligned} \frac{\partial \vec{V}}{\partial x} \cdot \nabla \mathfrak{M}_1 + \frac{\partial \vec{V}}{\partial y} \cdot \nabla \mathfrak{M}_2 - \vec{\mathfrak{M}} \cdot \nabla \left( \vec{V} \cdot \nabla \ln M \right) - f\zeta + \hat{k} \cdot \vec{\mathfrak{M}} \times \nabla f \\ = -\nabla \cdot \left\{ \chi \left[ \left( 1 + \frac{1}{\mu} \frac{\partial p_v}{\partial \eta} \right) M \nabla \Phi - \left( \eta \nabla M + \frac{M}{\mu} \nabla p_v \right) \frac{\partial \Phi}{\partial \eta} \right] \right\}, \end{aligned} \quad (3.17)$$

or, using indicial notation,

$$\frac{\partial u_i}{\partial x_j} \frac{\partial \mathfrak{M}_j}{\partial x_i} - \mathfrak{M}_i \frac{\partial}{\partial x_i} \left( \frac{u_j}{M} \frac{\partial M}{\partial x_j} \right) - f \epsilon_{3ij} \frac{\partial \mathfrak{M}_j}{\partial x_i} + \epsilon_{3ij} \mathfrak{M}_i \frac{\partial f}{\partial x_j} = -\frac{\partial P_i}{\partial x_i}. \quad (3.18)$$

According to Fig. 3.6, this reduction in complexity of the LHS of the balance equation does not radically alter its value, except on the smallest scales, as has already been alluded to. The top panel (Fig. 3.6a) shows the sum of all of the terms A through D, whereas the middle panel (Fig. 3.6b) shows the LHS of Eq. (3.18). The original LHS tends to be a bit noisier, but there are no important extrema that are not reproduced in the proper locations by the simplified version. The bottom panel (Fig. 3.6c), which shows the sum of the terms neglected, verifies that this is the case.

It appears that, at least in the ideal case, simplifying the left hand side of Eq. (3.14) does not carry any ill effects on the large scale. Is the same true in the real world? To check, the same test was repeated for Test Case 2, and the results are displayed in Fig. 3.7. The pattern does appear to hold. The simplified LHS (Fig. 3.7b) is smoother than the full LHS (Fig. 3.7a), indicating that again the simplification is basically a matter of removing noise from the imbalance equation. That noise (Fig. 3.7c), however, has a greater magnitude relative to the full LHS in this real-world case.

Note that, by making this simplification, all terms involving  $\mathfrak{D}$ , the divergence of the mass flow, have vanished. Thus, we can rewrite the mass flow  $\vec{\mathfrak{M}}$  in terms of the streamfunction  $\psi$  using the definition  $\vec{\mathfrak{M}} = \hat{k} \times \nabla \psi$ , or, using indicial notation,  $\mathfrak{M}_i = \epsilon_{3ji} \frac{\partial \psi}{\partial x_j}$ . We apply this substitution term

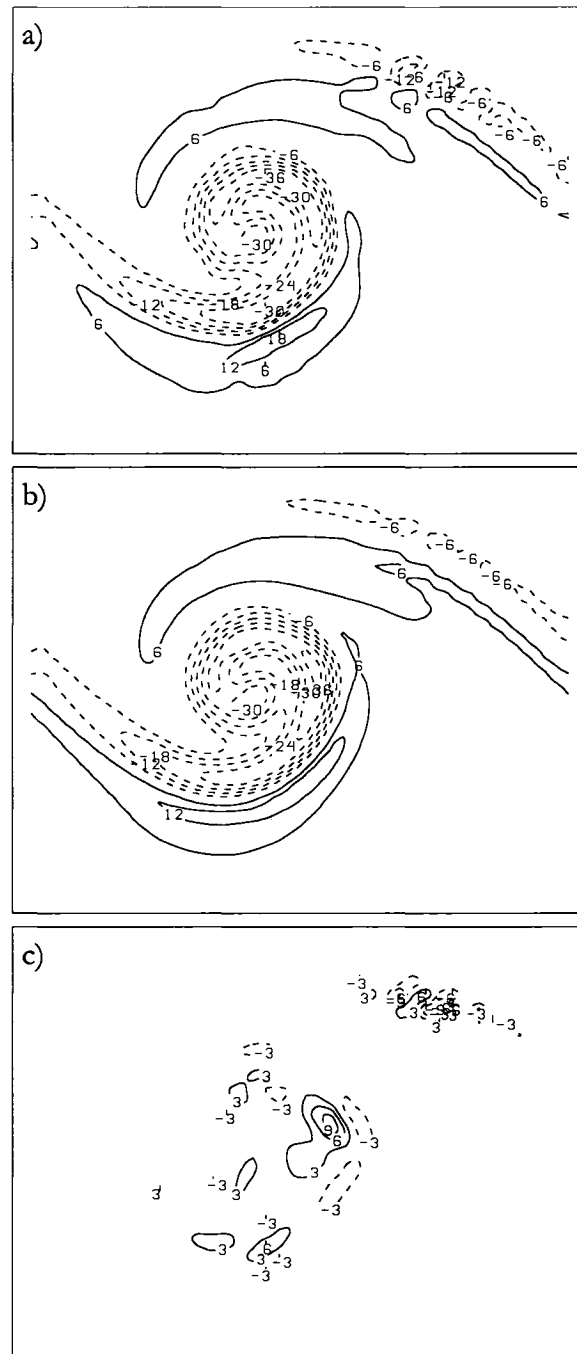


Figure 3.6: Examination of the left-hand side of the balance equation for Test Case 1 at hour 117. (a) The sum of the left-hand side of Eq. (3.14) [contour interval  $6 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)]. (b) Same as (a) but including terms B1, B2, C, and D only. (c) The graphical difference of panels (a) and (b) [contour interval  $3 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)].

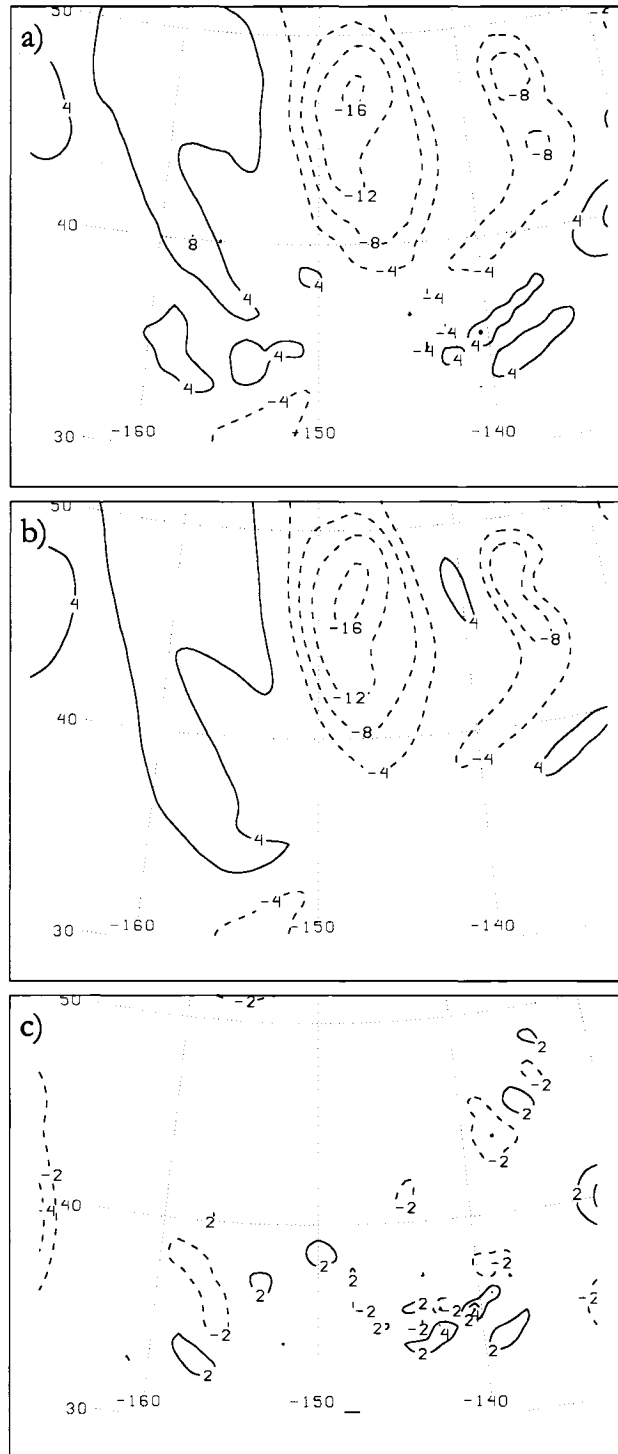


Figure 3.7: Same as Fig. 3.6 but for Test Case 2 at hour 12, and with (a–b) contour interval  $4 \times 10^{-9} \text{ s}^{-2}$ , and (c) contour interval  $2 \times 10^{-9} \text{ s}^{-2}$ .

by term to the LHS of Eq. (3.18) as follows:

$$\begin{aligned}
\frac{\partial u_i}{\partial x_j} \frac{\partial \mathfrak{M}_j}{\partial x_i} &= \frac{\partial u_k}{\partial x_i} \frac{\partial \mathfrak{M}_i}{\partial x_k} \\
&= \frac{\partial}{\partial x_i} \left( \frac{1}{M} \epsilon_{3lk} \frac{\partial \psi}{\partial x_l} \right) \frac{\partial}{\partial x_k} \left( \epsilon_{3ji} \frac{\partial \psi}{\partial x_j} \right) \\
&= \epsilon_{ij3} \epsilon_{kl3} \frac{\partial^2 \psi}{\partial x_j \partial x_k} \frac{\partial}{\partial x_i} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right)
\end{aligned} \tag{3.19}$$

$$\begin{aligned}
-\mathfrak{M}_i \frac{\partial}{\partial x_i} \left( \frac{u_j}{M} \frac{\partial M}{\partial x_j} \right) &= -\mathfrak{M}_i \frac{\partial}{\partial x_i} \left( \frac{u_k}{M} \frac{\partial M}{\partial x_k} \right) \\
&= -\epsilon_{3ji} \frac{\partial \psi}{\partial x_j} \frac{\partial}{\partial x_i} \left( \frac{1}{M^2} \epsilon_{3lk} \frac{\partial \psi}{\partial x_l} \frac{\partial M}{\partial x_k} \right) \\
&= \epsilon_{3ij} \frac{\partial \psi}{\partial x_j} \frac{\partial}{\partial x_i} \left\{ \epsilon_{3lk} \left[ \frac{1}{M} \frac{\partial^2 \psi}{\partial x_k \partial x_l} - \frac{\partial}{\partial x_k} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right) \right] \right\} \\
&= \epsilon_{3ij} \frac{\partial \psi}{\partial x_j} \frac{\partial}{\partial x_i} \left\{ \epsilon_{3lk} \left[ -\frac{\partial}{\partial x_k} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right) \right] \right\} \quad \left( \epsilon_{3lk} \frac{\partial^2 \psi}{\partial x_k \partial x_l} = 0 \right) \\
&= \epsilon_{ij3} \epsilon_{kl3} \frac{\partial \psi}{\partial x_j} \frac{\partial^2}{\partial x_i \partial x_k} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right)
\end{aligned} \tag{3.20}$$

$$\begin{aligned}
-f \epsilon_{3ij} \frac{\partial \mathfrak{M}_j}{\partial x_i} + \epsilon_{3ij} \mathfrak{M}_i \frac{\partial f}{\partial x_j} &= -f \epsilon_{3ij} \frac{\partial}{\partial x_i} \left( \epsilon_{3kj} \frac{\partial \psi}{\partial x_k} \right) + \epsilon_{3ij} \epsilon_{3ki} \frac{\partial \psi}{\partial x_k} \frac{\partial f}{\partial x_j} \\
&= -\epsilon_{ij3} \epsilon_{3kj} f \frac{\partial^2 \psi}{\partial x_i \partial x_k} + (\delta_{ik} \delta_{ji} - \delta_{ii} \delta_{jk}) \frac{\partial \psi}{\partial x_k} \frac{\partial f}{\partial x_j} \\
&= (\delta_{ij} \delta_{jk} - \delta_{ik} \delta_{jj}) f \frac{\partial^2 \psi}{\partial x_i \partial x_k} + \frac{\partial \psi}{\partial x_i} \frac{\partial f}{\partial x_i} - 2 \frac{\partial \psi}{\partial x_j} \frac{\partial f}{\partial x_j} \\
&= f \frac{\partial^2 \psi}{\partial x_j \partial x_j} - 2f \frac{\partial^2 \psi}{\partial x_i \partial x_i} - \frac{\partial \psi}{\partial x_i} \frac{\partial f}{\partial x_i} \\
&= -f \frac{\partial^2 \psi}{\partial x_i \partial x_i} - \frac{\partial \psi}{\partial x_i} \frac{\partial f}{\partial x_i} \\
&= -\frac{\partial}{\partial x_i} \left( f \frac{\partial \psi}{\partial x_i} \right),
\end{aligned} \tag{3.21}$$

where the relations  $\epsilon_{kij} \epsilon_{klm} = \delta_{il} \delta_{jm} - \delta_{im} \delta_{jl}$  and  $\delta_{ii} = 2$  have been used. Substituting Eqs. (3.19),

(3.20), and (3.21) into Eq. (3.18) and multiplying by  $-1$  give

$$\begin{aligned} \frac{\partial}{\partial x_i} \left( f \frac{\partial \psi}{\partial x_i} \right) - \epsilon_{ij3} \epsilon_{kl3} \frac{\partial^2 \psi}{\partial x_j \partial x_k} \frac{\partial}{\partial x_i} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right) - \epsilon_{ij3} \epsilon_{kl3} \frac{\partial \psi}{\partial x_j} \frac{\partial^2}{\partial x_i \partial x_k} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right) &= \frac{\partial P_i}{\partial x_i} \\ \frac{\partial}{\partial x_i} \left( f \frac{\partial \psi}{\partial x_i} \right) - \epsilon_{ij3} \epsilon_{kl3} \frac{\partial}{\partial x_k} \left[ \frac{\partial \psi}{\partial x_j} \frac{\partial}{\partial x_i} \left( \frac{1}{M} \frac{\partial \psi}{\partial x_l} \right) \right] &= \frac{\partial P_i}{\partial x_i}, \end{aligned}$$

which can be expanded to

$$\begin{aligned} \nabla \cdot (f \nabla \psi) &- \left\{ \frac{\partial}{\partial x} \left[ \frac{\partial \psi}{\partial y} \frac{\partial}{\partial x} \left( \frac{1}{M} \frac{\partial \psi}{\partial y} \right) - \frac{\partial \psi}{\partial x} \frac{\partial}{\partial y} \left( \frac{1}{M} \frac{\partial \psi}{\partial y} \right) \right] + \frac{\partial}{\partial y} \left[ \frac{\partial \psi}{\partial x} \frac{\partial}{\partial y} \left( \frac{1}{M} \frac{\partial \psi}{\partial x} \right) - \frac{\partial \psi}{\partial y} \frac{\partial}{\partial x} \left( \frac{1}{M} \frac{\partial \psi}{\partial x} \right) \right] \right\} \\ &= \nabla \cdot \left\{ \chi \left[ \left( 1 + \frac{1}{\mu} \frac{\partial p_v}{\partial \eta} \right) M \nabla \Phi - \left( \eta \nabla M + \frac{M}{\mu} \nabla p_v \right) \frac{\partial \Phi}{\partial \eta} \right] \right\}. \quad (3.22) \end{aligned}$$

The nonlinear imbalance, then, is defined as the LHS of Eq. (3.22) minus the RHS of Eq. (3.22).

A flow in perfect balance thus has a nonlinear imbalance that is identically zero. The greater the imbalance, the further the flow is from obeying the nonlinear balance equation.

## 3.2 Discretization

To compute the imbalance, we need to discretize it. To simplify the presentation, the nonlinear imbalance will be written as

$$\text{IMB} = l_1 - l_2 - l_3 - r, \quad (3.23)$$

where IMB is the imbalance,  $l_1 = \nabla \cdot (f \nabla \psi)$ ,  $l_2 = \frac{\partial}{\partial x} \left[ \frac{\partial \psi}{\partial y} \frac{\partial}{\partial x} \left( \frac{1}{M} \frac{\partial \psi}{\partial y} \right) - \frac{\partial \psi}{\partial x} \frac{\partial}{\partial y} \left( \frac{1}{M} \frac{\partial \psi}{\partial y} \right) \right]$ ,  $l_3 = \frac{\partial}{\partial y} \left[ \frac{\partial \psi}{\partial x} \frac{\partial}{\partial y} \left( \frac{1}{M} \frac{\partial \psi}{\partial x} \right) - \frac{\partial \psi}{\partial y} \frac{\partial}{\partial x} \left( \frac{1}{M} \frac{\partial \psi}{\partial x} \right) \right]$ , and  $r = \nabla \cdot \left( M \nabla \Phi - \eta \nabla M \frac{\partial \Phi}{\partial \eta} \right)$ , ignoring contributions from



water substance for the time being. The following discrete operators are used:

$$\begin{aligned}
(\delta_x \varphi)_{i,j}^k &= \frac{\varphi_{i+\frac{1}{2},j}^k - \varphi_{i-\frac{1}{2},j}^k}{\Delta x} \\
(\delta_y \varphi)_{i,j}^k &= \frac{\varphi_{i,j+\frac{1}{2}}^k - \varphi_{i,j-\frac{1}{2}}^k}{\Delta x} \\
(\delta_\eta \varphi)_{i,j}^k &= \frac{\varphi_{i,j}^{k+\frac{1}{2}} - \varphi_{i,j}^{k-\frac{1}{2}}}{\eta_{k+\frac{1}{2}} - \eta_{k-\frac{1}{2}}} \\
(\overline{\varphi}^x)_{i,j}^k &= \frac{\varphi_{i-\frac{1}{2},j}^k + \varphi_{i+\frac{1}{2},j}^k}{2} \\
(\overline{\varphi}^y)_{i,j}^k &= \frac{\varphi_{i,j-\frac{1}{2}}^k + \varphi_{i,j+\frac{1}{2}}^k}{2} \\
(\overline{\varphi}^{xy})_{i,j}^k &= \frac{\varphi_{i-\frac{1}{2},j-\frac{1}{2}}^k + \varphi_{i+\frac{1}{2},j-\frac{1}{2}}^k + \varphi_{i-\frac{1}{2},j+\frac{1}{2}}^k + \varphi_{i+\frac{1}{2},j+\frac{1}{2}}^k}{4},
\end{aligned}$$

where  $\varphi$  is any discrete variable and fractional indices are replaced by integers according to the discretization scheme in Fig. 2.1. Similar interpolation operators are also defined for the vertical direction.

Considering  $l_1$ , we find that it can be discretized as

$$l_1 = m^2 \left[ \delta_x \left( \frac{\overline{f}^x}{m^u} \overline{m^v} \delta_x \psi^{xy} \right) + \delta_y \left( \frac{\overline{f}^y}{m^v} \overline{m^u} \delta_y \psi^{xy} \right) \right]. \quad (3.24)$$

Here,  $m^u$  represents the map factor valid at u-points, and  $m^v$  represents the map factor valid at

v-points. Expanding Eq. (3.24) on any  $\eta$  surface gives

$$\begin{aligned}
(l_1)_{i,j} = & \frac{m_{i,j}^2}{8(\Delta x)^2} \left\{ \frac{f_{i,j} + f_{i+1,j}}{m_{i,j}^u} [m_{i,j-1}^v (\psi_{i,j-1} - \psi_{i-1,j-1}) + m_{i+1,j-1}^v (\psi_{i+1,j-1} - \psi_{i,j-1}) \right. \\
& + m_{i,j}^v (\psi_{i,j} - \psi_{i-1,j}) + m_{i+1,j}^v (\psi_{i+1,j} - \psi_{i,j})] - \frac{f_{i-1,j} + f_{i,j}}{m_{i-1,j}^u} [m_{i-1,j-1}^v (\psi_{i-1,j-1} - \psi_{i-2,j-1}) \\
& + m_{i,j-1}^v (\psi_{i,j-1} - \psi_{i-1,j-1}) + m_{i-1,j}^v (\psi_{i-1,j} - \psi_{i-2,j}) + m_{i,j}^v (\psi_{i,j} - \psi_{i-1,j})] \\
& + \frac{f_{i,j} + f_{i,j+1}}{m_{i,j}^v} [m_{i-1,j}^u (\psi_{i-1,j} - \psi_{i-1,j-1}) + m_{i,j}^u (\psi_{i,j} - \psi_{i,j+1}) \\
& + m_{i-1,j+1}^u (\psi_{i-1,j+1} - \psi_{i-1,j}) + m_{i,j+1}^u (\psi_{i,j+1} - \psi_{i,j})] \\
& - \frac{f_{i,j-1} + f_{i,j}}{m_{i,j-1}^v} [m_{i-1,j-1}^u (\psi_{i-1,j-1} - \psi_{i-1,j-2}) + m_{i,j-1}^u (\psi_{i,j-1} - \psi_{i,j-2}) \\
& \left. + m_{i-1,j}^u (\psi_{i-1,j} - \psi_{i-1,j-1}) + m_{i,j}^u (\psi_{i,j} - \psi_{i,j-1})] \right\}. \tag{3.25}
\end{aligned}$$

The discretizations for the rest of the terms follow.

$$l_2 = m\delta_x \left[ (m^u)^2 \delta_y \psi \delta_x \left( \frac{1}{M} \overline{m^u \delta_y \psi^x} \right) - \overline{m^v \delta_x \psi^{xy}} m^u \delta_y \left( \frac{1}{M^{xy}} \overline{m^u \delta_y \psi^y} \right) \right], \tag{3.26}$$

and its expansion is

$$\begin{aligned}
(l_2)_{i,j} = & \frac{m_{i,j}}{2(\Delta x)^4} < m_{i,j}^u \{ m_{i,j}^u [\psi_{i,j} - \psi_{i,j-1}] [m_{i,j}^u \left( \frac{1}{M_{i+1,j}} - \frac{1}{M_{i,j}} \right) (\psi_{i,j} - \psi_{i,j-1}) \\
& + \frac{m_{i+1,j}^u}{M_{i+1,j}} (\psi_{i+1,j} - \psi_{i+1,j-1}) - \frac{m_{i-1,j}^u}{M_{i,j}} (\psi_{i-1,j} - \psi_{i-1,j-1})] - [m_{i,j-1}^v (\psi_{i,j-1} - \psi_{i-1,j-1}) \\
& + m_{i+1,j-1}^v (\psi_{i+1,j-1} - \psi_{i,j-1}) + m_{i,j}^v (\psi_{i,j} - \psi_{i-1,j}) + m_{i+1,j}^v (\psi_{i+1,j} - \psi_{i,j})] \\
& [c_{i,j} (m_{i,j+1}^u (\psi_{i,j+1} - \psi_{i,j}) + m_{i,j}^u (\psi_{i,j} - \psi_{i,j-1})) \\
& - c_{i,j-1} (m_{i,j}^u (\psi_{i,j} - \psi_{i,j-1}) + m_{i,j-1}^u (\psi_{i,j-1} - \psi_{i,j-2}))]] \} \\
& - m_{i-1,j}^u \{ m_{i-1,j}^u [\psi_{i-1,j} - \psi_{i-1,j-1}] [m_{i-1,j}^u \left( \frac{1}{M_{i,j}} - \frac{1}{M_{i-1,j}} \right) (\psi_{i-1,j} - \psi_{i-1,j-1}) \\
& + \frac{m_{i,j}^u}{M_{i,j}} (\psi_{i,j} - \psi_{i,j-1}) - \frac{m_{i-2,j}^u}{M_{i-1,j}} (\psi_{i-2,j} - \psi_{i-2,j-1})] - [m_{i-1,j-1}^v (\psi_{i-1,j-1} - \psi_{i-2,j-1}) \\
& + m_{i,j-1}^v (\psi_{i,j-1} - \psi_{i-1,j-1}) + m_{i-1,j}^v (\psi_{i-1,j} - \psi_{i-2,j}) + m_{i,j}^v (\psi_{i,j} - \psi_{i-1,j})] \\
& [c_{i-1,j} (m_{i-1,j+1}^u (\psi_{i-1,j+1} - \psi_{i-1,j}) + m_{i-1,j}^u (\psi_{i-1,j} - \psi_{i-1,j-1})) \\
& - c_{i-1,j-1} (m_{i-1,j}^u (\psi_{i-1,j} - \psi_{i-1,j-1}) + m_{i-1,j-1}^u (\psi_{i-1,j-1} - \psi_{i-1,j-2}))]] \} >, \quad (3.27)
\end{aligned}$$

where  $c_{i,j} = (M_{i,j} + M_{i+1,j} + M_{i,j+1} + M_{i+1,j+1})^{-1}$ .

$$l_3 = m\delta_y \left[ (m^v)^2 \delta_x \psi \delta_y \left( \frac{1}{M} \overline{m^v \delta_x \psi^y} \right) - \overline{m^u \delta_y \psi^{xy}} m^v \delta_x \left( \frac{1}{\overline{M^{xy}}} \overline{m^v \delta_x \psi^x} \right) \right], \quad (3.28)$$

and its expansion is similar to that of  $l_2$ . Refer to Part 10 of the Appendix to see it. Finally,

$$r = m^2 \left[ \delta_x \left( \overline{\delta_x \Phi^\eta M^x} - \eta \delta_x M \overline{\delta_\eta \Phi^x} \right) + \delta_y \left( \overline{\delta_y \Phi^\eta M^y} - \eta \delta_y M \overline{\delta_\eta \Phi^y} \right) \right], \quad (3.29)$$

and its expansion is

$$\begin{aligned}
r_{i,j}^k = & \frac{m_{i,j}^2}{2(\Delta x)^2} \left\{ \frac{1}{2} \left[ \left( \Phi_{i+1,j}^{k-1} - \Phi_{i,j}^{k-1} + \Phi_{i+1,j}^k - \Phi_{i,j}^k \right) (M_{i,j} + M_{i+1,j}) \right. \right. \\
& - \left( \Phi_{i,j}^{k-1} - \Phi_{i-1,j}^{k-1} + \Phi_{i,j}^k - \Phi_{i-1,j}^k \right) (M_{i-1,j} + M_{i,j}) \\
& + \left( \Phi_{i,j+1}^{k-1} - \Phi_{i,j}^{k-1} + \Phi_{i,j+1}^k - \Phi_{i,j}^k \right) (M_{i,j} + M_{i,j+1}) \\
& - \left. \left( \Phi_{i,j}^{k-1} - \Phi_{i,j-1}^{k-1} + \Phi_{i,j}^k - \Phi_{i,j-1}^k \right) (M_{i,j-1} + M_{i,j}) \right] + \frac{\tilde{\eta}^k}{\eta^k - \eta^{k-1}} \left[ - (M_{i+1,j} - M_{i,j}) \right. \\
& \left( \Phi_{i,j}^k - \Phi_{i,j}^{k-1} + \Phi_{i+1,j}^k - \Phi_{i+1,j}^{k-1} \right) + (M_{i,j} - M_{i-1,j}) \left( \Phi_{i-1,j}^k - \Phi_{i-1,j}^{k-1} + \Phi_{i,j}^k - \Phi_{i,j}^{k-1} \right) \\
& - (M_{i,j+1} - M_{i,j}) \left( \Phi_{i,j}^k - \Phi_{i,j}^{k-1} + \Phi_{i+1,j}^k - \Phi_{i+1,j}^{k-1} \right) \\
& \left. \left. + (M_{i,j} - M_{i,j-1}) \left( \Phi_{i,j-1}^k - \Phi_{i,j-1}^{k-1} + \Phi_{i,j}^k - \Phi_{i,j}^{k-1} \right) \right] \right\}. \tag{3.30}
\end{aligned}$$

These discretizations were then applied to Test Case 1 to determine the degree of imbalance in the flow at hour 117. Figure 3.8 shows the results. The top panel (Fig. 3.8a), showing the combination of terms  $l_1$ ,  $l_2$ , and  $l_3$ , should be compared to Fig. 3.6b. The difference between these two figures is small, and is due to the fact that, in the current case, the streamfunction is being used rather than the original wind data. [Note that Eq. (3.22) has been multiplied by  $-1$  relative to Eq. (3.14).] The impact of this is that the results are smoother when the streamfunction is used. The center panel (Fig. 3.8b) shows term  $r$ , and the lower panel (Fig. 3.8c) shows the imbalance, which is quite reminiscent of Fig. 3.6c, as expected.

Turning our attention now to Test Case 2, the same procedure was carried out, and the analogous results are displayed in Fig. 3.9. In this real-world case, regions of relatively high imbalance are more widely dispersed throughout the domain, with hints that areas near the boundaries tend to be less in balance (possibly reflecting the imbalanced nature of the WRF boundary conditions rather than any boundary effects from the imbalance computation itself). In both cases, the imbalance is on the order of  $10^{-9} \text{ s}^{-2}$ .

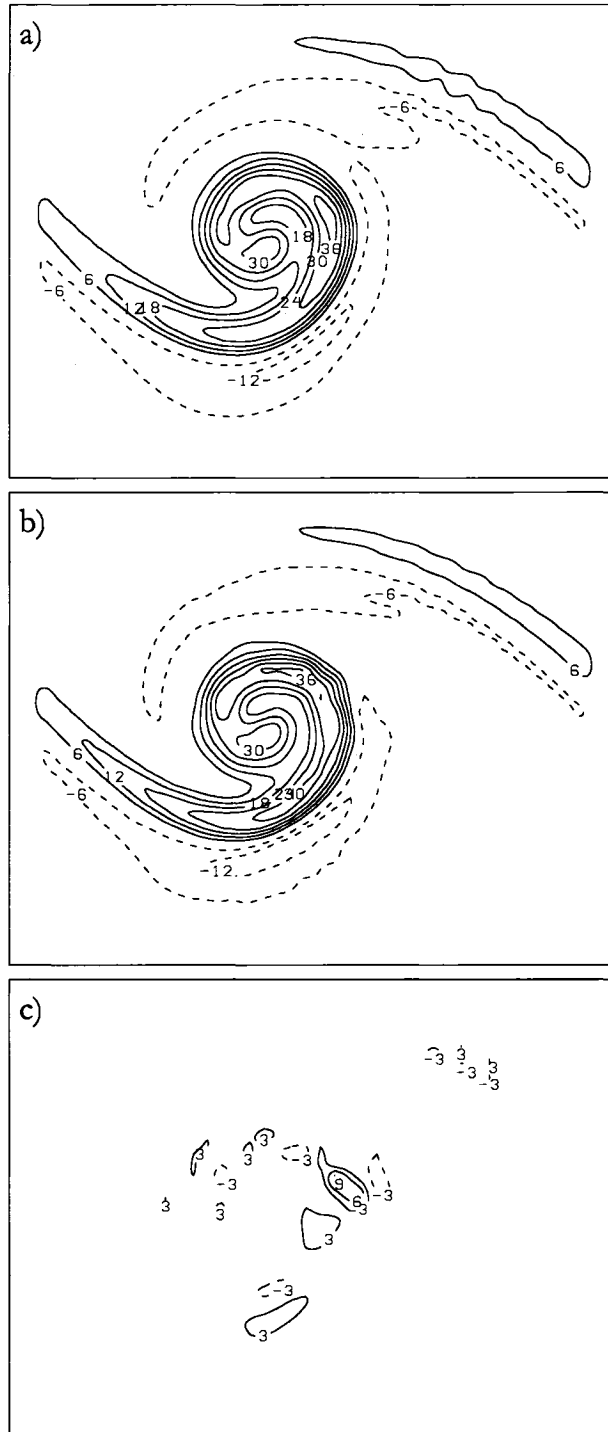


Figure 3.8: Determination of imbalance for Test Case 1. (a)  $l_1 - l_2 - l_3$  [contour interval  $6 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)], (b) same as (a) but for  $r$ , and (c) nonlinear imbalance  $(l_1 - l_2 - l_3 - r)$  [contour interval  $3 \times 10^{-9} \text{ s}^{-2}$ , positive (negative) values solid (dashed)].

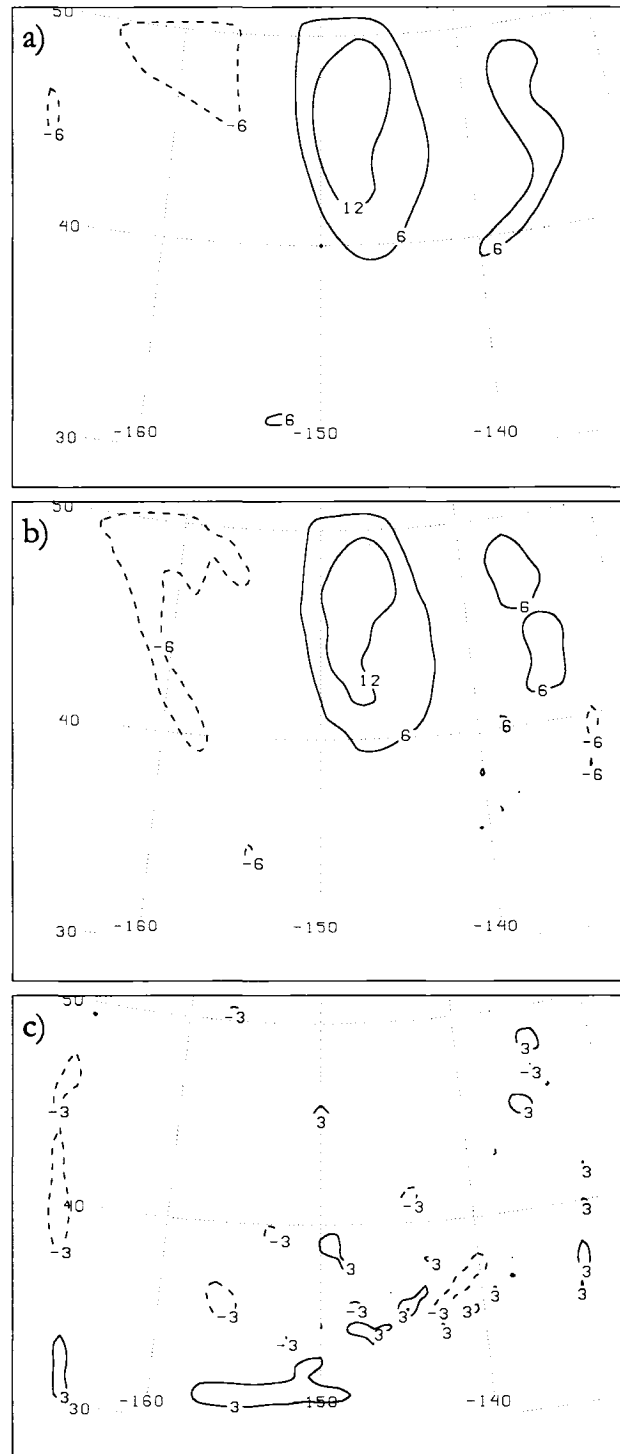


Figure 3.9: Same as Fig. 3.8 but for Test Case 2.

### 3.3 Tangent linear and adjoint

This calculation of the imbalance can be thought of as the result of multiplying a giant  $m \times n$  matrix (where  $m$  is equal to the number of  $\psi$  points in the domain plus the number of  $\Phi$  points in the domain and  $n$  is equal to the number of grid boxes in the domain) by a vector of length  $m$  containing the values of  $\psi$  and  $\Phi$  at all of their respective grid points. It is not strictly a matrix multiplication because of the nonlinear nature of the calculation; some values in the “matrix” are dependent on the values of  $\psi$  and  $\Phi$  at various locations in perhaps complicated ways. Instead, we call this pseudo-matrix the nonlinear imbalance operator, and denote it as  $\tilde{B}$ . However, if we take the derivative of IMB with respect to each  $\psi$  and  $\Phi$ , we can construct the tangent linear matrix related to  $\tilde{B}$ . Left multiplying a perturbation vector of values for  $\delta\psi$  and  $\delta\Phi$  by the tangent linear matrix produces an output vector of  $\delta\text{IMB}$  corresponding to a change in the imbalance at each grid point.  $B^*$ , the adjoint of  $\tilde{B}$ , is the transpose of this tangent linear matrix.

To illustrate, consider the following example, adapted from one given by Kalnay (2003). Suppose the state vector consists of  $\mathbf{X} = \begin{pmatrix} \psi_1 \\ \psi_2 \end{pmatrix}$  and the imbalance operator is defined as

$$\text{IMB} = \tilde{B}\mathbf{X} = \psi_1 \sin(\psi_2^2) + c_p \psi_1. \quad (3.31)$$

The tangent linear version of this operator tells us how the imbalance changes ( $\delta\text{IMB}$ ) as a result of changes to the dependent variables ( $\delta\psi_1$  and  $\delta\psi_2$ ). This can be written as

$$\delta\text{IMB} = (\sin(\psi_2^2) + c_p) \delta\psi_1 + 2\psi_1 \psi_2 \cos(\psi_2^2) \delta\psi_2, \quad (3.32)$$

where the general formula

$$\delta\text{IMB} = \sum_k \frac{\partial \text{IMB}}{\partial \psi_k} \delta\psi_k \quad (3.33)$$

has been used. The key here is that all of the  $\frac{\partial \text{IMB}}{\partial \psi_k}$  must be derived (or in the case of imbalance,

$\frac{\partial \text{IMB}}{\partial X_k}$ ). Eq. (3.32) is then converted to matrix form:

$$\delta \text{IMB} = \begin{pmatrix} \sin(\psi_2^2) + c_p & 2\psi_1\psi_2 \cos(\psi_2^2) \end{pmatrix} \begin{pmatrix} \delta\psi_1 \\ \delta\psi_2 \end{pmatrix} \quad (3.34)$$

Implicit in this equation is the fact that the  $\delta\psi_k$  are left unchanged. To reflect this, and to more easily derive the adjoint, the matrix form is expanded to

$$\begin{pmatrix} \delta\psi_1 \\ \delta\psi_2 \\ \delta \text{IMB} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \sin(\psi_2^2) + c_p & 2\psi_1\psi_2 \cos(\psi_2^2) & 0 \end{pmatrix} \begin{pmatrix} \delta\psi_1 \\ \delta\psi_2 \\ \delta \text{IMB} \end{pmatrix}. \quad (3.35)$$

The  $3 \times 3$  matrix in Eq. (3.35) is the tangent linear matrix for this case. To determine the adjoint of  $\tilde{B}$ , the tangent linear matrix is transposed, resulting in

$$\begin{pmatrix} \delta^*\psi_1 \\ \delta^*\psi_2 \\ \delta^*\text{IMB} \end{pmatrix}^{(1)} = \begin{pmatrix} 1 & 0 & \sin(\psi_2^2) + c_p \\ 0 & 1 & 2\psi_1\psi_2 \cos(\psi_2^2) \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta^*\psi_1 \\ \delta^*\psi_2 \\ \delta^*\text{IMB} \end{pmatrix}^{(0)}, \quad (3.36)$$

where asterisks indicate adjoint variables. The adjoint can then be programmed using the assignments

$$\begin{cases} \delta^*\psi_1 := \delta^*\psi_1 + [\sin(\psi_2^2) + c_p] \delta^*\tilde{B} \\ \delta^*\psi_2 := \delta^*\psi_2 + 2\psi_1\psi_2 \cos(\psi_2^2) \delta^*\tilde{B} \\ \delta^*\text{IMB} := 0 \end{cases}$$

where the values of the nonadjoint variables ( $\psi_1$  and  $\psi_2$ ) are taken from their current best guess. (This is the basic state about which the operators have been linearized.)

Thus, in the case of the nonlinear imbalance operator, we must derive  $\frac{\partial \text{IMB}}{\partial X_k}$  for all  $k$ . These



derivatives are then used in the tangent linear version to compute  $\delta\text{IMB}$  according to the pattern

$$\begin{aligned}
\delta\text{IMB}_{i,j}^k = & a_{i,j}^k \delta\psi_{i-1,j-2}^k + b_{i,j}^k \delta\psi_{i,j-2}^k + c_{i,j}^k \delta\psi_{i-2,j-1}^k + d_{i,j}^k \delta\psi_{i-1,j-1}^k + e_{i,j}^k \delta\psi_{i,j-1}^k \\
& + f_{i,j}^k \delta\psi_{i+1,j-1}^k + g_{i,j}^k \delta\psi_{i-2,j}^k + h_{i,j}^k \delta\psi_{i-1,j}^k + i_{i,j}^k \delta\psi_{i,j}^k + j_{i,j}^k \delta\psi_{i+1,j}^k \\
& + k_{i,j}^k \delta\psi_{i+1,j}^k + l_{i,j}^k \delta\psi_{i-1,j+1}^k + m_{i,j}^k \delta\psi_{i,j+1}^k + n_{i,j}^k \delta\Phi_{i,j-1}^{k-1} + o_{i,j}^k \delta\Phi_{i-1,j}^{k-1} \\
& + p_{i,j}^k \delta\Phi_{i,j}^{k-1} + q_{i,j}^k \delta\Phi_{i+1,j}^{k-1} + r_{i,j}^k \delta\Phi_{i,j+1}^{k-1} + s_{i,j}^k \delta\Phi_{i,j-1}^{k+1} + t_{i,j}^k \delta\Phi_{i,j-1}^{k+1} \\
& + u_{i,j}^k \delta\Phi_{i,j}^{k+1} + v_{i,j}^k \delta\Phi_{i+1,j}^{k+1} + w_{i,j}^k \delta\Phi_{i,j+1}^{k+1},
\end{aligned} \tag{3.37}$$

where the  $a_{i,j}^k - w_{i,j}^k$  are derived from the full expansion of the nonlinear balance operator. (A computer algebra system assisted in deriving the coefficients.) In general, these coefficients are too lengthy for inclusion here. One of the few exceptions is  $n_{i,j}^k$ , which can be written as

$$n_{i,j}^k = -\frac{m_{i,j}^2}{2(\Delta x)^2} \left[ \frac{\tilde{\eta}^k}{\eta^k - \eta^{k-1}} (M_{i,j-1} - M_{i,j}) + \frac{1}{2} (M_{i,j-1} + M_{i,j}) \right]. \tag{3.38}$$

For the complete result, refer to Part 10 of the Appendix.

The tangent linear version of the imbalance operator can be tested by comparing how the tangent linear version predicts a perturbation will affect the imbalance to the actual change in the imbalance upon adding that perturbation. For Test Case 2, a perturbation of  $5 \times 10^6 \text{ m}^2 \text{ s}^{-1}$  was added to the streamfunction at gridpoint (5,6), level  $\eta = 0.525$ . In addition, a perturbation of  $50 \text{ m}^2 \text{ s}^{-2}$  was added to the geopotential at gridpoint (5,6), level  $\eta = 0.55$ . The results of this test are shown in Figure 3.10. The left half of the figure contains results for  $\eta = 0.525$ , whereas the right half of the figure contains results for  $\eta = 0.575$ . The top panels (Figs. 3.10a–b) show the imbalance (zoomed into the southwestern part of the domain). The introduction of the perturbations has led to a large increase in the imbalance in their vicinity. Note how the patterns away from the perturbation match those in Fig. 3.9c. The second row of panels (Figs. 3.10c–d) isolates the perturbations themselves. The perturbations are not symmetric because the background fields of geopotential and streamfunction vary from place to place. Finally, the bottom row (Figs. 3.10e–f) displays the output from the tangent linear version of the imbalance operator. Although the

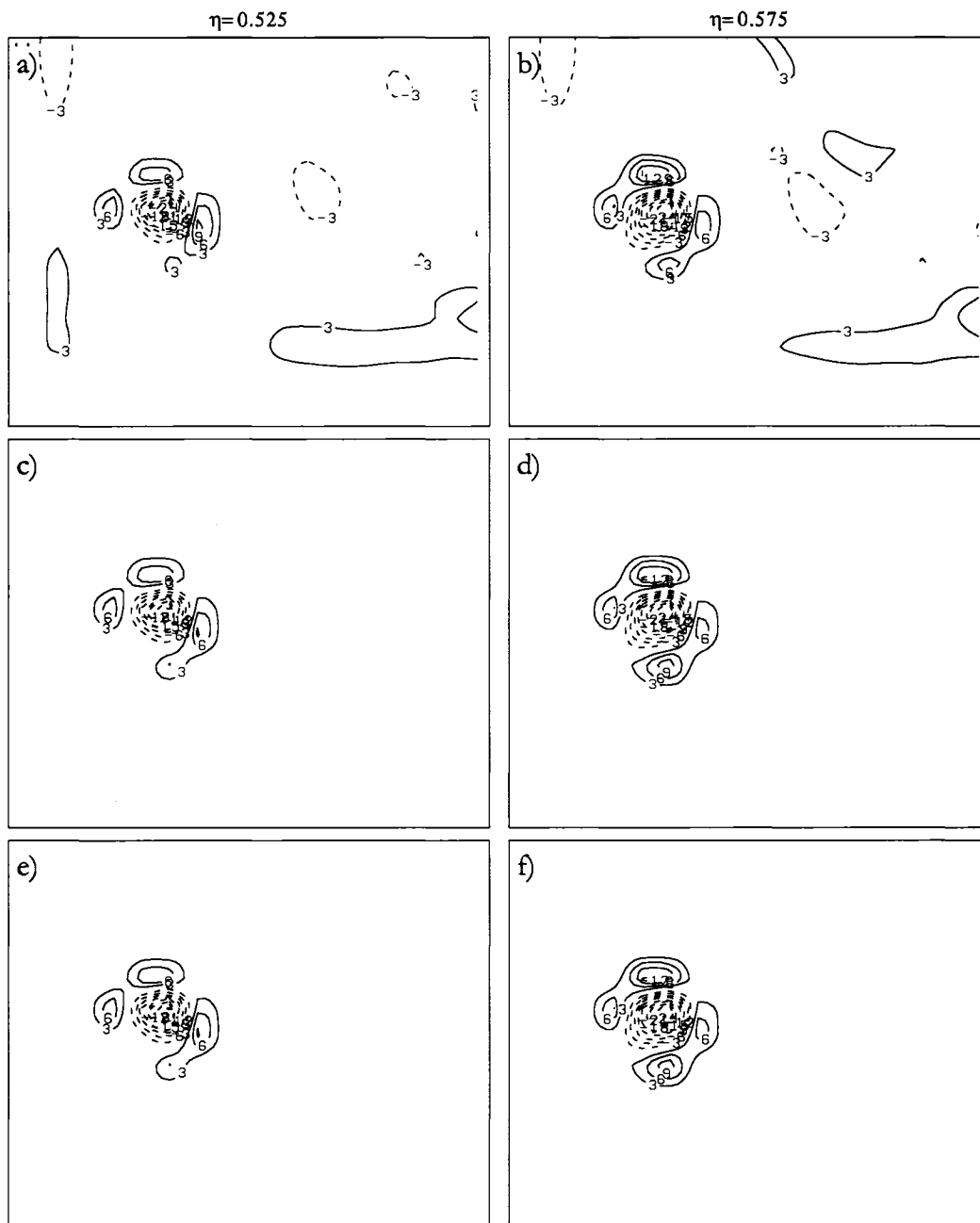


Figure 3.10: Checking the tangent linear version of the imbalance operator for Test Case 2. (a) Result of imbalance operator at  $\eta = 0.525$  after adding the perturbations described in the text [contour interval  $3 \times 10^{-9} \text{ s}^{-2}$ , with positive (negative) values solid (dashed)]. (b) Same as panel (a) but for  $\eta = 0.575$ . (c) Same as (a) but showing the difference on the imbalance operator between perturbed and nonperturbed input. (d) Same as (c) but for  $\eta = 0.575$ . (e) Same as (a) but showing output from the tangent linear version of the imbalance operator. (f) Same as (e) but for  $\eta = 0.575$ .

tangent linear output is only an approximation of the effects of the perturbations, the behavior of the imbalance operator is apparently only weakly nonlinear, since the bottom two rows match nearly identically. Similar checks were made in other locations, all of which showed very close correspondence between the tangent linear estimated change in the imbalance, and the actual change that resulted from the perturbations.

The adjoint version of the imbalance operator was then derived following the principles outlined in the simplified example above, and its implementation is also shown in Part 10 of the Appendix. One of the assignment statements in the adjoint routine is

$$\delta^* \Phi_{i,j-1}^{k-1} := \delta^* \Phi_{i,j-1}^{k-1} - \frac{m_{i,j}^2}{2(\Delta x)^2} \left[ \frac{\tilde{\eta}^k}{\eta^k - \eta^{k-1}} (M_{i,j-1} - M_{i,j}) + \frac{1}{2} (M_{i,j-1} + M_{i,j}) \right] \delta^* \text{IMB}_{i,j}^k,$$

associated with the portion of the tangent linear routine shown in Eq. (3.38). The standard adjoint check was performed following Kalnay (2003). The check makes use of the identity

$$(B\delta\mathbf{X})^T (B\delta\mathbf{X}) = (\delta\mathbf{X})^T [B^* (B\delta\mathbf{X})], \quad (3.39)$$

where  $B$  is the tangent linear version of the nonlinear balance operator. If the adjoint is accurate, both sides of the equation should agree to within machine precision, and this was indeed the case.

## Chapter 4

# The PV operator and its adjoint

### 4.1 Derivation of operator

Potential vorticity is defined most generally as

$$q = \alpha \left( \nabla \theta \cdot \vec{\xi} \right), \quad (4.1)$$

where  $\vec{\xi}$  is the three-dimensional absolute vorticity vector. For large-scale flow, the dot product can be simplified to use the vertical components of the associated vectors only; the result in Cartesian coordinates is

$$q \approx \alpha \frac{\partial \theta}{\partial z} \left( f + \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right). \quad (4.2)$$

To transform this expression to the WRF terrain-following coordinate, the hydrostatic equation and the chain rule is used, resulting in

$$q \approx -\frac{1}{\mu} \frac{\partial \Phi}{\partial \eta} \frac{\partial \theta}{\partial z} \left( f + \frac{\partial v}{\partial x} - \frac{\partial \eta}{\partial \Phi} \frac{\partial \Phi}{\partial x} \frac{\partial v}{\partial \eta} - \frac{\partial u}{\partial y} + \frac{\partial \eta}{\partial \Phi} \frac{\partial \Phi}{\partial y} \frac{\partial u}{\partial \eta} \right). \quad (4.3)$$

This expression is still not ready for use in the PV inversion routine, since the only dependent variables should be streamfunction and geopotential. This can be accomplished by replacing  $\frac{\partial \theta}{\partial z}$  with a new expression and using our definition of the streamfunction.

A new expression for  $\frac{\partial\theta}{\partial z}$  can be derived by equating geometric height with geopotential height and then using the definition of geopotential height, giving

$$\frac{\partial\theta}{\partial z} \approx g \frac{\partial\theta}{\partial\Phi}. \quad (4.4)$$

Next, the definition of potential temperature along with the thickness equation and the definition of  $\eta$  allow us to write

$$\theta^k \approx \frac{\Phi^k - \Phi^{k-1}}{R \ln \left( \frac{p_t + \eta^{k-1}\mu}{p_t + \eta^k\mu} \right)} \left( \frac{p_0}{p_t + \eta^k\mu} \right)^{\frac{R}{c_p}}, \quad (4.5)$$

where superscript  $k$ 's indicate the vertical level, water substance has been neglected, and  $\tilde{\eta}^k$  represents the value of  $\eta$  on a half-level.

Potential temperature is provided as output from the WRF model, so the degree to which Eq. (4.5) accurately approximates the potential temperature can be quantified. Figure 4.1 compares the potential temperature field with its approximation for Test Cases 1 and 2. Considering Test Case 1 (Figs. 4.1a,c), it can be seen that the approximation to the potential temperature is visually indistinct from its actual distribution. The approximation is also quite good for Test Case 2 (Figs. 4.1b,d). However, careful inspection reveals that the approximated values of  $\theta$  are often slightly higher than the values of  $\theta$  taken directly from the model output. This bias is an effect of neglecting water substance (and in particular, the vapor pressure) in Eq. (4.5). By excluding the vapor pressure from the computation, the (total) pressure used in computing the potential temperature is slightly underestimated, leading to an overestimation of  $\theta$ . This effect is most pronounced near the surface, where vapor pressure is typically maximized, and decreases with height. An implication of this is that the static stability, as estimated from geopotential, will tend to be lower than it actually is. Introducing the vapor pressure into the approximation for  $\theta$  is left for future work.

Now that the necessary ingredients are in place, the discretized form of the potential vorticity

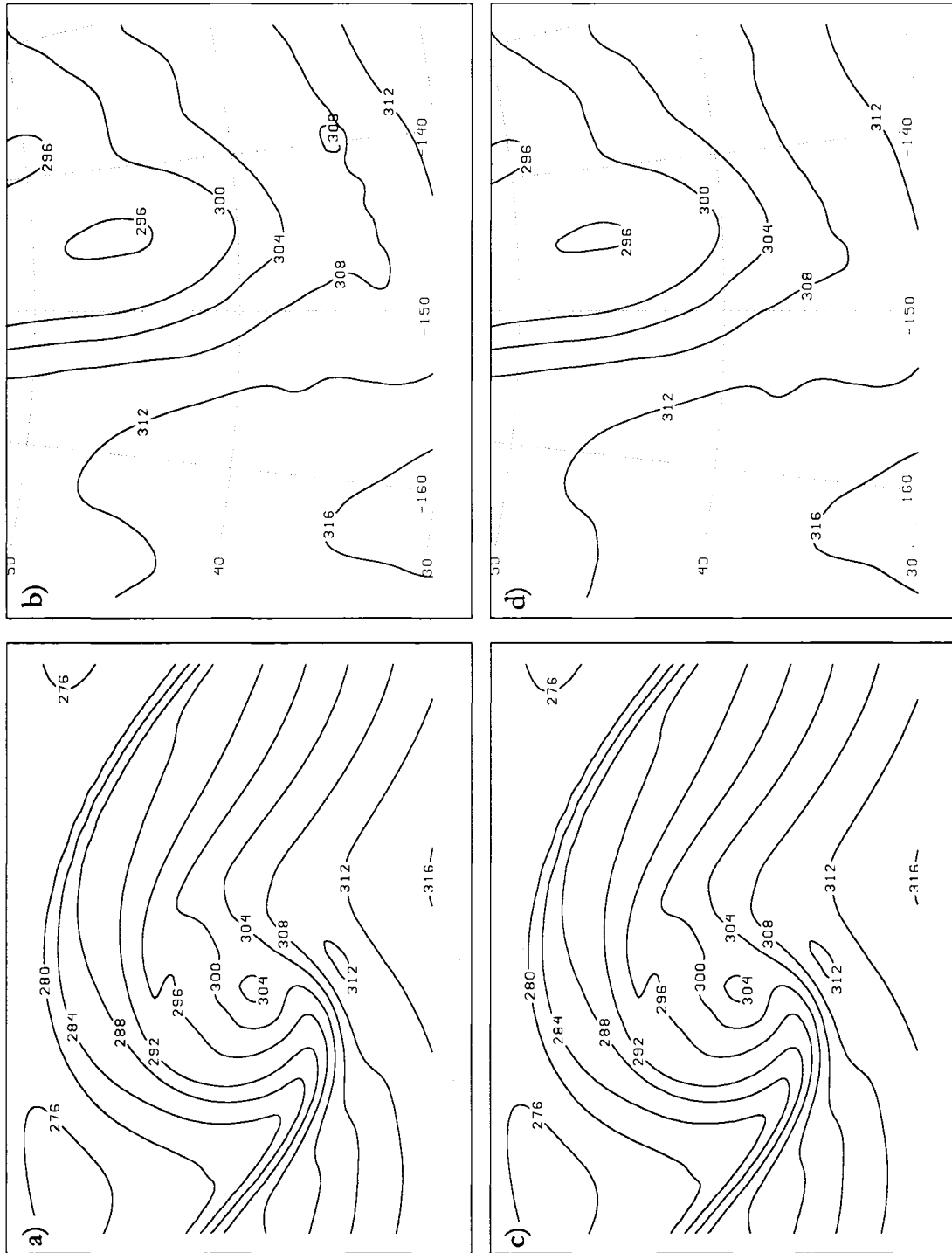


Figure 4.1: Potential temperature (contour interval 4 K) from (a) model output from Test Case 1, (b) model output from Test Case 2, (c) Eq. (4.5) applied to Test Case 1, and (d) Eq. (4.5) applied to Test Case 2.

can be written as

$$q \approx -\frac{g}{\mu} \delta_\eta \Phi \delta_\Phi \bar{\theta}^\eta \left\{ f + \delta_x \left[ \overline{\frac{1}{M^y}} \delta_x \psi \right]^{xy} + \partial_y \left[ \overline{\frac{1}{M^x}} \delta_y \psi \right]^{xy} - \delta_\Phi \eta \left[ \overline{\delta_x \Phi^{x\eta}} \partial_\eta \left( \overline{\frac{1}{M^y}} \delta_x \psi \right)^y + \overline{\delta_y \Phi^{y\eta}} \partial_\eta \left( \overline{\frac{1}{M^x}} \delta_y \psi \right)^x \right] \right\}, \quad (4.6)$$

where  $\bar{\theta}^\eta$  at level  $k$  is given by  $\frac{1}{2} (\theta^k + \theta^{k+1})$  and  $\theta^k$  is given by Eq. (4.5). The expansion of this equation is quite lengthy; the interested reader is referred to Part 10 of the Appendix for more details.

The discretized version of the potential vorticity given by Eq. (4.6) was compared to a discrete version of Eq. (4.3) to determine its accuracy. Figure 4.2 shows that, in general, the potential vorticity based solely on  $\psi$  and  $\Phi$  is quite similar to the less approximated version. Upon closer inspection, it can be seen that, in general, the magnitude of the PV is slightly less when based on Eq. (4.6) in comparison to Eq. (4.3). For instance, the 1 PVU contour in the upper-right portion of Fig. 4.2b is smaller than the corresponding contour in Fig. 4.2a. This is confirmed in Fig. 4.2c. The reduction in PV is primarily a result of the reduction in the static stability introduced by neglecting vapor pressure in the expression for potential temperature, Eq. (4.5).

## 4.2 Tangent linear and adjoint

The tangent linear version of the PV operator is derived in a manner analogous to that of the imbalance operator. The level of complexity is increased somewhat, however, as the analogous

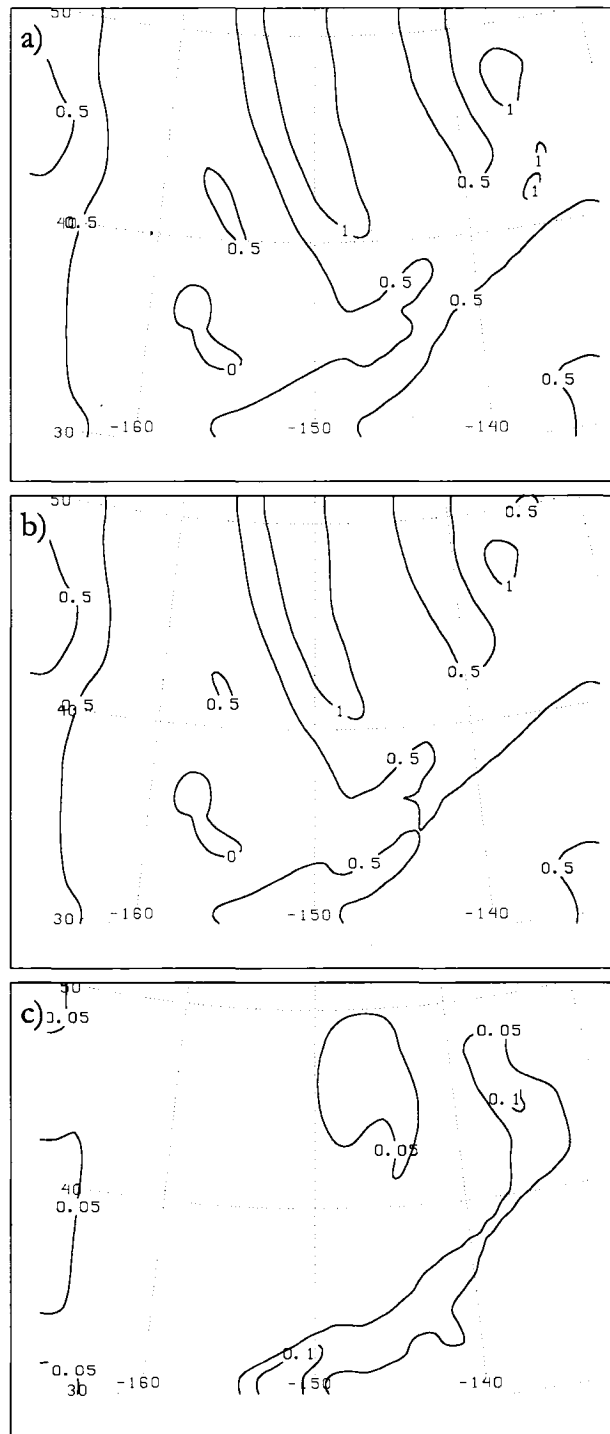


Figure 4.2: (a) Potential vorticity calculated from Eq. (4.3) for Test Case 2 (contour interval 0.5 PVU). (b) Same as (a) but calculated from Eq. (4.6). (c) The graphical difference between (a) and (b) (contour interval 0.05 PVU, zero contour omitted).



expression to Eq. (3.37) is

$$\begin{aligned}
\delta q_{i,j}^k = & a_{i,j}^k \delta \psi_{i-1,j-1}^{k-1} + b_{i,j}^k \delta \psi_{i,j-1}^{k-1} + c_{i,j}^k \delta \psi_{i-1,j}^{k-1} + d_{i,j}^k \delta \psi_{i,j}^{k-1} + e_{i,j}^k \delta \psi_{i-1,j-2}^k \\
& + f_{i,j}^k \delta \psi_{i,j-2}^k + g_{i,j}^k \delta \psi_{i-2,j-1}^k + h_{i,j}^k \delta \psi_{i-1,j-1}^k + i_{i,j}^k \delta \psi_{i,j-1}^k + j_{i,j}^k \delta \psi_{i+1,j-1}^k \\
& + k_{i,j}^k \delta \psi_{i-2,j}^k + l_{i,j}^k \delta \psi_{i-1,j}^k + m_{i,j}^k \delta \psi_{i,j}^k + n_{i,j}^k \delta \psi_{i+1,j}^k + o_{i,j}^k \delta \psi_{i-1,j+1}^k \\
& + p_{i,j}^k \delta \psi_{i,j+1}^k + q_{i,j}^k \delta \psi_{i-1,j+1}^{k+1} + r_{i,j}^k \delta \psi_{i,j+1}^{k+1} + s_{i,j}^k \delta \psi_{i-1,j+1}^{k+1} + t_{i,j}^k \delta \psi_{i,j+1}^{k+1} \\
& + u_{i,j}^k \delta \Phi_{i,j}^{k-2} + v_{i,j}^k \delta \Phi_{i,j-1}^{k-1} + w_{i,j}^k \delta \Phi_{i-1,j}^{k-1} + x_{i,j}^k \delta \Phi_{i,j}^{k-1} + y_{i,j}^k \delta \Phi_{i+1,j}^{k-1} \\
& + z_{i,j}^k \delta \Phi_{i,j+1}^{k-1} + \alpha_{i,j}^k \delta \Phi_{i,j-1}^k + \beta_{i,j}^k \delta \Phi_{i-1,j}^k + \gamma_{i,j}^k \delta \Phi_{i,j}^k + \sigma_{i,j}^k \delta \Phi_{i+1,j}^k \\
& + \tau_{i,j}^k \delta \Phi_{i,j+1}^k + \nu_{i,j}^k \delta \Phi_{i,j}^{k+1}, \tag{4.7}
\end{aligned}$$

where the  $a_{i,j}^k - \nu_{i,j}^k$  are derived from the full expansion of the potential vorticity operator.

As an example showing a typical form for some of the coefficients,

$$e_{i,j}^k = \frac{m_{i-1,j-1}^{\psi} c_{i-1,j-1}}{2 (\Delta x)^2} \left[ \kappa_{i,j}^{k-1} \left( \Phi_{i,j}^{k-1} - \Phi_{i,j}^{k-2} \right) - \kappa_{i,j}^{k+1} \left( \Phi_{i,j}^{k+1} - \Phi_{i,j}^k \right) \right], \tag{4.8}$$

where  $c_{i,j} = \frac{m_{i,j}^u}{M_{i,j} + M_{i+1,j}}$  and  $\kappa_{i,j}^k = \left[ \frac{p_0}{p_t + \eta^k m_{i,j}} \right]^{\frac{R}{c_p}} \left[ R \ln \left( \frac{p_t + \eta^{k-1} m_{i,j}}{p_t + \eta^k m_{i,j}} \right) \right]^{-1}$ . For the complete result, refer to Part 10 of the Appendix.

The same perturbations employed during testing of the tangent linear form of the imbalance operator were used to test the tangent linear form of the PV operator. Figure 4.3 displays the results of a test analogous to that displayed in Fig. 3.10. Evidently, the same perturbations that produced a large change in the imbalance operator have a more muted effect on PV. At the  $\eta = 0.525$  level, shown in the left column of the figure, the perturbations change the PV on the order of 0.5 PVU. Curiously, the changes in PV extend in a more pronounced way towards the southwest from the central perturbed gridpoint, whereas the changes in the imbalance extended in that direction the least. Changes to the PV at the  $\eta = 0.575$  level are even more muted still (Figs. 4.3b,d,f). The PV operator appears to be more nonlinear than the imbalance operator, since the tangent linear prediction of  $\delta PV$  (Figs. 4.3e,f) is more noticeably different than the actual  $\delta PV$  (Figs. 4.3c,d)

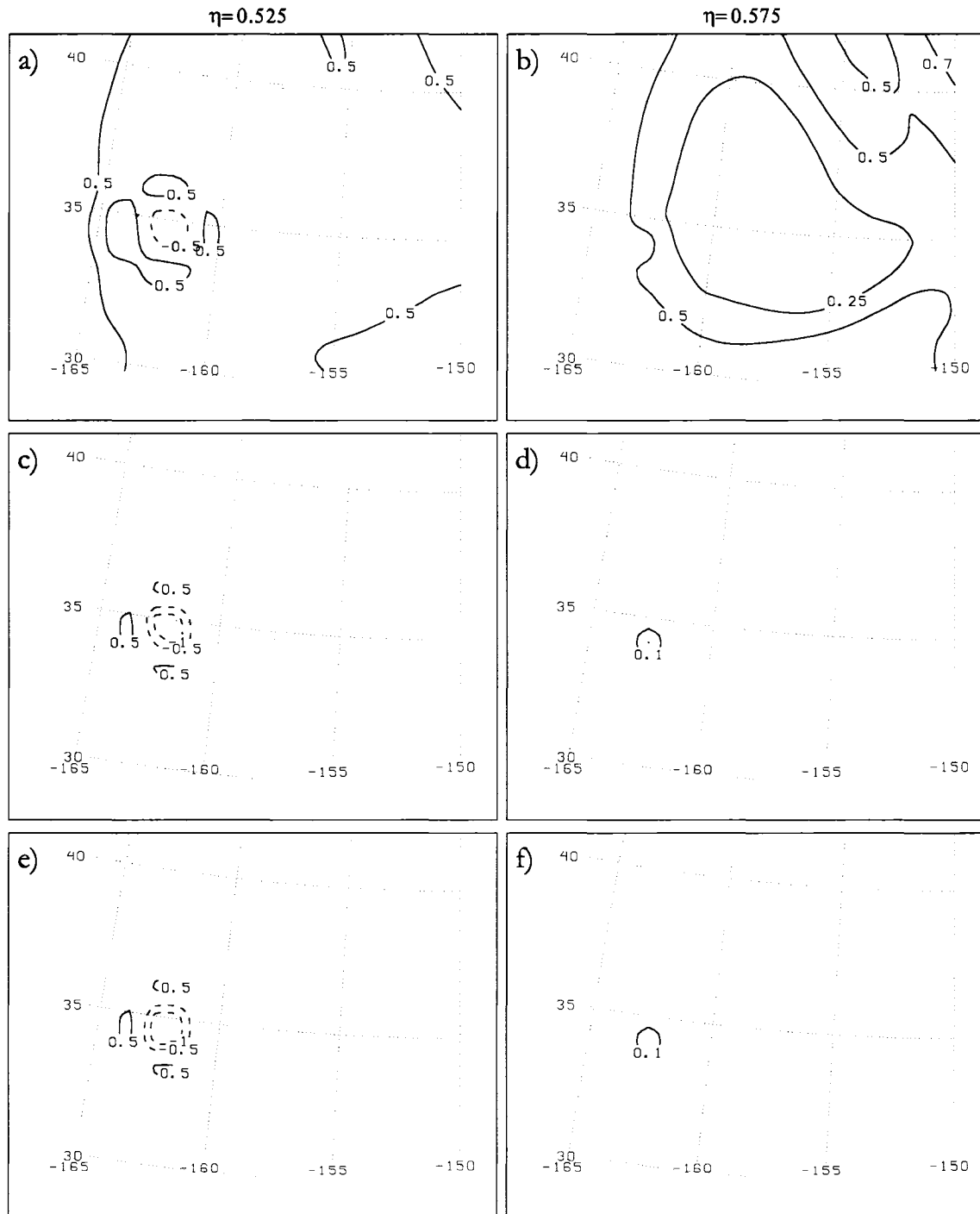


Figure 4.3: Same as Fig. 3.10 except showing the tangent linear version of the potential vorticity operator [contour interval 0.25 PVU, positive (negative) values solid (dashed), except (b) contour interval 0.25 PVU, and (d) and (f) contour interval 0.1 PVU].

than was the case for the imbalance. As in the case of the imbalance, checks such as the one shown above were made for different perturbations, and the results were similar, indicating that the tangent linear version of the PV operator is accurate.

The adjoint of the PV operator,  $P^*$ , was derived in a process analogous to the one described in the previous chapter for the imbalance adjoint. One of the assignment statements in the adjoint routine is

$$\delta^* \psi_{i-1,j-2}^k := \delta^* \psi_{i-1,j-2}^k + \frac{m_{i-1,j-1}^\psi c_{i-1,j-1}}{2(\Delta x)^2} \left[ \kappa_{i,j}^{k-1} (\Phi_{i,j}^{k-1} - \Phi_{i,j}^{k-2}) - \kappa_{i,j}^{k+1} (\Phi_{i,j}^{k+1} - \Phi_{i,j}^k) \right] \delta^* \tilde{P}_{i,j}^k, \quad (4.9)$$

and the complete routine can be found in Part 10 of the Appendix.

## Chapter 5

# Putting the pieces together: PV inversion

### 5.1 Review

Recall that we are attempting to minimize the cost function given by Eq. (2.8) using its gradient, given by Eq. (2.9). Now that the adjoints of the nonlinear balance and PV operators have been implemented and tested, we are ready to do so.

### 5.2 Procedure

The value of  $\epsilon$  has heretofore been left undefined. However, based on our knowledge that PV has values in SI units on the order of  $10^{-6} \text{ m}^2 \text{ K kg}^{-1} \text{ s}^{-1}$  and that the imbalance operator typically has values on the order of  $10^{-9} \text{ s}^{-2}$ , we can make the following inferences. First, consider how accurate the calculated PV should be relative to any given PV distribution in a typical grid box. If we set this accuracy to 0.1 PVU, then the first term in the cost function will have a value of about  $10^{-14} \text{ m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^{-2}$  times the number of gridboxes in the domain when the solution has been reached. On the other hand, the second term in the cost function will have a value of about  $10^{-18} \text{ s}^{-4}$  times the number of gridboxes in the domain when the solution has been reached. This suggests that

$\varepsilon$  should have a value on the order of  $10^4 \text{ m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^2$ . Values greater than this will enforce greater nonlinear balance at the expense of a greater mismatch between the calculated PV and the given PV distribution, whereas values less than this will more heavily weight the match between the calculated and given PV distributions. During the work described here,  $\varepsilon \equiv 5000 \text{ m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^2$ , giving the PV match a bit more weight.

The lateral boundary conditions are given by fixed values of  $\psi$  and  $\Phi$  that match those in the input data. In other words, only the interior values of  $\psi$  and  $\Phi$  are allowed to fluctuate as the minimization is carried out. At the lower boundary,  $\Phi$  is defined by the elevation of the earth's surface, and it is held fixed at the upper boundary. An alternative upper boundary condition that could be implemented would be to set the temperature in the top layer to a fixed value representative of the lower stratosphere (the WRF model top is usually 50 hPa), and set  $\frac{\partial \Phi}{\partial \eta}$  accordingly. The streamfunction is only involved in horizontal derivatives, so it is free to vary in the top and bottom layers. Finally, the mass of the atmosphere in each column serves as a boundary condition that determines the pressure at each grid point. In other words,  $\mu$  is specified for all  $i$  and  $j$ .

The cost function is minimized through the method of steepest descent. Starting from a first guess, the gradient of the cost function,  $\nabla J$ , is computed. A line search is then performed in the direction opposite to that of the gradient (given by  $\mathbf{Y} = -\frac{\nabla J}{|\nabla J|}$ ) until the vector  $\Delta \mathbf{X} = \alpha \mathbf{Y}$  that minimizes  $J$  along that line is found. The line search is carried out by considering  $\alpha$  to lie in an interval ranging from zero to some large real number. By subdividing this interval repeatedly, while checking the value of the cost function for each trial  $\Delta \mathbf{X}$ , a reasonable value for  $\alpha$  may be found. The current implementation stops the line search when the width of the interval divided by its midpoint is less than  $10^{-5}$ . The current guess for  $\mathbf{X}$  is replaced by  $\mathbf{X} + \Delta \mathbf{X}$ , and the process is repeated until  $J$  is minimized.  $J$  is considered minimized when the value of  $J$  increases after an iteration or after 10 000 iterations have been performed.

## 5.3 Results

Three trials of the PV inversion technique have been carried out, all of which are based off of Test Case 2.

### 5.3.1 Trial 1

In the first test, the first guesses for the streamfunction and geopotential match their “true” values. Thus, the output of the inversion should nearly equal the input. Over the course of the minimization, the cost function decreased from  $1.62 \times 10^{-10} \text{ m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^{-2}$  to  $3.76 \times 10^{-11} \text{ m}^4 \text{ K}^2 \text{ kg}^{-2} \text{ s}^{-2}$ . The results of this inversion are shown in Fig. 5.1. The top panels (Figs. 5.1a–b) show actual conditions diagnosed from the model output for Test Case 2. The geopotential heights are shown rather than geopotential for familiarity, but geopotential is the field used during inversion. The bottom panels (Figs. 5.1c–d) show the output of the inversion routine. These “balanced” heights and streamfunction values are nearly identical to their original values, indicating that the original fields were nearly balanced to begin with. (We already knew this based on Fig. 3.9c.)

### 5.3.2 Trial 2

For a more stringent test of the inversion technique, the streamfunction field was altered on the interior, the intention being to use the boundary values of the streamfunction to determine a plausible first-guess streamfunction field via interpolation. The interpolation was later found to be improperly programmed, resulting in large gradients of  $\psi$  along the boundary in the input field. Nevertheless, a robust inversion routine should not be dependent on how good the initial guesses are. When given the altered streamfunction field as an initial guess (the geopotential initial guess was left identical to the model output), the inversion routine again minimized the cost function, as shown in Fig. 5.2. Notice, however, that the minimum value of the cost function is much greater than that found during the first trial. This suggests the inversion is not very accurate, and this is confirmed when the inverted fields are displayed (Fig. 5.3). Again, the top panels show the original fields of geopotential height and streamfunction. If the inversion were able to reproduce

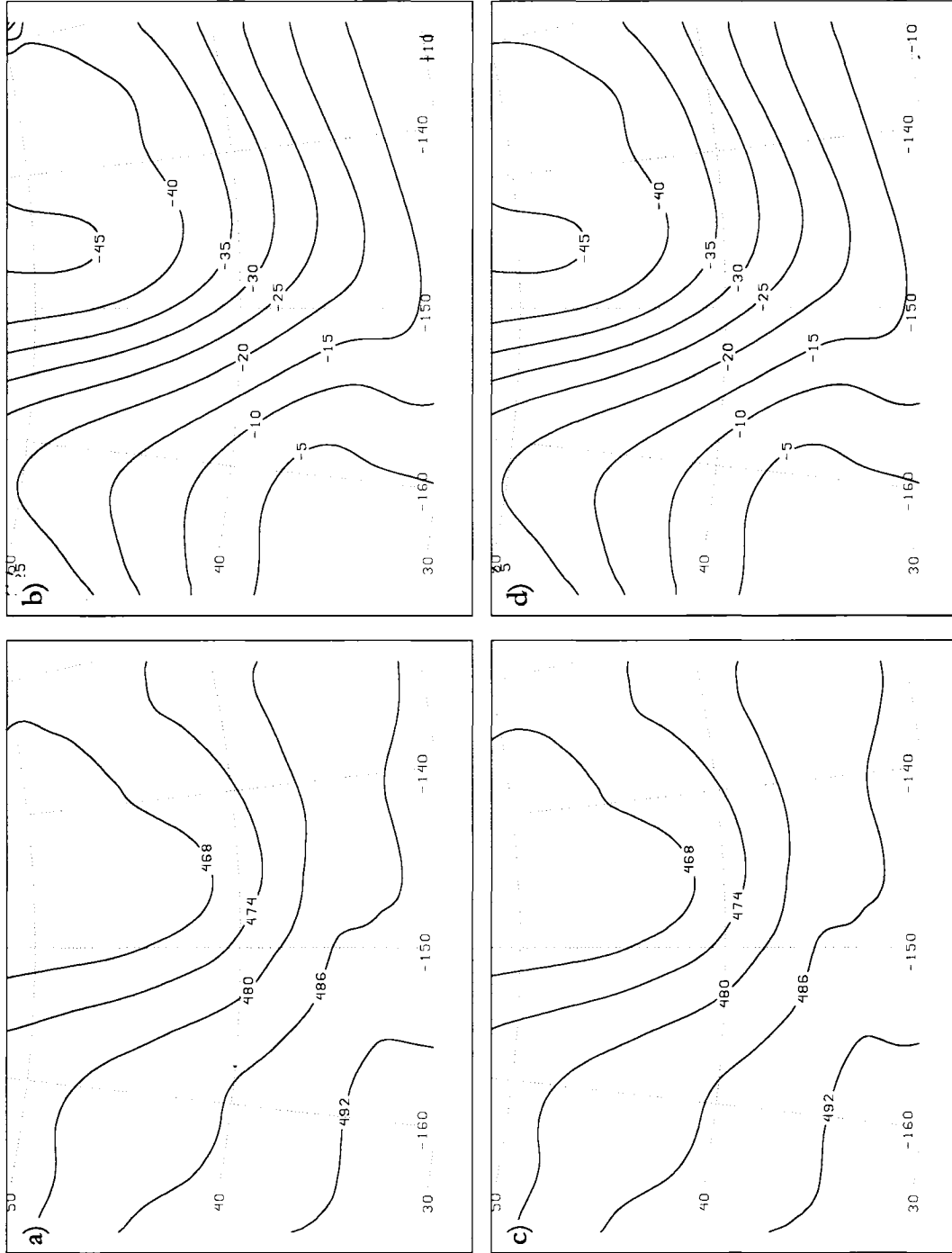


Figure 5.1: A trivial case of PV inversion, as described in the text. (a) Diagnosed geopotential heights at  $\eta = 0.525$  (contour interval 6 dam). (b) Diagnosed streamfunction at  $\eta = 0.525$  (contour interval  $5 \times 10^6 \text{ m}^2 \text{ s}^{-1}$ ). (c) Same as (a) except showing the result of the inversion. (d) Same as (b) except showing the result of the inversion.

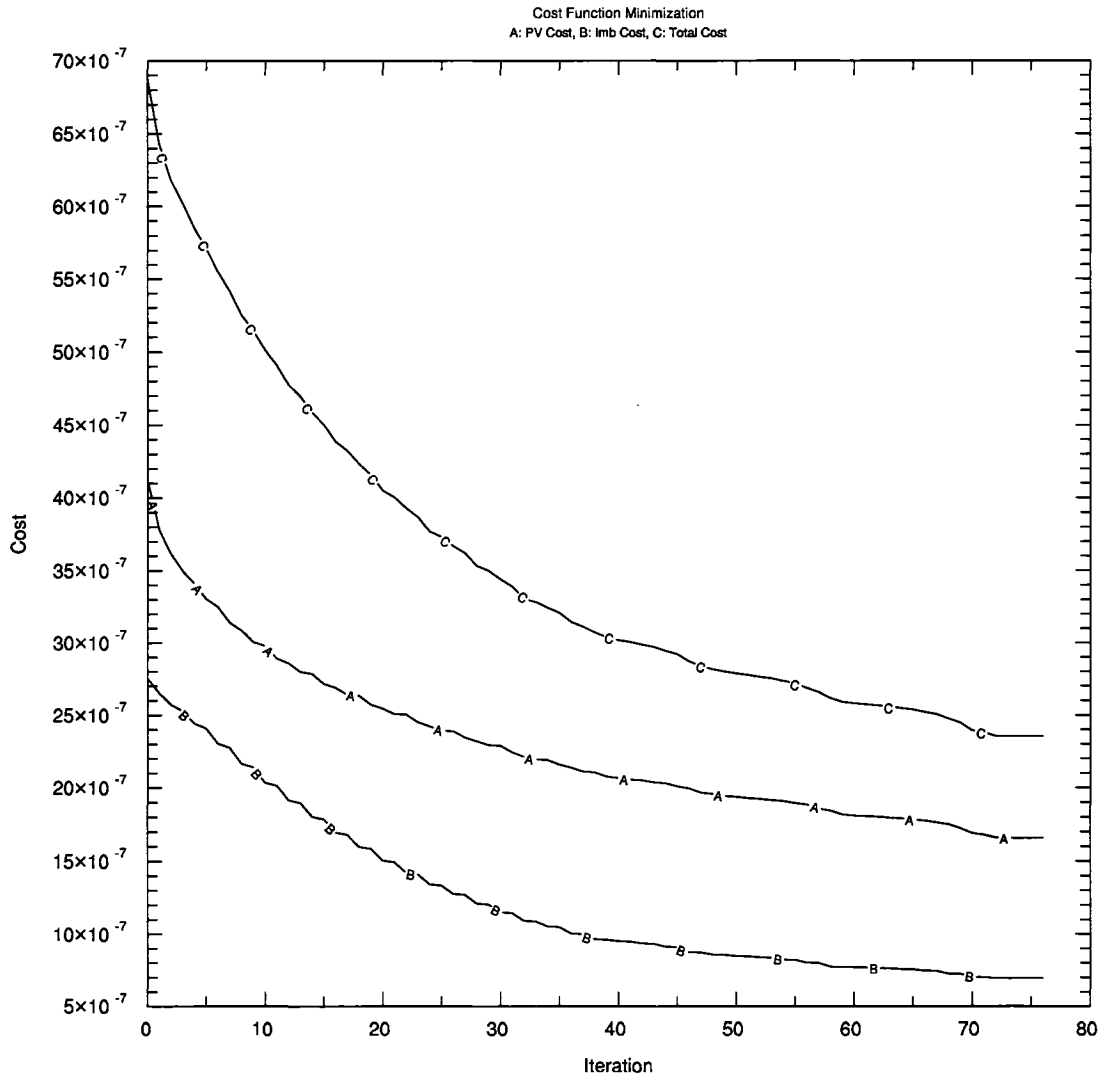


Figure 5.2: Reduction in the cost function during the inversion process for the second trial. All values are in  $\text{m}^4 \text{K}^2 \text{kg}^{-2} \text{s}^{-2}$ . The line labeled “A” (“B”) represents that part of the cost function due to the PV (imbalance) operator. The line labeled “C” represents the total cost function.





the full nondivergent flow, the bottom panels would look nearly identical. However, in this case, the streamfunction bears only a slight resemblance to the model-derived streamfunction, and the heights, which started out identical to Fig. 5.3a, now have an extreme minimum in the northwest corner of the domain. Despite the clear unrealistic nature of the inverted flow, the cost function did decrease, which seems puzzling at first. Another oddity is that the inverted  $\psi$  distribution has not changed visibly from its initial guess (not shown). The existence of the geopotential (and geopotential height) minimum in the northwest corner suggests that there may be an issue with the handling of the boundary conditions (although the absence of this problem in the previous trial makes this conclusion suspect). However, a physical explanation may exist as well.

As seen in Fig. 5.3d, which closely matches the initial guess of  $\psi$ , there is a very large streamfunction gradient in the northwest corner of the domain. Through nonlinear balance, that large gradient should be reflected by an intense height gradient as well. So, to reduce the imbalance, and thus the cost function, a large geopotential anomaly is introduced during the minimization. On the other hand, the fact that geopotential is greatly changed while streamfunction is not raises its own questions.

### 5.3.3 Trial 3

After noticing the interpolation of  $\psi$  was not properly constructed, a third trial was undertaken with a more plausible initial guess for  $\psi$ . As in the previous two trials, the cost function decreased during the inversion (Fig. 5.4). The manner in which the cost function decreased, however, was quite different from the previous trial. The more realistic guess for  $\psi$  results in a smaller cost function throughout the inversion, particularly for that part of the cost function due to the imbalance operator. However, this cost function is not as small as was seen during the first trial. In addition, after an initial decrease, the imbalance portion of the cost function increases as the total cost function decreases. Finally, note that the cost function minimum is not that much smaller than its original value in this case.

The results of the inversion in terms of streamfunction and geopotential height are shown in Fig. 5.5. As in the previous trial, the inverted streamfunction (Fig. 5.5d) is not visually distinct from

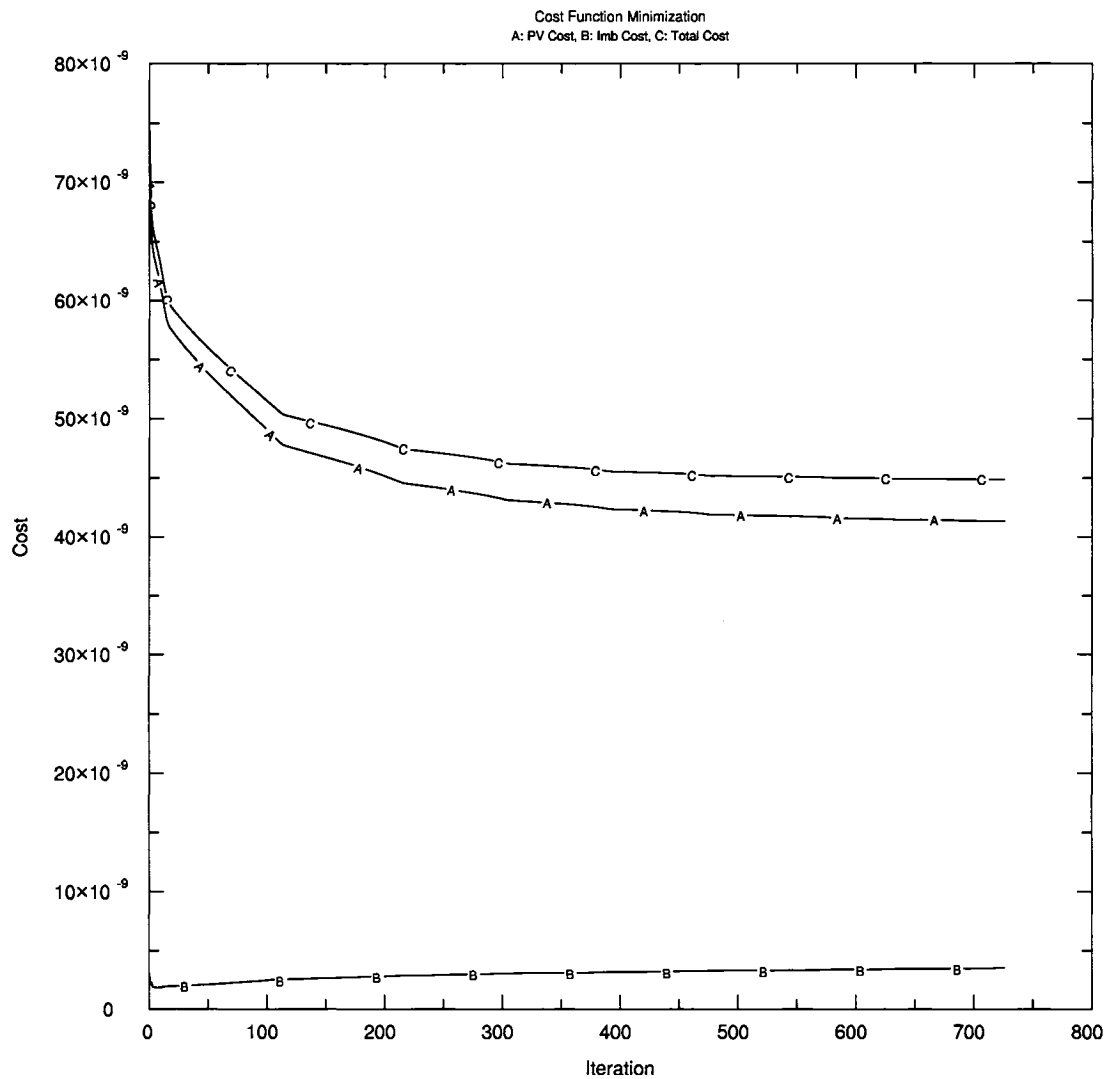


Figure 5.4: Same as Fig. 5.2, but for the third trial.

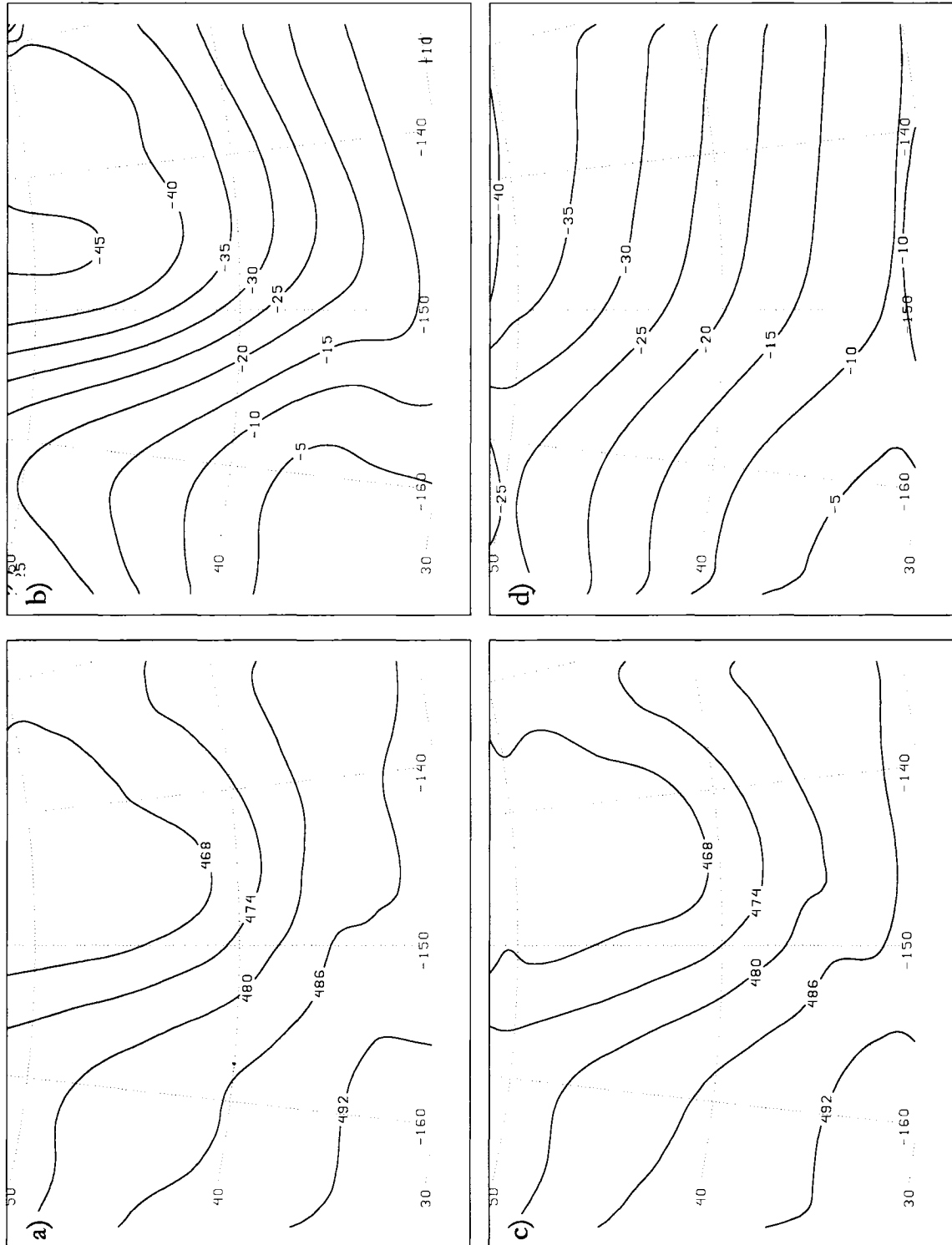


Figure 5.5: Same as Fig. 5.3 but for the third trial.

the first guess (not shown). The geopotential heights do change from their first guess (Fig. 5.5c), but generally not for the better, as noise has been added near the boundaries in some areas, especially in the north. The extreme minimum in the northwest corner is not present in the third trial, however.

These results suggest one of two possibilities. Are there errors in the formulation of the inversion technique, or is the minimization becoming trapped at local minima of the cost function that are distinct from the global minimum? Prospects for answering this question are discussed in the next chapter.

## Chapter 6

### Discussion

As we have seen, the success of the PV inversion routine appears to greatly depend on the degree to which the first guesses are realistic, but is that the only factor at play? Trial 3 showed a case where the partitioning of the cost function between the PV mismatch and the imbalance was heavily weighted toward the PV mismatch. Would a better result occur if these terms were more equally weighted? To check,  $\varepsilon$  was increased by a factor of thirty to bring the terms closer together, and the trial was rerun. The results are shown in Figure 6.1. Even with the increase in the value of  $\varepsilon$ , the results of the inversion are still unphysical. The geopotential heights (Fig. 6.1a) are now even further from their original values (Fig. 5.5a), but the streamfunction (Fig. 6.1b) continues to remain nearly unchanged. It appears that the heights are being brought closer into balance with the streamfunction (Figs. 6.1a–b look more similar) when instead the streamfunction is the field that should be undergoing the most change to enforce balance, given the first guesses and the PV distribution used. A look at the cost function (Fig. 6.1c) reveals that the imbalance decreased rapidly over the first few iterations, so that by roughly iteration 20, the PV mismatch portion of the cost function was again the dominant contributor to  $J$ . The PV mismatch changed little throughout the course of the minimization.

Although the minimization procedure may be finding local minima correctly, further more stringent tests need to be carried out to verify the accuracy of the adjoint operators and the cost function gradient computation. To further test the adjoint operators, and the tangent linear op-

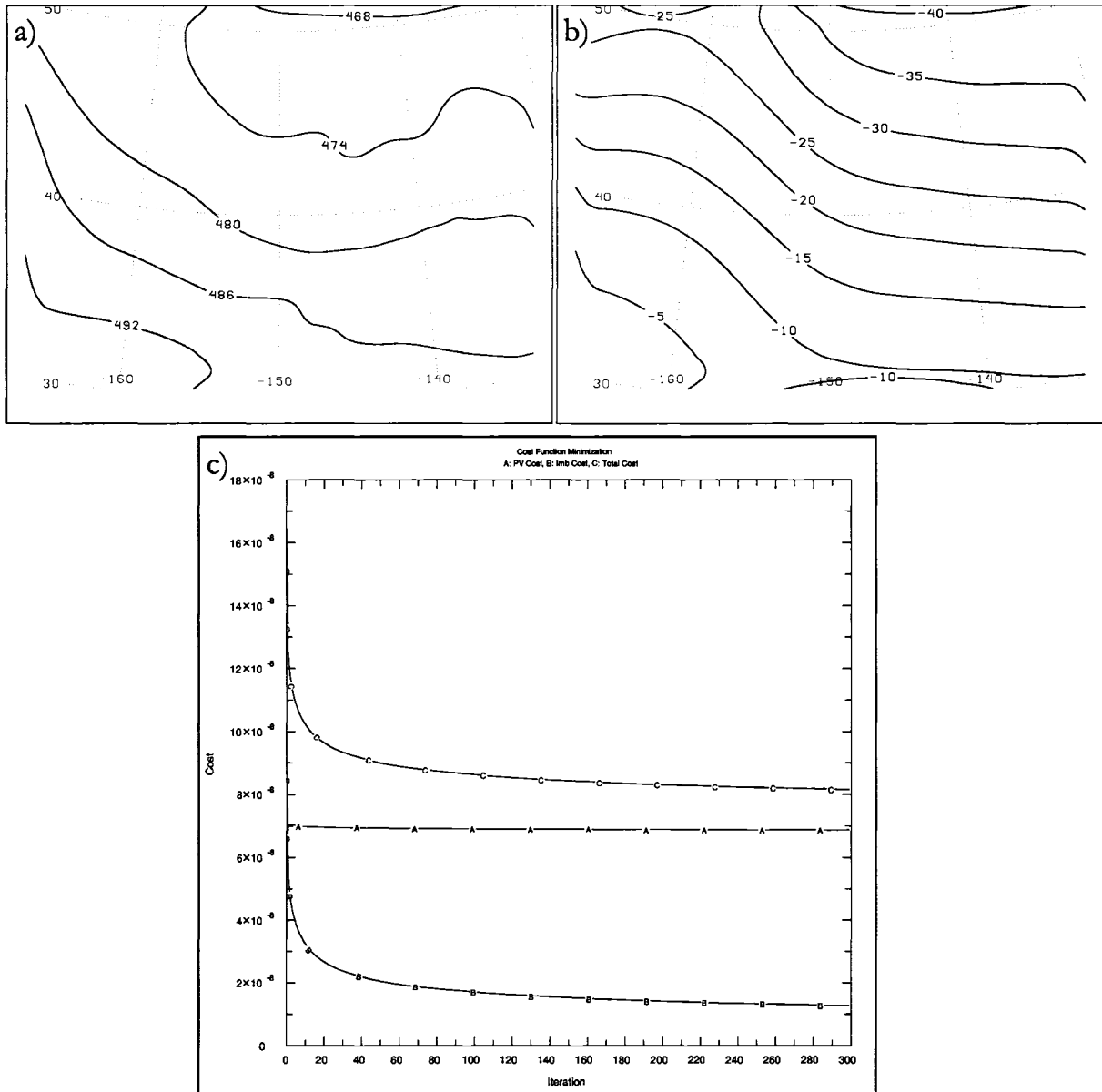


Figure 6.1: The results of Trial 3 with a thirtyfold increase in  $\varepsilon$ . (a) Same as Fig. 5.5c. (b) Same as Fig. 5.5d. (c) Same as Fig. 5.4.

erators from which they are derived, the following checks can be performed. First, small random perturbations drawn from a Gaussian distribution can be added to the state vector  $\mathbf{X}$  at all grid-points (rather than just a few as is currently done). Output of the tangent linear operator can be checked against output from the full operator at all levels. Differences between the two should be proportional to the magnitude of the perturbations as long as the perturbations are neither too small (where machine roundoff plays a role) nor too large (where the tangent linear operator is no longer appropriate). In other words, considering the imbalance operator as an example, the following equation should be verified (Kalnay 2003):

$$\delta\text{IMB} = \mathbf{B}\delta\mathbf{X} = \tilde{B}(\mathbf{X} + \delta\mathbf{X}) - \tilde{B}\mathbf{X} + O(\langle \delta\mathbf{X}, \delta\mathbf{X} \rangle^2) \quad (6.1)$$

These more complex perturbations can also be used as a more rigorous check of the adjoint, using the same procedure outlined earlier.

The gradient of the cost function can also be checked by comparing the output of Eq. (2.9) against a numerically estimated  $\nabla J$ . This numerical estimate can be carried out by incrementing the value of  $\mathbf{X}$  gridpoint by gridpoint and then computing the resulting change in the cost function. Collecting the changes in the cost function, sensitivity maps can be constructed showing how the cost function would change given a change in  $\psi$  or  $\Phi$  at particular gridpoints. Maps based on the output of Eq. (2.9) should look similar.

Even if the gradient computation were found to be incorrect, the question of whether the PV minimization process is prone to becoming trapped in a local minimum may be addressed by using the numerically estimated  $\nabla J$  during the minimization. This procedure would be very computationally expensive (which is why the adjoints are necessary in the first place), but the relatively small domain of Test Case 2 would make it feasible.

It could also be questioned whether the right tests are being performed to begin with. After all, when applied to morphological data assimilation, it is the PV distribution that is changed, not the initial guess of streamfunction. Although one would hope that the inversion would not be so fragile as to only work when the initial guesses are close to being in balance, perhaps that is the



case. Thus, this sort of testing also needs to be done.

In addition to the specific details regarding the inversion discussed above, the fact that  $\mu$  must be specified as a boundary condition raises questions of its own. Specifying  $\mu$  is equivalent to specifying the surface pressure (disregarding contributions from the vapor pressure and any nonhydrostatic effects), which is somewhat unsatisfactory. If PV is warped within the interior of the domain, shouldn't the surface pressures also change as a result of the new PV distribution? In the DE method, the surface pressure is also specified (by setting the bottom level to, say, 1000 hPa), but the geopotential heights at that level are allowed to vary. In the method presented here however, the geopotential heights are fixed to the elevation of the surface of the Earth. Generalizing the PV inversion technique so that  $\mu$  need not be specified is left for future research.

The warping aspect of morphological data assimilation could also use some improvement. For instance, a more physically pleasing weighting function would be based on geometric distance on the globe instead of the difference in latitude/longitude coordinates. The warp should also be defined on a Lambert conformal grid to match the domain being used for runs of the WRF model.

Although the cutoff in the current weighting function has not resulted in discontinuities in the warped PV field, it would be better, at least theoretically, to use a function that goes to zero smoothly. For instance,  $w$  could be defined as

$$w = \begin{cases} \left(\frac{lp}{a+d}\right)^b & d < 4.5^\circ \\ (5.5^\circ - d) \left(\frac{lp}{a+d}\right)^b & 4.5^\circ \leq d \leq 5.5^\circ \\ 0 & d > 5.5^\circ \end{cases} \quad (6.2)$$

Finally, some words should be said regarding how the results of the PV inversion can be introduced into a WRF model run. If the procedure is being undertaken as a retrospective case study, a six-hour forecast of the WRF can be run, and the distribution of the potential vorticity can be computed. The warping procedure can then be carried out, followed by PV inversion to obtain background fields of geopotential heights, winds, and temperatures. These background fields can be combined with observations using WRF 3DVar system to produce an analysis, and,

from that analysis, a run of the WRF can be executed that has been based on morphological data assimilation. It is better to warp and invert PV before employing 3DVar because the main use of the warping is to remove gross displacement errors from the forecast. Once this is done, then 3DVar is well suited to removing the remaining more randomly distributed, flow independent errors.

The situation would be different in a pseudo-operational environment. In that case, the forecast error is unknown at the time the PV is being warped, so there is little sense of where the initial conditions may need to be changed. Thus, tests of initial condition sensitivity such as those performed by Beare et al. (2003) may be necessary to determine which areas should be focused on in particular during the warping process. Additionally, knowledge of the forecast errors six and twelve hours into the forecast could be useful in a pseudo-operational setting. If the forecast improvement is targeted at the 3–5 day forecast period, there is still time to use the 6–12-hr forecast errors to alter the original forecast, and those alterations may produce better results than unaltered forecasts initialized six or twelve hours later.

## Chapter 7

# Conclusions

Morphological data assimilation is the process of using visual information to produce initial conditions for a numerical weather prediction run. It is an approach complementary to that of traditional data assimilation, which is purely statistical in nature. In essence, traditional methods see the trees, but not the forest, at least not explicitly. On the other hand, morphological data assimilation sees the forest, and uses that information to create a plausible distribution of trees. PV inversion is then the process by which the trees are distributed throughout the forest. As Swarbrick (2001) showed, morphological data assimilation alone cannot produce superior forecasts. Similarly, as McMurdie and Mass (2004) showed, standard assimilation techniques do not lead to superior forecasts, either. The hope is that through combining morphological and standard assimilation techniques, an estimate for the state of the atmosphere at an initialization time can be found that is consistently closer to reality than can be determined using either technique alone.

Additionally, morphological data assimilation can be used during case studies to ask more complicated “what if” questions. In addition to asking, “What if the PV anomaly were doubled?”, as Roebber et al. (2002) do, one can ask “What if the PV anomaly had a different shape?”

Of course, all of these applications are predicated on a robust PV inversion routine. Once such a routine is in place, the promise of morphological data assimilation will finally be able to be tested.

## Chapter 8

# Appendix

Contained within the appendix is the Fortran code used to carry out potential vorticity inversion. The code originated from an earlier program intended to convert output from the WRF model into a form suitable for viewing using the GEneral Meteorological PAcKage (GEMPAK). In a sense, this is still what is done. After all, we need to be able to display the results of PV inversion just as much as we need to display wind vectors, temperatures, or any other variable derivable from WRF model output. Each section contains one file (usually a Fortran 90 module) along with a brief description of its purpose.

### 8.1 File parameters

This module contains a few default settings used to find the namelist file that controls how the program is run.

```
module wrf2gem_parameters
  implicit none
  save
  integer,          parameter :: NLUnit = 19
  character(len=*), parameter :: NLFName = "convert.nl"
end module wrf2gem_parameters
```

## 8.2 Date utilities

This module contains functions used to help calculate dates.

```

module dateutil
  implicit none
  save
  private
  public :: hr_diff
  public :: operator(<), operator(>), operator(==)
  type, public :: datestamp
    integer :: yr, mth, day, hr
  end type datestamp
  interface operator(<)
    module procedure datestamp_lt
  end interface
  interface operator(>)
    module procedure datestamp_gt
  end interface
  interface operator(==)
    module procedure datestamp_eq
  end interface
contains ! =====
  ! hr_diff returns the number of hours elapsed from d1 to d2.
  ! If the second time is before the first, -1 is returned.
  integer function hr_diff(d2, d1)
    type(datestamp), intent(in) :: d2, d1
    integer, parameter :: MaxHours = 999
    type(datestamp) :: d
    integer :: i
    ! Make sure the second time doesn't come before the first time
    if (d2 < d1) then
      hr_diff = -1
    else if (d2 == d1) then ! Times match
      hr_diff = 0
    else
      d = d1
      hr_diff = 0
      do i = 1, MaxHours ! Progress forward in time hour by hour
        hr_diff = hr_diff + 1
        d%hr = d%hr + 1
        if (d%hr > 23) then ! New day
          d%hr = 0
          d%day = d%day + 1
          if (d%day > days_in_month(d%mth, d%yr)) then ! New month
            d%day = 1
            d%mth = d%mth + 1
            if (d%mth > 12) then ! New year
              d%mth = 1
            end if
          end if
        end if
      end do
    end if
  end function hr_diff
end module dateutil

```

```

        d%yr = d%yr + 1
    end if
end if
end if
! Check to see if it is the same time
if (d == d2) exit
end do
end if
end function hr_diff
! =====
! days_in_month returns the number of days month has given the year.
integer function days_in_month(month, year)
    integer, intent(in) :: month, year
    select case (month)
    case (4,6,9,11)
        days_in_month = 30
    case (1,3,5,7,8,10,12)
        days_in_month = 31
    case (2)
        if (mod(year, 4) == 0 .and. &
            (mod(year, 100) /= 0 .or. mod(year, 400) == 0)) then
            days_in_month = 29
        else
            days_in_month = 28
        end if
    case default
        stop "Invalid input to days_in_month!"
    end select
end function days_in_month
! =====
! datestamp_lt returns .true. if d1 occurs before d2, .false. otherwise.
logical function datestamp_lt(d1, d2)
    type(datestamp), intent(in) :: d1, d2
    if (d1%yr > d2%yr .or. d1%mth > d2%mth .and. d1%yr == d2%yr &
        .or. d1%day > d2%day .and. d1%mth == d2%mth .and. d1%yr == d2%yr &
        .or. d1%hr >= d2%hr .and. d1%day == d2%day .and. d1%mth == d2%mth &
        .and. d1%yr == d2%yr) then
        datestamp_lt = .false.
    else
        datestamp_lt = .true.
    end if
end function datestamp_lt
! =====
! datestamp_gt returns .true. if d1 occurs after d2, .false. otherwise.
logical function datestamp_gt(d1, d2)
    type(datestamp), intent(in) :: d1, d2
    if (d1%yr > d2%yr .or. d1%mth > d2%mth .and. d1%yr == d2%yr &
        .or. d1%day > d2%day .and. d1%mth == d2%mth .and. d1%yr == d2%yr &
        .or. d1%hr > d2%hr .and. d1%day == d2%day .and. d1%mth == d2%mth &
        .and. d1%yr == d2%yr) then

```

```

        datestamp_gt = .true.
    else
        datestamp_gt = .false.
    end if
end function datestamp_gt
! =====
! datestamp_eq returns .true. if d1 equals d2, .false. otherwise.
logical function datestamp_eq(d1, d2)
    type(datestamp), intent(in) :: d1, d2
    if (d1%yr == d2%yr .and. d1%mth == d2%mth .and. d1%day == d2%day &
        .and. d1%hr == d2%hr) then
        datestamp_eq = .true.
    else
        datestamp_eq = .false.
    end if
end function datestamp_eq
end module dateutil

```

### 8.3 Meteorological diagnostics

This module contains functions that compute all nontrivial, but typical, meteorological quantities.

For instance, a function that converts mixing ratio to specific humidity is included.

```

module diagnostics
    implicit none
    save
    private
    public :: vint, hydro_interp, mix_ratio_to_spec_hum, temp_from_theta_p, &
        surface_pres, density, sea_pres, lifted_index, mean_rh, &
        unstag_hght, theta_phi, pvor
    public :: RGas, Grav, Kappa, Eps
    ! Some meteorological constants (set to the values used by WRF)
    real, parameter :: RGas = 287., Grav = 9.81, KappaR = 3.5, &
        SpecHeatPresDry = KappaR * RGas, &
        Kappa = RGas / SpecHeatPresDry, Gamma = .0065, &
        LatHeatVap = 2.5e6, RVap = 461.6, Eps = RGas / RVap, &
        EpsR = RVap / RGas, Missing = -9999., EpsRM1 = EpsR - 1
    interface mix_ratio_to_spec_hum
        module procedure mrtsh3d
        module procedure mrtsh4d
    end interface
    interface temp_from_theta_p
        module procedure tftp0d
        module procedure tftp1d
        module procedure tftp2d
        module procedure tftp3d
    end interface

```

```

      module procedure tftp4d
    end interface
  interface density
    module procedure dens3d
    module procedure dens4d
  end interface
contains ! Public Functions =====
! vint takes a vertical profile (prof) and interpolates it to regular levels
! starting at first with an interval of delta using a profile of values for
! the interpolating vertical coordinate variable (vCoordVals) defined at the
! same points as prof. The interpolated output is returned as interpProf.
! interpType = 1 --> linear interpolation, interpType = 2 --> interpolate
! with respect to -ln(vertical coordinate value).
subroutine vint(prof, vCoordVals, first, delta, interpType, interpProf)
  real, dimension(:), intent(in) :: prof, vCoordVals
  integer,          intent(in)  :: first, delta, interpType
  real, dimension(:), intent(out) :: interpProf
  real, dimension(2) :: vCoordBounds, profBounds
  real :: level
  integer :: nz, np, k
  nz = size(prof)
  np = size(interpProf)
  ! Validate input
  if (nz /= size(vCoordVals)) then
    print *, "Error: Array extent mismatch in vint!"
    stop
  end if
  ! Initialize output
  interpProf = Missing
  ! Interpolate
  do k = 1, np
    level = first + (k-1) * delta
    ! Skip this level if we don't have data for it
    if (level < minval(vCoordVals) .or. level > maxval(vCoordVals)) cycle
    call find_bounds(vCoordVals, prof, level, vCoordBounds, profBounds)
    if (interpType == 2) then ! Handle logarithmic interpolation case
      vCoordBounds = -log(vCoordBounds)
      level = -log(level)
    end if
    interpProf(k) = lin_int(vCoordBounds(1), vCoordBounds(2), &
      profBounds(1), profBounds(2), level)
  end do
end subroutine vint
! =====
! hydro_interp hydrostatically interpolates geopotential heights on pressure
! surfaces underground. Provided as input are the height (zs; m), pressure
! (ps; hPa), and temperature (ts; K) in the lowest WRF model layer, as well
! as the lowest (in altitude) pressure surface desired (pBot) and the
! pressure level spacing (dp). The routine takes the currently computed
! geopotential heights on pressure surfaces (z), and interpolates to those

```



```

! grid points determined to be underground.
subroutine hydro_interp(zs, ps, ts, pBot, dp, z)
  real, dimension(:,:,:), intent(in) :: zs, ps, ts
  integer, intent(in) :: pBot, dp
  real, dimension(:,:,:,:), intent(inout) :: z
  real, parameter :: LapseRate = .0065, Expo = RGas * LapseRate / Grav
  integer :: nz, k, p
  nz = size(z,3)
  ! Validate
  if (any(shape(zs) /= shape(ps)) .or. any(shape(zs) /= shape(ts)) .or. &
      size(z,1) /= size(zs,1) .or. size(z,2) /= size(zs,2) .or. &
      size(z,4) /= size(zs,3)) stop "Array mismatch in hydro_interp!"
  do k = 1, nz
    if (all(z /= Missing)) exit ! Are we above the highest terrain?
    p = pBot + (k-1) * dp
    where (z(:, :, k, :) == Missing)
      z(:, :, k, :) = zs - ts / LapseRate * ((p/ps)**Expo - 1)
    end where
  end do
end subroutine hydro_interp
! =====
! Both mrtsh3d and mrtsh4d convert mixing ratio (kg/kg) to specific
! humidity (kg/kg).
function mrtsh3d(w) result (q)
  real, dimension(:,:), intent(in) :: w
  real, dimension(size(w,1),size(w,2),size(w,3)) :: q
  q = w / (1 + w)
end function mrtsh3d
! .....
function mrtsh4d(w) result (q)
  real, dimension(:,:,:,:), intent(in) :: w
  real, dimension(size(w,1),size(w,2),size(w,3),size(w,4)) :: q
  q = w / (1 + w)
end function mrtsh4d
! =====
! tftp0d, tftp1d, tftp2d, tftp3d, and tftp4d calculate the temperature (K)
! given the potential temperature (th; K) and pressure (p; Pa or hPa
! depending on si flag).
function tftp0d(th, p, si) result (t)
  real, intent(in) :: th, p
  logical, intent(in), optional :: si
  real :: t
  real :: p0r
  ! Set appropriate p0
  p0r = 1e-5 ! Default is SI units (p0 = 100000)
  if (present(si)) then
    if (.not. si) p0r = .001 ! (p0 = 1000)
  end if
  ! Calculate temperature (exp and log faster than ** on my machine)
  t = th * exp(Kappa * log(p * p0r))

```

```

end function tftp0d
! .....
function tftp1d(th, p, si) result (t)
  real, dimension(:), intent(in)          :: th, p
  logical,          intent(in), optional :: si
  real, dimension(size(p,1))              :: t
  real :: p0r
  ! Make sure arrays match up
  if (any(shape(th) /= shape(p))) stop "Array extent mismatch in tftp2d!"
  ! Set appropriate p0
  p0r = 1e-5 ! Default is SI units (p0 = 100000)
  if (present(si)) then
    if (.not. si) p0r = .001 ! (p0 = 1000)
  end if
  ! Calculate temperature
  t = th * (p * p0r) ** Kappa
end function tftp1d
! .....
function tftp2d(th, p, si) result (t)
  real, dimension(:, :), intent(in)          :: th, p
  logical,          intent(in), optional :: si
  real, dimension(size(p,1),size(p,2))      :: t
  real :: p0r
  ! Make sure arrays match up
  if (any(shape(th) /= shape(p))) stop "Array extent mismatch in tftp2d!"
  ! Set appropriate p0
  p0r = 1e-5 ! Default is SI units (p0 = 100000)
  if (present(si)) then
    if (.not. si) p0r = .001 ! (p0 = 1000)
  end if
  ! Calculate temperature
  t = th * (p * p0r) ** Kappa
end function tftp2d
! .....
function tftp3d(th, p, si) result (t)
  real, dimension(:, :, :), intent(in)          :: th, p
  logical,          intent(in), optional :: si
  real, dimension(size(p,1),size(p,2),size(p,3)) :: t
  real :: p0r
  ! Make sure arrays match up
  if (any(shape(th) /= shape(p))) stop "Array extent mismatch in tftp3d!"
  ! Set appropriate p0
  p0r = 1e-5 ! Default is SI units (p0 = 100000)
  if (present(si)) then
    if (.not. si) p0r = .001 ! (p0 = 1000)
  end if
  ! Calculate temperature
  t = th * (p * p0r) ** Kappa
end function tftp3d
! .....

```

```

function tftp4d(th, p, si) result (t)
  real, dimension(:,:,:,:), intent(in)          :: th, p
  logical,          intent(in), optional         :: si
  real, dimension(size(p,1),size(p,2),size(p,3),size(p,4)) :: t
  real :: p0r
  ! Make sure arrays match up
  if (any(shape(th) /= shape(p))) stop "Array extent mismatch in tftp4d!"
  ! Set appropriate p0
  p0r = 1e-5 ! Default is SI units (p0 = 100000)
  if (present(si)) then
    if (.not. si) p0r = .001 ! (p0 = 1000)
  end if
  ! Calculate temperature
  t = th * (p * p0r) ** Kappa
end function tftp4d
! =====
! surface_pres calculates the surface pressure (Pa) given the following data:
! ht-   Geopotential height (m) at lowest two w (full) levels   (x,y,z,t)
! p1-   Pressure (Pa) at lowest mass (half) level                (x,y,t)
! q1-   Specific humidity (kg/kg) at lowest mass level           (x,y,t)
! t1-   Temperature (K) at lowest mass level                     (x,y,t)
! sfcq- Specific humidity (kg/kg) at surface/2-m above ground   (x,y,t)
! It is assumed that the array extents match appropriately.
function surface_pres(ht, p1, q1, t1, sfcq)
  real, dimension(:,:,:,:), intent(in)          :: ht
  real, dimension(:,:,:,:), intent(in)          :: p1, q1, t1, sfcq
  real, dimension(size(p1,1),size(p1,2),size(p1,3)) :: surface_pres
  real, dimension(size(t1,1),size(t1,2),size(t1,3)) :: tvirt
  tvirt = (1 + .608 * .5 * (sfcq + q1)) * t1
  surface_pres = p1 * exp((Grav / (RGas * tvirt)) &
    * .5 * (ht(:,:,2,:) - ht(:,:,1,:)))
end function surface_pres
! =====
! dens3d and dens4d calculate the density (kg/m**3) using the ideal gas law
! given:
! p- Pressure (Pa)
! t- Temperature (K)
! q- Water vapor mixing ratio (kg/kg)
! It is assumed that the array extents match appropriately.
function dens3d(p, t, q)
  real, dimension(:,:,:,:), intent(in)          :: p, t, q
  real, dimension(size(p,1),size(p,2),size(p,3)) :: dens3d
  dens3d = p / (t * (RGas + q * RVap))
  dens3d = dens3d * (1 + q)
end function dens3d
! .....
function dens4d(p, t, q)
  real, dimension(:,:,:,:), intent(in)          :: p, t, q
  real, dimension(size(p,1),size(p,2),size(p,3),size(p,4)) :: dens4d
  dens4d = p / (t * (RGas + q * RVap))

```

```

    dens4d = dens4d * (1 + q)
end function dens4d
! =====
! sea_pres is taken from the WRF NCL tools. The inputs are:
! zs-   Geopotential height (m) on w (full) levels      (x,y,z,t)
! theta- Potential temperature (K) on mass (half) levels (x,y,z,t)
! p-    Pressure (Pa) on mass levels                    (x,y,z,t)
! q-    Specific humidity (kg/kg) on mass levels        (x,y,z,t)
! It is assumed that the array extents match appropriately.
function sea_pres(zs, theta, p, q)
  ! Estimate sea level pressure.
  real, dimension(:,:,:), intent(in)      :: zs, theta, p, q
  real, dimension(size(p,1),size(p,2),size(p,4)) :: sea_pres
  real, dimension(size(p,1),size(p,2),size(p,3),size(p,4)) :: t, z
  real, dimension(size(p,1),size(p,2)) :: tSurf, tSeaLevel
  integer, dimension(size(p,1),size(p,2)) :: level
  ! Specific constants for assumptions made in this routine:
  real, parameter :: Tc = 273.16+17.5, PConst = 10000
  ! Local variables:
  real :: plo, phi, tlo, thi, zlo, zhi, pAtPConst, tAtPConst, zAtPConst
  integer :: nx, ny, nz, nt
  integer :: i, j, k, klo, khi, l
  logical :: l1, l2, l3, found
  nx = size(p,1)
  ny = size(p,2)
  nz = size(p,3)
  nt = size(p,4)
  ! Convert potential temperature to temperature
  t = temp_from_theta_p(theta, p)
  ! Unstagger z
  z(:, :, 1:nz, :) = .5 * (zs(:, :, 1:nz, :) + zs(:, :, 2:nz+1, :))
  ! Grand loop over all times
  do l = 1, nt
    ! Find least zeta level that is PConst Pa above the surface. We later
    ! use this level to extrapolate a surface pressure and temperature,
    ! which is supposed to reduce the effect of the diurnal heating cycle in
    ! the pressure field.
    do j = 1, ny
      do i = 1, nx
        level(i,j) = -1
        k = 1
        found = .false.
        do while( (.not. found) .and. (k <= nz))
          if ( p(i,j,k,l) < p(i,j,1,l)-PConst ) then
            level(i,j) = k
            found = .true.
          end if
          k = k+1
        end do
        if ( level(i,j) == -1 ) then

```

```

        print '(A,I4,A)', 'Troubles finding level ', &
            nint(PConst)/100, ' above ground.'
        print '(A,I4,A,I4,A)', &
            'Problems first occur at (',i,',',j,')'
        print '(A,F6.1,A)', &
            'Surface pressure = ',p(i,j,1,1)/100, ' hPa.'
        stop 'Error_in_finding_100_hPa_up'
    end if
end do
end do
! Get temperature PConst Pa above surface. Use this to extrapolate
! the temperature at the surface and down to sea level.
do j = 1 , ny
    do i = 1 , nx
        klo = max ( level(i,j) - 1 , 1 )
        khi = min ( klo + 1 , nz - 1 )
        if ( klo == khi ) then
            print '(A)', 'Trapping levels are weird.'
            print '(A,I3,A,I3,A)', 'klo = ',klo,', khi = ',khi, &
                ': and they should not be equal.'
            stop 'Error_trapping_levels'
        end if
        plo = p(i,j,klo,1)
        phi = p(i,j,khi,1)
        tlo = t(i,j,klo,1) * (1. + 0.608 * q(i,j,klo,1) )
        thi = t(i,j,khi,1) * (1. + 0.608 * q(i,j,khi,1) )
        zlo = z(i,j,klo,1)
        zhi = z(i,j,khi,1)
        pAtPConst = p(i,j,1,1) - PConst
        TAtPConst = thi - (thi-tlo) * log(pAtPConst/phi) * log(plo/phi)
        zAtPConst = zhi - (zhi-zlo) * log(pAtPConst/phi) * log(plo/phi)
        tSurf(i,j) = tAtPConst * (p(i,j,1,1)/pAtPConst)**(Gamma*RGas/Grav)
        tSeaLevel(i,j) = tAtPConst + Gamma * zAtPConst
    end do
end do
! If we follow a traditional computation, there is a correction to the
! sea level temperature if both the surface and sea level temperatures
! are *too* hot.
do j = 1 , ny
    do i = 1 , nx
        l1 = tSeaLevel(i,j) < Tc
        l2 = tSurf(i,j) <= Tc
        l3 = .not. l1
        if ( l2 .and. l3 ) then
            tSeaLevel(i,j) = Tc
        else
            tSeaLevel(i,j) = Tc - 0.005 * (tSurf(i,j) - Tc)**2
        end if
    end do
end do
end do

```

```

! The grand finale: ta da!
sea_pres(:, :, 1) = p(:, :, 1, 1) * &
    exp((2.*Grav*z(:, :, 1, 1)) / (RGas*(tSeaLevel + tSurf)))
end do
end function sea_pres
! =====
! lifted_index calculates the lifted index (K) based on the supplied profiles
! of potential temperature (theta; K), pressure (p; hPa), and mixing ratio
! (w; kg/kg). Array indices are ordered from the ground up. The lowest 4
! model levels are mixed to produce the hypothetical parcel, which is then
! lifted adiabatically to saturation, and pseudoadiabatically thereafter.
function lifted_index(theta, p, w) result (li)
    real, dimension(:), intent(in) :: theta, p, w
    real :: li
    real, dimension(2) :: pBounds, thBounds
    real :: dens, thMix, wMix, sm, tk, pLCL, thes, tParcel, tEnv
    integer :: nz, k
    nz = size(theta)
    ! Check that array extents match
    if (nz /= size(p) .or. nz /= size(w)) &
        stop "Array mismatch in lifted_index."
    ! Set lifted index to 0 if we don't have enough data
    if (nz < 4 .or. p(1) < 500.) then
        li = 0
    else
        ! Determine temperature of pseudoadiabatically lifted parcel at 500 mb
        ! Step 1: Mix the lowest 4 model levels
        thMix = 0; wMix = 0; sm = 0
        do k = 1, 4
            dens = density_thp(theta(k), p(k)) ! Effect of water vapor ignored
            thMix = thMix + dens * theta(k)
            wMix = wMix + dens * w(k)
            sm = sm + dens
        end do
        thMix = thMix / sm
        wMix = wMix / sm
        ! Step 2: Lift parcel to LCL
        tk = temp_from_theta_p(thMix, p(1), si=.false.)
        ! This equation is derived from Curry & Webster (1999, p. 174)
        pLCL = 1000 * (temp_lcl(tk, rh(wMix, p(1), tk)) / thMix)**KappaR
        ! Step 3: Lift parcel to 500 mb
        thes = theta_es(thMix, pLCL)
        tParcel = temp_from_theta_p(theta_es_to_theta_fast(thes, 500.), 500., &
            si=.false.)
        ! Get environmental temperature at 500 mb
        call find_bounds(p, theta, 500., pBounds, thBounds)
        tEnv = temp_from_theta_p(lin_int(pBounds(1), pBounds(2), thBounds(1), &
            thBounds(2), 500.), 500., si=.false.)
        li = tEnv - tParcel
    end if
end if

```

```

end function lifted_index
! =====
! mean_rh computes the mean relative humidity (%) between pBot hPa and pTop
! hPa based on the supplied profiles of potential temperature (theta; K),
! pressure (p; hPa), and mixing ratio (w; kg/kg). Also provided is the
! surface pressure (pSfc; hPa). Array indices are ordered from the ground
! up. The mean is weighted by the thickness of each layer.
function mean_rh(theta, p, w, pSfc, pBot, pTop) result (mrh)
  real, dimension(:), intent(in) :: theta, p, w
  real,                intent(in) :: pSfc, pBot, pTop
  real                :: mrh
  real, dimension(0:size(theta)-1) :: pMid
  real                :: pThick, relHum, totPresThick, totRH
  integer             :: nz, k
  nz = size(theta)
  ! Check that array extents match
  if (nz /= size(p) .or. nz /= size(w)) stop "Array mismatch in mean_rh."
  ! Calculate pressure at midpoints
  pMid(0) = pSfc
  pMid(1:nz-1) = .5 * (p(1:nz-1) + p(2:nz))
  ! Calculate mean RH
  totPresThick = 0; totRH = 0
  do k = 1, nz-1
    if (pMid(k) > pBot) cycle
    if (pMid(k-1) < pTop) exit
    ! Calculate layer thickness
    if (pMid(k-1) > pBot) then
      pThick = pBot - pMid(k)
    else if (pMid(k) < pTop) then
      pThick = pMid(k-1) - pTop
    else
      pThick = pMid(k-1) - pMid(k)
    end if
    relHum = rh(w(k), p(k), temp_from_theta_p(theta(k), p(k), si=.false.))
    totRH = totRH + pThick * relHum
    totPresThick = totPresThick + pThick
  end do
  if (totPresThick > .01) then
    mrh = 100 * totRH / totPresThick
  else
    mrh = 0
  end if
end function mean_rh
! =====
! unstag_hght unstaggers the geopotential height from w levels (zs) given
! by znw to u levels (z) given by znu. mu and pt provide additional
! information needed to do an interpolation based on ln p.
function unstag_hght(zs, znw, znu, mu, pt) result (z)
  real, dimension(:), intent(in) :: zs, znw, znu
  real,                intent(in) :: mu, pt

```

```

real, dimension(size(znu))      :: z
real, dimension(size(znw)) :: pw
real, dimension(size(znu)) :: pu
integer :: k
! Validate input
if (size(zs) /= size(znw) .or. size(znw) /= size(znu) + 1) &
  print *, "Warning: Array extent mismatch in unstag_hght!"
! Initialize pressures
do k = 1, size(znw)
  if (k < size(znw)) pu(k) = pd(znu(k))
  pw(k) = pd(znw(k))
end do
! Interpolate to half levels
do k = 1, size(znu)
  z(k) = lin_int(pw(k), pw(k+1), zs(k), zs(k+1), pu(k))
end do
contains
real function pd(eta)
  real, intent(in) :: eta
  pd = pt + mu * eta
end function pd
end function unstag_hght
! =====
subroutine theta_phi(phi, mu, etaz, etam, pt, th)
  real, dimension(:, :, 0:., :), intent(in) :: phi
  real, dimension(:, :, :), intent(in) :: mu
  real, dimension(0:), intent(in) :: etaz
  real, dimension(:), intent(in) :: etam
  real, intent(in) :: pt
  real, dimension(size(phi,1), size(phi,2), size(etam), size(phi,4)) :: th
  real, parameter :: p0 = 100000.
  integer :: nx, ny, nz, nt, i, j, k, t
  nx = size(phi,1)
  ny = size(phi,2)
  nz = size(etam)
  nt = size(phi,4)
  forall (i = 1:nx, j = 1:ny, k = 1:nz, t = 1:nt)
    th(i,j,k,t) = (phi(i,j,k,t) - phi(i,j,k-1,t)) &
      * (p0 / (pt + etam(k) * mu(i,j,t)))**Kappa &
      / (RGas * log((pt + etaz(k-1) * mu(i,j,t)) &
        / (pt + etaz(k) * mu(i,j,t))))
  end forall
end subroutine theta_phi
! =====
subroutine pvor(u, v, hght, theta, rho, f, mf, etau, etaw, dx, q)
  real, dimension(:, :, :, :), intent(in) :: u, v, hght, theta, rho
  real, dimension(:, :), intent(in) :: f, mf
  real, dimension(:), intent(in) :: etau, etaw
  real, intent(in) :: dx
  real, dimension(size(v,1), size(u,2), size(u,3), size(u,4)), intent(out) :: q

```



```

real, dimension(:,:,:), allocatable :: dthetadeta, dzdx, dzdy, dvdeteta, &
                                         dudeta, vort, detadz

real, dimension(3) :: c
real :: dy, detal, detah, mfavg
real :: dvdeavg, dudeavg, dzdxavg, dzdyavg, detadzavg
integer :: nx, ny, nz, nt, i, j, k, t
nx = size(q,1)
ny = size(q,2)
nz = size(q,3)
nt = size(q,4)
dy = dx
print *, "Calculating intermediate quantities globally..."
! Calculate the intermediate quantities globally
allocate(dthetadeta(2:nx-1,2:ny-1,2:nz-1,nt), detadz(nx,ny,2:nz-1,nt), &
         dvdeteta(nx,2:ny,2:nz-1,nt), dudeta(2:nx,ny,2:nz-1,nt))
do k = 2, nz-1
    detal = etau(k) - etau(k-1)
    detah = etau(k+1) - etau(k)
    c(1) = -detah / (detal * (detal+detah))
    c(2) = (detah**2 - detal**2) / (detal * detah * (detal+detah))
    c(3) = detal / (detah * (detal+detah))
    dthetadeta(2:nx-1,2:ny-1,k,:) = c(1)*theta(2:nx-1,2:ny-1,k-1,:) &
        + c(2)*theta(2:nx-1,2:ny-1,k,:) + c(3)*theta(2:nx-1,2:ny-1,k+1,:)
    dvdeteta(:,2:ny,k,:) = c(1)*v(:,2:ny,k-1,:) + c(2)*v(:,2:ny,k,:) &
        + c(3)*v(:,2:ny,k+1,:)
    dudeta(2:nx,:,k,:) = c(1)*u(2:nx,:,k-1,:) + c(2)*u(2:nx,:,k,:) &
        + c(3)*u(2:nx,:,k+1,:)
    detadz(:, :, k, :) = (etaw(k+1)-etaw(k)) / (hght(:, :, k+1, :)-hght(:, :, k, :))
end do
allocate(dzdx(2:nx,ny,2:nz,nt))
forall (i = 2:nx, k = 2:nz, t = 1:nt)
    dzdx(i,:,k,t) = .5 * (mf(i-1,:) + mf(i,:)) &
        * (hght(i,:,k,t) - hght(i-1,:,k,t)) / dx
end forall
allocate(dzdy(nx,2:ny,2:nz,nt))
forall (j = 2:ny, k = 2:nz, t = 1:nt)
    dzdy(:,j,k,t) = .5 * (mf(:,j-1) + mf(:,j)) &
        * (hght(:,j,k,t) - hght(:,j-1,k,t)) / dy
end forall
allocate(vort(2:nx,2:ny,2:nz-1,nt))
do t = 1, nt
    do k = 2, nz-1
        do j = 2, ny
            do i = 2, nx
                mfavg = .25 * (mf(i-1,j-1) + mf(i,j-1) + mf(i-1,j) + mf(i,j))
                detadzavg = .25 * (detadz(i-1,j-1,k,t) + &
                    detadz(i,j-1,k,t) + detadz(i-1,j,k,t) + detadz(i,j,k,t))
                dzdxavg = .25 * (dzdx(i,j-1,k,t) + dzdx(i,j,k,t) &
                    + dzdx(i,j-1,k+1,t) + dzdx(i,j,k+1,t))
                dzdyavg = .25 * (dzdy(i-1,j,k,t) + dzdy(i,j,k,t) &

```

```

        + dzdy(i-1,j,k+1,t) + dzdy(i,j,k+1,t))
    dvdeavg = .5 * (dvdeteta(i-1,j,k,t) + dvdeteta(i,j,k,t))
    dudeavg = .5 * (dudeteta(i,j-1,k,t) + dudeteta(i,j,k,t))
    vort(i,j,k,t) = mfavg * ((v(i,j,k,t) - v(i-1,j,k,t))/dx &
        + detadzavg * (dzdyavg*dudeavg - dzdxavg*dvdeavg) &
        - (u(i,j,k,t) - u(i,j-1,k,t))/dy)
    end do
end do
end do
print *, "Calculating PV..."
! We've got all the pieces... now combine them!
forall (i = 2:nx-1, j = 2:ny-1, k = 2:nz-1, t = 1:nt)
    q(i,j,k,t) = (detadz(i,j,k,t) * dthetadeta(i,j,k,t) &
        * (f(i,j) + .25 * (vort(i,j,k,t) + vort(i+1,j,k,t) &
        + vort(i,j+1,k,t) + vort(i+1,j+1,k,t)))) / rho(i,j,k,t)
end forall
deallocate(dthetadeta, dzdx, dzdy, dvdeteta, dudeteta, vort, detadz)
! Set boundary values to interior neighbors
print *, "Filling boundaries..."
q(2:nx-1, (/1,ny/), 2:nz-1,:) = q(2:nx-1, (/2,ny-1/), 2:nz-1,:)
q((/1,nx/), :, 2:nz-1,:) = q((/2,nx-1/), :, 2:nz-1,:)
q(:, :, (/1,nz/), :) = q(:, :, (/2,nz-1/), :)
end subroutine pvor
! =====
! Private functions (called only from this module) go below this line.
! =====
! density_thp uses the ideal gas law to compute the density of dry air given
! its potential temperature (theta; K) and pressure (p; hPa)
real function density_thp(theta, p)
    real, intent(in) :: theta, p
    real :: t
    t = temp_from_theta_p(theta, p, si=.false.)
    density_thp = 100 * p / (RGas * t) ! * 100 since p is in hPa
end function density_thp
! =====
! theta_es computes the saturated equivalent potential temperature given
! potential temperature (theta; K) and pressure (p; hPa)
real function theta_es(theta, p)
    real, intent(in) :: theta, p
    real :: tmpk, es, ws
    tmpk = temp_from_theta_p(theta, p, si=.false.)
    es = sat_vap_pres(tmpk)
    ws = mixr(es, p)
    theta_es = theta * exp(LatHeatVap * ws / (SpecHeatPresDry * tmpk))
end function theta_es
! =====
! theta_es_to_theta_fast computes the potential temperature given the
! saturated equivalent potential temperature (thes; K) and the
! pressure (p; hPa), using a lookup table.

```

```

function theta_es_to_theta_fast(thes, p) result(theta)
  real, intent(in) :: thes, p
  real              :: theta
  integer, dimension(2), parameter :: PLim = (/ 50, 1020 /), &
                                         TLim = (/ 2500, 3500 /)

  real, dimension(PLim(1):PLim(2),TLim(1):TLim(2)), save :: lookup
  real      :: t
  integer :: i, j
  logical :: firstTime = .true.
  ! Establish lookup table.
  if (firstTime) then
    firstTime = .false.
    do j = TLim(1), TLim(2)
      t = .1 * j
      do i = PLim(1), PLim(2)
        lookup(i,j) = theta_es_to_theta(t, real(i))
      end do
    end do
  end if
  ! Are we within lookup table?
  t = 10 * thes
  if (t > TLim(1) .and. t < TLim(2) .and. p > PLim(1) .and. p < PLim(2)) then
    i = int(p)
    j = int(t)
    theta = bilin_int(lookup(i,j), lookup(i,j+1), lookup(i+1,j), &
                     lookup(i+1,j+1), p-i, t-j)
  else
    theta = theta_es_to_theta(thes, p)
  end if
end function theta_es_to_theta_fast
! theta_es_to_theta computes the potential temperature given the saturated
! equivalent potential temperature (thes; K) and the pressure (p; hPa),
! using the secant method.
function theta_es_to_theta(thes, p) result(theta)
  real, intent(in) :: thes, p
  real              :: theta
  real, parameter :: Delta = 1e-4, Epsil = 1e-4
  integer, parameter :: MaxIter = 200
  real      :: a, b, fa, fb, s
  integer :: k
  ! See if we're so dry that theta_es approximately equals theta
  a = thes
  if (abs(theta_es(a, p) - a) < .05) then
    theta = a
    return
  end if
  ! Make initial guess
  a = temp_from_theta_p(thes, p, si=.false.)
  b = a - 5
  fa = f(a)

```

```

fb = f(b)
! Here's the secant method, at the end of which a is the temperature.
do k = 2, MaxIter
  if (abs(fa) > abs(fb)) then
    call swap(a, b)
    call swap(fa, fb)
  end if
  s = (b - a) / (fb - fa)
  b = a
  fb = fa
  a = a - fa * s
  fa = f(a)
  if (abs(fa) < Epsil .or. abs(b-a) < Delta) exit
end do
theta = a * (1000. / p)**Kappa
contains
! This function has a zero when the temperature satisfies the equation
! for saturated equivalent potential temperature.
real function f(t)
  real, intent(in) :: t
  real :: es
  real :: ws
  es = sat_vap_pres(t)
  ws = mixr(es, p)
  ! Compare f to function theta_es
  f = thes - t * (1000./p)**kappa * &
    exp(LatHeatVap * ws / (SpecHeatPresDry * t))
end function f
end function theta_es_to_theta
! =====
! mixr calculates the mixing ratio (kg/kg) given the vapor pressure (e) and
! pressure (p). e and p must have the same units.
real function mixr(e, p)
  real, intent(in) :: e, p
  mixr = Eps * e / (p - e)
end function mixr
! =====
! vap_pres calculates the vapor pressure given the mixing ratio (w; kg/kg)
! and pressure (p). The vapor pressure will be in the same units as p.
real function vap_pres(w, p)
  real, intent(in) :: w, p
  vap_pres = w * p / (Eps + w)
end function vap_pres
! =====
! sat_vap_pres uses the data from Curry & Webster (1999, p.113) to compute
! the saturation vapor pressure (hPa) given temperature (tmpk; K).
real function sat_vap_pres(tmpk)
  real, intent(in) :: tmpk
  real, dimension(7), parameter :: a = (/ 6.1117675, &
    .443986062, .143053301e-1, .265027242e-3, .302246994e-5, &

```

```

        .203886313e-7, .638780966e-10 /)
real    :: psum, tdiff
integer :: i
tdiff = tmpk - 273.15
if (tdiff < -50) then
    sat_vap_pres = 6.11 * exp(LatHeatVap / RVap * (1. / 273.16 - 1. / tmpk))
else
    psum = 0
    do i = 7, 2, -1
        psum = psum + a(i) * tdiff**(i-1)
    end do
    sat_vap_pres = psum + a(1)
end if
end function sat_vap_pres
! =====
! temp_lcl calculates the temperature at the LCL given a parcel's temperature
! (tmpk; K) and relative humidity (rh; 0<rh<1) using Curry & Webster (1999,
! Eq. 6.30).
real function temp_lcl(tmpk, rh)
    real, intent(in) :: tmpk, rh
    real :: a, b
    a = 1. / (tmpk - 55)
    b = log(rh) / 2840.
    temp_lcl = 1. / (a - b) + 55
end function temp_lcl
! =====
! rh calculates the relative humidity (0<rh<1) given the mixing ratio
! (w; kg/kg), pressure (p; hPa), and temperature (tmpk; K).
real function rh(w, p, tmpk)
    real, intent(in) :: w, p, tmpk
    real :: es, e
    es = sat_vap_pres(tmpk)
    e = vap_pres(w, p)
    rh = e/es
end function rh
! =====
! swap interchanges the values of a and b.
subroutine swap(a, b)
    real, intent(inout) :: a, b
    real :: temp
    temp = a
    a = b
    b = temp
end subroutine swap
! =====
! Given profiles of variables a (aProf) and b (bProf) and a target value of
! variable a (a), find_bounds outputs two adjacent values in aProf that
! bound a (aBound) and the associated values of b (bBound).
subroutine find_bounds(aProf, bProf, a, aBound, bBound)
    real, dimension(:), intent(in) :: aProf, bProf

```

```

real,          intent(in)  :: a
real, dimension(2), intent(out) :: aBound, bBound
integer :: i, aSize
logical :: found
! Make sure input arrays match
aSize = size(aProf)
if (aSize /= size(bProf)) then
  print *, "aProf and bProf have different sizes!", aSize, size(bProf)
  stop
end if
! Find bounds
found = .false.
do i = 1, aSize-1
  if (aProf(i) >= a .and. aProf(i+1) < a .or. &
      aProf(i) <= a .and. aProf(i+1) > a) then
    found = .true.
    aBound(1) = aProf(i)
    aBound(2) = aProf(i+1)
    bBound(1) = bProf(i)
    bBound(2) = bProf(i+1)
    exit
  end if
end do
if (.not. found) then
  print *, "Could not find bounds!"
  print *, aProf, a
  stop
end if
end subroutine find_bounds
! =====
! lin_int performs linear interpolation given two data points (x1,y1) and
! (x2,y2) and a known value for x.
real function lin_int(x1, x2, y1, y2, x)
  real, intent(in) :: x1, x2, y1, y2, x
  if (x1 /= x2) then
    lin_int = y1 + (y2-y1)*(x-x1)/(x2-x1)
  else
    lin_int = y1
  end if
end function lin_int
! =====
! The first four values are the known quantities in the four corners.
! a and b are the fraction of the way we are between the left and right
! and lower and upper sides, respectively.
real function bilin_int(ll, lr, ul, ur, a, b)
  real, intent(in) :: ll, lr, ul, ur, a, b
  real, dimension(4) :: w
  w(1) = (1-a) * (1-b)
  w(2) = a * (1-b)
  w(3) = (1-a) * b

```

```

w(4) = a * b
    bilin_int = w(1) * ll + w(2) * lr + w(3) * ul + w(4) * ur
end function bilin_int
! =====

```

## 8.4 GEMPAK utilities

The GEMPAK module contains the data and routines necessary to interface with the GEMPAK library. For instance, a subroutine that creates a GEMPAK file is contained here.

```

module gempak
  use wrf2gem_parameters, only: NLUnit, NLFName
  implicit none
  save
  private
  public :: init_gem, create_gemfile, write_gempak, close_and_exit_gem
  integer, parameter      :: navsz = 256, ianlsz = 128, ihdrsz = 2
  real, dimension(navsz)  :: rnvblk, rnvblk2
  real, dimension(ianlsz) :: anlblk = 0
  integer, dimension(2)   :: level, ighdr = 0
  character(len=20), dimension(2) :: gdatm
  logical                 :: pack = .true., rewrit = .true.
contains ! =====
  ! init_gem initializes the GEMPAK interface.
  subroutine init_gem()
    integer :: status
    call in_bdta(status)
    call handle_gerr(status, "ID_BDTA")
  end subroutine init_gem
  ! =====
  ! create_gemfile creates a new GEMPAK file, using the following information:
  ! o Projection type (proj)
  ! o Grid extent in x and y dimensions (nx, ny)
  ! o Latitude/longitude in lower-left and upper-right grid corners (lat, lon)
  ! o Additional angles necessary for the map projection (ang)
  ! If successful, a file handle (gid) is assigned and provided as output.
  ! If the GEMPAK file already exists, the program will either use it or abort
  ! depending on the user's preference. However, if gsFlag = .false., the grid
  ! navigation of the current GEMPAK file does not match what's needed;
  ! otherwise there's no problem.
  subroutine create_gemfile(proj, nx, ny, lat, lon, ang, gid, gsFlag)
    integer,          intent(in)  :: proj, nx, ny
    real, dimension(2), intent(in) :: lat, lon
    real, dimension(3), intent(in) :: ang
    integer,          intent(out) :: gid
    logical,          intent(out) :: gsFlag

```

```

integer                :: maxGrd = 5000, status
character(len=80)      :: gFName = "wrf.gem"
character(len=3)       :: projName
logical                :: overwrite = .false., nlExist, gemExist, angFlg
namelist /GEMFILE/ gFName, maxGrd, pack, overwrite
! Check for namelist existence, then read it.
inquire (file=NLFName, exist=nlExist)
if (nlExist) then
  open (unit=NLUnit, file=NLFName, iostat=status)
  if (status /= 0) stop "There is a problem opening the namelist file."
  read (unit=NLUnit, nml=GEMFILE, iostat=status)
  if (status /= 0) stop "There is a problem reading the namelist file."
  close (unit=NLUnit)
end if
! If the GEMPAK file already exists, follow user's wishes.
inquire (file=gFName, exist=gemExist)
if (gemExist) then
  if (overwrite) then
    print *, "GEMPAK file already exists! wrf2gem will attempt to add to&
      & and/or overwrite it."
  else
    stop "GEMPAK file already exists!"
  end if
end if
! Set up map projection
select case(proj)
case (0)
  projName = "CED"
  angFlg = .false.
case (1)
  if (ang(1) < 0 .and. ang(3) < 0) then
    projName = "SCC"
  else
    projName = "LCC"
  end if
  angFlg = .true.
case (3)
  projName = "MER"
  angFlg = .false.
case default
  print *, "This projection not yet implemented."
  stop
end select
! Call appropriate gemlib routines to create or add to the GEMPAK file.
call gr_mnav(projName, nx, ny, lat(1), lon(1), lat(2), lon(2), ang(1), &
  ang(2), ang(3), angFlg, rnvblk, status)
call handle_gerr(status, "GR_MNAV")
if (gemExist) then
  call gd_opnr(gFName, gid, navsz, rnvblk2, ianlsz, anlblk, ihdrsz, &
    maxGrd, status)

```



```

        call handle_gerr(status, "GD_OPNR")
        ! Make sure grid navigation blocks match.
        call gr_cnav(rnvblk, rnvblk2, navsz, gsFlag, status)
        call handle_gerr(status, "GR_CNAV")
    else
        call gd_cref(gFName, navsz, rnvblk, ianlsz, anlblk, ihdrsz, maxGrd, &
            gid, status)
        call handle_gerr(status, "GD_CREF")
        gsFlag = .true.
    end if
end subroutine create_gemfile
! =====
! write_gempak writes a 2D array of data (grid) to the GEMPAK file associated
! with id. The extents of grid are specified by nx and ny. The time at which
! the array is valid is specified in GEMPAK form in gdat1. It is assumed that
! the grid does not have two times associated with it. GEMPAK level
! information is provided by lev1 and lev2, and the GEMPAK parameter name is
! given by parm.
subroutine write_gempak(id, grid, nx, ny, gdat1, lev1, lev2, cord, parm)
    integer,          intent(in) :: id, nx, ny, lev1, lev2, cord
    real, dimension(nx,ny), intent(in) :: grid
    character(len=20), intent(in) :: gdat1
    character(len=12), intent(in) :: parm
    integer :: status
    ! Initialize GEMPAK level and date/time information
    level(1) = lev1 ; level(2) = lev2
    gdattm(1) = gdat1 ; gdattm(2) = " "
    ! Call appropriate gemlib routines
    if (pack) then
        call gd_wpgd(id, grid, nx, ny, ighdr, gdattm, level, cord, parm, &
            rewrit, 1, 16, status)
        call handle_gerr(status, "GD_WPGD")
    else
        call gd_wdat(id, grid, nx, ny, ighdr, gdattm, level, cord, parm, &
            rewrit, status)
        call handle_gerr(status, "GD_WDAT")
    end if
end subroutine write_gempak
! =====
! close_and_exit_gem closes the GEMPAK file associated with igdfln as well as
! the gemlib interface.
subroutine close_and_exit_gem(igdfln)
    integer, intent(in) :: igdfln
    integer :: status
    call gd_clos(igdfln, status)
    call handle_gerr(status, "GD_CLOS")
    call ip_exit(status)
    call handle_gerr(status, "IP_EXIT")
end subroutine close_and_exit_gem
! =====

```

```

! handle_gerr examines the result code from a gemlib routine. If necessary,
! it then prints an error message and aborts the program.
subroutine handle_gerr(status, routine)
  integer,          intent(in) :: status
  character(len=*), intent(in) :: routine
  if (status /= 0) then
    write (*,"(a,a,i3)") routine, ": status = ", status
    stop
  end if
end subroutine handle_gerr
end module gempak

```

## 8.5 Bookkeeping utilities

The utilities in this module are sometimes used to compute the number of gridpoints in one direction, taking into account the grid staggering.

```

module maths_utils
  implicit none
  private
  public :: cds1, cds2
contains
  pure integer function cds1(nPoints, dim)
    integer, intent(in) :: nPoints, dim
    if (dim == 1) then
      cds1 = nPoints - 1
    else
      cds1 = nPoints
    end if
  end function cds1
  !.....
  pure integer function cds2(nPoints, dim)
    integer, intent(in) :: nPoints, dim
    if (dim == 2) then
      cds2 = nPoints - 1
    else
      cds2 = nPoints
    end if
  end function cds2
end module maths_utils

```

## 8.6 Mathematical utilities

This module contains functions that carry out some common finite differences.

```

module maths
  use maths_utils, only: cds1, cds2
  implicit none
  private
  public :: mean, cen_diff_stag, cen_diff, uneven_deriv, calc_mfpsi
  real, parameter, public :: Half = 0.5, One = 1.
  real, parameter          :: Quarter = 0.25
contains
  real function mean(data)
    real, dimension(:,,:), intent(in) :: data
    mean = sum(data) / size(data)
  end function mean
  ! =====
  function cen_diff_stag(data, rdx, dim)
    real, dimension(:,,:), intent(in) :: data
    real,
    integer,
    integer, intent(in) :: dim
    real, dimension(cds1(size(data,1),dim),cds2(size(data,2),dim)) :: cen_diff_stag
    integer :: nPoints
    nPoints = size(data,dim) - 1
    select case (dim)
    case (1)
      cen_diff_stag = rdx * (data(2:nPoints+1,:) - data(1:nPoints,:))
    case (2)
      cen_diff_stag = rdx * (data(:,2:nPoints+1) - data(:,1:nPoints))
    case default
      stop "Error in dim value in cen_diff_stag!"
    end select
  end function cen_diff_stag
  ! =====
  function cen_diff(d, dxr, dim)
    real, dimension(:,:,:,:), intent(in) :: d
    real,
    integer,
    integer, intent(in) :: dim
    real, dimension(size(d,1),size(d,2),size(d,3),size(d,4)) :: cen_diff
    integer :: n
    n = size(d,dim)
    select case (dim)
    case (1)
      cen_diff(2:n-1,:,:,) = Half * dxr * (d(3:n,:,:,) - d(1:n-2,:,:,))
      cen_diff((/1,n/),:,:,) = dxr * (d((/2,n/),:,:,) - d((/1,n-1/),:,:,))
    case (2)
      cen_diff(:,2:n-1,:,:) = Half * dxr * (d(:,3:n,:,:) - d(:,1:n-2,:,:,))
      cen_diff(:,(/1,n/),:,:) = dxr * (d(:,(/2,n/),:,:) - d(:,(/1,n-1/),:,:,))
    case (4)
      cen_diff(:,:,:,2:n-1) = Half * dxr * (d(:,:,:,3:n) - d(:,:,:,1:n-2))
      cen_diff(:,:,:,(/1,n/)) = dxr * (d(:,:,:,(/2,n/)) - d(:,:,:,(/1,n-1/)))
    case default
      stop "Bad dim in cen_diff!"
    end select
  end function

```

```

end function cen_diff
! =====
function uneven_deriv(d, coordVal, dim)
  real, dimension(:,:,:), intent(in) :: d
  real, dimension(:), intent(in) :: coordVal
  integer, intent(in) :: dim
  real, dimension(size(d,1),size(d,2),size(d,3),size(d,4)) :: uneven_deriv
  real, dimension(2:size(coordVal)) :: deltaCoord
  real, dimension(3) :: c
  integer :: n, i
  n = size(d,dim)
  if (n /= size(coordVal)) stop "Arrays don't conform in uneven_deriv!"
  deltaCoord = coordVal(2:) - coordVal(:n-1)
  select case (dim)
  case (3)
    do i = 2, n - 1
      c = deriv_coeff(deltaCoord(i), deltaCoord(i+1))
      uneven_deriv(:, :, i, :) = c(1) * d(:, :, i-1, :) + c(2) * d(:, :, i, :) + &
        c(3)*d(:, :, i+1, :)
    end do
    uneven_deriv(:, :, (/1,n/), :) = (d(:, :, (/2,n/), :) - &
      d(:, :, (/1,n-1/), :)) / spread(spread(spread(coordVal((/2,n/)) - &
        coordVal((/1,n-1/)),1,size(d,1)),2,size(d,2)),4,size(d,4))
  case default
    stop "This dim value not implemented in uneven_deriv!"
  end select
end function uneven_deriv
! =====
function calc_mfpsi(mfu, mfv)
  real, dimension(:,:), intent(in) :: mfu, mfv
  real, dimension(size(mfv,1)-1,size(mfu,2)-1) :: calc_mfpsi
  integer :: nx, ny!, i, j
  nx = size(mfv,1)
  ny = size(mfu,2)
  if (size(mfv,2) /= ny-1 .or. size(mfu,1) /= nx-1) stop "Mismatch in mfpsi!"
  calc_mfpsi = Quarter * (mfu(:, :ny-1)+mfu(:, 2:) + mfv(:nx-1, :)+mfv(2:, :))
end function calc_mfpsi
! =====
! A function to calculate the coefficients for taking a first derivative
! with variable grid spacing
pure function deriv_coeff(d1, d2)
  real, intent(in) :: d1, d2 ! Grid spacings adj. to derivative pt.
  real, dimension(3) :: deriv_coeff
  deriv_coeff(1) = -d2 / (d1 * (d1+d2))
  deriv_coeff(2) = (d2**2 - d1**2) / (d1 * d2 * (d1+d2))
  deriv_coeff(3) = d1 / (d2 * (d1+d2))
end function deriv_coeff
end module maths

```

## 8.7 Floating point kinds

This module defines the single and double precision floating point kinds. Single (double) precision values are guaranteed to have at least five (ten) digits of accuracy.

```
module kind_types
  implicit none
  save
  integer, parameter :: sp = selected_real_kind(5)
  integer, parameter :: dp = selected_real_kind(10)
end module kind_types
```

## 8.8 Simple constants

This module selects a floating point representation and then defines simple numerals and fractions in terms of that selection.

```
module current_kind
  use kind_types, only: cp => dp
  implicit none
  save
  real(cp), parameter :: Zero = 0._cp, One = 1._cp, Quarter = .25_cp, &
    Four = 4._cp, Half = .5_cp, Eighth = .125_cp, &
    Two = 2._cp, Third = 1._cp / 3._cp
end module current_kind
```

## 8.9 User input and netCDF utilities

This module contains subroutines that handle reading WRF model output in netCDF format as well as the main utilities that determine how the program will operate during the current run.

```
module wrf2gem_subs
  use netcdf, only: nf90_open, nf90_inq_dimid, nf90_inquire_dimension, &
    nf90_get_att, nf90_inq_varid, nf90_get_var, nf90_close, &
    nf90_inquire_variable
  use netcdf, only: NF90_NoWrite, NF90_NoErr, NF90_Global, NF90_StrError, &
    NF90_Max_Var_Dims
  use dateutil, only: hr_diff
  use dateutil, only: operator(<), operator(>)
  use dateutil, only: datestamp
```

```

use wrf2gem_parameters, only: NLUnit, NLFName
implicit none
save
private
public :: get_out_fields, get_num_files, open_wrf_file, get_grid_info, &
         update_grid_info, get_times, close_wrf_file, handle_err, &
         mem_error, read_values, read_values_const, freq_divisible
public :: NLUnit, NLFName
integer, parameter :: MaxOuts = 200
integer,          dimension(MaxOuts), target :: outFields
character(len=80), dimension(MaxOuts), target :: fName
interface read_values
    module procedure rv0d
    module procedure rv1d
    module procedure rv2d
    module procedure rv3d
    module procedure rv4d
end interface
interface read_values_const
    module procedure rv0dc
    module procedure rv1dc
    module procedure rv2dc
    module procedure rv3dc
end interface
contains ! =====
! get_out_fields reads the namelist to determine what variables should be
! output to the GEMPAK file. It returns the number of variables to output
! (num) and a pointer to an array of variable IDs (ptr). If there are no
! variables to output, the status of ptr does not change. The subroutine
! also determines how pressure interpolation should be handled, giving
! values for the bottom and top pressure levels (pb, pt) and the pressure
! interval (dp).
subroutine get_out_fields(num, ptr, pb, pt, dp)
    integer, intent(out) :: num
    integer, dimension(:), pointer :: ptr
    integer, intent(out) :: pb, pt, dp
    integer :: status
    logical :: nlExist
    namelist /NUMOUTS/ num
    namelist /VARS/ outFields, pb, pt, dp
    ! The default if there is no namelist information is to output nothing.
    num = 0
    ! Defaults for pressure level interpolation information
    pb = 1000; pt = 200; dp = 50
    ! Check for namelist existence, then read it.
    inquire (file=NLFName, exist=nlExist)
    if (nlExist) then
        open (unit=NLUnit, file=NLFName, iostat=status)
        if (status /= 0) stop "There is a problem opening the namelist file."
        read (unit=NLUnit, nml=NUMOUTS, iostat=status)
    
```

```

        if (status /= 0) stop "There is a problem reading NUMOUTS from the &
            &namelist file."
        if (num > MaxOuts) stop "There are too many outputs!"
        read (unit=NLUnit, nml=VARS, iostat=status)
        if (status /= 0) stop "There is a problem reading VARS from the &
            &namelist file."
        close (unit=NLUnit)
        if (num > 0) ptr => outFields(1:num)
    end if
end subroutine get_out_fields
! =====
! get_num_files reads the namelist to determine what WRF files should be
! read for conversion. It returns the number of files to convert (numFiles)
! and a pointer to an array of filenames (point). If there are no files to
! convert, the status of point does not change.
subroutine get_num_files(numFiles, point)
    integer, intent(out) :: numFiles
    character(len=80), dimension(:), pointer :: point
    integer :: status
    logical :: nlExist
    namelist /OUTFILES/ numFiles
    namelist /WRFFILE/ fName
    ! The default if there is no namelist information is to output nothing.
    numFiles = 0
    ! Check for namelist existence, then read it.
    inquire (file=NLFName, exist=nlExist)
    if (nlExist) then
        open (unit=NLUnit, file=NLFName, iostat=status)
        if (status /= 0) stop "There is a problem opening the namelist file."
        read (unit=NLUnit, nml=OUTFILES, iostat=status)
        if (status /= 0) stop "There is a problem reading OUTFILES from the &
            &namelist file."
        if (numFiles > MaxOuts) stop "There are too many files to process!"
        read (unit=NLUnit, nml=WRFFILE, iostat=status)
        if (status /= 0) stop "There is a problem reading WRFFILE from the &
            &namelist file."
        close (unit=NLUnit)
        if (numFiles > 0) point => fName(1:numFiles)
    end if
end subroutine get_num_files
! =====
! open_wrf_file opens the WRF history file (fil) and returns its handle (id).
subroutine open_wrf_file(fil, id)
    character(len=80), intent(in) :: fil
    integer,          intent(out) :: id
    integer :: status
    status = nf90_open(fil, NF90_NoWrite, id)
    call handle_err(status)
end subroutine open_wrf_file
! =====

```

```

! get_grid_info gathers the following info from the WRF history file (ncid):
! o Model grid extent in x, y, and z directions (nx, ny, nz)
! o Number of times in the file (nt)
! o Map projection type (proj)
! o Latitude/longitude in lower-left and upper-right grid corners (lat, lon)
! o Projection angle info (ang)
subroutine get_grid_info(ncid, nx, ny, nz, nt, proj, lat, lon, ang)
  integer,          intent(in)  :: ncid
  integer,          intent(out) :: nx, ny, nz, nt, proj
  real, dimension(2), intent(out) :: lat, lon
  real, dimension(3), intent(out) :: ang
  character(len=11), dimension(4), parameter :: DimName = (/ "west_east  ", &
                                                             "south_north", "bottom_top ", "Time      " /)

  integer, dimension(4) :: temp
  integer                :: i, varid, status
  do i = 1, 4
    status = nf90_inq_dimid(ncid, trim(DimName(i)), varid)
    call handle_err(status)
    status = nf90_inquire_dimension(ncid, varid, len=temp(i))
    call handle_err(status)
  end do
  nx = temp(1)
  ny = temp(2)
  nz = temp(3)
  nt = temp(4)
  status = nf90_get_att(ncid, NF90_Global, "MAP_PROJ", proj)
  call handle_err(status)
  status = nf90_inq_varid(ncid, "XLONG", varid)
  call handle_err(status)
  status = nf90_get_var(ncid, varid, lon(1), start = (/ 1, 1, 1 /))
  call handle_err(status)
  status = nf90_get_var(ncid, varid, lon(2), start = (/ nx, ny, 1 /))
  call handle_err(status)
  status = nf90_inq_varid(ncid, "XLAT", varid)
  call handle_err(status)
  status = nf90_get_var(ncid, varid, lat(1), start = (/ 1, 1, 1 /))
  call handle_err(status)
  status = nf90_get_var(ncid, varid, lat(2), start = (/ nx, ny, 1 /))
  call handle_err(status)
  select case (proj)
  case (0) ! Ideal x/y
    ang = 0
    lon = (/ -100. - nx/2., -100. + nx/2. /)
    lat = (/ 45. - ny/2., 45. + ny/2. /)
  case (1) ! LCC
    status = nf90_get_att(ncid, NF90_Global, "TRUELAT1", ang(1))
    call handle_err(status)
    status = nf90_get_att(ncid, NF90_Global, "CEN_LON", ang(2))
    call handle_err(status)
    status = nf90_get_att(ncid, NF90_Global, "TRUELAT2", ang(3))

```



```

        call handle_err(status)
    case (3) ! Mercator
        ang = 0
    case default
        write (*,"(a,i2)") "proj = ", proj
        print *, "This map projection has not yet been implemented."
        stop
    end select
end subroutine get_grid_info
! =====
! update_grid_info compares grid size and prejection settings from the WRF
! history file (ncid) with the following previously saved settings:
! o Model grid extent in x, y, and z directions (nx, ny, nz)
! o Map projection type (proj)
! o Latitude/longitude in lower-left and upper-right grid corners (lat, lon)
! o Projection angle info (ang)
! If there is not an exact match, ok is set to .false.; otherwise, ok is
! .true. The number of times in the WRF file (nt) is also provided as output.
subroutine update_grid_info(ncid, nx, ny, nz, proj, lat, lon, ang, nt, ok)
    integer,          intent(in)  :: ncid, nx, ny, nz, proj
    real, dimension(2), intent(in) :: lat, lon
    real, dimension(3), intent(in) :: ang
    integer,          intent(out) :: nt
    logical,          intent(out) :: ok
    character(len=11), dimension(4), parameter :: DimName = (/ "west_east  ", &
                                                                "south_north", "bottom_top ", "Time      " /)
    real,    dimension(3) :: angl
    real,    dimension(2) :: lonl, latl
    integer, dimension(4) :: temp
    integer                                :: i, varid, status, nxl, nyl, nzl, projl
    do i = 1, 4
        status = nf90_inq_dimid(ncid, trim(DimName(i)), varid)
        call handle_err(status)
        status = nf90_inquire_dimension(ncid, varid, len=temp(i))
        call handle_err(status)
    end do
    nxl = temp(1)
    nyl = temp(2)
    nzl = temp(3)
    nt = temp(4)
    status = nf90_get_att(ncid, NF90_Global, "MAP_PROJ", projl)
    call handle_err(status)
    status = nf90_inq_varid(ncid, "XLONG", varid)
    call handle_err(status)
    status = nf90_get_var(ncid, varid, lonl(1), start = (/ 1, 1, 1 /))
    call handle_err(status)
    status = nf90_get_var(ncid, varid, lonl(2), start = (/ nxl, nyl, 1 /))
    call handle_err(status)
    status = nf90_inq_varid(ncid, "XLAT", varid)
    call handle_err(status)

```

```

status = nf90_get_var(ncid, varid, lat1(1), start = (/ 1, 1, 1 /))
call handle_err(status)
status = nf90_get_var(ncid, varid, lat1(2), start = (/ nxl, nyl, 1 /))
call handle_err(status)
select case (proj1)
case (0) ! Ideal x/y
    angl = 0
    lon1 = (/ -100. - nx/2., -100. + nx/2. /)
    lat1 = (/ 45. - ny/2., 45. + ny/2. /)
case (1) ! LCC
    status = nf90_get_att(ncid, NF90_Global, "TRUELAT1", angl(1))
    call handle_err(status)
    status = nf90_get_att(ncid, NF90_Global, "CEN_LON", angl(2))
    call handle_err(status)
    status = nf90_get_att(ncid, NF90_Global, "TRUELAT2", angl(3))
    call handle_err(status)
case (3) ! Mercator
    angl = 0
case default
    write (*,"(a,i2)") "proj = ", proj
    print *, "This map projection has not yet been implemented."
    stop
end select
! Check for exact match
ok = (nx1 == nx .and. nyl == ny .and. nz1 == nz .and. &
      proj1 == proj .and. all(lat1 == lat) .and. all(lon1 == lon) .and. &
      all(ang1 == ang))
end subroutine update_grid_info
! =====
! get_times uses information from the WRF history file (id) and the namelist
! to create an appropriate array of output times in GEMPAK form (times). In
! addition, the number of outputs per 12 hours (timesPer12hr) is provided for
! output. Also provided as input is the list of history files (outFiles),
! and the index of the current history file being processed (outInd). This
! subroutine assumes the history file contains either output every n hours,
! where n is some positive integer, or output for one time.
subroutine get_times(outInd, outFiles, id, times, timesPer12hr)
    integer,                                intent(in)      :: outInd
    character(len=80), dimension(:), intent(in)      :: outFiles
    integer,                                intent(inout)   :: id
    character(len=*), dimension(:), intent(out)       :: times
    integer,                                intent(out)     :: timesPer12hr
    integer, parameter :: y1 = 1, y2 = 4, m1 = 6, m2 = 7, d1 = 9, d2 = 10, &
                          h1 = 12, h2 = 13
    character(len=19), dimension(size(times,1)) :: wrfTimes
    type(datestamp) :: fileInit, runInit, adj, temp
    integer :: numTimes, varid, status, adjID, interval, ihr1, i
    character(len=19) :: adjTimes, startDate
    character(len=3)  :: fhr
    character(len=2)  :: chr1, chr2

```

```

! Initialization
numTimes = size(times,1)
! Read WRF history file times
! The format for these times is yyyy-mm-dd_hh:mm:ss
status = nf90_inq_varid(id, "Times", varid)
call handle_err(status)
status = nf90_get_var(id, varid, wrfTimes)
call handle_err(status)
! Determine initial time from the history file
status = nf90_get_att(id, nf90_global, "START_DATE", startDate)
call handle_err(status)
fileInit%yr = char_to_int(wrfTimes(1)(y1:y2))
runInit%yr = char_to_int( startDate(y1:y2))
fileInit%mth = char_to_int(wrfTimes(1)(m1:m2))
runInit%mth = char_to_int( startDate(m1:m2))
fileInit%day = char_to_int(wrfTimes(1)(d1:d2))
runInit%day = char_to_int( startDate(d1:d2))
fileInit%hr = char_to_int(wrfTimes(1)(h1:h2))
runInit%hr = char_to_int( startDate(h1:h2))
! Calculate interval between outputs in hours, and number of outputs per 12
! hours.
if (numTimes > 1) then
  adj%yr = char_to_int(wrfTimes(2)(y1:y2))
  adj%mth = char_to_int(wrfTimes(2)(m1:m2))
  adj%day = char_to_int(wrfTimes(2)(d1:d2))
  adj%hr = char_to_int(wrfTimes(2)(h1:h2))
else if (size(outFiles) > 1) then
  call close_wrf_file(id) ! My system can't handle two open netCDF files!
  if (outInd > 1) then
    call open_wrf_file(outFiles(outInd-1), adjID)
  else
    call open_wrf_file(outFiles(2), adjID)
  end if
  status = nf90_inq_varid(adjID, "Times", varid)
  call handle_err(status)
  status = nf90_get_var(id, varid, adjTimes)
  call handle_err(status)
  call close_wrf_file(adjID)
  call open_wrf_file(outFiles(outInd), id)
  adj%yr = char_to_int(adjTimes(y1:y2))
  adj%mth = char_to_int(adjTimes(m1:m2))
  adj%day = char_to_int(adjTimes(d1:d2))
  adj%hr = char_to_int(adjTimes(h1:h2))
else
  adj = fileInit
end if
if (adj > fileInit) then
  interval = hr_diff(adj, fileInit)
else if (adj < fileInit) then
  interval = hr_diff(fileInit, adj)

```

```

else
    interval = 12 ! Set so that timesPer12hr = 1
end if
timesPer12hr = 12 / interval
! Calculate forecast hour to which the first time in history file
! corresponds
ihr1 = hr_diff(fileInit, runInit)
if (ihr1 < 0) then
    print *, "WARNING: Can't handle forecast hour calculation properly!"
    ihr1 = 0
end if
! Form GEMPAK times for all of the times in the history file
! GEMPAK format is yymmdd/hhmmFfff
do i = 1, numTimes
    times(i)(1:4) = int_to_char2(runInit%yr) // int_to_char2(runInit%mth)
    times(i)(5:7) = int_to_char2(runInit%day) // "/"
    times(i)(8:12) = int_to_char2(runInit%hr) // wrfTimes(1)(15:16) // "F"
    write (unit=fhr, fmt="(i3)") ihr1
    select case (ihr1)
    case (0:9)
        fhr(1:2) = "00"
    case (10:99)
        fhr(1:1) = "0"
    end select
    times(i)(13:15) = fhr
    ihr1 = ihr1 + interval
end do
end subroutine get_times
! =====
! close_wrf_file closes the WRF history file specified by id.
subroutine close_wrf_file(id)
    integer, intent(in) :: id
    integer :: status
    status = nf90_close(id)
    call handle_err(status)
end subroutine close_wrf_file
! =====
! handle_err examines the result code from a netCDF function (status). If
! necessary, it then prints a netCDF error message and aborts the program.
subroutine handle_err(status)
    integer, intent(in) :: status
    if (status /= NF90_NoErr) then
        print *, trim(NF90_strerror(status))
        print *, "Stopped"
        stop
    end if
end subroutine handle_err
! =====
! mem_error examines the result code from a memory allocation/deallocation
! (status). id determines whether allocation or deallocation was attempted.

```

```

! string specifies the program unit that attempted the memory operation.  If
! necessary, an error message is printed and the program is aborted.
subroutine mem_error(status, id, string)
  integer, intent(in) :: status, id
  character(len=*), intent(in) :: string
  if (status /= 0) then
    select case (id)
    case (1)
      print *, "Memory allocation error in ", string
    case (2)
      print *, "Memory deallocation error in ", string
    case default
      print *, "Memory error in ", string
    end select
    stop
  end if
end subroutine mem_error

! =====
! rv0d supplies a scalar (value) containing the variable specified by varName
! from the netCDF file associated with id.
subroutine rv0d(id, varName, value)
  integer,          intent(in)  :: id
  character(len=*), intent(in)  :: varName
  real,             intent(out) :: value
  integer, dimension(NF90_Max_Var_Dims) :: dimIDs
  integer :: varid, status, nDims, ncSize
  ! Get array extents from the netCDF file
  status = nf90_inq_varid(id, varName, varid)
  call handle_err(status)
  status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
  call handle_err(status)
  status = nf90_inquire_dimension(id, dimIDs(1), len = ncSize)
  call handle_err(status)
  ! Make sure array extents match!
  if (ncSize /= 1 .or. nDims /= 0) then
    print *, "Array dimensions are incorrect in read_values."
    stop
  end if
  ! Read the variable from the netCDF file
  status = nf90_get_var(id, varid, value)
  call handle_err(status)
end subroutine rv0d

! .....
! rv1d supplies a 1D array (values) containing the variable specified by
! varName from the netCDF file associated with id.
subroutine rv1d(id, varName, values)
  integer,          intent(in)  :: id
  character(len=*), intent(in)  :: varName
  real, dimension(:), intent(out) :: values
  integer, parameter :: NumDims = 1

```

```

integer, dimension(NF90_Max_Var_Dims) :: dimIDs
integer, dimension(NumDims)           :: sizes, ncSizes
integer :: i, varid, status, nDims
! Determine array extents that wrf2gem expects
sizes = shape(values)
! Get array extents from the netCDF file
status = nf90_inq_varid(id, varName, varid)
call handle_err(status)
status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
call handle_err(status)
do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
end do
! Make sure array extents match!
if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
    "Array dimensions are incorrect in read_values."
! Read the variable from the netCDF file
status = nf90_get_var(id, varid, values)
call handle_err(status)
end subroutine rv1d
! .....
! rv2d is identical to rv1d, but for a 2D array.
subroutine rv2d(id, varName, values)
integer,          intent(in) :: id
character(len=*), intent(in) :: varName
real, dimension(:, :), intent(out) :: values
integer, parameter :: NumDims = 2
integer, dimension(NF90_Max_Var_Dims) :: dimIDs
integer, dimension(NumDims)           :: sizes, ncSizes
integer :: i, varid, status, nDims
! Determine array extents that wrf2gem expects
sizes = shape(values)
! Get array extents from the netCDF file
status = nf90_inq_varid(id, varName, varid)
call handle_err(status)
status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
call handle_err(status)
do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
end do
! Make sure array extents match!
if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
    "Array dimensions are incorrect in read_values."
! Read the variable from the netCDF file
status = nf90_get_var(id, varid, values)
call handle_err(status)
end subroutine rv2d
! .....

```

```

! rv3d is identical to rv2d, but for a 3D array.
subroutine rv3d(id, varName, values)
  integer,          intent(in)  :: id
  character(len=*), intent(in)  :: varName
  real, dimension(:, :, :), intent(out) :: values
  integer, parameter :: NumDims = 3
  integer, dimension(NF90_Max_Var_Dims) :: dimIDs
  integer, dimension(NumDims)          :: sizes, ncSizes
  integer :: i, varid, status, nDims
  ! Determine array extents that wrf2gem expects
  sizes = shape(values)
  ! Get array extents from the netCDF file
  status = nf90_inq_varid(id, varName, varid)
  call handle_err(status)
  status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
  call handle_err(status)
  do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
  end do
  ! Make sure array extents match!
  if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
    "Array dimensions are incorrect in read_values."
  ! Read the variable from the netCDF file
  status = nf90_get_var(id, varid, values)
  call handle_err(status)
end subroutine rv3d
! .....
! rv4d is identical to rv2d, but for a 4D array.
subroutine rv4d(id, varName, values)
  integer,          intent(in)  :: id
  character(len=*), intent(in)  :: varName
  real, dimension(:, :, :, :), intent(out) :: values
  integer, parameter :: NumDims = 4
  integer, dimension(NF90_Max_Var_Dims) :: dimIDs
  integer, dimension(NumDims)          :: sizes, ncSizes
  integer :: i, varid, status, nDims
  ! Determine array extents that wrf2gem expects
  sizes = shape(values)
  ! Get array extents from the netCDF file
  status = nf90_inq_varid(id, varName, varid)
  call handle_err(status)
  status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
  call handle_err(status)
  do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
  end do
  ! Make sure array extents match!
  if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &

```

```

        "Array dimensions are incorrect in read_values."
        ! Read the variable from the netCDF file
        status = nf90_get_var(id, varid, values)
        call handle_err(status)
    end subroutine rv4d
! =====
! Same as rv0dc, but accounting for the fact that the value doesn't change
! with time.
subroutine rv0dc(id, varName, value)
    integer,          intent(in)  :: id
    character(len=*), intent(in)  :: varName
    real,             intent(out) :: value
    integer, parameter :: NumDims = 2
    real,   dimension(:, :), allocatable :: temp
    integer, dimension(NF90_Max_Var_Dims) :: dimIDs
    integer, dimension(NumDims)           :: sizes, ncSizes
    integer :: status, varid, nt, i, nDims
    ! Get number of times in data file
    status = nf90_inq_dimid(id, "Time", varid)
    call handle_err(status)
    status = nf90_inquire_dimension(id, varid, len=nt)
    call handle_err(status)
    ! Determine array extents that wrf2gem expects
    allocate(temp(1,nt), stat=status)
    call mem_error(status, 1, "rv0dc")
    sizes = shape(temp)
    ! Get array extents from the netCDF file
    status = nf90_inq_varid(id, varName, varid)
    call handle_err(status)
    status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
    call handle_err(status)
    do i = 1, NumDims
        status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
        call handle_err(status)
    end do
    ! Make sure array extents match!
    if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
        "Array dimensions are incorrect in read_values."
    ! Read the variable from the netCDF file
    status = nf90_get_var(id, varid, temp)
    call handle_err(status)
    value = temp(1,1)
end subroutine rv0dc
! .....
! Same as rv0dc, but for a 1D array.
subroutine rv1dc(id, varName, values)
    integer,          intent(in)  :: id
    character(len=*), intent(in)  :: varName
    real, dimension(:), intent(out) :: values
    integer, parameter :: NumDims = 2

```



```

real,    dimension(:,:), allocatable :: temp
integer, dimension(NF90_Max_Var_Dims) :: dimIDs
integer, dimension(NumDims)           :: sizes, ncSizes
integer :: status, varid, nt, i, nDims
! Get number of times in data file
status = nf90_inq_dimid(id, "Time", varid)
call handle_err(status)
status = nf90_inquire_dimension(id, varid, len=nt)
call handle_err(status)
! Determine array extents that wrf2gem expects
allocate(temp(size(values),nt), stat=status)
call mem_error(status, 1, "rv1dc")
sizes = shape(temp)
! Get array extents from the netCDF file
status = nf90_inq_varid(id, varName, varid)
call handle_err(status)
status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
call handle_err(status)
do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
end do
! Make sure array extents match!
if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
    "Array dimensions are incorrect in read_values."
! Read the variable from the netCDF file
status = nf90_get_var(id, varid, temp)
call handle_err(status)
values = temp(:,1)
end subroutine rv1dc
! .....
! Same as rv0dc, but for a 2D array.
subroutine rv2dc(id, varName, values)
integer,          intent(in)  :: id
character(len=*), intent(in)  :: varName
real, dimension(:,:), intent(out) :: values
integer, parameter :: NumDims = 3
real,    dimension(:,:,:), allocatable :: temp
integer, dimension(NF90_Max_Var_Dims) :: dimIDs
integer, dimension(NumDims)           :: sizes, ncSizes
integer :: status, varid, nt, i, nDims
! Get number of times in data file
status = nf90_inq_dimid(id, "Time", varid)
call handle_err(status)
status = nf90_inquire_dimension(id, varid, len=nt)
call handle_err(status)
! Determine array extents that wrf2gem expects
allocate(temp(size(values,1),size(values,2),nt), stat=status)
call mem_error(status, 1, "rv2dc")
sizes = shape(temp)

```

```

! Get array extents from the netCDF file
status = nf90_inq_varid(id, varName, varid)
call handle_err(status)
status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
call handle_err(status)
do i = 1, NumDims
    status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
    call handle_err(status)
end do
! Make sure array extents match!
if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
    "Array dimensions are incorrect in read_values."
! Read the variable from the netCDF file
status = nf90_get_var(id, varid, temp)
call handle_err(status)
values = temp(:, :, 1)
end subroutine rv2dc
! .....
! Same as rv0dc, but for a 3D array.
subroutine rv3dc(id, varName, values)
    integer,          intent(in) :: id
    character(len=*), intent(in) :: varName
    real, dimension(:, :, :), intent(out) :: values
    integer, parameter :: NumDims = 4
    real, dimension(:, :, :, :), allocatable :: temp
    integer, dimension(NF90_Max_Var_Dims) :: dimIDs
    integer, dimension(NumDims) :: sizes, ncSizes
    integer :: status, varid, nt, i, nDims
    ! Get number of times in data file
    status = nf90_inq_dimid(id, "Time", varid)
    call handle_err(status)
    status = nf90_inquire_dimension(id, varid, len=nt)
    call handle_err(status)
    ! Determine array extents that wrf2gem expects
    allocate(temp(size(values,1),size(values,2),size(values,3),nt),stat=status)
    call mem_error(status, 1, "rv3dc")
    sizes = shape(temp)
    ! Get array extents from the netCDF file
    status = nf90_inq_varid(id, varName, varid)
    call handle_err(status)
    status = nf90_inquire_variable(id, varid, ndims = nDims, dimids = dimIDs)
    call handle_err(status)
    do i = 1, NumDims
        status = nf90_inquire_dimension(id, dimIDs(i), len = ncSizes(i))
        call handle_err(status)
    end do
    ! Make sure array extents match!
    if (any(ncSizes /= sizes) .or. nDims /= NumDims) stop &
        "Array dimensions are incorrect in read_values."
    ! Read the variable from the netCDF file

```

```

    status = nf90_get_var(id, varid, temp)
    call handle_err(status)
    values = temp(:, :, :, 1)
end subroutine rv3dc
! =====
! Given that there are p outputs per q hours, can the outputs be subdivided
! into blocks of r hours? This function answers that question.
! Yes = .true., and No = .false.
! Negative values for p, q, or r will also give .false. as output.
logical function freq_divisible(p, q, r)
    integer, intent(in) :: p, q, r
    integer :: gcf, pReduced, qReduced, outputTimestep
    ! Make sure p, q, and r are all positive
    if (p <= 0 .or. q <= 0 .or. r <= 0) then
        freq_divisible = .false.
        return
    end if
    ! Reduce the fraction p/q to lowest terms.
    gcf = greatest_common_factor(p, q)
    pReduced = p / gcf
    qReduced = q / gcf
    ! First test: If r is divisible by q, then certainly there is a subdivision
    ! into r; in fact, there are pr blocks.
    ! Second test: If p doesn't divide into q, then the output timestep is not
    ! an integer, which is not allowed if r is not divisible by q.
    ! Third test: If r is a multiple of the output timestep, we've found the
    ! subdivision we're looking for.
    if (mod(r, qReduced) == 0) then
        freq_divisible = .true.
    else if (mod(qReduced, pReduced) /= 0) then
        freq_divisible = .false.
    else
        outputTimestep = qReduced / pReduced
        if (mod(r, outputTimestep) == 0) then
            freq_divisible = .true.
        else
            freq_divisible = .false.
        end if
    end if
end function freq_divisible
! =====
! greatest_common_factor uses the Euclidean algorithm to compute the GCF of
! integers a and b. It returns zero if both a and b are zero. Otherwise, it
! returns a positive integer.
recursive function greatest_common_factor(a, b) result (gcf)
    integer, intent(in) :: a, b
    integer :: gcf
    if (b == 0) then
        gcf = a
    else

```

```

        gcf = greatest_common_factor(b, mod(a,b))
    end if
end function greatest_common_factor
! =====
! This function converts the input character string (ch) to an integer.
integer function char_to_int(ch)
    character(len=*), intent(in) :: ch
    character(len=*), parameter :: digs = "0123456789 -", frmt = "(i9)"
    integer :: chLen, i, ind
    character(len=len(ch)) :: chl
    chLen = len(ch)
    chl = ch
    if (chLen < 1) then
        char_to_int = 0
        return
    end if
    ! Replace bad characters with spaces
    do i = 1, chLen
        if (scan(chl(i:i),digs) == 0) chl(i:i) = " "
    end do
    ! Make sure only spaces precede a minus sign
    ind = scan(chl,"-")
    chl(1:ind-1) = " "
    ! Make sure no additional minus signs occur
    do i = ind+1, chLen
        if (chl(i:i) == "-") chl(i:i) = " "
    end do
    read (chl, frmt) char_to_int
end function char_to_int
! =====
! This function converts a positive integer (num) into a character of
! length 2. If num < 0, it's absolute value is used. If num has more than
! 2 digits, it is truncated such that only the ones and tens digits convert.
! A leading zero is added if necessary.
character(len=2) function int_to_char2(num)
    integer, intent(in) :: num
    integer :: num1
    num1 = abs(num)
    if (num1 > 99) num1 = mod(num1,100)
    write(unit=int_to_char2, fmt="(i2)") num1
    if (num1 < 10) int_to_char2(1:1) = "0"
end function int_to_char2
end module wrf2gem_subs

```

## 8.10 Wind partitioning and PV inversion utilities

This module performs the bulk of the work involved in partitioning the flow into nondivergent and irrotational parts. It also contains the routines responsible for calculating the potential vorticity and nonlinear imbalance operators, their tangent linear and adjoint versions, and the PV inversion routine itself.

```

module partition_wind
  use current_kind, only: cp, Zero, One, Quarter, Four, Half, Eighth, Two, Third
  use diagnostics, only: RGas, Grav, Kappa
  implicit none
  private
  public :: partition, balance_lhs, balance_lhs_exp, balance_lhs_tlm, &
           balance_lhs_adj, balance_rhs_simp, balance_rhs_simp_exp, &
           balance_rhs_simp_tlm, balance_rhs_simp_adj, imbalance_simp_adj, &
           imbalance_exp, pv_exp, pv_tlm, pv_adj, solve_phi_psi
contains
  ! Extents should be as follows:
  ! us/mu - (0:nx,ny) ; vs/mv - (nx,0:ny) ; div/mchi - (nx,ny)
  ! vort/mpsi - (nx-1,ny-1) ; chi - (0:nx+1,0:ny+1) ; psi - (0:nx,0:ny)
  subroutine partition(us, vs, div, vort, mchi, mpsi, mu, mv, phill, rdx, chi, psi)
    real(cp), dimension(0:,:), intent(in) :: us, mu
    real(cp), dimension(:,0:), intent(in) :: vs, mv
    real(cp), dimension(:,,:), intent(in) :: div, vort, mchi, mpsi
    real(cp), intent(in) :: phill, rdx
    real(cp), dimension(0:size(us,1),0:size(vs,2)), intent(out) :: chi
    real(cp), dimension(0:size(vs,1),0:size(us,2)), intent(out) :: psi
    real(cp), parameter :: Eps = 1e-4_cp
    integer, parameter :: MaxIter = 50000
    character(len=*), parameter :: Mismatch = "mismatch in partition!"
    real(cp), dimension(:,:,:), allocatable :: ps, ch
    real(cp), dimension(size(div,1),size(div,2)) :: divForcing, divCheck
    real(cp), dimension(0:size(div,1),size(div,2)) :: uu
    real(cp), dimension(size(div,1),0:size(div,2)) :: vv
    real(cp), dimension(size(div,1)-1,size(div,2)) :: u
    real(cp), dimension(size(div,1),size(div,2)-1) :: v
    real(cp), dimension(size(vort,1),size(vort,2)) :: vortForcing, vortCheck
    real(cp) :: dx, dx2, err
    integer :: nx, ny, prev, cur, i, j, k
    nx = size(vs,1)
    ny = size(us,2)
    dx = One / rdx
    dx2 = dx**2
    divForcing = dx2 * div / mchi**2
    vortForcing = dx2 * vort / mpsi**2
    ! Sanity checks
    if (any(shape(us) /= shape(mu))) stop "us mu " // Mismatch
  end subroutine partition
end module partition_wind

```

```

if (any(shape(vs) /= shape(mv))) stop "vs mv " // Mismatch
if (any(shape(div) /= shape(mchi))) stop "div " // Mismatch
if (any(shape(vort) /= shape(mpsi))) stop "vort " // Mismatch
if (any(shape(mchi) /= (/ nx, ny /))) stop "mchi " // Mismatch
if (any(shape(mpsi) /= (/ nx-1, ny-1 /))) stop "mpsi " // Mismatch
if (any(shape(chi) /= (/ nx+2, ny+2 /))) stop "chi " // Mismatch
if (any(shape(psi) /= (/ nx+1, ny+1 /))) stop "psi " // Mismatch
! Initialize
allocate(ch(0:nx+1,0:ny+1,2), ps(0:nx,0:ny,2))
prev = 1; cur = 2
! First guess
ch = Zero ; ps = Zero
do k = 1, MaxIter
  prev = 2 - mod(k,2)
  cur = 2 - mod(k+1,2)
  ! Compute boundary values of psi
  ! ps(0,1) is fixed at zero
  ps(0,2:,cur) = ps(1,2:,prev) + &
    (ch(1,3:,prev) - ch(1,2:ny,prev)) - dx * vs(1,2:) / mv(1,2:)
  ps(1:,ny,cur) = ps(1:,ny-1,prev) + &
    (ch(2:,ny,prev) - ch(1:nx,ny,prev)) - dx * us(1:,ny) / mu(1:,ny)
  ps(nx:,ny-1,cur) = ps(nx-1:,ny-1,prev) - &
  ps(1:nx-1,0,cur) = ps(1:nx-1,1,prev) - &
    (ch(2:nx,1,prev)-ch(1:nx-1,1,prev)) + dx*us(1:nx-1,1)/mu(1:nx-1,1)
  ! Compute interior values of psi
  call update_sor(ps, vortForcing, prev, cur)
  ! Compute interior values of chi
  call update_sor(ch, divForcing, prev, cur)
  ! Check current errors
  ! First u
  u = mu(1:nx-1,:) * (ch(2:nx,1:ny,cur) - ch(1:nx-1,1:ny,cur) - &
    (ps(1:nx-1,1:,cur) - ps(1:nx-1:,ny-1,cur))) / dx
  ! Then v
  v = mv(:,1:ny-1) * (ps(1:,1:ny-1,cur) - ps(:,nx-1,1:ny-1,cur) + &
    ch(1:nx,2:ny,cur) - ch(1:nx,1:ny-1,cur)) / dx
  err = sqrt(sum(abs(u-us(1:nx-1,:))**2) + sum(abs(v-vs(:,1:ny-1))**2))
  if(err < Eps) exit
  ! Check for errors in vorticity and divergence
  vortCheck = mpsi**2 * ( &
    ps(:,nx-2,1:ny-1,cur) + ps(2:,1:ny-1,cur) + ps(1:nx-1:,ny-2,cur) + &
    ps(1:nx-1,2:,cur) - Four * ps(1:nx-1,1:ny-1,cur) ) / dx2
  divCheck = mchi**2 * ( &
    ch(:,nx-1,1:ny,cur) + ch(2:,1:ny,cur) + ch(1:nx:,ny-1,cur) + &
    ch(1:nx,2:,cur) - Four * ch(1:nx,1:ny,cur) ) / dx2
  if (mod(k,250) == 0) write (*, "(a,i6,a,3es12.5)" &
    "Iteration", k, ". Errors in Wnd/Vor/Div: ", err, &
    maxval(abs(vortCheck-vort)), maxval(abs(divCheck-div))
  ! Finally, calculate divergence from psi and vorticity from chi
  uu = -mu * (ps(:,1:,cur) - ps(:,ny-1,cur)) / dx
  vv = mv * (ps(1:,: ,cur) - ps(nx-1:,: ,cur)) / dx

```

```

divCheck = mchi**2 * rdx * ( &
    uu(1:, :)/mu(1:, :) - uu(:nx-1, :)/mu(:nx-1, :) + &
    vv(:, 1:)/mv(:, 1:) - vv(:, :ny-1)/mv(:, :ny-1) )
uu = mu * (ch(1:, 1:ny, cur) - ch(:nx, 1:ny, cur)) / dx
vv = mv * (ch(1:nx, 1:, cur) - ch(1:nx, :ny, cur)) / dx
vortCheck = mpsi**2 * rdx * &
    (vv(2:, 1:ny-1)/mv(2:, 1:ny-1) - vv(:nx-1, 1:ny-1)/mv(:nx-1, 1:ny-1) - &
    (uu(1:nx-1, 2:)/mu(1:nx-1, 2:) - uu(1:nx-1, :ny-1)/mu(1:nx-1, :ny-1)))
if (mod(k, 250) == 0) write (*, "(a, 2es12.5)") &
    "          Div from Psi / Vor from Chi: ", maxval(abs(divCheck)), &
    maxval(abs(vortCheck))
end do
if (k > MaxIter) stop "Too many iterations!"
chi = ch(:, :, cur)
psi = ps(:, :, cur)
end subroutine partition
! =====
subroutine update_sor(data, forcing, prev, cur)
real(cp), dimension(0:, 0:, :), intent(inout) :: data
real(cp), dimension(:, :), intent(in) :: forcing
integer, intent(in) :: prev, cur
real(cp), parameter :: Omega = 1.8_cp
integer :: nx, ny, j, i
nx = size(forcing, 1)
ny = size(forcing, 2)
! Sanity checks
if (all(shape(data) /= (/ nx+2, ny+2, 2/))) stop "Mismatch in update_sor!"
if (prev + cur /= 3 .or. cur /= 1 .and. cur /= 2) stop "prev/cur error!"
do j = 1, ny-1
    do i = 1, nx-1
        data(i, j, cur) = data(i, j, prev) + Quarter * Omega * ( &
            data(i-1, j, cur) + data(i+1, j, prev) + data(i, j-1, cur) + &
            data(i, j+1, prev) - Four * data(i, j, prev) - forcing(i, j) )
    end do
    ! i = nx special
    data(i, j, cur) = data(i, j, prev) + Quarter * Omega * ( &
        data(i-1, j, cur) + data(i+1, j, cur) + data(i, j-1, cur) + &
        data(i, j+1, prev) - Four * data(i, j, prev) - forcing(i, j) )
    end do
    ! j = ny special
    do i = 1, nx-1
        data(i, j, cur) = data(i, j, prev) + Quarter * Omega * ( &
            data(i-1, j, cur) + data(i+1, j, prev) + data(i, j-1, cur) + &
            data(i, j+1, cur) - Four * data(i, j, prev) - forcing(i, j) )
    end do
    ! (nx, ny) special
    data(i, j, cur) = data(i, j, prev) + Quarter * Omega * ( &
        data(i-1, j, cur) + data(i+1, j, cur) + data(i, j-1, cur) + &
        data(i, j+1, cur) - Four * data(i, j, prev) - forcing(i, j) )
end subroutine update_sor

```

```

! =====
subroutine balance_lhs(psi, f, mCap, m, mu, mv, rdx, lhs)
  real(cp), dimension(0:,0:,0:), intent(in) :: psi      ! (0:nx,0:ny)
  real(cp), dimension(:,0:), intent(in) :: f, mCap, m   ! (1:nx,1:ny)
  real(cp), dimension(0:,0:), intent(in) :: mu
  real(cp), dimension(:,0:), intent(in) :: mv           ! (1:nx,0:ny)
  real(cp), dimension(0:,0:), intent(in) :: rdx
  real, dimension(size(f,1),size(f,2),size(psi,3)), intent(out) :: lhs
  character(len=*), parameter :: Mismatch = "mismatch in balance_lhs!"
  real(cp), dimension(:,0:), allocatable :: mdxp, mdyp, c, fmvdxp, fmudyp, t1, &
                                         mdx1, mdx2, t2, mdy1, mdy2, t3

  integer :: nx, ny, nz, k, i, j
  nx = size(psi,1) - 1
  ny = size(psi,2) - 1
  nz = size(psi,3)
  ! Sanity checks
  if (any(shape(f) /= (/ nx, ny /)) stop "f " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) stop "m " // Mismatch
  if (any(shape(mu) /= (/ nx+1, ny /)) stop "mu " // Mismatch
  if (any(shape(mv) /= (/ nx, ny+1 /)) stop "mv " // Mismatch
  allocate(mdxp(nx,0:ny), mdyp(0:nx,ny), c(nx-1,ny-1))
  allocate(fmvdxp(nx-1,2:ny-1), fmudyp(2:nx-1,ny-1))
  allocate(t1(2:nx-1,2:ny-1), t2(2:nx-1,2:ny-1), t3(2:nx-1,2:ny-1))
  allocate(mdx1(nx-1,2:ny-1), mdx2(nx-1,2:ny-1))
  allocate(mdy1(2:nx-1,ny-1), mdy2(2:nx-1,ny-1))
  lhs = Zero
  do k = 1, nz
    mdxp = mv * xdiff(psi(:,0:k), rdx)
    mdyp = mu * ydiff(psi(0:,0:k), rdx)
    fmvdxp = xavg(f(:,2:ny-1)) * xyavg(mdxp(:,1:ny-1)) / mu(1:nx-1,2:ny-1)
    fmudyp = yavg(f(2:nx-1,:)) * xyavg(mdyp(1:nx-1,:)) / mv(2:nx-1,1:ny-1)
    t1 = m(2:nx-1,2:ny-1)**2 * (xdiff(fmvdxp, rdx) + ydiff(fmudyp, rdx))
    forall (i = 1:nx-1, j = 2:ny-1)
      mdx1(i,j) = Half * mu(i,j)**2 * rdx**3 &
        * ( mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) * psi(i,j,k)**2 &
          - Two * mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) * psi(i,j-1,k) &
            * psi(i,j,k) &
          + mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) * psi(i,j-1,k)**2 &
          + mu(i+1,j)/mCap(i+1,j) * psi(i,j,k) * psi(i+1,j,k) &
          - mu(i+1,j)/mCap(i+1,j) * psi(i,j-1,k) * psi(i+1,j,k) &
          - mu(i+1,j)/mCap(i+1,j) * psi(i,j,k) * psi(i+1,j-1,k) &
          + mu(i+1,j)/mCap(i+1,j) * psi(i,j-1,k) * psi(i+1,j-1,k) &
          - mu(i-1,j)/mCap(i,j) * psi(i,j,k) * psi(i-1,j,k) &
          + mu(i-1,j)/mCap(i,j) * psi(i,j-1,k) * psi(i-1,j,k) &
          + mu(i-1,j)/mCap(i,j) * psi(i,j,k) * psi(i-1,j-1,k) &
          - mu(i-1,j)/mCap(i,j) * psi(i,j-1,k) * psi(i-1,j-1,k) )
    end forall
    forall (i = 1:nx-1, j = 1:ny-1)
      c(i,j) = One / (mCap(i,j) + mCap(i+1,j) + mCap(i,j+1) + mCap(i+1,j+1))
    end forall
  end do

```



```

forall (i = 1:nx-1, j = 2:ny-1)
  mdx2(i,j) = Half * mu(i,j) * rdx**3 &
    * ( mv(i,j-1) * psi(i,j-1,k) - mv(i,j-1) * psi(i-1,j-1,k) &
      + mv(i+1,j-1) * psi(i+1,j-1,k) - mv(i+1,j-1) * psi(i,j-1,k) &
      + mv(i,j) * psi(i,j,k) - mv(i,j) * psi(i-1,j,k) &
      + mv(i+1,j) * psi(i+1,j,k) - mv(i+1,j) * psi(i,j,k) ) &
    * ( c(i,j) * mu(i,j+1) * psi(i,j+1,k) - c(i,j) * mu(i,j+1) &
      * psi(i,j,k) + c(i,j) * mu(i,j) * psi(i,j,k) - c(i,j) * mu(i,j) &
      * psi(i,j-1,k) - c(i,j-1) * mu(i,j) * psi(i,j,k) + c(i,j-1) &
      * mu(i,j) * psi(i,j-1,k) - c(i,j-1) * mu(i,j-1) * psi(i,j-1,k) &
      + c(i,j-1) * mu(i,j-1) * psi(i,j-2,k) )
end forall
forall (i = 2:nx-1, j = 2:ny-1)
  t2(i,j) = Half * rdx**4 * m(i,j) * ( &
    mu(i,j) * (mu(i,j) * (psi(i,j,k) - psi(i,j-1,k)) * ( &
      mu(i,j) * (One/mCap(i+1,j)-One/mCap(i,j)) * (psi(i,j,k) &
      - psi(i,j-1,k)) + mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k) &
      - psi(i+1,j-1,k)) - mu(i-1,j)/mCap(i,j) * (psi(i-1,j,k) &
      - psi(i-1,j-1,k))) - (mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) &
      + mv(i+1,j-1) * (psi(i+1,j-1,k) - psi(i,j-1,k)) + &
      mv(i,j) * (psi(i,j,k) - psi(i-1,j,k)) + &
      mv(i+1,j) * (psi(i+1,j,k) - psi(i,j,k))) * ( &
      c(i,j) * (mu(i,j+1) * (psi(i,j+1,k) - psi(i,j,k)) + &
      mu(i,j) * (psi(i,j,k) - psi(i,j-1,k))) - &
      c(i,j-1) * (mu(i,j) * (psi(i,j,k) - psi(i,j-1,k)) + &
      mu(i,j-1) * (psi(i,j-1,k) - psi(i,j-2,k)))) - &
      mu(i-1,j) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) * ( &
      mu(i-1,j)*(One/mCap(i,j)-One/mCap(i-1,j)) &
      * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
      + mu(i,j)/mCap(i,j) * (psi(i,j,k) - psi(i,j-1,k)) - &
      mu(i-2,j)/mCap(i-1,j) * (psi(i-2,j,k) - psi(i-2,j-1,k))) - ( &
      mv(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-2,j-1,k)) + &
      mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) + &
      mv(i-1,j) * (psi(i-1,j,k) - psi(i-2,j,k)) + &
      mv(i,j) * (psi(i,j,k) - psi(i-1,j,k))) * ( &
      c(i-1,j) * (mu(i-1,j+1) * (psi(i-1,j+1,k) - psi(i-1,j,k)) + &
      mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k))) - &
      c(i-1,j-1) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) + &
      mu(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-1,j-2,k))))))
end forall
mdy1 = mv(2:nx-1,1:ny-1) * mdxp(2:nx-1,1:ny-1) &
  * ydiff(yavg(mdxp(2:nx-1,:)) / mCap(2:nx-1,:), rdx)
mdy2 = mv(2:nx-1,1:ny-1) * xyavg(mdy1(1:nx-1,:)) &
  * xdiff(xavg(mdxp(:,1:ny-1)) / xyavg(mCap), rdx)
t3 = m(2:nx-1,2:ny-1) * ydiff(mdy1 - mdy2, rdx)
lhs(2:nx-1,2:ny-1,k) = t1 - t2 - t3
end do
deallocate(mdy2, mdy1, mdx2, mdx1, t3, t2, t1, fmudyp, fmvdxp, mdy1, mdxp)
end subroutine balance_lhs
! =====

```

```

subroutine balance_lhs_exp(psi, f, mCap, m, mu, mv, rdx, lhs)
  real(cp), dimension(0:,0:,:), intent(in) :: psi      ! (0:nx,0:ny)
  real(cp), dimension(:,,:), intent(in) :: f, mCap, m   ! (1:nx,1:ny)
  real(cp), dimension(0:,:), intent(in) :: mu
  real(cp), dimension(:,0:), intent(in) :: mv          ! (1:nx,0:ny)
  real(cp), intent(in) :: rdx
  real, dimension(size(f,1),size(f,2),size(psi,3)), intent(out) :: lhs
  character(len=*), parameter :: Mismatch = "mismatch in balance_lhs!"
  real(cp), dimension(:, :, :), allocatable :: t1, t2, t3
  real(cp), dimension(:, :), allocatable :: c
  integer :: nx, ny, nz, i, j, k
  nx = size(psi,1) - 1
  ny = size(psi,2) - 1
  nz = size(psi,3)
  ! Sanity checks
  if (any(shape(f) /= (/ nx, ny /)) stop "f " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) stop "m " // Mismatch
  if (any(shape(mu) /= (/ nx+1, ny /)) stop "mu " // Mismatch
  if (any(shape(mv) /= (/ nx, ny+1 /)) stop "mv " // Mismatch
  allocate(t1(2:nx-1,2:ny-1,nz), t2(2:nx-1,2:ny-1,nz), t3(2:nx-1,2:ny-1,nz))
  allocate(c(nx-1,ny-1))
  forall (i = 1:nx-1, j = 1:ny-1)
    c(i,j) = One / (mCap(i,j) + mCap(i+1,j) + mCap(i,j+1) + mCap(i+1,j+1))
  end forall
  lhs = Zero
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    t1(i,j,k) = Eighth * rdx**2 * m(i,j)**2 * (
      (f(i, j) + f(i+1,j)) * (
        mv(i, j-1) * (psi(i, j-1,k) - psi(i-1,j-1,k)) +
        mv(i+1,j-1) * (psi(i+1,j-1,k) - psi(i, j-1,k)) +
        mv(i, j) * (psi(i, j, k) - psi(i-1,j, k)) +
        mv(i+1,j) * (psi(i+1,j, k) - psi(i, j, k)) ) / mu(i, j) -
      (f(i-1,j) + f(i,j)) * (
        mv(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-2,j-1,k)) +
        mv(i, j-1) * (psi(i, j-1,k) - psi(i-1,j-1,k)) +
        mv(i-1,j) * (psi(i-1,j, k) - psi(i-2,j, k)) +
        mv(i, j) * (psi(i, j, k) - psi(i-1,j, k)) ) / mu(i-1,j) +
      (f(i, j) + f(i,j+1)) * (
        mu(i-1,j) * (psi(i-1,j, k) - psi(i-1,j-1,k)) +
        mu(i, j) * (psi(i, j, k) - psi(i, j-1,k)) +
        mu(i-1,j+1) * (psi(i-1,j+1,k) - psi(i-1,j, k)) +
        mu(i, j+1) * (psi(i, j+1,k) - psi(i, j, k)) ) / mv(i, j) -
      (f(i,j-1) + f(i,j)) * (
        mu(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-1,j-2,k)) +
        mu(i, j-1) * (psi(i, j-1,k) - psi(i, j-2,k)) +
        mu(i-1,j) * (psi(i-1,j, k) - psi(i-1,j-1,k)) +
        mu(i, j) * (psi(i, j, k) - psi(i, j-1,k)) ) / mv(i,j-1) )
    t2(i,j,k) = Half * rdx**4 * m(i,j) * (
      mu(i,j) * (mu(i,j) * (psi(i,j,k) - psi(i,j-1,k)) &
        * (mu(i,j) * (One/mCap(i+1,j)-One/mCap(i,j)) * (psi(i,j,k) &

```

```

- psi(i,j-1,k)) + mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k) &
- psi(i+1,j-1,k)) - mu(i-1,j)/mCap(i,j) * (psi(i-1,j,k) &
- psi(i-1,j-1,k))) - (mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) &
+ mv(i+1,j-1) * (psi(i+1,j-1,k) - psi(i,j-1,k)) + mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i+1,j) * (psi(i+1,j,k)-psi(i,j,k))) &
* (c(i,j) * (mu(i,j+1) * (psi(i,j+1,k) - psi(i,j,k)) + mu(i,j) &
* (psi(i,j,k) - psi(i,j-1,k))) - c(i,j-1) * (mu(i,j) * (psi(i,j,k) &
- psi(i,j-1,k)) + mu(i,j-1) * (psi(i,j-1,k) - psi(i,j-2,k)))) &
- mu(i-1,j) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) &
* (mu(i-1,j) * (One/mCap(i,j)-One/mCap(i-1,j)) * (psi(i-1,j,k) &
- psi(i-1,j-1,k)) + mu(i,j)/mCap(i,j) * (psi(i,j,k) - psi(i,j-1,k)) &
- mu(i-2,j)/mCap(i-1,j) * (psi(i-2,j,k) - psi(i-2,j-1,k))) &
- (mv(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-2,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) &
+ mv(i-1,j) * (psi(i-1,j,k) - psi(i-2,j,k)) + mv(i,j) * (psi(i,j,k) &
- psi(i-1,j,k))) * (c(i-1,j) * (mu(i-1,j+1) * (psi(i-1,j+1,k) &
- psi(i-1,j,k)) + mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k))) &
- c(i-1,j-1) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-1,j-2,k))))))
t3(i,j,k) = Half * rdx**4 * m(i,j) * ( &
mv(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) * (One/mCap(i,j+1) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j+1) * (psi(i,j+1,k) &
- psi(i-1,j+1,k))) - One/mCap(i,j) * (mv(i,j-1) * (psi(i,j-1,k) &
- psi(i-1,j-1,k)) + mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)))) &
- (mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) + mu(i,j) * (psi(i,j,k) &
- psi(i,j-1,k)) + mu(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) &
+ mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k))) * (c(i,j) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i+1,j) * (psi(i+1,j,k) &
- psi(i,j,k))) - c(i-1,j) * (mv(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)))) - mv(i,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) * (One/mCap(i,j) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i,j) * (psi(i,j,k) &
- psi(i-1,j,k))) - One/mCap(i,j-1) * (mv(i,j-2) * (psi(i,j-2,k) &
- psi(i-1,j-2,k)) + mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) &
- (mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k)) + mu(i,j-1) &
* (psi(i,j-1,k)-psi(i,j-2,k)) + mu(i-1,j) * (psi(i-1,j,k) &
- psi(i-1,j-1,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k))) * (c(i,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k))) - c(i-1,j-1) * (mv(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-2,j-1,k)) + mv(i,j-1) * (psi(i,j-1,k) &
- psi(i-1,j-1,k))))))
end forall
lhs(2:nx-1,2:ny-1,:) = t1 - t2 - t3
deallocate(c, t3, t2, t1)
end subroutine balance_lhs_exp
! =====
subroutine balance_lhs_tlm(dps, psi, f, mCap, m, mu, mv, rdx, dlhs)
real(cp), dimension(0:,0:,:), intent(in) :: dps, psi ! (0:nx,0:ny)
real(cp), dimension(:,,:), intent(in) :: f, mCap, m ! (1:nx,1:ny)
real(cp), dimension(0:,:), intent(in) :: mu

```

```

real(cp), dimension(:,0:), intent(in) :: mv ! (1:nx,0:ny)
real(cp), intent(in) :: rdx
real, dimension(size(f,1),size(f,2),size(dps,3)), intent(out) :: dlhs
character(len=*), parameter :: Mismatch = "mismatch in balance_lhs_tlm!"
real(cp), dimension(:,:,:), allocatable :: dps, t1, t2, t3
real(cp), dimension(:,:), allocatable :: c
integer :: nx, ny, nz, i, j, k
nx = size(dps,1) - 1
ny = size(dps,2) - 1
nz = size(dps,3)
! Sanity checks
if (any(shape(f) /= (/ nx, ny /) )) stop "f " // Mismatch
if (any(shape(m) /= (/ nx, ny /) )) stop "m " // Mismatch
if (any(shape(mu) /= (/ nx+1, ny /) )) stop "mu " // Mismatch
if (any(shape(mv) /= (/ nx, ny+1 /) )) stop "mv " // Mismatch
allocate(t1(2:nx-1,2:ny-1,nz), t2(2:nx-1,2:ny-1,nz), t3(2:nx-1,2:ny-1,nz))
allocate(dps(0:nx,0:ny,nz))
allocate(c(nx-1,ny-1))
dps = Zero
dps(0:nx,0:ny,:) = Zero
dlhs = Zero ! Really just setting the i/j boundaries.
forall (i = 1:nx-1, j = 1:ny-1)
  c(i,j) = One / (mCap(i,j) + mCap(i+1,j) + mCap(i,j+1) + mCap(i+1,j+1))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
  t1(i,j,k) = m(i,j)**2 * Eighth * rdx**2 * ( &
    dps(i-1,j-2,k) * mu(i-1,j-1) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
    dps(i, j-2,k) * mu(i, j-1) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
    dps(i-2,j-1,k) * mv(i-1,j-1) / mu(i-1,j) * (f(i-1,j) + f(i,j)) - &
    dps(i-1,j-1,k) * ( mv(i,j-1) / mu(i, j) * (f(i,j) + f(i+1,j)) + &
      (f(i-1,j) + f(i,j)) / mu(i-1,j) * (mv(i-1,j-1) - mv(i,j-1)) + &
      mu(i-1,j) / mv(i, j) * (f(i,j) + f(i,j+1)) + &
      (f(i,j-1) + f(i,j)) / mv(i,j-1) * (mu(i-1,j-1) - mu(i-1,j))) + &
    dps(i, j-1,k) * (-mv(i,j-1) / mu(i-1,j) * (f(i-1,j) + f(i,j)) + &
      (f(i,j) + f(i+1,j)) / mu(i, j) * (mv(i,j-1) - mv(i+1,j-1)) - &
      mu(i, j) / mv(i, j) * (f(i,j) + f(i,j+1)) + &
      (f(i,j-1) + f(i,j)) / mv(i,j-1) * (mu(i, j) - mu(i, j-1))) + &
    dps(i+1,j-1,k) * mv(i+1,j-1) / mu(i, j) * (f(i,j) + f(i+1,j)) + &
    dps(i-2,j, k) * mv(i-1, j) / mu(i-1,j) * (f(i-1,j) + f(i,j)) - &
    dps(i-1,j, k) * ( mv(i, j) / mu(i, j) * (f(i,j) + f(i+1,j)) + &
      (f(i-1,j) + f(i,j)) / mu(i-1,j) * (mv(i-1,j) - mv(i, j)) + &
      mu(i-1,j) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
      (f(i,j) + f(i,j+1)) / mv(i, j) * (mu(i-1,j+1) - mu(i-1,j))) + &
    dps(i, j, k) * (-mv(i, j) / mu(i-1,j) * (f(i-1,j) + f(i,j)) + &
      (f(i,j) + f(i+1,j)) / mu(i, j) * (mv(i, j) - mv(i+1,j)) - &
      mu(i, j) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
      (f(i,j) + f(i,j+1)) / mv(i, j) * (mu(i, j) - mu(i,j+1))) + &
    dps(i+1,j, k) * mv(i+1, j) / mu(i, j) * (f(i,j) + f(i+1,j)) + &
    dps(i-1,j+1,k) * mu(i-1,j+1) / mv(i, j) * (f(i,j) + f(i,j+1)) + &

```

```

    dpsi(i, j+1,k) * mu(i, j+1) / mv(i, j) * (f(i,j) + f(i,j+1)))
t2(i,j,k) = m(i,j) * Half * rdx**4 * ( &
    dpsi(i-1,j-2,k) * c(i-1,j-1) * mu(i-1,j-1) * mu(i-1,j) * ( &
    (psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) &
    * mv(i-1,j) + (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1) &
    + (psi(i-1,j-1,k)-psi(i-2,j-1,k))*mv(i-1,j-1)))
t2(i,j,k) = t2(i,j,k) - m(i,j) * Half * rdx**4 * ( &
    dpsi(i,j-2,k) * c(i,j-1) * mu(i,j-1) * mu(i,j) * ((psi(i+1,j,k) &
    - psi(i,j,k)) * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) + &
    (psi(i+1,j-1,k)-psi(i,j-1,k))*mv(i+1,j-1) + (psi(i,j-1,k) &
    - psi(i-1,j-1,k))*mv(i,j-1)))
t2(i,j,k) = t2(i,j,k) - m(i,j) * Half * rdx**4 * ( &
    dpsi(i-2,j-1,k) * mu(i-1,j) * ((c(i-1,j) * ((psi(i-1,j+1,k) &
    - psi(i-1,j,k))*mu(i-1,j+1) + (psi(i-1,j,k)-psi(i-1,j-1,k))*mu(i-1,j)) &
    - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) + &
    (psi(i-1,j-1,k)-psi(i-1,j-2,k))*mu(i-1,j-1))) * mv(i-1,j-1) &
    + mu(i-2,j)*mu(i-1,j)/mCap(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
    dpsi(i-1,j-1,k) * (mu(i,j) * ((c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) &
    * mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
    * ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
    * mu(i,j-1))) * mv(i,j-1) + mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i,j,k) &
    - psi(i,j-1,k))) - mu(i-1,j) * ((c(i-1,j) * mu(i-1,j) + c(i-1,j-1) &
    * (mu(i-1,j-1) - mu(i-1,j))) * ((psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) &
    + (psi(i-1,j,k)-psi(i-2,j,k)) * mv(i-1,j) + (psi(i,j-1,k) &
    - psi(i-1,j-1,k))*mv(i,j-1) + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) &
    * mv(i-1,j-1)) - (c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) &
    * mu(i-1,j+1) + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) &
    - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
    + (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) &
    * (mv(i-1,j-1) - mv(i,j-1)) - mu(i-1,j) * (mu(i,j)/mCap(i,j) &
    * (psi(i,j,k)-psi(i,j-1,k)) + mu(i-1,j) * (One/mCap(i,j) &
    - One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
    - mu(i-2,j)/mCap(i-1,j)*(psi(i-2,j,k)-psi(i-2,j-1,k))) - mu(i-1,j)**2 &
    * (One/mCap(i,j) - One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
    dpsi(i,j-1,k) * (mu(i,j) * ((c(i,j) * mu(i,j) + c(i,j-1) * (mu(i,j-1) &
    - mu(i,j))) * ((psi(i+1,j,k)-psi(i,j,k)) * mv(i+1,j) + (psi(i,j,k) &
    - psi(i-1,j,k)) * mv(i,j) + (psi(i+1,j-1,k)-psi(i,j-1,k))*mv(i+1,j-1) &
    + (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1)) - (c(i,j) * ((psi(i,j+1,k) &
    - psi(i,j,k))*mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) &
    - c(i,j-1)*((psi(i,j,k)-psi(i,j-1,k))*mu(i,j)+(psi(i,j-1,k) &
    - psi(i,j-2,k))*mu(i,j-1))) * (mv(i,j-1) - mv(i+1,j-1)) - mu(i,j) &
    * (mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k)-psi(i+1,j-1,k)) + mu(i,j) &
    * (One/mCap(i+1,j) - One/mCap(i,j)) * (psi(i,j,k)-psi(i,j-1,k)) &
    - mu(i-1,j)/mCap(i,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) - mu(i,j)**2 &
    * (One/mCap(i+1,j) - One/mCap(i,j)) * (psi(i,j,k)-psi(i,j-1,k))) &
    - mu(i-1,j) * ( -(c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) &
    * mu(i-1,j+1) + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) &
    - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &

```

```

+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i,j-1) &
- mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i+1,j-1,k) * mu(i,j) * ( -(c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) &
* mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
* ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
* mu(i,j-1))) * mv(i+1,j-1) - mu(i,j)*mu(i+1,j)/mCap(i+1,j) &
* (psi(i,j,k)-psi(i,j-1,k))))
t2(i,j,k) = t2(i,j,k) - m(i,j) * Half * rdx**4 * ( &
dpsi(i-2,j,k) * mu(i-1,j) * ((c(i-1,j) * ((psi(i-1,j+1,k) &
- psi(i-1,j,k)) * mu(i-1,j+1) + (psi(i-1,j,k)-psi(i-1,j-1,k)) &
* mu(i-1,j)) - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) &
* mu(i-1,j) + (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) &
* mv(i-1,j) - mu(i-2,j)*mu(i-1,j)/mCap(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-1,j,k) * (mu(i,j) * ((c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) &
* mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
* ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
* mu(i,j-1))) * mv(i,j) - mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i,j,k) &
-psi(i,j-1,k))) - mu(i-1,j) * (-(c(i-1,j) * (mu(i-1,j) - mu(i-1,j+1)) &
- c(i-1,j-1) * mu(i-1,j)) * ((psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) &
+ (psi(i-1,j,k)-psi(i-2,j,k)) * mv(i-1,j) + (psi(i,j-1,k) &
- psi(i-1,j-1,k)) * mv(i,j-1) + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) &
* mv(i-1,j-1)) - (c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) &
* mu(i-1,j+1) + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) &
- c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) &
* (mv(i-1,j) - mv(i,j)) + mu(i-1,j) * (mu(i,j)/mCap(i,j) &
* (psi(i,j,k)-psi(i,j-1,k)) + mu(i-1,j) &
* (One/mCap(i,j) - One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- mu(i-2,j)/mCap(i-1,j) * (psi(i-2,j,k)-psi(i-2,j-1,k))) &
+ mu(i-1,j)**2 * (One/mCap(i,j) - One/mCap(i-1,j)) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i,j,k) * (mu(i,j) * (-(c(i,j) * (mu(i,j)-mu(i,j+1)) &
- c(i,j-1)*mu(i,j)) * ((psi(i+1,j,k)-psi(i,j,k)) * mv(i+1,j) &
+ (psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) + (psi(i+1,j-1,k)-psi(i,j-1,k)) &
* mv(i+1,j-1) + (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1)) - (c(i,j) &
* ((psi(i,j+1,k)-psi(i,j,k))*mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k)) &
* mu(i,j)) - c(i,j-1) * ((psi(i,j,k)-psi(i,j-1,k)) * mu(i,j) &
+ (psi(i,j-1,k)-psi(i,j-2,k)) * mu(i,j-1))) * (mv(i,j) - mv(i+1,j)) &
+ mu(i,j) * (mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k)-psi(i+1,j-1,k)) &
+ mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) &
* (psi(i,j,k)-psi(i,j-1,k)) - mu(i-1,j)/mCap(i,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))) + mu(i,j)**2 &
* (One/mCap(i+1,j) - One/mCap(i,j)) &
* (psi(i,j,k)-psi(i,j-1,k))) - mu(i-1,j) &
* (mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- (c(i,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k))*mu(i-1,j+1)) &

```

```

+ (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) - c(i-1,j-1) &
* ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i,j))))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i+1,j,k) * mu(i,j) * (mu(i,j)*mu(i+1,j)/mCap(i+1,j) &
* (psi(i,j,k)-psi(i,j-1,k)) - (c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) &
* mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) &
- c(i,j-1) * ((psi(i,j,k)-psi(i,j-1,k)) * mu(i,j) &
+ (psi(i,j-1,k)-psi(i,j-2,k)) * mu(i,j-1))) * mv(i+1,j)))
t2(i,j,k) = t2(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-1,j+1,k) * c(i-1,j) * mu(i-1,j) * mu(i-1,j+1) &
* ((psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) &
* mv(i-1,j) + (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1) &
+ (psi(i-1,j-1,k)-psi(i-2,j-1,k))*mv(i-1,j-1)))
t2(i,j,k) = t2(i,j,k) - m(i,j) * Half * rdx**4 * ( &
dpsi(i,j+1,k) * c(i,j) * mu(i,j) * mu(i,j+1) &
* ((psi(i+1,j,k)-psi(i,j,k)) * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) &
* mv(i,j) + (psi(i+1,j-1,k)-psi(i,j-1,k)) * mv(i+1,j-1) &
+ (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1)))
t3(i,j,k) = m(i,j) * Half * rdx**4 * ( &
-dpsi(i-1,j-2,k) * mv(i,j-1) * (mu(i-1,j-1) * (c(i,j-1) &
* (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)))) &
- c(i-1,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) &
+ mv(i,j-2)/mCap(i,j-1) * mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))))
t3(i,j,k) = t3(i,j,k) - m(i,j) * Half * rdx**4 * ( &
dpsi(i,j-2,k) * mv(i,j-1) * (mu(i,j-1) * (c(i,j-1) * (mv(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k)) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)))) &
- c(i-1,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) &
- mv(i,j-2)/mCap(i,j-1) * mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-2,j-1,k) * c(i-1,j-1) * mv(i-1,j-1) * mv(i,j-1) &
* (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-1,j-1,k) * (mv(i,j) * (mu(i-1,j) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j)*(mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))+mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) + mv(i,j)*mv(i,j-1)/mCap(i,j) &
* (psi(i,j,k)-psi(i-1,j,k))) - mv(i,j-1) * (-mv(i,j-1) &
* (One/mCap(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k))) - One/mCap(i,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i,j-2)*(psi(i,j-2,k)-psi(i-1,j-2,k)))) - (mu(i-1,j-1)-mu(i-1,j)) &
* (c(i,j-1) * (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &

```

```

+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) + mv(i,j-1)**2 &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) * (One/mCap(i,j-1) - One/mCap(i,j)) &
+ (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) &
* (c(i,j-1) * mv(i,j-1) + c(i-1,j-1) * (mv(i-1,j-1) - mv(i,j-1))))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i,j-1,k) * mv(i,j) * (mu(i,j) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j)*(mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))+mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) - mv(i,j)*mv(i,j-1)/mCap(i,j) &
* (psi(i,j,k)-psi(i-1,j,k))) - mv(i,j-1) * (mv(i,j-1) &
* (One/mCap(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k))) - One/mCap(i,j-1) * (mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i,j-2) &
* (psi(i,j-2,k)-psi(i-1,j-2,k)))) - (mu(i,j-1) - mu(i,j)) * (c(i,j-1) &
* (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) &
- (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) * (c(i,j-1) &
* (mv(i,j-1) - mv(i+1,j-1)) - c(i-1,j-1) * mv(i,j-1) + m(i,j-1)**2 &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) * (One/mCap(i,j) - One/mCap(i,j-1))))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i+1,j-1,k) * c(i,j-1) * mv(i,j-1) * mv(i+1,j-1) &
* (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))))
t3(i,j,k) = t3(i,j,k) - m(i,j) * Half * rdx**4 * ( &
dpsi(i-2,j,k) * c(i-1,j) * mv(i-1,j) * mv(i,j) &
* (mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) &
+ mu(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) &
+ mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-1,j,k) * mv(i,j) * (-mu(i-1,j) - mu(i-1,j+1)) * (c(i,j) &
* (mv(i+1,j) * (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k))) - c(i-1,j) * (mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) - mv(i,j) * (One/mCap(i,j+1) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j+1) &
* (psi(i,j+1,k)-psi(i-1,j+1,k))) - One/mCap(i,j) * (mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &

```



```

* (psi(i,j-1,k)-psi(i-1,j-1,k)))) + mv(i,j)**2 &
* (psi(i,j,k)-psi(i-1,j,k)) * (One/mCap(i,j) - One/mCap(i,j+1)) &
- (mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
* (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * (-c(i,j) * mv(i,j) &
- c(i-1,j) * (mv(i-1,j) - mv(i,j)))) - mv(i,j-1) &
* (-mv(i,j-1)*mv(i,j)/mCap(i,j) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
- mu(i-1,j) * (c(i,j-1) * (mv(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k))) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k))))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i,j,k) * (mv(i,j) * (-mu(i,j) - mu(i,j+1)) * (c(i,j) &
* (mv(i+1,j) * (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k))) - c(i-1,j) * (mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) - (mu(i,j+1) &
* (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
* (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * (c(i,j) * (mv(i,j) &
- mv(i+1,j)) - c(i-1,j) * mv(i,j)) + mv(i,j) * (One/mCap(i,j+1) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j+1) &
* (psi(i,j+1,k)-psi(i-1,j+1,k))) - One/mCap(i,j) * (mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)))) - mv(i,j)**2 &
* (psi(i,j,k)-psi(i-1,j,k)) * (One/mCap(i,j) - One/mCap(i,j+1))) &
- mv(i,j-1) * (mv(i,j-1)*mv(i,j)/mCap(i,j) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) - mu(i,j) * (c(i,j-1) &
* (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) &
- c(i-1,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k))))))
t3(i,j,k) = t3(i,j,k) - m(i,j) * Half * rdx**4 * ( &
dpsi(i+1,j,k) * c(i,j) * mv(i,j) * mv(i+1,j) &
* (mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
* (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i-1,j+1,k) * mv(i,j) * (-mu(i-1,j+1) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)))) &
- mv(i,j)*mv(i,j+1)/mCap(i,j+1) * (psi(i,j,k)-psi(i-1,j,k))))
t3(i,j,k) = t3(i,j,k) + m(i,j) * Half * rdx**4 * ( &
dpsi(i,j+1,k) * mv(i,j) * (mv(i,j)*mv(i,j+1)/mCap(i,j+1) &
* (psi(i,j,k)-psi(i-1,j,k)) - mu(i,j+1) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j) * (mv(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k))))))

```

```

end forall
dlhs(2:nx-1,2:ny-1,:) = t1 - t2 - t3
deallocate(c, dpsi, t3, t2, t1)
end subroutine balance_lhs_tlm
! =====
subroutine balance_lhs_adj(t1, psi, f, mCap, m, mu, mv, rdx, ps)
  real(cp), dimension(:,:), intent(in) :: t1      ! (1:nx,1:ny)
  real(cp), dimension(0:,0:,:), intent(in) :: psi  ! (0:nx,0:ny)
  real(cp), dimension(:,:), intent(in) :: f, mCap, m ! (1:nx,1:ny)
  real(cp), dimension(0:,:), intent(in) :: mu
  real(cp), dimension(:,0:), intent(in) :: mv      ! (1:nx,0:ny)
  real(cp), intent(in) :: rdx
  real(cp), dimension(0:size(t1,1),0:size(t1,2),size(t1,3)), intent(out) :: ps
  character(len=*), parameter :: Mismatch = "mismatch in balance_lhs_adj!"
  real(cp), dimension(:,:), allocatable :: c, cc, dd
  integer :: nx, ny, nz, i, j, k
  nx = size(t1,1)
  ny = size(t1,2)
  nz = size(t1,3)
  ! Sanity checks
  if (any(shape(psi) /= (/ nx+1, ny+1, nz /))) stop "psi " // Mismatch
  if (any(shape(f) /= (/ nx, ny /))) stop "f " // Mismatch
  if (any(shape(mCap) /= (/ nx, ny /))) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /))) stop "m " // Mismatch
  if (any(shape(mu) /= (/ nx+1, ny /))) stop "mu " // Mismatch
  if (any(shape(mv) /= (/ nx, ny+1 /))) stop "mv " // Mismatch
  allocate(c(nx-1,ny-1), cc(nx,ny), dd(nx,ny))
  forall (i = 1:nx-1, j = 1:ny-1)
    c(i,j) = One / (mCap(i,j) + mCap(i+1,j) + mCap(i,j+1) + mCap(i+1,j+1))
  end forall
  ps = Zero
  cc = m**2 * Eighth * rdx**2
  dd = m * Half * rdx**4
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    ps(i-1,j-2,k) = ps(i-1,j-2,k) + cc(i,j) * &
      mu(i-1,j-1) / mv(i,j-1) * (f(i,j-1) + f(i, j)) * t1(i,j,k) &
    ps(i-1,j-2,k) = ps(i-1,j-2,k) - dd(i,j) * &
      c(i-1,j-1) * mu(i-1,j-1) * mu(i-1,j) * ((psi(i,j,k)-psi(i-1,j,k)) &
      * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) * mv(i-1,j) &
      + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1) &
      + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) * mv(i-1,j-1)) * t1(i,j,k) &
    ps(i-1,j-2,k) = ps(i-1,j-2,k) + dd(i,j) * &
      mv(i,j-1) * (mu(i-1,j-1) * (c(i,j-1) * (mv(i+1,j-1) &
      * (psi(i+1,j-1,k)-psi(i,j-1,k)) + mv(i,j-1) &
      * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
      * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
      + mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) &
      + mv(i,j-2)/mCap(i,j-1) * mv(i,j-1) &
      * (psi(i,j-1,k)-psi(i-1,j-1,k))) * t1(i,j,k) &
    ps(i ,j-2,k) = ps(i ,j-2,k) + cc(i,j) * &

```

```

    mu(i, j-1) / mv(i,j-1) * (f(i,j-1) + f(i, j)) * t1(i,j,k)
ps(i, j-2,k) = ps(i, j-2,k) + dd(i,j) * &
    c(i,j-1) * mu(i,j-1) * mu(i,j) * ((psi(i+1,j,k)-psi(i,j,k)) &
    * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) &
    + (psi(i+1,j-1,k)-psi(i,j-1,k)) * mv(i+1,j-1) &
    + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1)) * t1(i,j,k)
ps(i, j-2,k) = ps(i, j-2,k) + dd(i,j) * &
    mv(i,j-1) * (mu(i,j-1) * (c(i,j-1) * (mv(i+1,j-1) &
    * (psi(i+1,j-1,k)-psi(i,j-1,k)) + mv(i,j-1) &
    * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) * (mv(i,j-1) &
    * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i-1,j-1) &
    * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) - mv(i,j-2)/mCap(i,j-1) &
    * mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) * t1(i,j,k)
ps(i-2,j-1,k) = ps(i-2,j-1,k) + cc(i,j) * &
    mv(i-1,j-1) / mu(i-1,j) * (f(i-1,j) + f(i, j)) * t1(i,j,k)
ps(i-2,j-1,k) = ps(i-2,j-1,k) + dd(i,j) * &
    mu(i-1,j) * ((c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) * mu(i-1,j+1) &
    + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) - c(i-1,j-1) &
    * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
    + (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i-1,j-1) &
    + mu(i-2,j)*mu(i-1,j)/mCap(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) &
    * t1(i,j,k)
ps(i-2,j-1,k) = ps(i-2,j-1,k) - dd(i,j) * &
    c(i-1,j-1) * mv(i-1,j-1) * mv(i,j-1) &
    * (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
    + mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
    + mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
    + mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) * t1(i,j,k)
ps(i-1,j-1,k) = ps(i-1,j-1,k) - cc(i,j) * ( &
    mv(i, j-1) / mu(i, j) * (f(i,j) + f(i+1,j)) + &
    (f(i-1,j) + f(i,j)) / mu(i-1,j) * (mv(i-1,j-1) - mv(i,j-1)) + &
    mu(i-1, j) / mv(i, j) * (f(i,j) + f(i,j+1)) + &
    (f(i,j-1) + f(i,j)) / mv(i,j-1) * (mu(i-1,j-1) - mu(i-1,j)) ) &
    * t1(i,j,k)
ps(i-1,j-1,k) = ps(i-1,j-1,k) - dd(i,j) * &
    (mu(i,j) * ((c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) * mu(i,j+1) &
    + (psi(i,j,k)-psi(i,j-1,k)) * mu(i,j)) - c(i,j-1) &
    * ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
    * mu(i,j-1))) * mv(i,j-1) + mu(i-1,j)*mu(i,j)/mCap(i,j) &
    * (psi(i,j,k)-psi(i,j-1,k))) - mu(i-1,j) * ((c(i-1,j) * mu(i-1,j) &
    + c(i-1,j-1) * (mu(i-1,j-1) - mu(i-1,j))) &
    * ((psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) &
    * mv(i-1,j) + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1) &
    + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) * mv(i-1,j-1)) - (c(i-1,j) &
    * ((psi(i-1,j+1,k)-psi(i-1,j,k)) * mu(i-1,j+1) &
    + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) &
    - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
    + (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) &
    * (mv(i-1,j-1) - mv(i,j-1)) - mu(i-1,j) * (mu(i,j)/mCap(i,j) &
    * (psi(i,j,k)-psi(i,j-1,k)) + mu(i-1,j) * (One/mCap(i,j) &

```

```

- One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- mu(i-2,j)/mCap(i-1,j)*(psi(i-2,j,k)-psi(i-2,j-1,k))) - mu(i-1,j)**2 &
* (One/mCap(i,j) - One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k))) &
* t1(i,j,k)
ps(i-1,j-1,k) = ps(i-1,j-1,k) - dd(i,j) * &
(mv(i,j) * (mu(i-1,j) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j)*(mv(i,j)*(psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) + mv(i,j)*mv(i,j-1)/mCap(i,j) &
* (psi(i,j,k)-psi(i-1,j,k))) - mv(i,j-1) * (-mv(i,j-1) &
* (One/mCap(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k))) - One/mCap(i,j-1) * (mv(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k))) + mv(i,j-2) &
* (psi(i,j-2,k)-psi(i-1,j-2,k)))) - (mu(i-1,j-1) - mu(i-1,j)) &
* (c(i,j-1) * (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) + mv(i,j-1)**2 &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) * (One/mCap(i,j-1) - One/mCap(i,j)) &
+ (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) + mu(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k)) + mu(i,j-1) &
* (psi(i,j-1,k)-psi(i,j-2,k)) + mu(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-1,j-2,k))) * (c(i,j-1) * mv(i,j-1) &
+ c(i-1,j-1) * (mv(i-1,j-1) - mv(i,j-1)))) * t1(i,j,k)
ps(i ,j-1,k) = ps(i ,j-1,k) + cc(i,j) * ( &
-mv(i, j-1) / mu(i-1,j) * (f(i-1,j) + f(i,j)) + &
(f(i,j) + f(i+1,j)) / mu(i, j) * (mv(i,j-1) - mv(i+1,j-1)) - &
mu(i, j) / mv(i, j) * (f(i,j) + f(i,j+1)) + &
(f(i,j-1) + f(i,j)) / mv(i,j-1) * (mu(i, j) - mu(i, j-1)) ) &
* t1(i,j,k)
ps(i ,j-1,k) = ps(i ,j-1,k) - dd(i,j) * &
(mu(i,j) * ((c(i,j) * mu(i,j) + c(i,j-1) * (mu(i,j-1) - mu(i,j))) &
* ((psi(i+1,j,k)-psi(i,j,k)) * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) &
* mv(i,j) + (psi(i+1,j-1,k)-psi(i,j-1,k)) * mv(i+1,j-1) &
+ (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1)) - (c(i,j) &
* ((psi(i,j+1,k)-psi(i,j,k))*mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k)) &
* mu(i,j)) - c(i,j-1)*((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) &
+ (psi(i,j-1,k)-psi(i,j-2,k))*mu(i,j-1))) * (mv(i,j-1) - mv(i+1,j-1)) &
- mu(i,j) * (mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k)-psi(i+1,j-1,k)) &
+ mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) &
* (psi(i,j,k)-psi(i,j-1,k)) - mu(i-1,j)/mCap(i,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))) - mu(i,j)**2 * (One/mCap(i+1,j) &
- One/mCap(i,j)) * (psi(i,j,k)-psi(i,j-1,k))) - mu(i-1,j) * (-( &
c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) * mu(i-1,j+1) &
+ (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) - c(i-1,j-1) &
* ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i,j-1) &
- mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)))) &
* t1(i,j,k)
ps(i ,j-1,k) = ps(i ,j-1,k) - dd(i,j) * &

```

```

(mv(i,j) * (mu(i,j) * (c(i,j) * (mv(i+1,j)*(psi(i+1,j,k)-psi(i,j,k)) &
+ mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) - c(i-1,j)*(mv(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j)*(psi(i-1,j,k)-psi(i-2,j,k)))) &
- mv(i,j)*mv(i,j-1)/mCap(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
- mv(i,j-1) * (mv(i,j-1) * (One/mCap(i,j) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) - One/mCap(i,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + mv(i,j-2) &
* (psi(i,j-2,k)-psi(i-1,j-2,k)))) - (mu(i,j-1) - mu(i,j)) * (c(i,j-1) &
* (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) - c(i-1,j-1) &
* (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))) &
- (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) * (c(i,j-1) &
* (mv(i,j-1) - mv(i+1,j-1)) - c(i-1,j-1) * mv(i,j-1)) &
+ m(i,j-1)**2 * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
* (One/mCap(i,j) - One/mCap(i,j-1)))) * t1(i,j,k)
ps(i+1,j-1,k) = ps(i+1,j-1,k) + cc(i,j) * &
mv(i+1,j-1) / mu(i, j) * (f(i,j) + f(i+1,j)) * t1(i,j,k)
ps(i+1,j-1,k) = ps(i+1,j-1,k) - dd(i,j) * &
mu(i,j) * (-(c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) * mu(i,j+1) &
+ (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
* ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
* mu(i,j-1))) * mv(i+1,j-1) - mu(i,j)*mu(i+1,j)/mCap(i+1,j) &
* (psi(i,j,k)-psi(i,j-1,k))) * t1(i,j,k)
ps(i+1,j-1,k) = ps(i+1,j-1,k) - dd(i,j) * &
c(i,j-1) * mv(i,j-1) * mv(i+1,j-1) &
* (mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) * t1(i,j,k)
ps(i-2,j ,k) = ps(i-2,j ,k) + cc(i,j) * &
mv(i-1,j ) / mu(i-1,j) * (f(i-1,j) + f(i,j)) * t1(i,j,k)
ps(i-2,j ,k) = ps(i-2,j ,k) + dd(i,j) * &
mu(i-1,j) * ((c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) * mu(i-1,j+1) &
+ (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) - c(i-1,j-1) &
* ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i-1,j) &
- mu(i-2,j)*mu(i-1,j)/mCap(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) &
* t1(i,j,k)
ps(i-2,j ,k) = ps(i-2,j ,k) + dd(i,j) * &
c(i-1,j) * mv(i-1,j) * mv(i,j) * (mu(i,j+1) &
* (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
* (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * t1(i,j,k)
ps(i-1,j ,k) = ps(i-1,j ,k) - cc(i,j) * ( &
mv(i ,j ) / mu(i, j) * (f(i,j) + f(i+1,j)) + &

```

```

      (f(i-1,j) + f(i,j)) / mu(i-1,j) * (mv(i-1,j) - mv(i, j)) + &
      mu(i-1,j) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
      (f(i,j) + f(i,j+1)) / mv(i, j) * (mu(i-1,j+1) - mu(i-1,j)) ) &
      * t1(i,j,k)
ps(i-1,j ,k) = ps(i-1,j ,k) - dd(i,j) * &
      (mu(i,j) * ((c(i,j) * ((psi(i,j+1,k)-psi(i,j,k)) * mu(i,j+1) &
      + (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
      * ((psi(i,j,k)-psi(i,j-1,k))*mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
      * mu(i,j-1))) * mv(i,j) - mu(i-1,j)*mu(i,j)/mCap(i,j) &
      * (psi(i,j,k)-psi(i,j-1,k))) - mu(i-1,j) * (-c(i-1,j) * (mu(i-1,j) &
      - mu(i-1,j+1)) - c(i-1,j-1) * mu(i-1,j)) * ((psi(i,j,k)-psi(i-1,j,k)) &
      * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) * mv(i-1,j) &
      + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1) &
      + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) * mv(i-1,j-1)) &
      - (c(i-1,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k)) * mu(i-1,j+1) &
      + (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) &
      - c(i-1,j-1) * ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
      + (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * (mv(i-1,j) &
      - mv(i,j)) + mu(i-1,j) * (mu(i,j)/mCap(i,j) &
      * (psi(i,j,k)-psi(i,j-1,k)) + mu(i-1,j) * (One/mCap(i,j) &
      - One/mCap(i-1,j)) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
      - mu(i-2,j)/mCap(i-1,j) * (psi(i-2,j,k)-psi(i-2,j-1,k))) &
      + mu(i-1,j)**2 * (One/mCap(i,j) - One/mCap(i-1,j)) &
      * (psi(i-1,j,k)-psi(i-1,j-1,k)))) * t1(i,j,k)
ps(i-1,j ,k) = ps(i-1,j ,k) - dd(i,j) * &
      (mv(i,j) * (-mu(i-1,j) - mu(i-1,j+1)) * (c(i,j) * (mv(i+1,j) &
      * (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
      - c(i-1,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
      * (psi(i-1,j,k)-psi(i-2,j,k)))) - mv(i,j) * (One/mCap(i,j+1) &
      * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j+1) &
      * (psi(i,j+1,k)-psi(i-1,j+1,k))) - One/mCap(i,j) * (mv(i,j) &
      * (psi(i,j,k)-psi(i-1,j,k)) + mv(i,j-1) &
      * (psi(i,j-1,k)-psi(i-1,j-1,k)))) + mv(i,j)**2 &
      * (psi(i,j,k)-psi(i-1,j,k)) * (One/mCap(i,j) - One/mCap(i,j+1)) &
      - (mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
      * (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
      + mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * (-c(i,j) * mv(i,j) &
      - c(i-1,j) * (mv(i-1,j) - mv(i,j)))) - mv(i,j-1) &
      * (-mv(i,j-1)*mv(i,j)/mCap(i,j) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
      - mu(i-1,j) * (c(i,j-1) * (mv(i+1,j-1)*(psi(i+1,j-1,k)-psi(i,j-1,k)) &
      + mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) &
      - c(i-1,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
      + mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))))) * t1(i,j,k)
ps(i ,j ,k) = ps(i ,j ,k) + cc(i,j) * ( &
      -mv(i ,j ) / mu(i-1,j) * (f(i-1,j) + f(i,j)) + &
      (f(i,j) + f(i+1,j)) / mu(i, j) * (mv(i, j) - mv(i+1,j)) - &
      mu(i ,j ) / mv(i,j-1) * (f(i,j-1) + f(i,j)) + &
      (f(i,j) + f(i,j+1)) / mv(i, j) * (mu(i, j) - mu(i,j+1)) ) &
      * t1(i,j,k)
ps(i ,j ,k) = ps(i ,j ,k) - dd(i,j) * &

```

```

      (mu(i,j) * (-c(i,j) * (mu(i,j)-mu(i,j+1)) - c(i,j-1)*mu(i,j)) &
* ((psi(i+1,j,k)-psi(i,j,k)) * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) &
* mv(i,j) + (psi(i+1,j-1,k)-psi(i,j-1,k))*mv(i+1,j-1) &
+ (psi(i,j-1,k)-psi(i-1,j-1,k))*mv(i,j-1)) - (c(i,j) &
* ((psi(i,j+1,k)-psi(i,j,k))*mu(i,j+1) + (psi(i,j,k)-psi(i,j-1,k)) &
* mu(i,j)) - c(i,j-1) * ((psi(i,j,k)-psi(i,j-1,k)) * mu(i,j) &
+ (psi(i,j-1,k)-psi(i,j-2,k)) * mu(i,j-1))) * (mv(i,j) - mv(i+1,j)) &
+ mu(i,j) * (mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k)-psi(i+1,j-1,k)) &
+ mu(i,j) * (One/mCap(i+1,j) - One/mCap(i,j)) &
* (psi(i,j,k)-psi(i,j-1,k)) - mu(i-1,j)/mCap(i,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))) + mu(i,j)**2 * (One/mCap(i+1,j) &
- One/mCap(i,j)) * (psi(i,j,k)-psi(i,j-1,k)) - mu(i-1,j) &
* (mu(i-1,j)*mu(i,j)/mCap(i,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- (c(i,j) * ((psi(i-1,j+1,k)-psi(i-1,j,k))*mu(i-1,j+1) &
+ (psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j)) - c(i-1,j-1) &
* ((psi(i-1,j,k)-psi(i-1,j-1,k)) * mu(i-1,j) &
+ (psi(i-1,j-1,k)-psi(i-1,j-2,k)) * mu(i-1,j-1))) * mv(i,j))) &
* t1(i,j,k)
ps(i ,j ,k) = ps(i ,j ,k) - dd(i,j) * &
(mv(i,j) * (-mu(i,j) - mu(i,j+1)) * (c(i,j) * (mv(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
- c(i-1,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
* (psi(i-1,j,k)-psi(i-2,j,k)))) - (mu(i,j+1) &
* (psi(i,j+1,k)-psi(i,j,k)) + mu(i-1,j+1) &
* (psi(i-1,j+1,k)-psi(i-1,j,k)) + mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * (c(i,j) * (mv(i,j) &
- mv(i+1,j)) - c(i-1,j) * mv(i,j)) + mv(i,j) * (One/mCap(i,j+1) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i,j+1) * (psi(i,j+1,k)-psi(i-1,j+1,k))) &
- One/mCap(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)))) &
- mv(i,j)**2 * (psi(i,j,k)-psi(i-1,j,k)) &
* (One/mCap(i,j) - One/mCap(i,j+1))) - mv(i,j-1) &
* (mv(i,j-1)*mv(i,j)/mCap(i,j) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
- mu(i,j) * (c(i,j-1) * (mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) &
- c(i-1,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)))))) * t1(i,j,k)
ps(i+1,j ,k) = ps(i+1,j ,k) + cc(i,j) * &
mv(i+1,j ) / mu(i ,j ) * (f(i,j) + f(i+1,j)) * t1(i,j,k)
ps(i+1,j ,k) = ps(i+1,j ,k) - dd(i,j) * &
mu(i,j) * (mu(i,j)*mu(i+1,j)/mCap(i+1,j) * (psi(i,j,k)-psi(i,j-1,k)) &
- (c(i,j) * ((psi(i,j+1,k)-psi(i,j,k))*mu(i,j+1) &
+ (psi(i,j,k)-psi(i,j-1,k))*mu(i,j)) - c(i,j-1) &
* ((psi(i,j,k)-psi(i,j-1,k)) * mu(i,j) + (psi(i,j-1,k)-psi(i,j-2,k)) &
* mu(i,j-1))) * mv(i+1,j)) * t1(i,j,k)
ps(i+1,j ,k) = ps(i+1,j ,k) + dd(i,j) * &
c(i,j) * mv(i,j) * mv(i+1,j) * (mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) &
+ mu(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) &
+ mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &

```

```

      + mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k))) * t1(i,j,k)
ps(i-1,j+1,k) = ps(i-1,j+1,k) + cc(i,j) *                                &
      mu(i-1,j+1) / mv(i, j) * (f(i,j) + f(i,j+1)) * t1(i,j,k)
ps(i-1,j+1,k) = ps(i-1,j+1,k) - dd(i,j) *                                &
      c(i-1,j) * mu(i-1,j) * mu(i-1,j+1) * ((psi(i,j,k)-psi(i-1,j,k)) &
      * mv(i,j) + (psi(i-1,j,k)-psi(i-2,j,k)) * mv(i-1,j) &
      + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1) &
      + (psi(i-1,j-1,k)-psi(i-2,j-1,k)) * mv(i-1,j-1)) * t1(i,j,k)
ps(i-1,j+1,k) = ps(i-1,j+1,k) - dd(i,j) *                                &
      mv(i,j) * (-mu(i-1,j+1) * (c(i,j) * (mv(i+1,j) &
      * (psi(i+1,j,k)-psi(i,j,k)) + mv(i,j)*(psi(i,j,k)-psi(i-1,j,k))) &
      - c(i-1,j) * (mv(i,j)*(psi(i,j,k)-psi(i-1,j,k)) + mv(i-1,j) &
      * (psi(i-1,j,k)-psi(i-2,j,k)))) - mv(i,j)*mv(i,j+1)/mCap(i,j+1) &
      * (psi(i,j,k)-psi(i-1,j,k))) * t1(i,j,k)
ps(i, j+1,k) = ps(i, j+1,k) + cc(i,j) *                                &
      mu(i, j+1) / mv(i, j) * (f(i,j) + f(i,j+1)) * t1(i,j,k)
ps(i, j+1,k) = ps(i, j+1,k) + dd(i,j) *                                &
      c(i,j) * mu(i,j) * mu(i,j+1) * ((psi(i+1,j,k)-psi(i,j,k)) &
      * mv(i+1,j) + (psi(i,j,k)-psi(i-1,j,k)) * mv(i,j) &
      + (psi(i+1,j-1,k)-psi(i,j-1,k)) * mv(i+1,j-1) &
      + (psi(i,j-1,k)-psi(i-1,j-1,k)) * mv(i,j-1)) * t1(i,j,k)
ps(i, j+1,k) = ps(i, j+1,k) - dd(i,j) *                                &
      mv(i,j) * (mv(i,j)*mv(i,j+1)/mCap(i,j+1) * (psi(i,j,k)-psi(i-1,j,k)) &
      - mu(i,j+1) * (c(i,j) * (mv(i+1,j) * (psi(i+1,j,k)-psi(i,j,k)) &
      + mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
      - c(i-1,j) * (mv(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
      + mv(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)))))) * t1(i,j,k)
end forall
ps(/0,nx/),:,:) = Zero
ps(1:nx-1,/0,ny/),:) = Zero
deallocate(dd, cc, c)
end subroutine balance_lhs_adj
! =====
subroutine balance_rhs_simp(phi, mCap, m, etaw, etam, rdx, rhs)
  real(cp), dimension(:,:), intent(in) :: phi
  real(cp), dimension(:,:), intent(in) :: mCap, m
  real(cp), dimension(0:), intent(in) :: etaw
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: rdx
  real, dimension(size(phi,1),size(phi,2),size(etam)), intent(out) :: rhs
  character(len=*), parameter :: Mismatch = "mismatch in balance_rhs_simp!"
  real(cp), dimension(:,:), allocatable :: dpde, e1, e2
  integer :: nx, ny, nz, k
  nx = size(phi,1)
  ny = size(phi,2)
  nz = size(phi,3) - 1
  ! Sanity checks
  if (any(shape(mCap) /= (/ nx, ny /))) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /))) stop "m " // Mismatch
  if (size(etaw) /= nz+1) stop "etaw " // Mismatch

```



```

if (size(etam) /= nz) stop "etam " // Mismatch
allocate(dpde(nx,ny), e1(1:nx-1,2:ny-1), e2(2:nx-1,1:ny-1))
rhs = Zero
do k = 1, nz
  dpde = (phi(:, :, k) - phi(:, :, k-1)) / (etaw(k) - etaw(k-1))
  e1 = Half*(xdiff(phi(:, 2:ny-1, k-1), rdx)+xdiff(phi(:, 2:ny-1, k), rdx)) * &
    xavg(mCap(:, 2:ny-1)) - etam(k) * xdiff(mCap(:, 2:ny-1), rdx) * &
    xavg(dpde(:, 2:ny-1))
  e2 = Half*(ydiff(phi(2:nx-1, :, k-1), rdx)+ydiff(phi(2:nx-1, :, k), rdx)) * &
    yavg(mCap(2:nx-1, :)) - etam(k) * ydiff(mCap(2:nx-1, :), rdx) * &
    yavg(dpde(2:nx-1, :))
  rhs(2:nx-1, 2:ny-1, k) = m(2:nx-1, 2:ny-1)**2 * &
    (xdiff(e1, rdx) + ydiff(e2, rdx))
end do
deallocate(e2, e1, dpde)
end subroutine balance_rhs_simp
! =====
subroutine balance_rhs_simp_exp(phi, mCap, m, etaw, etam, rdx, rhs)
  real(cp), dimension(:, :, 0:), intent(in) :: phi
  real(cp), dimension(:, :), intent(in) :: mCap, m
  real(cp), dimension(0:), intent(in) :: etaw
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: rdx
  real, dimension(size(phi,1),size(phi,2),size(etam)), intent(out) :: rhs
  character(len=*) , parameter :: Mismatch = "mismatch in balance_rhs_simp_exp!"
  integer :: nx, ny, nz, i, j, k
  nx = size(phi,1)
  ny = size(phi,2)
  nz = size(phi,3) - 1
  ! Sanity checks
  if (any(shape(mCap) /= (/ nx, ny /)) ) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) ) stop "m " // Mismatch
  if (size(etaw) /= nz+1) stop "etaw " // Mismatch
  if (size(etam) /= nz) stop "etam " // Mismatch
  rhs = Zero
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    rhs(i,j,k) = m(i,j)**2 * Half * rdx**2 * (Half &
      * ((phi(i+1,j,k-1)-phi(i,j,k-1) + phi(i+1,j,k)-phi(i,j,k)) &
      * (mCap(i,j) + mCap(i+1,j)) - (phi(i,j,k-1)-phi(i-1,j,k-1) &
      + phi(i,j,k)-phi(i-1,j,k)) * (mCap(i-1,j) + mCap(i,j)) &
      + (phi(i,j+1,k-1)-phi(i,j,k-1) + phi(i,j+1,k)-phi(i,j,k)) &
      * (mCap(i,j) + mCap(i,j+1)) - (phi(i,j,k-1)-phi(i,j-1,k-1) &
      + phi(i,j,k)-phi(i,j-1,k)) * (mCap(i,j-1) + mCap(i,j))) + etam(k) &
      / (etaw(k) - etaw(k-1)) * (-mCap(i+1,j) - mCap(i,j)) &
      * (phi(i,j,k)-phi(i,j,k-1) + phi(i+1,j,k)-phi(i+1,j,k-1)) &
      + (mCap(i,j) - mCap(i-1,j)) * (phi(i-1,j,k)-phi(i-1,j,k-1) &
      + phi(i,j,k)-phi(i,j,k-1)) - (mCap(i,j+1) - mCap(i,j)) &
      * (phi(i,j,k)-phi(i,j,k-1) + phi(i,j+1,k)-phi(i,j+1,k-1)) &
      + (mCap(i,j) - mCap(i,j-1)) * (phi(i,j-1,k)-phi(i,j-1,k-1) &
      + phi(i,j,k)-phi(i,j,k-1)))

```

```

end forall
end subroutine balance_rhs_simp_exp
! =====
subroutine balance_rhs_simp_tlm(dph, mCap, m, etaw, etam, rdx, drhs)
  real(cp), dimension(:,:,:), intent(in) :: dph
  real(cp), dimension(:,:),   intent(in) :: mCap, m
  real(cp), dimension(0:),    intent(in) :: etaw
  real(cp), dimension(:),     intent(in) :: etam
  real(cp),                   intent(in) :: rdx
  real, dimension(size(dph,1),size(dph,2),size(etam)), intent(out) :: drhs
  character(len=*), parameter :: Mismatch = "mismatch in balance_rhs_simp_tlm!"
  real(cp), dimension(:,:,:), allocatable :: dphi
  integer :: nx, ny, nz, i, j, k
  nx = size(dph,1)
  ny = size(dph,2)
  nz = size(dph,3) - 1
  ! Sanity checks
  if (any(shape(mCap) /= (/ nx, ny /)) ) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) ) stop "m " // Mismatch
  if (size(etaw) /= nz+1) stop "etaw " // Mismatch
  if (size(etam) /= nz) stop "etam " // Mismatch
  allocate(dphi(nx,ny,0:nz))
  dphi = dph
  dphi((/1,nx/),:,:) = Zero
  dphi(2:nx-1,(/1,ny/),:) = Zero
  dphi(2:nx-1,2:ny-1,(/0,nz/)) = Zero
  drhs = Zero ! Really just setting the boundaries
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    drhs(i,j,k) = m(i,j)**2 * Half * rdx**2 * ( &
      dphi(i,j-1,k-1) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i,j-1) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i,j-1))) &
      + dphi(i-1,j,k-1) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i-1,j) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i-1,j))) &
      + dphi(i,j,k-1) * (etam(k) / (etaw(k) - etaw(k-1))) * (mCap(i,j-1) &
      + mCap(i-1,j) - Four * mCap(i,j) + mCap(i+1,j) + mCap(i,j+1)) &
      - Half * (mCap(i,j-1) + mCap(i-1,j) + Four * mCap(i,j) &
      + mCap(i+1,j) + mCap(i,j+1))) &
      + dphi(i+1,j,k-1) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i+1,j) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i+1,j))) &
      + dphi(i,j+1,k-1) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i,j+1) - mCap(i,j)) + Half * (mCap(i,j+1) + mCap(i,j))) &
      + dphi(i,j-1,k) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i,j) - mCap(i,j-1)) + Half * (mCap(i,j) + mCap(i,j-1))) &
      + dphi(i-1,j,k) * (etam(k) / (etaw(k) - etaw(k-1))) &
      * (mCap(i,j) - mCap(i-1,j)) + Half * (mCap(i,j) + mCap(i-1,j))) &
      + dphi(i,j,k) * (etam(k) / (etaw(k) - etaw(k-1))) * (Four * mCap(i,j) &
      - mCap(i,j-1) - mCap(i-1,j) - mCap(i+1,j) - mCap(i,j+1)) &
      - Half * (mCap(i,j-1) + mCap(i-1,j) + Four * mCap(i,j) &
      + mCap(i+1,j) + mCap(i,j+1))) &
      + dphi(i+1,j,k) * (etam(k) / (etaw(k) - etaw(k-1))) &

```

```

      * (mCap(i,j) - mCap(i+1,j)) + Half * (mCap(i,j) + mCap(i+1,j))) &
+ dphi(i,j+1,k) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j) - mCap(i,j+1)) + Half * (mCap(i,j) + mCap(i,j+1)))
end forall
deallocate(dphi)
end subroutine balance_rhs_simp_tlm
! =====
subroutine balance_rhs_simp_adj(adjrhs, mCap, m, etaw, etam, rdx, ph)
  real(cp), dimension(:,:,:), intent(in) :: adjrhs
  real(cp), dimension(:,:), intent(in) :: mCap, m
  real(cp), dimension(0:), intent(in) :: etaw
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: rdx
  real(cp), dimension(size(adjrhs,1),size(adjrhs,2),0:size(adjrhs,3)), intent(out) :: ph
  character(len=*) , parameter :: Mismatch = "mismatch in balance_rhs_simp_adj!"
  real(cp), dimension(:,:), allocatable :: c
  integer :: nx, ny, nz, i, j, k
  nx = size(adjrhs,1)
  ny = size(adjrhs,2)
  nz = size(adjrhs,3)
  ! Sanity checks
  if (any(shape(mCap) /= (/ nx, ny /)) ) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) ) stop "m " // Mismatch
  if (size(etaw) /= nz+1) stop "etaw " // Mismatch
  if (size(etam) /= nz) stop "etam " // Mismatch
  allocate(c(nx,ny))
  ph = Zero
  c = m**2 * Half * rdx**2
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    ph(i,j-1,k-1) = ph(i,j-1,k-1) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j-1) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i,j-1))) &
      * adjrhs(i,j,k)
    ph(i-1,j,k-1) = ph(i-1,j,k-1) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i-1,j) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i-1,j))) &
      * adjrhs(i,j,k)
    ph(i,j,k-1) = ph(i,j,k-1) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j-1) + mCap(i-1,j) - Four * mCap(i,j) + mCap(i+1,j) &
      + mCap(i,j+1)) - Half * (mCap(i,j-1) + mCap(i-1,j) + Four * mCap(i,j) &
      + mCap(i+1,j) + mCap(i,j+1))) * adjrhs(i,j,k)
    ph(i+1,j,k-1) = ph(i+1,j,k-1) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i+1,j) - mCap(i,j)) + Half * (mCap(i,j) + mCap(i+1,j))) &
      * adjrhs(i,j,k)
    ph(i,j+1,k-1) = ph(i,j+1,k-1) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j+1) - mCap(i,j)) + Half * (mCap(i,j+1) + mCap(i,j))) &
      * adjrhs(i,j,k)
    ph(i,j-1,k) = ph(i,j-1,k) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j) - mCap(i,j-1)) + Half * (mCap(i,j) + mCap(i,j-1))) &
      * adjrhs(i,j,k)
    ph(i-1,j,k) = ph(i-1,j,k) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j) - mCap(i-1,j)) + Half * (mCap(i,j) + mCap(i-1,j))) &

```

```

      * adjrhs(i,j,k)
      ph(i,j,k) = ph(i,j,k) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) * (Four &
      * mCap(i,j) - mCap(i,j-1) - mCap(i-1,j) - mCap(i+1,j) - mCap(i,j+1)) &
      - Half * (mCap(i,j-1) + mCap(i-1,j) + Four * mCap(i,j) + mCap(i+1,j) &
      + mCap(i,j+1))) * adjrhs(i,j,k)
      ph(i+1,j,k) = ph(i+1,j,k) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j) - mCap(i+1,j)) + Half * (mCap(i,j) + mCap(i+1,j))) &
      * adjrhs(i,j,k)
      ph(i,j+1,k) = ph(i,j+1,k) + c(i,j) * (etam(k) / (etaw(k) - etaw(k-1)) &
      * (mCap(i,j) - mCap(i,j+1)) + Half * (mCap(i,j) + mCap(i,j+1))) &
      * adjrhs(i,j,k)
    end forall
    ph((/1,nx/),:,:) = Zero
    ph(2:nx-1,(/1,ny/),:) = Zero
    ph(2:nx-1,2:ny-1,(/0,nz/)) = Zero
    deallocate(c)
  end subroutine balance_rhs_simp_adj
! =====
  subroutine imbalance_simp_adj(adjimb, psi, f, mCap, m, mu, mv, etaw, etam, rdx, &
    adjpsi, adjphi)
    real(cp), dimension(:,:,:), intent(in) :: adjimb
    real(cp), dimension(0:0,:,:), intent(in) :: psi
    real(cp), dimension(:,:), intent(in) :: f, mCap, m
    real(cp), dimension(0:,:), intent(in) :: mu
    real(cp), dimension(:,0:), intent(in) :: mv
    real(cp), dimension(0:), intent(in) :: etaw
    real(cp), dimension(:), intent(in) :: etam
    real(cp), intent(in) :: rdx
    real(cp), dimension(0:size(adjimb,1),0:size(adjimb,2),size(adjimb,3)), intent(out) :: &
      adjpsi
    real(cp), dimension(size(adjimb,1),size(adjimb,2),0:size(adjimb,3)), intent(out) :: &
      adjphi
    character(len=*), parameter :: Mismatch = "mismatch in balance_simp_adj!"
    integer :: nx, ny, nz
    nx = size(adjimb,1)
    ny = size(adjimb,2)
    nz = size(adjimb,3)
    ! Sanity checks
    if (any(shape(psi) /= (/ nx+1, ny+1, nz /) )) stop "psi " // Mismatch
    if (any(shape(f) /= (/ nx, ny /) )) stop "f " // Mismatch
    if (any(shape(mCap) /= (/ nx, ny /) )) stop "mCap " // Mismatch
    if (any(shape(m) /= (/ nx, ny /) )) stop "m " // Mismatch
    if (any(shape(mu) /= (/ nx+1, ny /) )) stop "mu " // Mismatch
    if (any(shape(mv) /= (/ nx, ny+1 /) )) stop "mv " // Mismatch
    if (size(etaw) /= nz+1) stop "etaw " // Mismatch
    if (size(etam) /= nz) stop "etam " // Mismatch
    call balance_lhs_adj(adjimb, psi, f, mCap, m, mu, mv, rdx, adjpsi)
    call balance_rhs_simp_adj(adjimb, mCap, m, etaw, etam, rdx, adjphi)
    adjphi = -adjphi
  end subroutine imbalance_simp_adj

```

```

! =====
subroutine imbalance_exp(psi, phi, f, mCap, m, mu, mv, etaw, etam, rdx, imb)
  real(cp), dimension(0:,0:,0:), intent(in) :: psi      ! (0:nx,0:ny)
  real(cp), dimension(:,0:), intent(in) :: phi
  real(cp), dimension(:,0:), intent(in) :: f, mCap, m    ! (1:nx,1:ny)
  real(cp), dimension(0:,0:), intent(in) :: mu
  real(cp), dimension(:,0:), intent(in) :: mv            ! (1:nx,0:ny)
  real(cp), dimension(0:), intent(in) :: etaw
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: rdx
  real(cp), dimension(size(f,1),size(f,2),size(psi,3)), intent(out) :: imb
  character(len=*), parameter :: Mismatch = "mismatch in imbalance_exp!"
  real(cp), dimension(:,0:), allocatable :: t1, t2, t3
  real(cp), dimension(:,0:), allocatable :: c
  integer :: nx, ny, nz, i, j, k
  nx = size(psi,1) - 1
  ny = size(psi,2) - 1
  nz = size(psi,3)
  ! Sanity checks
  if (any(shape(f) /= (/ nx, ny /)) stop "f " // Mismatch
  if (any(shape(mCap) /= (/ nx, ny /)) stop "mCap " // Mismatch
  if (any(shape(m) /= (/ nx, ny /)) stop "m " // Mismatch
  if (any(shape(mu) /= (/ nx+1, ny /)) stop "mu " // Mismatch
  if (any(shape(mv) /= (/ nx, ny+1 /)) stop "mv " // Mismatch
  if (any(shape(mCap) /= (/ nx, ny /)) stop "mCap " // Mismatch
  if (size(etaw) /= nz+1) stop "etaw " // Mismatch
  if (size(etam) /= nz) stop "etam " // Mismatch
  allocate(t1(2:nx-1,2:ny-1,nz), t2(2:nx-1,2:ny-1,nz), t3(2:nx-1,2:ny-1,nz))
  allocate(c(nx-1,ny-1))
  forall (i = 1:nx-1, j = 1:ny-1)
    c(i,j) = One / (mCap(i,j) + mCap(i+1,j) + mCap(i,j+1) + mCap(i+1,j+1))
  end forall
  imb = Zero
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    t1(i,j,k) = Eighth * rdx**2 * m(i,j)**2 * (
      (f(i, j) + f(i+1,j)) * (
        mv(i, j-1) * (psi(i, j-1,k) - psi(i-1,j-1,k)) +
        mv(i+1,j-1) * (psi(i+1,j-1,k) - psi(i, j-1,k)) +
        mv(i, j) * (psi(i, j, k) - psi(i-1,j, k)) +
        mv(i+1,j) * (psi(i+1,j, k) - psi(i, j, k)) ) / mu(i, j) -
      (f(i-1,j) + f(i,j)) * (
        mv(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-2,j-1,k)) +
        mv(i, j-1) * (psi(i, j-1,k) - psi(i-1,j-1,k)) +
        mv(i-1,j) * (psi(i-1,j, k) - psi(i-2,j, k)) +
        mv(i, j) * (psi(i, j, k) - psi(i-1,j, k)) ) / mu(i-1,j) +
      (f(i, j) + f(i,j+1)) * (
        mu(i-1,j) * (psi(i-1,j, k) - psi(i-1,j-1,k)) +
        mu(i, j) * (psi(i, j, k) - psi(i, j-1,k)) +
        mu(i-1,j+1) * (psi(i-1,j+1,k) - psi(i-1,j, k)) +
        mu(i, j+1) * (psi(i, j+1,k) - psi(i, j, k)) ) / mv(i, j) -

```

```

(f(i,j-1) + f(i,j )) * (
mu(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-1,j-2,k)) +
mu(i, j-1) * (psi(i, j-1,k) - psi(i, j-2,k)) +
mu(i-1,j ) * (psi(i-1,j, k) - psi(i-1,j-1,k)) +
mu(i, j ) * (psi(i, j, k) - psi(i, j-1,k)) ) / mv(i,j-1) )
t2(i,j,k) = Half * rdx**4 * m(i,j) * ( &
mu(i,j) * (mu(i,j) * (psi(i,j,k) - psi(i,j-1,k)) &
* (mu(i,j) * (One/mCap(i+1,j)-One/mCap(i,j)) &
* (psi(i,j,k) - psi(i,j-1,k)) &
+ mu(i+1,j)/mCap(i+1,j) * (psi(i+1,j,k) - psi(i+1,j-1,k)) &
- mu(i-1,j)/mCap(i,j) * (psi(i-1,j,k) - psi(i-1,j-1,k))) &
- (mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) &
+ mv(i+1,j-1) * (psi(i+1,j-1,k) - psi(i,j-1,k)) &
+ mv(i,j) * (psi(i,j,k) - psi(i-1,j,k)) + mv(i+1,j) &
* (psi(i+1,j,k) - psi(i,j,k))) &
* (c(i,j) * (mu(i,j+1) * (psi(i,j+1,k) - psi(i,j,k)) &
+ mu(i,j) * (psi(i,j,k) - psi(i,j-1,k))) - c(i,j-1) &
* (mu(i,j) * (psi(i,j,k) - psi(i,j-1,k)) &
+ mu(i,j-1) * (psi(i,j-1,k) - psi(i,j-2,k)))) &
- mu(i-1,j) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) &
* (mu(i-1,j) * (One/mCap(i,j)-One/mCap(i-1,j)) &
* (psi(i-1,j,k) - psi(i-1,j-1,k)) &
+ mu(i,j)/mCap(i,j) * (psi(i,j,k) - psi(i,j-1,k)) &
- mu(i-2,j)/mCap(i-1,j) * (psi(i-2,j,k) - psi(i-2,j-1,k))) &
- (mv(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-2,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k) - psi(i-1,j-1,k)) &
+ mv(i-1,j) * (psi(i-1,j,k) - psi(i-2,j,k)) &
+ mv(i,j) * (psi(i,j,k) - psi(i-1,j,k))) &
* (c(i-1,j) * (mu(i-1,j+1) * (psi(i-1,j+1,k) - psi(i-1,j,k)) &
+ mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k))) &
- c(i-1,j-1) * (mu(i-1,j) * (psi(i-1,j,k) - psi(i-1,j-1,k)) &
+ mu(i-1,j-1) * (psi(i-1,j-1,k) - psi(i-1,j-2,k))))))
t3(i,j,k) = Half * rdx**4 * m(i,j) * ( &
mv(i,j) * (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) * (One/mCap(i,j+1) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i,j+1) * (psi(i,j+1,k)-psi(i-1,j+1,k))) &
- One/mCap(i,j) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
- (mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
+ mu(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) &
+ mu(i,j+1) * (psi(i,j+1,k)-psi(i,j,k))) * (c(i,j) &
* (mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ mv(i+1,j) * (psi(i+1,j,k)-psi(i,j,k))) &
- c(i-1,j) * (mv(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ mv(i,j) * (psi(i,j,k)-psi(i-1,j,k)))) &
- mv(i,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
* (One/mCap(i,j) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i,j) * (psi(i,j,k)-psi(i-1,j,k))) &
- One/mCap(i,j-1) * (mv(i,j-2) * (psi(i,j-2,k)-psi(i-1,j-2,k)) &

```

```

+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))) &
- (mu(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k)) &
+ mu(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k)) &
+ mu(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
+ mu(i,j) * (psi(i,j,k)-psi(i,j-1,k))) &
* (c(i,j-1) * (mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ mv(i+1,j-1) * (psi(i+1,j-1,k)-psi(i,j-1,k))) &
- c(i-1,j-1) * (mv(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)) &
+ mv(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k))))))
end forall
imb(2:nx-1,2:ny-1,:) = t1 - t2 - t3
deallocate(c, t3, t2, t1)
forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
  imb(i,j,k) = imb(i,j,k) - m(i,j)**2 * Half * rdx**2 * (Half &
    * ((phi(i+1,j,k-1)-phi(i,j,k-1) + phi(i+1,j,k)-phi(i,j,k)) &
    * (mCap(i,j) + mCap(i+1,j)) - (phi(i,j,k-1)-phi(i-1,j,k-1) &
    + phi(i,j,k)-phi(i-1,j,k)) * (mCap(i-1,j) + mCap(i,j)) &
    + (phi(i,j+1,k-1)-phi(i,j,k-1) + phi(i,j+1,k)-phi(i,j,k)) &
    * (mCap(i,j) + mCap(i,j+1)) - (phi(i,j,k-1)-phi(i,j-1,k-1) &
    + phi(i,j,k)-phi(i,j-1,k)) * (mCap(i,j-1) + mCap(i,j))) &
    + etam(k) / (etaw(k) - etaw(k-1)) * (-mCap(i+1,j) - mCap(i,j)) &
    * (phi(i,j,k)-phi(i,j,k-1) + phi(i+1,j,k)-phi(i+1,j,k-1)) &
    + (mCap(i,j) - mCap(i-1,j)) * (phi(i-1,j,k)-phi(i-1,j,k-1) &
    + phi(i,j,k)-phi(i,j,k-1)) - (mCap(i,j+1) - mCap(i,j)) &
    * (phi(i,j,k)-phi(i,j,k-1) + phi(i,j+1,k)-phi(i,j+1,k-1)) &
    + (mCap(i,j) - mCap(i,j-1)) * (phi(i,j-1,k)-phi(i,j-1,k-1) &
    + phi(i,j,k)-phi(i,j,k-1))))
  end forall
end subroutine imbalance_exp
! =====
subroutine pv_exp(psi, phi, f, mCap, m, mu, mv, etaz, etam, pt, rdx, q)
  real(cp), dimension(0:0,:,:), intent(in) :: psi      ! (0:nx,0:ny)
  real(cp), dimension(:, :, 0:), intent(in) :: phi
  real(cp), dimension(:, :, ), intent(in) :: f, mCap, m  ! (1:nx,1:ny)
  real(cp), dimension(0:,:), intent(in) :: mu
  real(cp), dimension(:, 0:), intent(in) :: mv          ! (1:nx,0:ny)
  real(cp), dimension(0:), intent(in) :: etaz
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: pt, rdx
  real, dimension(size(f,1),size(f,2),size(psi,3)), intent(out) :: q
  real(cp), parameter :: p0 = 100000._cp
  character(len=*) , parameter :: Mismatch = "mismatch in pv_exp!"
  real(cp), dimension(:, :, :), allocatable :: e
  real(cp), dimension(:, :, ), allocatable :: mpsi, c, d
  integer :: nx, ny, nz, i, j, k
  nx = size(psi,1) - 1
  ny = size(psi,2) - 1
  nz = size(psi,3)
  ! Sanity checks
  if (any(shape(phi) /= (/ nx, ny, nz+1 /)) ) stop "phi " // Mismatch

```

```

if (any(shape(f) /= (/ nx, ny /)) stop "f " // Mismatch
if (any(shape(mCap) /= (/ nx, ny /)) stop "mCap " // Mismatch
if (any(shape(m) /= (/ nx, ny /)) stop "m " // Mismatch
if (any(shape(mu) /= (/ nx+1, ny /)) stop "mu " // Mismatch
if (any(shape(mv) /= (/ nx, ny+1 /)) stop "mv " // Mismatch
if (size(etaz) /= nz+1) stop "etaz " // Mismatch
if (size(etam) /= nz) stop "etam " // Mismatch
allocate(e(2:nx-1,2:ny-1,nz))
allocate(mpsi(nx-1,ny-1), c(nx-1,ny), d(nx,ny-1))
forall (i = 1:nx-1, j = 1:ny-1)
    mpsi(i,j) = Quarter * (mu(i,j) + mu(i,j+1) + mv(i,j) + mv(i+1,j))
end forall
forall (i = 1:nx-1, j = 1:ny)
    c(i,j) = mu(i,j) / (mCap(i,j) + mCap(i+1,j))
end forall
forall (i = 1:nx, j = 1:ny-1)
    d(i,j) = mv(i,j) / (mCap(i,j) + mCap(i,j+1))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    e(i,j,k) = (p0 / (pt + etam(k) * m(i,j)))**Kappa &
    / (RGas * log((pt + etaz(k-1) * m(i,j)) / (pt + etaz(k) * m(i,j))))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 2:nz-1)
    q(i,j,k) = Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) &
    * ((phi(i,j,k-1)-phi(i,j,k-2)) * e(i,j,k-1) &
    - (phi(i,j,k+1)-phi(i,j,k)) * e(i,j,k+1)) * (f(i,j) + Half * rdx**2 &
    * (mpsi(i-1,j-1) * (d(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
    - d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
    * (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
    * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
    * (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
    * (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
    - c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
    * (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
    + c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
    * (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
    * (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
    + c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
    * (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
    * ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
    - phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
    + phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
    - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
    * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
    + (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
    - phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
    + phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
    - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
    * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1))))))
end forall

```



```

q((/1,nx/),2:ny-1,2:nz-1) = q((/2,nx-1/),2:ny-1,2:nz-1)
q(:,(/1,ny/),2:nz-1) = q(:,(/2,ny-1/),2:nz-1)
q(:,:(/1,nz/)) = q(:,:(/2,nz-1/))
end subroutine pv_exp
! =====
subroutine pv_tlm(dps, dph, psi, phi, f, mCap, m, mu, mv, etaz, etam, pt, rdx, &
    dpv)
    real(cp), dimension(0:,0:,,:), intent(in) :: dps, psi          ! (0:nx,0:ny)
    real(cp), dimension(:, :, 0:), intent(in) :: dph, phi
    real(cp), dimension(:, :), intent(in) :: f, mCap, m          ! (1:nx,1:ny)
    real(cp), dimension(0:, :), intent(in) :: mu
    real(cp), dimension(:, 0:), intent(in) :: mv                ! (1:nx,0:ny)
    real(cp), dimension(0:), intent(in) :: etaz
    real(cp), dimension(:), intent(in) :: etam
    real(cp), intent(in) :: pt, rdx
    real, dimension(size(f,1),size(f,2),size(psi,3)), intent(out) :: dpv
    real(cp), parameter :: p0 = 100000._cp
    character(len=*), parameter :: Mismatch = "mismatch in pv_tlm!"
    real(cp), dimension(:, :, :), allocatable :: dpspsi, dphi, e
    real(cp), dimension(:, :), allocatable :: mpsi, c, d
    integer :: nx, ny, nz, i, j, k
    nx = size(psi,1) - 1
    ny = size(psi,2) - 1
    nz = size(psi,3)
    ! Sanity checks
    if (any(shape(dps) /= (/ nx+1, ny+1, nz /) )) stop "dps " // Mismatch
    if (any(shape(psi) /= (/ nx+1, ny+1, nz /) )) stop "psi " // Mismatch
    if (any(shape(dph) /= (/ nx, ny, nz+1 /) )) stop "dph " // Mismatch
    if (any(shape(phi) /= (/ nx, ny, nz+1 /) )) stop "phi " // Mismatch
    if (any(shape(f) /= (/ nx, ny /) )) stop "f " // Mismatch
    if (any(shape(mCap) /= (/ nx, ny /) )) stop "mCap " // Mismatch
    if (any(shape(m) /= (/ nx, ny /) )) stop "m " // Mismatch
    if (any(shape(mu) /= (/ nx+1, ny /) )) stop "mu " // Mismatch
    if (any(shape(mv) /= (/ nx, ny+1 /) )) stop "mv " // Mismatch
    if (size(etaz) /= nz+1) stop "etaz " // Mismatch
    if (size(etam) /= nz) stop "etam " // Mismatch
    allocate(dpspsi(0:nx,0:ny,nz), dphi(nx,ny,0:nz), e(2:nx-1,2:ny-1,nz))
    allocate(mspsi(nx-1,ny-1), c(nx-1,ny), d(nx,ny-1))
    dpspsi = dps
    dpspsi((/0,nx/),:,:) = Zero
    dpspsi(1:nx-1,(/0,ny/),:) = Zero
    dphi = dph
    dphi((/1,nx/),:,:) = Zero
    dphi(2:nx-1,(/1,ny/),:) = Zero
    dphi(2:nx-1,2:ny-1,(/0,nz/)) = Zero
    dpv = Zero ! Really just setting the boundaries.
    forall (i = 1:nx-1, j = 1:ny-1)
        mpsi(i,j) = Quarter * (mu(i,j) + mu(i,j+1) + mv(i,j) + mv(i+1,j))
    end forall
    forall (i = 1:nx-1, j = 1:ny)

```

```

      c(i,j) = mu(i,j) / (mCap(i,j) + mCap(i+1,j))
    end forall
  forall (i = 1:nx, j = 1:ny-1)
    d(i,j) = mv(i,j) / (mCap(i,j) + mCap(i,j+1))
  end forall
  forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    e(i,j,k) = (p0 / (pt + etam(k) * m(i,j)))**Kappa &
      / (RGas * log((pt + etaz(k-1) * m(i,j)) / (pt + etaz(k) * m(i,j))))
  end forall
  forall (i = 2:nx-1, j = 2:ny-1, k = 2:nz-1)
    dpv(i,j,k) = Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      -dphi(i,j,k-2) * e(i,j,k-1) * (f(i,j) + Half * rdx**2 &
        * (mpsi(i-1,j-1) * (d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) &
          - d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
            * (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
              * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
                * (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
                  * (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
                    - c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
                      * (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
                        + c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
                          * (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
                            * (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
                              + c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
                                * (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
                                  * ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
                                    - phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
                                      + phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j,k+1) &
                                        - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
                                          * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
                                            + (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
                                              - phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
                                                + phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
                                                  - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
                                                    * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1))))))
    dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      dphi(i,j-1,k-1) * Eighth * rdx**2 * mv(i,j-1) &
        / (phi(i,j,k)-phi(i,j,k-1)) &
          * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
            - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) &
              * (-psi(i,j,k-1) + psi(i,j-1,k-1) + psi(i,j,k+1) - psi(i,j-1,k+1)) &
                + c(i-1,j) * (-psi(i-1,j,k-1) + psi(i-1,j-1,k-1) + psi(i-1,j,k+1) &
                  - psi(i-1,j-1,k+1))))
    dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      dphi(i-1,j,k-1) * Eighth * rdx**2 * mu(i-1,j) &
        / (phi(i,j,k)-phi(i,j,k-1)) &
          * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
            - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) &
              * (-psi(i,j,k-1) + psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) &
                + d(i,j-1) * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) + psi(i,j-1,k+1) &
                  - psi(i-1,j-1,k+1))))
  end forall

```

```

- psi(i-1,j-1,k+1)))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dphi(i,j,k-1) * (e(i,j,k-1) * (f(i,j) + Half * rdx**2 &
    * (mpsi(i-1,j-1) * (d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) &
      - d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
        * (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
          * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
            * (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
              * (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
                - c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
                  * (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
                    + c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
                      * (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
                        * (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
                          + c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
                            * (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
                              * ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
                                - phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
                                  + phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
                                    - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
                                      * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
                                        + (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
                                          - phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
                                            + phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
                                              - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
                                                * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)))) &
                                                  + Half * rdx**2 * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
                                                    - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (-Quarter &
                                                      / (phi(i,j,k)-phi(i,j,k-1))**2 * ((mu(i,j) * (phi(i+1,j,k-1) &
                                                        - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
                                                          * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
                                                            * (d(i,j) * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) &
                                                              + psi(i-1,j,k-1)) + d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) &
                                                                - psi(i,j-1,k-1) + psi(i-1,j-1,k-1))) + (mv(i,j) * (phi(i,j+1,k-1) &
                                                                  - phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
                                                                    * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
                                                                      * (c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) &
                                                                        + psi(i,j-1,k-1)) + c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
                                                                          - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)))) - Quarter &
                                                                          / (phi(i,j,k)-phi(i,j,k-1)) * ((mu(i-1,j) - mu(i,j)) * (d(i,j) &
                                                                            * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1)) &
                                                                              + d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) &
                                                                                + psi(i-1,j-1,k-1))) + (mv(i,j-1) - mv(i,j)) * (c(i,j) &
                                                                                  * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
                                                                                    + c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
                                                                                      + psi(i-1,j-1,k-1))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  -dphi(i+1,j,k-1) * Eighth * rdx**2 * mu(i,j) &
  / (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
    * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &

```

```

      * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
      + psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) + d(i,j-1) &
      * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) + psi(i,j-1,k+1) &
      - psi(i-1,j-1,k+1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      -dphi(i,j+1,k-1) * Eighth * rdx**2 * mv(i,j) &
      / (phi(i,j,k)-phi(i,j,k-1)) &
      * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
      - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (psi(i,j,k+1) &
      - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) + c(i-1,j) &
      * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
      + psi(i-1,j-1,k-1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      dphi(i,j-1,k) * Eighth * rdx**2 * mv(i,j-1) &
      / (phi(i,j,k)-phi(i,j,k-1)) &
      * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
      - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (-psi(i,j,k-1) &
      + psi(i,j-1,k-1) + psi(i,j,k+1) - psi(i,j-1,k+1)) + c(i-1,j) &
      * (-psi(i-1,j,k-1) + psi(i-1,j-1,k-1) + psi(i-1,j,k+1) &
      - psi(i-1,j-1,k+1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      dphi(i-1,j,k) * Eighth * rdx**2 * mu(i-1,j) &
      / (phi(i,j,k)-phi(i,j,k-1)) &
      * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
      - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
      + psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) + d(i,j-1) &
      * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) + psi(i,j-1,k+1) &
      - psi(i-1,j-1,k+1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
      dphi(i,j,k) * (e(i,j,k+1) * (f(i,j) + Half * rdx**2 * (mpsi(i-1,j-1) &
      * (d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) - d(i-1,j-1) &
      * (psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
      * (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
      * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
      * (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
      * (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
      - c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
      * (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
      + c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
      * (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
      * (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
      + c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
      * (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
      * ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
      - phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
      + phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
      - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
      * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
      + (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
      - phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &

```

```

+ phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)))) &
+ Half * rdx**2 * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (Quarter &
/ (phi(i,j,k)-phi(i,j,k-1))**2 * ((mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
* (d(i,j) * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) &
+ psi(i-1,j,k-1)) + d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) &
- psi(i,j-1,k-1) + psi(i-1,j-1,k-1))) + (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
* (c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) &
+ psi(i,j-1,k-1)) + c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
- psi(i-1,j,k-1) + psi(i-1,j-1,k-1)))) - Quarter &
/ (phi(i,j,k)-phi(i,j,k-1)) * ((mu(i-1,j) - mu(i,j)) * (d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1)) &
+ d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) &
+ psi(i-1,j-1,k-1))) + (mv(i,j-1) - mv(i,j)) * (c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
+ c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
+ psi(i-1,j-1,k-1))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dphi(i+1,j,k) * Eighth * rdx**2 * mu(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
+ psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) + d(i,j-1) &
* (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) + psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dphi(i,j+1,k) * Eighth * rdx**2 * mv(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (psi(i,j,k+1) &
- psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) + c(i-1,j) &
* (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
+ psi(i-1,j-1,k-1))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dphi(i,j,k+1) * e(i,j,k+1) * (f(i,j) + Half * rdx**2 &
* (mpsi(i-1,j-1) * (d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) &
- d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
- c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &

```

```

* (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
* (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
* ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
+ phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
+ (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
- phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
+ phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dpsi(i-1,j-1,k-1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j-1) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
+ c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dpsi(i,j-1,k-1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
- d(i,j-1) * (mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) &
- phi(i,j,k)) + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) &
+ phi(i,j,k) - phi(i-1,j,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dpsi(i-1,j,k-1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
- c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
- dpsi(i,j,k-1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (-d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
- c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k))))))

```

```

dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-1,j-2,k) * Half * rdx**2 * mpsi(i-1,j-1) * c(i-1,j-1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i,j-2,k) * Half * rdx**2 * mpsi(i,j-1) * c(i,j-1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-2,j-1,k) * Half * rdx**2 * mpsi(i-1,j-1) * d(i-1,j-1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-1,j-1,k) * Half * rdx**2 * &
  (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j-1) * d(i,j-1) &
  + mpsi(i-1,j-1) * (-d(i,j-1) - d(i-1,j-1) - c(i-1,j) - c(i-1,j-1)) &
  + mpsi(i-1,j) * c(i-1,j)))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i,j-1,k) * Half * rdx**2 * &
  (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j-1) * (-d(i+1,j-1) - d(i,j-1) &
  - c(i,j) - c(i,j-1)) + mpsi(i-1,j-1) * d(i,j-1) + mpsi(i,j) * c(i,j)))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i+1,j-1,k) * Half * rdx**2 * mpsi(i,j-1) * d(i+1,j-1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-2,j,k) * Half * rdx**2 * mpsi(i-1,j) * d(i-1,j) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-1,j,k) * Half * rdx**2 * (e(i,j,k-1) &
  * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j) * d(i,j) + mpsi(i-1,j) &
  * (-d(i,j) - d(i-1,j) - c(i-1,j+1) - c(i-1,j)) &
  + mpsi(i-1,j-1) * c(i-1,j)))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i,j,k) * Half * rdx**2 * (e(i,j,k-1) &
  * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j) * (-d(i+1,j) - d(i,j) &
  - c(i,j+1) - c(i,j)) + mpsi(i-1,j) * d(i,j) + mpsi(i,j-1) * c(i,j)))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i+1,j,k) * Half * rdx**2 * mpsi(i,j) * d(i+1,j) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dpsi(i-1,j+1,k) * Half * rdx**2 * mpsi(i-1,j) * c(i-1,j+1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))

```

```

dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  dps(i,j+1,k) * Half * rdx**2 * mpsi(i,j) * c(i,j+1) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
  - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  -dps(i,i-1,j-1,k+1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (-d(i,j-1) * (mu(i,j) &
  * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) &
  + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
  - phi(i-1,j,k))) - c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) &
  - phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
  * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  -dps(i,i,j-1,k+1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j-1) * (mu(i,j) &
  * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) &
  + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
  - phi(i-1,j,k))) - c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
  + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) &
  - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  -dps(i,i-1,j,k+1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) &
  - phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
  * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
  - d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) &
  - phi(i,j,k)) + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) &
  + phi(i,j,k) - phi(i-1,j,k))))))
dpv(i,j,k) = dpv(i,j,k) + Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1))) * ( &
  -dps(i,i,j,k+1) * Eighth * rdx**2 / (phi(i,j,k)-phi(i,j,k-1)) &
  * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
  * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) &
  - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
  * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
  + c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
  - phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
  + phi(i,j,k) - phi(i,j-1,k))))))
end forall
deallocate(d, c, mpsi, e, dphi, dps)
end subroutine pv_tlm
! =====
subroutine pv_adj(adjpv, psi, phi, f, mCap, m, mu, mv, etaz, etam, pt, rdx, ps, &
  ph)
real(cp), dimension(:,:,:), intent(in) :: adjpv
real(cp), dimension(0:,0:,:), intent(in) :: psi ! (0:nx,0:ny)
real(cp), dimension(:,:,0:), intent(in) :: phi
real(cp), dimension(:,:), intent(in) :: f, mCap, m ! (1:nx,1:ny)

```



```

real(cp), dimension(0:,:), intent(in) :: mu
real(cp), dimension(:,0:), intent(in) :: mv ! (1:nx,0:ny)
real(cp), dimension(0:), intent(in) :: etaz
real(cp), dimension(:,), intent(in) :: etam
real(cp), intent(in) :: pt, rdx
real(cp), dimension(0:size(phi,1),0:size(phi,2),size(psi,3)), intent(out) :: ps
real(cp), dimension(size(phi,1),size(phi,2),0:size(psi,3)), intent(out) :: ph
real(cp), parameter :: p0 = 100000._cp
character(len=*), parameter :: Mismatch = "mismatch in pv_adj!"
real(cp), dimension(:,:,:), allocatable :: b, e
real(cp), dimension(:,:), allocatable :: mpsi, c, d
integer :: nx, ny, nz, i, j, k
nx = size(psi,1) - 1
ny = size(psi,2) - 1
nz = size(psi,3)
! Sanity checks
if (any(shape(adjpv) /= (/ nx, ny, nz /)) stop "adjpv " // Mismatch
if (any(shape(psi) /= (/ nx+1, ny+1, nz /)) stop "psi " // Mismatch
if (any(shape(phi) /= (/ nx, ny, nz+1 /)) stop "phi " // Mismatch
if (any(shape(f) /= (/ nx, ny /)) stop "f " // Mismatch
if (any(shape(mCap) /= (/ nx, ny /)) stop "mCap " // Mismatch
if (any(shape(m) /= (/ nx, ny /)) stop "m " // Mismatch
if (any(shape(mu) /= (/ nx+1, ny /)) stop "mu " // Mismatch
if (any(shape(mv) /= (/ nx, ny+1 /)) stop "mv " // Mismatch
if (size(etaz) /= nz+1) stop "etaz " // Mismatch
if (size(etam) /= nz) stop "etam " // Mismatch
allocate(b(2:nx-1,2:ny-1,2:nz-1), e(2:nx-1,2:ny-1,nz))
allocate(mpsi(nx-1,ny-1), c(nx-1,ny), d(nx,ny-1))
ps = 0.
ph = 0.
forall (i = 1:nx-1, j = 1:ny-1)
    mpsi(i,j) = Quarter * (mu(i,j) + mu(i,j+1) + mv(i,j) + mv(i+1,j))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 2:nz-1)
    b(i,j,k) = Half * Grav / (m(i,j) * (etaz(k)-etaz(k-1)))
end forall
forall (i = 1:nx-1, j = 1:ny)
    c(i,j) = mu(i,j) / (mCap(i,j) + mCap(i+1,j))
end forall
forall (i = 1:nx, j = 1:ny-1)
    d(i,j) = mv(i,j) / (mCap(i,j) + mCap(i,j+1))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 1:nz)
    e(i,j,k) = (p0 / (pt + etam(k) * m(i,j)))**Kappa &
    / (RGas * log((pt + etaz(k-1) * m(i,j)) / (pt + etaz(k) * m(i,j))))
end forall
forall (i = 2:nx-1, j = 2:ny-1, k = 2:nz-1)
    ph(i,j,k-2) = ph(i,j,k-2) - b(i,j,k) * (e(i,j,k-1) * (f(i,j) + Half &
    * rdx**2 * (mpsi(i-1,j-1) * (d(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
    - d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &

```

```

* (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- c(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) &
* (d(i+1,j-1)*(psi(i+1,j-1,k)-psi(i,j-1,k)) &
- d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) &
+ c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
- c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) &
+ mpsi(i-1,j)*(d(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
- d(i-1,j)*(psi(i-1,j,k)-psi(i-2,j,k)) &
+ c(i-1,j+1)*(psi(i-1,j+1,k)-psi(i-1,j,k)) &
- c(i-1,j)*(psi(i-1,j,k)-psi(i-1,j-1,k))) &
+ mpsi(i,j) * (d(i+1,j)*(psi(i+1,j,k)-psi(i,j,k)) &
- d(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
+ c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) &
- c(i,j) * (psi(i,j,k)-psi(i,j-1,k))) &
- Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
* ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
+ phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) &
+ d(i,j) * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) &
+ psi(i-1,j,k-1))) + (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) &
* (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
+ psi(i-1,j-1,k-1)) + c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) &
- psi(i,j,k-1) + psi(i,j-1,k-1)))))) * adjpv(i,j,k)
ph(i,j-1,k-1) = ph(i,j-1,k-1) + b(i,j,k) * (Eighth * rdx**2 * mv(i,j-1) &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (-psi(i,j,k-1) &
+ psi(i,j-1,k-1) + psi(i,j,k+1) - psi(i,j-1,k+1)) + c(i-1,j) &
* (-psi(i-1,j,k-1) + psi(i-1,j-1,k-1) + psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1)))) * adjpv(i,j,k)
ph(i-1,j,k-1) = ph(i-1,j,k-1) + b(i,j,k) * (Eighth * rdx**2 * mu(i-1,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
+ psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) + d(i,j-1) &
* (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) + psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1)))) * adjpv(i,j,k)
ph(i,j,k-1) = ph(i,j,k-1) + b(i,j,k) * (e(i,j,k-1) * (f(i,j) + Half &
* rdx**2 * (mpsi(i-1,j-1) * (d(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
- d(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k)) &
+ c(i-1,j) * (psi(i-1,j,k)-psi(i-1,j-1,k)) &
- c(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-1,j-2,k))) &
+ mpsi(i,j-1) * (d(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k)) &
- d(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) &
* (psi(i,j,k)-psi(i,j-1,k)) &
- c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) &

```

```

* (d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
- d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ c(i-1,j+1)*(psi(i-1,j+1,k)-psi(i-1,j,k)) &
- c(i-1,j)*(psi(i-1,j,k)-psi(i-1,j-1,k))) &
+ mpsi(i,j) * (d(i+1,j)*(psi(i+1,j,k)-psi(i,j,k)) &
- d(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
+ c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) &
- c(i,j) * (psi(i,j,k)-psi(i,j-1,k))) &
- Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
* ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
+ phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
+ (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
- phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
+ phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)))) &
+ Half * rdx**2 * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (-Quarter &
/ (phi(i,j,k)-phi(i,j,k-1))**2 * ((mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
* (d(i,j) * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) &
+ psi(i-1,j,k-1)) + d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) &
- psi(i,j-1,k-1) + psi(i-1,j-1,k-1))) + (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
* (c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) &
+ psi(i,j-1,k-1)) + c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
- psi(i-1,j,k-1) + psi(i-1,j-1,k-1)))) - Quarter &
/ (phi(i,j,k)-phi(i,j,k-1)) * ((mu(i-1,j) - mu(i,j)) * (d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1)) &
+ d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) &
+ psi(i-1,j-1,k-1))) + (mv(i,j-1) - mv(i,j)) * (c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
+ c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
+ psi(i-1,j-1,k-1)))))) * adjpv(i,j,k)
ph(i+1,j,k-1) = ph(i+1,j,k-1) - b(i,j,k) * (Eighth * rdx**2 * mu(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
+ psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) &
+ d(i,j-1) * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) &
+ psi(i,j-1,k+1) - psi(i-1,j-1,k+1)))) * adjpv(i,j,k)
ph(i,j+1,k-1) = ph(i,j+1,k-1) - b(i,j,k) * (Eighth * rdx**2 * mv(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (psi(i,j,k+1) &

```

```

- psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
+ c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
- psi(i-1,j,k-1) + psi(i-1,j-1,k-1))) * adjpv(i,j,k)
ph(i,j-1,k) = ph(i,j-1,k) + b(i,j,k) * (Eighth * rdx**2 * mv(i,j-1) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (-psi(i,j,k-1) &
+ psi(i,j-1,k-1) + psi(i,j,k+1) - psi(i,j-1,k+1)) &
+ c(i-1,j) * (-psi(i-1,j,k-1) + psi(i-1,j-1,k-1) &
+ psi(i-1,j,k+1) - psi(i-1,j-1,k+1))) * adjpv(i,j,k)
ph(i-1,j,k) = ph(i-1,j,k) + b(i,j,k) * (Eighth * rdx**2 * mu(i-1,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
+ psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) &
+ d(i,j-1) * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) &
+ psi(i,j-1,k+1) - psi(i-1,j-1,k+1))) * adjpv(i,j,k)
ph(i,j,k) = ph(i,j,k) + b(i,j,k) * (e(i,j,k+1) * (f(i,j) + Half * rdx**2 &
* (mpsi(i-1,j-1) * (d(i,j-1)*(psi(i,j-1,k)-psi(i-1,j-1,k)) &
- d(i-1,j-1)*(psi(i-1,j-1,k)-psi(i-2,j-1,k)) + c(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
- c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ c(i-1,j+1) * (psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) - d(i,j) * (psi(i,j,k)-psi(i-1,j,k)) &
+ c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
* (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
* ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
+ phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
+ (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
- phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
+ phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) + c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)))) &
+ Half * rdx**2 * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (Quarter &
/ (phi(i,j,k)-phi(i,j,k-1))**2 * ((mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
* (d(i,j) * (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) &
+ psi(i-1,j,k-1)) + d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) &
- psi(i,j-1,k-1) + psi(i-1,j-1,k-1))) + (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &

```

```

* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
* (c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) &
+ psi(i,j-1,k-1)) + c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
- psi(i-1,j,k-1) + psi(i-1,j-1,k-1)))) - Quarter &
/ (phi(i,j,k)-phi(i,j,k-1)) * ((mu(i-1,j) - mu(i,j)) * (d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1)) &
+ d(i,j-1) * (psi(i,j-1,k+1) - psi(i-1,j-1,k+1) - psi(i,j-1,k-1) &
+ psi(i-1,j-1,k-1))) + (mv(i,j-1) - mv(i,j)) * (c(i,j) &
* (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
+ c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) - psi(i-1,j,k-1) &
+ psi(i-1,j-1,k-1)))))) * adjpv(i,j,k)
ph(i+1,j,k) = ph(i+1,j,k) - b(i,j,k) * (Eighth * rdx**2 * mu(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (-psi(i,j,k-1) &
+ psi(i-1,j,k-1) + psi(i,j,k+1) - psi(i-1,j,k+1)) &
+ d(i,j-1) * (-psi(i,j-1,k-1) + psi(i-1,j-1,k-1) &
+ psi(i,j-1,k+1) - psi(i-1,j-1,k+1)))) * adjpv(i,j,k)
ph(i,j+1,k) = ph(i,j+1,k) - b(i,j,k) * (Eighth * rdx**2 * mv(i,j) &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (psi(i,j,k+1) &
- psi(i,j-1,k+1) - psi(i,j,k-1) + psi(i,j-1,k-1)) &
+ c(i-1,j) * (psi(i-1,j,k+1) - psi(i-1,j-1,k+1) &
- psi(i-1,j,k-1) + psi(i-1,j-1,k-1)))) * adjpv(i,j,k)
ph(i,j,k+1) = ph(i,j,k+1) - b(i,j,k) * (e(i,j,k+1) * (f(i,j) + Half &
* rdx**2 * (mpsi(i-1,j-1) * (d(i,j-1) * (psi(i,j-1,k)-psi(i-1,j-1,k)) &
- d(i-1,j-1) * (psi(i-1,j-1,k)-psi(i-2,j-1,k))) + c(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k)) - c(i-1,j-1) &
* (psi(i-1,j-1,k)-psi(i-1,j-2,k))) + mpsi(i,j-1) * (d(i+1,j-1) &
* (psi(i+1,j-1,k)-psi(i,j-1,k)) - d(i,j-1) &
* (psi(i,j-1,k)-psi(i-1,j-1,k)) + c(i,j) * (psi(i,j,k)-psi(i,j-1,k)) &
- c(i,j-1) * (psi(i,j-1,k)-psi(i,j-2,k))) + mpsi(i-1,j) * (d(i,j) &
* (psi(i,j,k)-psi(i-1,j,k)) - d(i-1,j) * (psi(i-1,j,k)-psi(i-2,j,k)) &
+ c(i-1,j+1)*(psi(i-1,j+1,k)-psi(i-1,j,k)) - c(i-1,j) &
* (psi(i-1,j,k)-psi(i-1,j-1,k))) + mpsi(i,j) * (d(i+1,j) &
* (psi(i+1,j,k)-psi(i,j,k)) - d(i,j)*(psi(i,j,k)-psi(i-1,j,k)) &
+ c(i,j+1) * (psi(i,j+1,k)-psi(i,j,k)) - c(i,j) &
* (psi(i,j,k)-psi(i,j-1,k))) - Quarter / (phi(i,j,k)-phi(i,j,k-1)) &
* ((mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
) - phi(i-1,j,k)) + mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
+ phi(i+1,j,k) - phi(i,j,k))) * (d(i,j-1) * (psi(i,j-1,k+1) &
- psi(i-1,j-1,k+1) - psi(i,j-1,k-1) + psi(i-1,j-1,k-1)) + d(i,j) &
* (psi(i,j,k+1) - psi(i-1,j,k+1) - psi(i,j,k-1) + psi(i-1,j,k-1))) &
+ (mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
- phi(i,j-1,k)) + mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
+ phi(i,j+1,k) - phi(i,j,k))) * (c(i-1,j) * (psi(i-1,j,k+1) &
- psi(i-1,j-1,k+1) - psi(i-1,j,k-1) + psi(i-1,j-1,k-1)) &
+ c(i,j) * (psi(i,j,k+1) - psi(i,j-1,k+1) - psi(i,j,k-1) &
+ psi(i,j-1,k-1)))))) * adjpv(i,j,k)

```

```

ps(i-1,j-1,k-1) = ps(i-1,j-1,k-1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j-1) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
+ c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k)))) * adjpv(i,j,k)
ps(i,j-1,k-1) = ps(i,j-1,k-1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k))) &
- d(i,j-1) * (mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) &
- phi(i,j,k)) + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) &
+ phi(i,j,k) - phi(i-1,j,k)))) * adjpv(i,j,k)
ps(i-1,j,k-1) = ps(i-1,j,k-1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
- c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k)))) * adjpv(i,j,k)
ps(i,j,k-1) = ps(i,j,k-1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (-d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) &
- phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) &
* (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k))) &
- c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) &
- phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) &
+ phi(i,j,k) - phi(i,j-1,k)))) * adjpv(i,j,k)
ps(i-1,j-2,k) = ps(i-1,j-2,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i-1,j-1) &
* c(i-1,j-1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i,j-2,k) = ps(i,j-2,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i,j-1) &
* c(i,j-1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-2,j-1,k) = ps(i-2,j-1,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i-1,j-1) &
* d(i-1,j-1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-1,j-1,k) = ps(i-1,j-1,k) + b(i,j,k) * (Half * rdx**2 * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j-1) * d(i,j-1) &
+ mpsi(i-1,j-1) * (-d(i,j-1) - d(i-1,j-1) - c(i-1,j) - c(i-1,j-1)) &
+ mpsi(i-1,j) * c(i-1,j))) * adjpv(i,j,k)

```

```

ps(i,j-1,k) = ps(i,j-1,k) + b(i,j,k) * (Half * rdx**2 * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j-1) * (-d(i+1,j-1) - d(i,j-1) &
- c(i,j) - c(i,j-1)) + mpsi(i-1,j-1) * d(i,j-1) + mpsi(i,j) &
* c(i,j))) * adjpv(i,j,k)
ps(i+1,j-1,k) = ps(i+1,j-1,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i,j-1) &
* d(i+1,j-1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-2,j,k) = ps(i-2,j,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i-1,j) &
* d(i-1,j) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-1,j,k) = ps(i-1,j,k) + b(i,j,k) * (Half * rdx**2 * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j) * d(i,j) + mpsi(i-1,j) &
* (-d(i,j) - d(i-1,j) - c(i-1,j+1) - c(i-1,j)) + mpsi(i-1,j-1) &
* c(i-1,j))) * adjpv(i,j,k)
ps(i,j,k) = ps(i,j,k) + b(i,j,k) * (Half * rdx**2 * (e(i,j,k-1) &
* (phi(i,j,k-1)-phi(i,j,k-2)) - e(i,j,k+1) &
* (phi(i,j,k+1)-phi(i,j,k))) * (mpsi(i,j) * (-d(i+1,j) - d(i,j) &
- c(i,j+1) - c(i,j)) + mpsi(i-1,j) * d(i,j) + mpsi(i,j-1) &
* c(i,j))) * adjpv(i,j,k)
ps(i+1,j,k) = ps(i+1,j,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i,j) &
* d(i+1,j) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-1,j+1,k) = ps(i-1,j+1,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i-1,j) &
* c(i-1,j+1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i,j+1,k) = ps(i,j+1,k) + b(i,j,k) * (Half * rdx**2 * mpsi(i,j) &
* c(i,j+1) * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k)))) * adjpv(i,j,k)
ps(i-1,j-1,k+1) = ps(i-1,j-1,k+1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (-d(i,j-1) * (mu(i,j) &
* (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) &
+ mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k))) - c(i-1,j) * (mv(i,j) * (phi(i,j+1,k-1) &
- phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) &
* (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k)))) &
* adjpv(i,j,k)
ps(i,j-1,k+1) = ps(i,j-1,k+1) - b(i,j,k) * (Eighth * rdx**2 &
/ (phi(i,j,k)-phi(i,j,k-1)) &
* (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
- e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j-1) * (mu(i,j) &
* (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) &
+ mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
- phi(i-1,j,k))) - c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
+ phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) &
- phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k)))) &
* adjpv(i,j,k)
ps(i-1,j,k+1) = ps(i-1,j,k+1) - b(i,j,k) * (Eighth * rdx**2 &

```

```

      / (phi(i,j,k)-phi(i,j,k-1)) &
      * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
      - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (c(i-1,j) * (mv(i,j) &
      * (phi(i,j+1,k-1) - phi(i,j,k-1) + phi(i,j+1,k) - phi(i,j,k)) &
      + mv(i,j-1) * (phi(i,j,k-1) - phi(i,j-1,k-1) + phi(i,j,k) &
      - phi(i,j-1,k))) - d(i,j) * (mu(i,j) * (phi(i+1,j,k-1) - phi(i,j,k-1) &
      + phi(i+1,j,k) - phi(i,j,k)) + mu(i-1,j) * (phi(i,j,k-1) &
      - phi(i-1,j,k-1) + phi(i,j,k) - phi(i-1,j,k)))) * adjpv(i,j,k)
    ps(i,j,k+1) = ps(i,j,k+1) - b(i,j,k) * (Eighth * rdx**2 &
      / (phi(i,j,k)-phi(i,j,k-1)) &
      * (e(i,j,k-1) * (phi(i,j,k-1)-phi(i,j,k-2)) &
      - e(i,j,k+1) * (phi(i,j,k+1)-phi(i,j,k))) * (d(i,j) * (mu(i,j) &
      * (phi(i+1,j,k-1) - phi(i,j,k-1) + phi(i+1,j,k) - phi(i,j,k)) &
      + mu(i-1,j) * (phi(i,j,k-1) - phi(i-1,j,k-1) + phi(i,j,k) &
      - phi(i-1,j,k))) + c(i,j) * (mv(i,j) * (phi(i,j+1,k-1) - phi(i,j,k-1) &
      + phi(i,j+1,k) - phi(i,j,k)) + mv(i,j-1) * (phi(i,j,k-1) &
      - phi(i,j-1,k-1) + phi(i,j,k) - phi(i,j-1,k)))) * adjpv(i,j,k)
  end forall
  ph(/1,nx/),:,:) = Zero
  ph(2:nx-1,/1,ny/),:) = Zero
  ph(2:nx-1,2:ny-1,/0,nz/) = Zero
  ps(/0,nx/),:,:) = Zero
  ps(1:nx-1,/0,ny/),:) = Zero
  deallocate(d, c, mpsi, e, b)
end subroutine pv_adj
! =====
subroutine solve_phi_psi(pvGiven, psi, phi, f, mCap, mu, mfm, mfu, mfv, etaz, &
  etam, pt, rdx)
  real(cp), dimension(:,:,:), intent(in) :: pvGiven
  real(cp), dimension(0:0,:,:), intent(inout) :: psi ! (0:nx,0:ny)
  real(cp), dimension(:,:0:), intent(inout) :: phi
  real(cp), dimension(:,:), intent(in) :: f, mCap, mu, mfm ! (1:nx,1:ny)
  real(cp), dimension(0:,:), intent(in) :: mfu
  real(cp), dimension(:,0:), intent(in) :: mfv ! (1:nx,0:ny)
  real(cp), dimension(0:), intent(in) :: etaz
  real(cp), dimension(:), intent(in) :: etam
  real(cp), intent(in) :: pt, rdx
  real(cp), parameter :: e = 5000., MaxStep = 2187.
  character(len=*), parameter :: Mismatch = "mismatch in solve_phi_psi!"
  real(cp), dimension(:,:,:), allocatable :: imb, pv
  real(cp), dimension(:,:,:), allocatable :: gradphi1, gradphi2, gradphi, pphi, &
    phiTrial
  real(cp), dimension(:,:,:), allocatable :: gradpsi1, gradpsi2, gradpsi, ppsi, &
    psiTrial

  real, dimension(:,:,:), allocatable :: pv4
  real(cp) :: costa, costb, cost, costlb, costrb, costleft, costright, oldcost
  real(cp) :: alpha, alphalb, alpharb, alphaleft, alphasright
  integer :: nx, ny, nz, i, j, k
  nx = size(psi,1) - 1
  ny = size(psi,2) - 1

```



```

nz = size(psi,3)
! Sanity checks
if (any(shape(pvGiven) /= (/ nx, ny, nz /) )) stop "pvGiven " // Mismatch
if (any(shape(psi) /= (/ nx+1, ny+1, nz /) )) stop "psi " // Mismatch
if (any(shape(phi) /= (/ nx, ny, nz+1 /) )) stop "phi " // Mismatch
if (any(shape(f) /= (/ nx, ny /) )) stop "f " // Mismatch
if (any(shape(mCap) /= (/ nx, ny /) )) stop "mCap " // Mismatch
if (any(shape(mu) /= (/ nx, ny /) )) stop "mu " // Mismatch
if (any(shape(mfm) /= (/ nx, ny /) )) stop "mfm " // Mismatch
if (any(shape(mfu) /= (/ nx+1, ny /) )) stop "mfu " // Mismatch
if (any(shape(mfv) /= (/ nx, ny+1 /) )) stop "mfv " // Mismatch
if (size(etaz) /= nz+1) stop "etaz " // Mismatch
if (size(etam) /= nz) stop "etam " // Mismatch
allocate(imb(nx,ny,nz), pv(nx,ny,nz))
allocate(gradphi1(nx,ny,0:nz), gradphi2(nx,ny,0:nz), gradphi(nx,ny,0:nz), &
  pphi(nx,ny,0:nz), phiTrial(nx,ny,0:nz))
allocate(gradpsi1(0:nx,0:ny,nz), gradpsi2(0:nx,0:ny,nz), &
  gradpsi(0:nx,0:ny,nz), ppsi(0:nx,0:ny,nz), psiTrial(0:nx,0:ny,nz))
allocate(pv4(nx,ny,nz))
i = 1
do
  print *, "Iteration: ", i
  ! Calculate cost function
  call imbalance_exp(psi, phi, f, mCap, mfm, mfu, mfv, etaz, etam, rdx, imb)
  costb = Half * e * sum(imb**2)
  call pv_exp(psi, phi, f, mCap, mu, mfu, mfv, etaz, etam, pt, rdx, pv4)
  pv = pv4
  costa = Half * sum( (pv - pvGiven)**2 )
  cost = costa + costb
  write(*,*) " Cost A & B:", costa, costb
  write(*,*) " Total cost:", cost
  ! Calculate gradient of cost function
  call pv_adj(pv-pvGiven, psi, phi, f, mCap, mu, mfu, mfv, etaz, etam, pt, &
    rdx, gradpsi1, gradphi1)
  call imbalance_simp_adj(imb, psi, f, mCap, mfm, mfu, mfv, etaz, etam, &
    rdx, gradpsi2, gradphi2)
  gradphi = gradphi1 + e * gradphi2
  gradpsi = gradpsi1 + e * gradpsi2
  pphi = gradphi / sqrt(sum(gradphi**2)+sum(gradpsi**2))
  ppsi = gradpsi / sqrt(sum(gradphi**2)+sum(gradpsi**2))
  ! Perform line search
  alphasb = Zero
  costlb = cost
  alphasb = Maxstep
  phiTrial = phi - alpha * pphi
  psiTrial = psi - alpha * ppsi
  ! Calculate cost function
  call imbalance_exp(psiTrial, phiTrial, f, mCap, mfm, mfu, mfv, etaz, &
    etam, rdx, imb)
  costb = Half * e * sum(imb**2)

```

```

call pv_exp(psiTrial, phiTrial, f, mCap, mu, mfu, mfv, etaz, etam, pt, &
            rdx, pv4)
pv = pv4
costa = Half * sum( (pv - pvGiven)**2 )
costrb = costa + costb
oldcost = Half * (costlb + costrb)
do j = 1, 100
    alphaleft = alphasb + Third * (alpharb - alphasb)
    alphasright = alphasb + Two * Third * (alpharb - alphasb)
    ! Try left
    phiTrial = phi - alphaleft * pphi
    psiTrial = psi - alphaleft * ppsi
    call imbalance_exp(psiTrial, phiTrial, f, mCap, mfm, mfu, mfv, etaz, &
                      etam, rdx, imb)
    costb = Half * e * sum(imb**2)
    call pv_exp(psiTrial, phiTrial, f, mCap, mu, mfu, mfv, etaz, etam, pt, &
                rdx, pv4)
    pv = pv4
    costa = Half * sum( (pv - pvGiven)**2 )
    costleft = costa + costb
    ! Try right
    phiTrial = phi - alphasright * pphi
    psiTrial = psi - alphasright * ppsi
    call imbalance_exp(psiTrial, phiTrial, f, mCap, mfm, mfu, mfv, etaz, &
                      etam, rdx, imb)
    costb = Half * e * sum(imb**2)
    call pv_exp(psiTrial, phiTrial, f, mCap, mu, mfu, mfv, etaz, etam, pt, &
                rdx, pv4)
    pv = pv4
    costa = Half * sum( (pv - pvGiven)**2 )
    costright = costa + costb
    ! Which is best?
    if (costleft < costright) then
        alphasb = alphasright
    else
        alphasb = alphaleft
    end if
    alpha = Half * (alphasb + alphasb)
    if (abs(alphasb-alphasb)/(Half*(alphasb+alphasb)) < 1e-4) then
        print *, j, alpha
        exit
    end if
end do
if (Half * (costleft + costright) > oldcost) exit
oldcost = Half * (costleft + costright)
! Update phi and psi
phi = phi - alphasright * pphi
psi = psi - alphasright * ppsi
i = i + 1
if (i > 100) exit

```

```

end do
open(34, file="psphiinvert.dat", status="new", form="unformatted", &
    action="write")
write (34) nx, ny, nz
write (34) psi
write (34) phi
close(34)
end subroutine solve_phi_psi
! =====
function xdiff(data, rdx) result(res)
    real(cp), dimension(:,:), intent(in) :: data
    real(cp),          intent(in) :: rdx
    real(cp), dimension(size(data,1)-1,size(data,2)) :: res
    integer :: nx
    nx = size(data,1)-1
    res = rdx * (data(2,:) - data(:,nx,:))
end function xdiff
! =====
function ydiff(data, rdx) result(res)
    real(cp), dimension(:,:), intent(in) :: data
    real(cp),          intent(in) :: rdx
    real(cp), dimension(size(data,1),size(data,2)-1) :: res
    integer :: ny
    ny = size(data,2)-1
    res = rdx * (data(:,2:) - data(:,ny))
end function ydiff
! =====
function xyavg(data) result(res)
    real(cp), dimension(:,:), intent(in) :: data
    real(cp), dimension(size(data,1)-1,size(data,2)-1) :: res
    integer :: nx, ny
    nx = size(data,1)-1
    ny = size(data,2)-1
    res = Quarter * (data(:,nx,ny) + data(2,:,ny) + data(:,nx,2) + data(2,:,2))
end function xyavg
! =====
function xavg(data) result(res)
    real(cp), dimension(:,:), intent(in) :: data
    real(cp), dimension(size(data,1)-1,size(data,2)) :: res
    integer :: nx
    nx = size(data,1)-1
    res = Half * (data(:,nx,:) + data(2,:))
end function xavg
! =====
function yavg(data) result(res)
    real(cp), dimension(:,:), intent(in) :: data
    real(cp), dimension(size(data,1),size(data,2)-1) :: res
    integer :: ny
    ny = size(data,2)-1
    res = Half * (data(:,ny) + data(:,2))

```

```

    end function yavg
end module partition_wind

```

## 8.11 Output definitions

This module defines each meteorological quantity that can be output into a GEMPAK file. It also specifies how the various quantities can be computed.

```

module registry
  use wrf2gem_subs, only: read_values, read_values_const, mem_error,      &
                           freq_divisible, open_wrf_file, close_wrf_file
  use gempak,           only: write_gempak
  use diagnostics,      only: vint, hydro_interp, mix_ratio_to_spec_hum,  &
                           temp_from_theta_p, surface_pres, density, sea_pres, &
                           lifted_index, mean_rh, unstag_hght, theta_phi, pvor
  use diagnostics,      only: Grav, Eps
  use maths,            only: mean, cen_diff_stag, cen_diff, uneven_deriv, &
                           calc_mfps_i
  use maths,            only: Half, One
  use partition_wind,   only: partition, balance_lhs, balance_lhs_exp,    &
                           balance_lhs_tlm, balance_lhs_adj, balance_rhs_simp, &
                           balance_rhs_simp_exp, balance_rhs_simp_tlm,      &
                           balance_rhs_simp_adj, imbalance_simp_adj, pv_exp, &
                           pv_tlm, pv_adj, imbalance_exp, solve_phi_psi
  use current_kind, only: cp, Quarter
  implicit none
  save
  private
  public :: init_reg, output_var
  ! Encode GEMPAK vertical coordinate IDs
  integer, parameter :: HAGL = 1279738184, None = 0, Sgma = 4, Pres = 1
  ! Set up enumeration for the available variables
  integer, parameter :: LastWRF = 40, FirstDiag = 101, LastDiag = 260, &
                           FirstPresDiag = 501, LastPresDiag = 506
  integer, parameter :: TerHght = 1, SfcTemp = 2, SfcMixR = 3, &
                           EtaMass = 4, PBase = 5, PPert = 6, &
                           ThetaPert = 7, UStag = 8, VStag = 9, &
                           WStag = 10, SfcU = 11, SfcV = 12, &
                           MixR = 13, MixRCloud = 14, MixRIce = 15, &
                           ConvPre = 16, StabPre = 17, GeopPert = 18, &
                           GeopBase = 19, EtaW = 20, LandUse = 21, &
                           MuPert = 22, MuBase = 23, SfcTheta = 24, &
                           MixRRain = 25, MixRSnow = 26, MixRGraup = 27, &
                           PBLHgt = 28, SeaTemp = 29, SkinTemp = 30, &
                           SWGrd = 31, LWGrd = 32, GrdFx = 33, &
                           HeatFx = 34, MoisFx = 35, LaHeat = 36, &

```

```

        MapFacM = 37, MapFacU = 38, MapFacV = 39, &
        Coriolis = 40
integer, parameter :: SfcP = 101, SfcP2m = 102, SeaLevP = 103, &
        ConvPre1h = 104, TotPre1h = 105, TotPre3h = 106, &
        TotPre6h = 107, TotPre = 108, Pressure = 109, &
        Theta = 110, UUnstag = 111, VUnstag = 112, &
        CWtr = 113, WUnstag = 114, SfcTempF0 = 115, &
        SfcMixRF0 = 116, SfcUF0 = 117, SfcVF0 = 118, &
        LiftIdx = 119, MeanRH = 120, GeopHghtS = 121, &
        SfcSpHum = 122, SpHum = 123, TmpK = 124, &
        GeopHght = 125, DryAir = 126, MUCAPE = 127, &
        MULPL = 128, SBCAPE = 129, SBCIN = 130, &
        MixCAPE = 131, MixCIN = 132, MixLCL = 133, &
        MixLFC = 134, UStorm = 135, VStorm = 136, &
        StormRHel = 137, PreWater = 138, Refl = 139, &
        CompRefl = 140
integer, parameter :: MCap = 201, MScript1 = 202, MScript2 = 203, &
        Diverg = 204, DScript = 205, DDDt = 206, &
        EtaDot = 207, TermA2 = 208, TermA3 = 209, &
        TermA4 = 210, TermA5 = 211, TermB1 = 212, &
        TermB2 = 213, TermB3 = 214, TermC = 215, &
        TermD = 216, MS1Unstag = 217, MS2Unstag = 218, &
        TermE = 219, B1B2CD = 220, ABCD = 221, &
        Stream = 222, VelocPot = 223, DivPsi = 224, &
        MS1PsiUnstag = 225, MS2PsiUnstag = 226, &
        MS1ChiUnstag = 227, MS2ChiUnstag = 228, &
        BallHS = 229, BalRHSSimp = 230, ImbalSimp = 231, &
        BallHSExp = 232, BallHSTLM = 233, BallHSPert = 234, &
        BallHSAdj = 235, BalRHSSimpExp = 236, BalRHSSimpTLM = 237, &
        BalRHSSimpPert = 238, BalRHSSimpAdj = 239, ImbalSimpPert = 240, &
        ImbalSimpTLM = 241, ImbalSimpAdj = 242, ThetaFromPhi = 243, &
        PotVort = 244, PVExp = 245, PVTLM = 246, &
        PVPert = 247, PVAdj = 248, CostGradPhi = 249, &
        InvertPhi = 250, InvertPsi = 251
integer, parameter :: TmpKPres = 501, UUnstagPres = 502, &
        VUnstagPres = 503, WUnstagPres = 504, &
        GeopHghtPres = 505, SpHumPres = 506
! This structure contains information about the various WRF variables.
type wrf_var
    integer :: dimType, & ! 0-z,t; 1-x,y,t; 2-x,y,z,t; 3-xs,y,z,t;
    ! 4-x,ys,z,t; 5-x,y,zs,t
    lev1, lev2, & ! GEMPAK level parameters. Relevant only for
    ! dimType 1
    vertCordID ! GEMPAK vertical coordinate ID
    character(len=8) :: wrfName ! WRF variable name
    character(len=4) :: gemName ! GEMPAK variable name
    logical :: const ! T - Fixed with time; F - Varies with time
end type wrf_var
! This is an analogous structure for diagnosed variables.
type diag_var

```

```

    integer          :: dimType, lev1, lev2, vertCordID, diagID
    character(len=12) :: gemName
end type diag_var
! And finally a structure for diagnosed variables interpolated to pressure.
type pres_diag_var
    integer          :: interpType, & ! 1-linear interp; 2-log interp
    integer          :: diagID
    character(len=12) :: gemName
end type pres_diag_var
type(wrf_var), dimension>LastWRF :: wrfVar
type(diag_var), dimension=FirstDiag>LastDiag :: diagVar
type(pres_diag_var), dimension=FirstPresDiag>LastPresDiag :: presDiagVar
integer :: timesPer12hr, pBot, pTop, dp, np
contains ! =====
! init_reg initializes the registry, which contains the vital parameters for
! each variable or diagnostic, with the number of outputs per 12 hours (freq)
! and pressure interpolation information (pb, pt, deltap) provided as input.
subroutine init_reg(freq, pb, pt, deltap)
    integer, intent(in) :: freq, pb, pt, deltap
    wrfVar(TerHght)      = wrf_var(1, 0, -1, None, "HGT", "HGHT", .true.)
    wrfVar(SfcTemp)      = wrf_var(1, 2, -1, HAGL, "T2", "TMPK", .false.)
    wrfVar(SfcMixR)      = wrf_var(1, 2, -1, HAGL, "Q2", "MIXR", .false.)
    wrfVar(EtaMass)      = wrf_var(0, -1, -1, -1, "ZNU", "SGMA", .true.)
    wrfVar(PBase)        = wrf_var(2, -1, -1, Sgma, "PB", "PBAR", .true.)
    wrfVar(PPert)        = wrf_var(2, -1, -1, Sgma, "P", "PP", .false.)
    wrfVar(ThetaPert)    = wrf_var(2, -1, -1, Sgma, "T", "TMPO", .false.)
    wrfVar(UStag)        = wrf_var(3, -1, -1, Sgma, "U", "USTG", .false.)
    wrfVar(VStag)        = wrf_var(4, -1, -1, Sgma, "V", "VSTG", .false.)
    wrfVar(WStag)        = wrf_var(5, -1, -1, Sgma, "W", "WSTG", .false.)
    wrfVar(SfcU)         = wrf_var(1, 10, -1, HAGL, "U10", "UREL", .false.)
    wrfVar(SfcV)         = wrf_var(1, 10, -1, HAGL, "V10", "VREL", .false.)
    wrfVar(MixR)         = wrf_var(2, -1, -1, Sgma, "QVAPOR", "MIXR", .false.)
    wrfVar(MixRCloud)    = wrf_var(2, -1, -1, Sgma, "QCLOUD", "WC", .false.)
    wrfVar(MixRIce)      = wrf_var(2, -1, -1, Sgma, "QICE", "WI", .false.)
    wrfVar(ConvPre)      = wrf_var(1, 0, -1, None, "RAINC", "CTOT", .false.)
    wrfVar(StabPre)      = wrf_var(1, 0, -1, None, "RAINNC", "ETOT", .false.)
    wrfVar(GeopPert)     = wrf_var(5, -1, -1, Sgma, "PH", "HP", .false.)
    wrfVar(GeopBase)     = wrf_var(5, -1, -1, Sgma, "PHB", "HBAR", .true.)
    wrfVar(EtaW)         = wrf_var(0, -1, -1, -1, "ZNW", "SGMA", .true.)
    wrfVar(LandUse)      = wrf_var(1, 0, -1, None, "LU_INDEX", "LUC", .true.)
    wrfVar(MuPert)       = wrf_var(1, 0, -1, None, "MU", "MUP", .false.)
    wrfVar(MuBase)       = wrf_var(1, 0, -1, None, "MUB", "MUB", .true.)
    wrfVar(SfcTheta)     = wrf_var(1, 2, -1, HAGL, "TH2", "THTA", .false.)
    wrfVar(MixRRain)     = wrf_var(2, -1, -1, Sgma, "QRAIN", "WR", .false.)
    wrfVar(MixRSnow)     = wrf_var(2, -1, -1, Sgma, "QSNOW", "WS", .false.)
    wrfVar(MixRGraup)    = wrf_var(2, -1, -1, Sgma, "QGRAUP", "WG", .false.)
    wrfVar(PBLHgt)       = wrf_var(1, 0, -1, None, "PBLH", "PBLH", .false.)
    wrfVar(SeaTemp)      = wrf_var(1, 0, -1, None, "SST", "SST", .false.)
    wrfVar(SkinTemp)     = wrf_var(1, 0, -1, None, "TSK", "TSK", .false.)
    wrfVar(SWGrd)        = wrf_var(1, 0, -1, None, "SWDOWN", "SWD", .false.)

```

```

wrfVar(LWGrd)      = wrf_var(1, 0, -1, None, "GLW", "LWD", .false.)
wrfVar(GrdFx)      = wrf_var(1, 0, -1, None, "GRDFLX", "GRDF", .false.)
wrfVar(HeatFx)     = wrf_var(1, 0, -1, None, "HFX", "HFX", .false.)
wrfVar(MoisFx)     = wrf_var(1, 0, -1, None, "QFX", "QFX", .false.)
wrfVar(LaHeat)     = wrf_var(1, 0, -1, None, "LH", "LH", .false.)
wrfVar(MapFacM)    = wrf_var(1, 0, -1, None, "MAPFAC_M", "MFM", .true.)
wrfVar(MapFacU)    = wrf_var(3, 0, -1, None, "MAPFAC_U", "MFU", .true.)
wrfVar(MapFacV)    = wrf_var(4, 0, -1, None, "MAPFAC_V", "MFV", .true.)
wrfVar(Coriolis)   = wrf_var(1, 0, -1, None, "F", "F", .true.)
diagVar(SfcP)      = diag_var(1, 0, -1, None, SfcP, "PRES")
diagVar(SfcP2m)    = diag_var(1, 2, -1, HAGL, SfcP2m, "PRES")
diagVar(SeaLevP)   = diag_var(1, 0, -1, None, SeaLevP, "PMSL")
diagVar(ConvPre1h) = diag_var(1, 0, -1, None, ConvPre1h, "CO1M")
diagVar(TotPre1h)  = diag_var(1, 0, -1, None, TotPre1h, "PO1M")
diagVar(TotPre3h)  = diag_var(1, 0, -1, None, TotPre3h, "PO3M")
diagVar(TotPre6h)  = diag_var(1, 0, -1, None, TotPre6h, "PO6M")
diagVar(TotPre)    = diag_var(1, 0, -1, None, TotPre, "PTOT")
diagVar(Pressure)  = diag_var(2, -1, -1, Sgma, Pressure, "PRES")
diagVar(Theta)     = diag_var(2, -1, -1, Sgma, Theta, "THTA")
diagVar(UUnstag)   = diag_var(2, -1, -1, Sgma, UUnstag, "UREL")
diagVar(VUnstag)   = diag_var(2, -1, -1, Sgma, VUnstag, "VREL")
diagVar(CWtr)      = diag_var(2, -1, -1, Sgma, CWtr, "CWTR")
diagVar(WUnstag)   = diag_var(2, -1, -1, Sgma, WUnstag, "W")
diagVar(SfcTempF0) = diag_var(1, 2, -1, HAGL, SfcTempF0, "TMPK")
diagVar(SfcMixRF0) = diag_var(1, 2, -1, HAGL, SfcMixRF0, "MIXR")
diagVar(SfcUF0)    = diag_var(1, 10, -1, HAGL, SfcUF0, "UREL")
diagVar(SfcVF0)    = diag_var(1, 10, -1, HAGL, SfcVF0, "VREL")
diagVar(LiftIdx)   = diag_var(1, 0, -1, None, LiftIdx, "LFT4")
diagVar(MeanRH)    = diag_var(1, 850, 500, Pres, MeanRH, "RELH")
diagVar(GeopHghtS) = diag_var(5, -1, -1, Sgma, GeopHghtS, "HGHT")
diagVar(SfcSpHum)  = diag_var(1, 2, -1, HAGL, SfcSpHum, "SPFH")
diagVar(SpHum)     = diag_var(2, -1, -1, Sgma, SpHum, "SPFH")
diagVar(TmpK)      = diag_var(2, -1, -1, Sgma, TmpK, "TMPK")
diagVar(GeopHght)  = diag_var(2, -1, -1, Sgma, GeopHght, "HGHT")
diagVar(DryAir)    = diag_var(1, 0, -1, None, DryAir, "MU")
diagVar(MUCAPE)    = diag_var(1, 0, -1, None, MUCAPE, "MUCAPE")
diagVar(MULPL)     = diag_var(1, 0, -1, None, MULPL, "LPL")
diagVar(SBCAPE)    = diag_var(1, 0, -1, None, SBCAPE, "SBCAPE")
diagVar(SBCIN)     = diag_var(1, 0, -1, None, SBCIN, "SBCIN")
diagVar(MixCAPE)   = diag_var(1, 0, -1, None, MixCAPE, "MLCAPE")
diagVar(MixCIN)    = diag_var(1, 0, -1, None, MixCIN, "MLCIN")
diagVar(MixLCL)    = diag_var(1, 0, -1, None, MixLCL, "LCL")
diagVar(MixLFC)    = diag_var(1, 0, -1, None, MixLFC, "LFC")
diagVar(UStorm)    = diag_var(1, 0, -1, None, UStorm, "USTM")
diagVar(VStorm)    = diag_var(1, 0, -1, None, VStorm, "VSTM")
diagVar(StormRHel) = diag_var(1, 0, 1000, HAGL, StormRHel, "SRH")
diagVar(PreWater)  = diag_var(1, 0, -1, None, PreWater, "PW")
diagVar(Refl)      = diag_var(2, -1, -1, Sgma, Refl, "REFL")
diagVar(CompRefl)  = diag_var(1, 0, -1, None, CompRefl, "CREF")
diagVar(MCap)      = diag_var(1, 0, -1, None, MCap, "MCAP")

```

```

diagVar(MScript1) = diag_var(3, -1, -1, Sgma, MScript1, "MSCR1")
diagVar(MScript2) = diag_var(4, -1, -1, Sgma, MScript2, "MSCR2")
diagVar(Diverg)   = diag_var(2, -1, -1, Sgma, Diverg, "DIVG")
diagVar(DScript)  = diag_var(2, -1, -1, Sgma, DScript, "DSCR")
diagVar(DDDt)     = diag_var(2, -1, -1, Sgma, DDDt, "DDDT")
diagVar(EtaDot)   = diag_var(2, -1, -1, Sgma, EtaDot, "EDOT")
diagVar(TermA2)   = diag_var(2, -1, -1, Sgma, TermA2, "A2")
diagVar(TermA3)   = diag_var(2, -1, -1, Sgma, TermA3, "A3")
diagVar(TermA4)   = diag_var(2, -1, -1, Sgma, TermA4, "A4")
diagVar(TermA5)   = diag_var(2, -1, -1, Sgma, TermA5, "A5")
diagVar(TermB1)   = diag_var(2, -1, -1, Sgma, TermB1, "B1")
diagVar(TermB2)   = diag_var(2, -1, -1, Sgma, TermB2, "B2")
diagVar(TermB3)   = diag_var(2, -1, -1, Sgma, TermB3, "B3")
diagVar(TermC)    = diag_var(2, -1, -1, Sgma, TermC, "C")
diagVar(TermD)    = diag_var(2, -1, -1, Sgma, TermD, "D")
diagVar(MS1Unstag) = diag_var(2, -1, -1, Sgma, MS1Unstag, "MSCR1")
diagVar(MS2Unstag) = diag_var(2, -1, -1, Sgma, MS2Unstag, "MSCR2")
diagVar(TermE)    = diag_var(2, -1, -1, Sgma, TermE, "E")
diagVar(B1B2CD)   = diag_var(2, -1, -1, Sgma, B1B2CD, "B1B2CD")
diagVar(ABCD)     = diag_var(2, -1, -1, Sgma, ABCD, "ABCD")
diagVar(Stream)   = diag_var(2, -1, -1, Sgma, Stream, "STRFUN")
diagVar(VelocPot) = diag_var(2, -1, -1, Sgma, VelocPot, "VELPOT")
diagVar(DivPsi)   = diag_var(2, -1, -1, Sgma, DivPsi, "DIVPSI")
diagVar(MS1PsiUnstag) = diag_var(2, -1, -1, Sgma, MS1PsiUnstag, "MS1PSI")
diagVar(MS2PsiUnstag) = diag_var(2, -1, -1, Sgma, MS2PsiUnstag, "MS2PSI")
diagVar(MS1ChiUnstag) = diag_var(2, -1, -1, Sgma, MS1ChiUnstag, "MS1CHI")
diagVar(MS2ChiUnstag) = diag_var(2, -1, -1, Sgma, MS2ChiUnstag, "MS2CHI")
diagVar(BalLHS)   = diag_var(2, -1, -1, Sgma, BalLHS, "LHS")
diagVar(BalRHSSimp) = diag_var(2, -1, -1, Sgma, BalRHSSimp, "RHS")
diagVar(ImbalSimp) = diag_var(2, -1, -1, Sgma, ImbalSimp, "IMB")
diagVar(BalLHSExp) = diag_var(2, -1, -1, Sgma, BalLHSExp, "LHSE")
diagVar(BalLHSTLM) = diag_var(2, -1, -1, Sgma, BalLHSTLM, "LHSTLM")
diagVar(BalLHSPert) = diag_var(2, -1, -1, Sgma, BalLHSPert, "LHSP")
diagVar(BalLHSAdj) = diag_var(2, -1, -1, Sgma, BalLHSAdj, "LHSADJ")
diagVar(BalRHSSimpExp) = diag_var(2, -1, -1, Sgma, BalRHSSimpExp, "RHSE")
diagVar(BalRHSSimpTLM) = diag_var(2, -1, -1, Sgma, BalRHSSimpTLM, "RHSTLM")
diagVar(BalRHSSimpPert) = diag_var(2, -1, -1, Sgma, BalRHSSimpPert, "RHSP")
diagVar(BalRHSSimpAdj) = diag_var(2, -1, -1, Sgma, BalRHSSimpAdj, "RHSADJ")
diagVar(ImbalSimpPert) = diag_var(2, -1, -1, Sgma, ImbalSimpPert, "IMBP")
diagVar(ImbalSimpTLM) = diag_var(2, -1, -1, Sgma, ImbalSimpTLM, "IMBTLM")
diagVar(ImbalSimpAdj) = diag_var(2, -1, -1, Sgma, ImbalSimpAdj, "IMBADJ")
diagVar(ThetaFromPhi) = diag_var(2, -1, -1, Sgma, ThetaFromPhi, "THPHI")
diagVar(PotVort)    = diag_var(2, -1, -1, Sgma, PotVort, "PV")
diagVar(PVExp)      = diag_var(2, -1, -1, Sgma, PVExp, "PVE")
diagVar(PVTLM)      = diag_var(2, -1, -1, Sgma, PVTLM, "PVTLM")
diagVar(PVPert)     = diag_var(2, -1, -1, Sgma, PVPert, "PVP")
diagVar(PVAdj)      = diag_var(2, -1, -1, Sgma, PVAdj, "PVADJ")
diagVar(CostGradPhi) = diag_var(2, -1, -1, Sgma, CostGradPhi, "CGPHI")
diagVar(InvertPhi)  = diag_var(2, -1, -1, Sgma, InvertPhi, "PHII")
diagVar(InvertPsi)  = diag_var(2, -1, -1, Sgma, InvertPsi, "PSII")

```



```

presDiagVar(TmpKPres)      = pres_diag_var(1, TmpKPres, "TMPK")
presDiagVar(UUnstagPres)  = pres_diag_var(1, UUnstagPres, "UREL")
presDiagVar(VUnstagPres)  = pres_diag_var(1, VUnstagPres, "VREL")
presDiagVar(WUnstagPres)  = pres_diag_var(1, WUnstagPres, "W")
presDiagVar(GeopHghtPres) = pres_diag_var(2, GeopHghtPres, "HGHT")! 1 or 2?
presDiagVar(SpHumPres)    = pres_diag_var(1, SpHumPres, "SPFH")
timesPer12hr = freq
pBot = pb
pTop = pt
dp = deltap
np = (pBot - pTop) / dp + 1
end subroutine init_reg
! =====
! output_var writes a particular WRF variable or diagnostic to the GEMPAK
! file associated with gemid.  ncid contains the file handle for the WRF
! history file from which the WRF data is being read.  The WRF model
! dimensions are specified by nx, ny, nz.  var is the ID for the requested
! diagnostic.  ind is the current index of histFiles, so that histFiles(ind)
! is the pathname for the GEMPAK history file currently being read.  times
! is an array of output times in GEMPAK form, which corresponds to all of the
! times in the history file.
subroutine output_var(ncid, gemid, nx, ny, nz, var, ind, histFiles, times)
  integer, intent(in) :: ncid, gemid, nx, ny, nz, var, ind
  character(len=80), dimension(:), intent(in) :: histFiles
  character(len=15), dimension(:), intent(in) :: times
  character(len=*) , parameter :: fmt1 = "(2x,a,i3,a)"
  write (*,fmt1) "Writing field ", var, " to GEMPAK..."
  ! Call the appropriate subroutine to produce the required output for GEMPAK
  select case (var)
  case (1:LastWRF)
    if (wrfVar(var)%dimType == 1 .or. wrfVar(var)%dimType == 2 .or. &
        wrfVar(var)%dimType == 5) then
      call simple_out(ncid, gemid, nx, ny, nz, wrfVar(var), times)
    else
      print *, "It does not make sense to output this variable to GEMPAK. &
        &Skipping!"
    end if
  case (FirstDiag:LastDiag)
    if (diagVar(var)%dimType == 1 .or. diagVar(var)%dimType == 2 .or. &
        diagVar(var)%dimType == 5) then
      call diag_out(ncid, gemid, nx, ny, nz, ind, histFiles, &
        diagVar(var), times)
    else
      print *, "This diagnostic is not appropriate for output to GEMPAK."
      print *, "It's either staggered in x or y, or not a function of x &
        &y."
      print *, "Skipping!"
    end if
  case (FirstPresDiag:LastPresDiag)
    call pres_diag_out(ncid, gemid, nx, ny, nz, presDiagVar(var), times)
  end select
end subroutine output_var

```

```

case default
  write (*,fmt1) "Variable ", var, " is unknown. Skipping!"
end select
end subroutine output_var
! =====
! simple_out reads the variable var in from the WRF history file associated
! with ncid and writes it out to the GEMPAK file associated with gemid.
! Other inputs to the subroutine are the WRF model dimensions (nx,ny,nz) and
! the array of times in GEMPAK form to be output, which corresponds to all of
! the times in the history file.
subroutine simple_out(ncid, gemid, nx, ny, nz, var, times)
  integer,                                intent(in) :: ncid, gemid, nx, ny, nz
  type(wrf_var),                          intent(in) :: var
  character(len=15), dimension(:), intent(in) :: times
  real, dimension(:,:,:), allocatable :: out4d
  real, dimension(:,:), allocatable :: out3d
  real, dimension(:,:), allocatable :: elvl
  integer                                :: kMax, eta, nt, status, t, k, lev
  character(len=20)                       :: gdat1
  character(len=12)                       :: parm
  ! Initialization
  select case (var%dimType)
  case (1) ! x,y,t
    kMax = 0
  case (2) ! x,y,z,t
    kMax = nz
    eta = EtaMass
  case (5) ! x,y,zs,t
    kMax = nz + 1
    eta = EtaW
  case default
    ! GEMPAK only understands variables that depend on (unstaggered) x and y
    print *, "Invalid dimType in simple_out!"
    stop
  end select
  nt = size(times)
  parm = var%gemName
  ! Read variable, and, if necessary, the vertical levels it's at
  if (var%dimType == 1) then
    allocate (out3d(nx,ny,nt), stat=status)
    call mem_error(status, 1, "simple_out")
    call read_values(ncid, var%wrfName, out3d)
  else
    allocate (out4d(nx,ny,kMax,nt), elvl(kMax,nt), stat=status)
    call mem_error(status, 1, "simple_out")
    call read_values(ncid, var%wrfName, out4d)
    call read_values(ncid, wrfVar(eta)%wrfName, elvl)
  end if
  ! Write the variable to GEMPAK
  if (var%const) nt = 1 ! Output constant variables only at initial time

```

```

do t = 1, nt
  gdat1 = times(t)
  if (var%dimType == 1) then
    call write_gempak(gemid, out3d(:,:,t), nx, ny, gdat1, var%lev1, &
      var%lev2, var%vertCordID, parm)
  else
    do k = 1, kMax
      lev = nint(elvl(k,1) * 10000)
      call write_gempak(gemid, out4d(:,:,k,t), nx, ny, gdat1, lev, &
        var%lev2, var%vertCordID, parm)
    end do
  end if
end do
! Clean up
if (allocated(out3d)) then
  deallocate (out3d, stat=status)
else
  deallocate (out4d, elvl, stat=status)
end if
call mem_error(status, 2, "simple_out")
end subroutine simple_out
! =====
! diag_out is a driver routine that produces the diagnostic var based on data
! from the WRF history file associated with ncid. The diagnostic is written
! out to the GEMPAK file associated with gemid. Other inputs to the
! subroutine are the WRF model dimensions (nx,ny,nz), the current index (ind)
! of histFiles, so that histFiles(ind) is the pathname for the GEMPAK history
! file currently being read, and the array of times in GEMPAK form to be
! output, which corresponds to all of the times in the history file.
subroutine diag_out(ncid, gemid, nx, ny, nz, ind, histFiles, var, times)
  integer,          intent(in) :: ncid, gemid, nx, ny, nz, ind
  character(len=80), dimension(:), intent(in) :: histFiles
  type(diag_var),   intent(in) :: var
  character(len=15), dimension(:), intent(in) :: times
  real, dimension(:,:,:), allocatable :: out4d
  real, dimension(:,:), allocatable :: out3d
  real, dimension(:), allocatable :: elvl
  integer          :: kMax, eta, nt, status, prev, diff, t, k, lev
  character(len=20) :: gdat1
  logical          :: ok
  ! Initialization
  select case (var%dimType)
  case (1) ! x,y,t
    kMax = 0
  case (2) ! x,y,z,t
    kMax = nz
    eta = EtaMass
  case (5) ! x,y,zs,t
    kMax = nz + 1
    eta = EtaW

```

```

case default
    print *, "Invalid dimType in diag_out!"
    stop
end select
nt = size(times)
ok = .true.
if (var%dimType == 1) then
    allocate (out3d(nx,ny,nt), stat=status)
    call mem_error(status, 1, "diag_out")
else
    allocate (out4d(nx,ny,kMax,nt), elvl(kMax,nt), stat=status)
    call mem_error(status, 1, "diag_out")
    call read_values(ncid, wrfVar(eta)%wrfName, elvl)
end if
! Compute appropriate diagnostic
select case (var%diagID)
case (SfcP)
    out3d = diag_SfcP(ncid, nx, ny, nz, nt)
case (SfcP2m)
    out3d = diag_SfcP2m(ncid, nx, ny, nz, nt)
case (SeaLevP)
    out3d = diag_SeaLevP(ncid, nx, ny, nz, nt)
case (Pressure)
    out4d = diag_Pressure(ncid, nx, ny, nz, nt)
case (Theta)
    out4d = diag_Theta(ncid, nx, ny, nz, nt)
case (UUnstag)
    out4d = diag_UUnstag(ncid, nx, ny, nz, nt)
case (VUnstag)
    out4d = diag_VUnstag(ncid, nx, ny, nz, nt)
case (CWtr)
    out4d = diag_CWtr(ncid, nx, ny, nz, nt)
case (WUnstag)
    out4d = diag_WUnstag(ncid, nx, ny, nz, nt)
case (SfcTempF0)
    out3d = diag_SfcTempF0(ncid, nx, ny, nz, nt)
case (SfcMixRF0)
    out3d = diag_SfcMixRF0(ncid, nx, ny, nz, nt)
case (SfcUF0)
    out3d = diag_SfcUF0(ncid, nx, ny, nz, nt)
case (SfcVF0)
    out3d = diag_SfcVF0(ncid, nx, ny, nz, nt)
case (LiftIdx)
    out3d = diag_LiftIdx(ncid, nx, ny, nz, nt)
case (MeanRH)
    out3d = diag_MeanRH(ncid, nx, ny, nz, nt)
case (GeopHghtS)
    out4d = diag_GeopHghtS(ncid, nx, ny, nz, nt)
case (SfcSpHum)
    out3d = diag_SfcSpHum(ncid, nx, ny, nz, nt)

```

```

case (SpHum)
    out4d = diag_SpHum(ncid, nx, ny, nz, nt)
case (TmpK)
    out4d = diag_TmpK(ncid, nx, ny, nz, nt)
case (GeopHght)
    out4d = diag_GeopHght(ncid, nx, ny, nz, nt)
case (DryAir)
    out3d = diag_DryAir(ncid, nx, ny, nt)
case (MCAp)
    out3d = diag_MCAp(ncid, nx, ny, nt)
case (MScript1)
    out4d = diag_MScript1(ncid, nx, ny, nz, nt)
case (MScript2)
    out4d = diag_MScript2(ncid, nx, ny, nz, nt)
case (Diverg)
    out4d = diag_Diverg(ncid, nx, ny, nz, nt)
case (DScript)
    out4d = diag_DScript(ncid, nx, ny, nz, nt)
case (DDDt)
    out4d = diag_DDDt(ncid, nx, ny, nz, nt)
case (EtaDot)
    out4d = diag_EtaDot(ncid, nx, ny, nz, nt)
case (TermA2)
    out4d = diag_TermA2(ncid, nx, ny, nz, nt)
case (TermA3)
    out4d = diag_TermA3(ncid, nx, ny, nz, nt)
case (TermA4)
    out4d = diag_TermA4(ncid, nx, ny, nz, nt)
case (TermA5)
    out4d = diag_TermA5(ncid, nx, ny, nz, nt)
case (TermB1)
    out4d = diag_TermB1(ncid, nx, ny, nz, nt)
case (TermB2)
    out4d = diag_TermB2(ncid, nx, ny, nz, nt)
case (TermB3)
    out4d = diag_TermB3(ncid, nx, ny, nz, nt)
case (TermC)
    out4d = diag_TermC(ncid, nx, ny, nz, nt)
case (TermD)
    out4d = diag_TermD(ncid, nx, ny, nz, nt)
case (MS1Unstag)
    out4d = diag_MS1Unstag(ncid, nx, ny, nz, nt)
case (MS2Unstag)
    out4d = diag_MS2Unstag(ncid, nx, ny, nz, nt)
case (TermE)
    out4d = diag_TermE(ncid, nx, ny, nz, nt)
case (B1B2CD)
    out4d = diag_B1B2CD(ncid, nx, ny, nz, nt)
case (ABCD)
    out4d = diag_ABCD(ncid, nx, ny, nz, nt)

```

```

case (Stream)
    out4d = diag_Stream(ncid, nx, ny, nz, nt)
case (VelocPot)
    out4d = diag_VelocPot(ncid, nx, ny, nz, nt)
case (DivPsi)
    out4d = diag_DivPsi(ncid, nx, ny, nz, nt)
case (MS1PsiUnstag)
    out4d = diag_MS1PsiUnstag(ncid, nx, ny, nz, nt)
case (MS2PsiUnstag)
    out4d = diag_MS2PsiUnstag(ncid, nx, ny, nz, nt)
case (MS1ChiUnstag)
    out4d = diag_MS1ChiUnstag(ncid, nx, ny, nz, nt)
case (MS2ChiUnstag)
    out4d = diag_MS2ChiUnstag(ncid, nx, ny, nz, nt)
case (BallHS)
    out4d = diag_BallHS(ncid, nx, ny, nz, nt)
case (BalRHSSimp)
    out4d = diag_BalRHSSimp(ncid, nx, ny, nz, nt)
case (ImbalSimp)
    out4d = diag_ImbalSimp(ncid, nx, ny, nz, nt)
case (BallHSExp)
    out4d = diag_BallHSExp(ncid, nx, ny, nz, nt)
case (BallHSTLM)
    out4d = diag_BallHSTLM(ncid, nx, ny, nz, nt)
case (BallHSPert)
    out4d = diag_BallHSPert(ncid, nx, ny, nz, nt)
case (BallHSAdj)
    out4d = diag_BallHSAdj(ncid, nx, ny, nz, nt)
case (BalRHSSimpExp)
    out4d = diag_BalRHSSimpExp(ncid, nx, ny, nz, nt)
case (BalRHSSimpTLM)
    out4d = diag_BalRHSSimpTLM(ncid, nx, ny, nz, nt)
case (BalRHSSimpPert)
    out4d = diag_BalRHSSimpPert(ncid, nx, ny, nz, nt)
case (BalRHSSimpAdj)
    out4d = diag_BalRHSSimpAdj(ncid, nx, ny, nz, nt)
case (ImbalSimpPert)
    out4d = diag_ImbalSimpPert(ncid, nx, ny, nz, nt)
case (ImbalSimpTLM)
    out4d = diag_ImbalSimpTLM(ncid, nx, ny, nz, nt)
case (ImbalSimpAdj)
    out4d = diag_ImbalSimpAdj(ncid, nx, ny, nz, nt)
case (ThetaFromPhi)
    out4d = diag_ThetaFromPhi(ncid, nx, ny, nz, nt)
case (PotVort)
    out4d = diag_PotVort(ncid, nx, ny, nz, nt)
case (PVExp)
    out4d = diag_PVExp(ncid, nx, ny, nz, nt)
case (PVTLM)
    out4d = diag_PVTLM(ncid, nx, ny, nz, nt)

```

```

case (PVPert)
  out4d = diag_PVPert(ncid, nx, ny, nz, nt)
case (PVAdj)
  out4d = diag_PVAdj(ncid, nx, ny, nz, nt)
case (CostGradPhi)
  out4d = diag_CostGradPhi(ncid, nx, ny, nz, nt)
case (InvertPhi)
  out4d = diag_InvertPhi(ncid, nx, ny, nz, nt)
case (InvertPsi)
  out4d = diag_InvertPsi(ncid, nx, ny, nz, nt)
case default
  write (*,"(a,i3,a)") "WARNING: Field ", var%diagID, &
    " unknown in diag_out!"
end select
! Write the diagnostic out to GEMPAK
if (ok) then
  do t = 1, nt
    gdat1 = times(t)
    if (var%dimType == 1) then
      call write_gempak(gemid, out3d(:, :, t), nx, ny, gdat1, var%lev1, &
        var%lev2, var%vertCordID, var%gemName)
    else
      do k = 1, kMax
        lev = nint(elvl(k,1) * 10000)
        call write_gempak(gemid, out4d(:, :, k, t), nx, ny, gdat1, lev, &
          var%lev2, var%vertCordID, var%gemName)
      end do
    end if
  end do
end if
! Clean up
if (allocated(out3d)) then
  deallocate (out3d, stat=status)
else
  deallocate (out4d, elvl, stat=status)
end if
call mem_error(status, 2, "diag_out")
end subroutine diag_out
! =====
! pres_diag_out is a driver routine that produces the diagnostic var
! interpolated to pressure surfaces based on data from the WRF history file
! associated with ncid. The diagnostic is written out to the GEMPAK file
! associated with gemid. Other inputs to the subroutine are the WRF model
! dimensions (nx,ny,nz) and the array of times in GEMPAK form to be output,
! which corresponds to all of the times in the history file. np is obtained
! through use association.
subroutine pres_diag_out(ncid, gemid, nx, ny, nz, var, times)
  integer,          intent(in) :: ncid, gemid, nx, ny, nz
  type(pres_diag_var), intent(in) :: var
  character(len=15), dimension(:), intent(in) :: times

```

```

real, dimension(:,:,:), allocatable :: tk
real, dimension(nx,ny,nz+1,size(times)) :: outNoInterp, p
real, dimension(nx,ny,np,size(times)) :: outInterp
integer :: nt, t, j, i, status, k, lev
character(len=20) :: gdat1
nt = size(times)
! Compute appropriate diagnostic
select case (var%diagID)
case (TmpKPres)
  outNoInterp(:,:,1,:) = diag_SfcTempF0(ncid, nx, ny, nz, nt)
  outNoInterp(:,:,2:nz+1,:) = diag_TmpK(ncid, nx, ny, nz, nt)
case (UUnstagPres)
  outNoInterp(:,:,1,:) = diag_SfcUF0(ncid, nx, ny, nz, nt)
  outNoInterp(:,:,2:nz+1,:) = diag_UUnstag(ncid, nx, ny, nz, nt)
case (VUnstagPres)
  outNoInterp(:,:,1,:) = diag_SfcVF0(ncid, nx, ny, nz, nt)
  outNoInterp(:,:,2:nz+1,:) = diag_VUnstag(ncid, nx, ny, nz, nt)
case (WUnstagPres)
  outNoInterp(:,:,1,:) = 0
  outNoInterp(:,:,2:nz+1,:) = diag_WUnstag(ncid, nx, ny, nz, nt)
case (GeopHghtPres)
  call read_values(ncid, wrfVar(TerHght)%wrfName, outNoInterp(:,:,1,:))
  outNoInterp(:,:,2:nz+1,:) = diag_GeopHght(ncid, nx, ny, nz, nt)
case (SpHumPres)
  outNoInterp(:,:,1,:) = diag_SfcSpHum(ncid, nx, ny, nz, nt)
  outNoInterp(:,:,2:nz+1,:) = diag_SpHum(ncid, nx, ny, nz, nt)
case default
  write (*,"(a,i3,a)") "WARNING: Field ", var, &
    " unknown in pres_diag_out!"
end select
! Need pressure for interpolation
p(:,:,1,:) = diag_SfcP(ncid, nx, ny, nz, nt)
p(:,:,2:nz+1,:) = diag_Pressure(ncid, nx, ny, nz, nt)
do t = 1, nt
  do j = 1, ny
    do i = 1, nx
      call vint(outNoInterp(i,j,:,t), p(i,j,:,t), pBot, -dp, &
        var%interpType, outInterp(i,j,:,t))
    end do
  end do
end do
! Interpolate underground for heights
if (var%diagID == GeopHghtPres) then
  allocate (tk(nx,ny,nz,nt), stat=status)
  call mem_error(status, 1, "pres_diag_out")
  tk = diag_TmpK(ncid, nx, ny, nz, nt)
  call hydro_interp(outNoInterp(:,:,1,:), p(:,:,1,:), tk(:,:,1,:), pBot, &
    -dp, outInterp)
  deallocate (tk, stat=status)
  call mem_error(status, 2, "pres_diag_out")

```



```

end if
! Write the diagnostic out to GEMPAK
do t = 1, nt
  gdat1 = times(t)
  do k = 1, np
    lev = pBot - (k-1) * dp
    call write_gempak(gemid, outInterp(:, :, k, t), nx, ny, gdat1, lev, &
      -1, Pres, var%gemName)
  end do
end do
end subroutine pres_diag_out
! =====
! The next group of functions produce the various diagnostics. They all take
! as input the WRF history file handle (ncid), the WRF domain size
! (nx, ny, and sometimes nz), and the number of times contained in the
! history file (nt).
! diag_SfcP diagnoses the surface pressure (in hPa).
function diag_SfcP(ncid, nx, ny, nz, nt) result (ps)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nt) :: ps
  character(len=9), parameter :: funName = "diag_SfcP"
  real, dimension(:, :, :, :), allocatable :: p, w, t, ht
  real, dimension(nx,ny,2,nt) :: ht12
  real, dimension(nx,ny,nt) :: p1, q1, sfcq, t1
  integer :: status
  ! Read in pressure, saving only lowest level
  allocate (p(nx,ny,nz,nt), stat=status)
  call mem_error(status, 1, funName//" p")
  p = diag_Pressure(ncid, nx, ny, nz, nt, si=.true.) ! Do calculations in Pa
  p1 = p(:, :, 1, :)
  deallocate (p, stat=status)
  call mem_error(status, 2, funName//" p")
  ! Handle specific humidity
  allocate (w(nx,ny,nz,nt), stat=status)
  call mem_error(status, 1, funName//" w")
  call read_values(ncid, wrfVar(MixR)%wrfName, w)
  q1 = mix_ratio_to_spec_hum(w(:, :, 1, :))
  deallocate (w, stat=status)
  call mem_error(status, 2, funName//" w")
  sfcq = mix_ratio_to_spec_hum(diag_SfcMixRFO(ncid, nx, ny, nz, nt))
  ! Handle temperature
  allocate (t(nx,ny,nz,nt), stat=status)
  call mem_error(status, 1, funName//" t")
  t = diag_Theta(ncid, nx, ny, nz, nt)
  t1 = temp_from_theta_p(t(:, :, 1, :), p1)
  deallocate (t, stat=status)
  call mem_error(status, 2, funName//" t")
  ! Handle geopotential height
  allocate (ht(nx,ny,nz+1,nt), stat=status)
  call mem_error(status, 1, funName//" ht")

```

```

    ht = diag_GeopHghtS(ncid, nx, ny, nz, nt)
    ht12 = ht(:, :, 1:2, :)
    deallocate (ht, stat=status)
    call mem_error(status, 2, funName// " ht")
    ps = .01 * surface_pres(ht12, p1, q1, t1, sfcq) ! Convert to hPa
end function diag_SfcP
! =====
! diag_SfcP2m diagnoses the pressure (in hPa) 2 meters above the surface.
! The density is approximated by using surface pressure and 2-m temperature.
function diag_SfcP2m(ncid, nx, ny, nz, nt) result (ps2m)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nt) :: ps, t2, q, rhog, ps2m
    ps = diag_SfcP(ncid, nx, ny, nz, nt) * 100 ! In Pa
    t2 = diag_SfcTempF0(ncid, nx, ny, nz, nt)
    q = diag_SfcMixRF0(ncid, nx, ny, nz, nt)
    rhog = Grav * density(ps, t2, q)
    ! Use hydrostatic equation
    ps2m = .01 * (ps - 2*rhog) ! In hPa
end function diag_SfcP2m
! =====
! diag_SeaLevP diagnoses the sea level pressure (in hPa).
function diag_SeaLevP(ncid, nx, ny, nz, nt) result (slp)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nt) :: slp
    real, dimension(nx,ny,nz+1,nt) :: ht
    real, dimension(nx,ny,nz,nt)   :: th, p, w
    ! Read in WRF data
    ht = diag_GeopHghtS(ncid, nx, ny, nz, nt)
    th = diag_Theta(ncid, nx, ny, nz, nt)
    p = diag_Pressure(ncid, nx, ny, nz, nt, si=.true.) ! Do calculations in Pa
    call read_values(ncid, wrfVar(MixR)%wrfName, w)
    ! Compute sea level pressure
    slp = .01 * sea_pres(ht, th, p, mix_ratio_to_spec_hum(w)) ! Convert to hPa
end function diag_SeaLevP
! =====
! diag_Pressure diagnoses pressure. By default, the output is in hPa, but
! if si is true, the output is in Pa.
function diag_Pressure(ncid, nx, ny, nz, nt, si) result (p)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    logical, intent(in), optional :: si
    real, dimension(nx,ny,nz,nt) :: p
    real, dimension(nx,ny,nz,nt) :: pb
    logical                     :: hPa
    ! Set appropriate output unit flag
    if (present(si)) then
        hPa = .not. si
    else
        hPa = .true.
    end if
    ! Read in WRF data

```

```

call read_values(ncid, wrfVar(PBase)%wrfName, pb)
call read_values(ncid, wrfVar(PPert)%wrfName, p)
! Calculate pressure
p = p + pb
if (hPa) p = .01 * p
end function diag_Pressure
! =====
! diag_Theta diagnoses potential temperature (K).
function diag_Theta(ncid, nx, ny, nz, nt) result (theta)
integer, intent(in) :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: theta
call read_values(ncid, wrfVar(ThetaPert)%wrfName, theta)
theta = theta + 300 ! WRF stores theta as theta-300
end function diag_Theta
! =====
! diag_UUnstag diagnoses the wind component (m/s) in the x-coordinate
! direction at the mass points (grid box centers).
function diag_UUnstag(ncid, nx, ny, nz, nt) result (u)
integer, intent(in) :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: u
real, dimension(nx+1,ny,nz,nt) :: us
! Read staggered wind
call read_values(ncid, wrfVar(UStag)%wrfName, us)
! Unstagger it
u(1:nx, :, :, :) = .5 * (us(1:nx, :, :, :) + us(2:nx+1, :, :, :))
end function diag_UUnstag
! =====
! diag_VUnstag diagnoses the wind component (m/s) in the y-coordinate
! direction at the mass points (grid box centers).
function diag_VUnstag(ncid, nx, ny, nz, nt) result (v)
integer, intent(in) :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: v
real, dimension(nx,ny+1,nz,nt) :: vs
! Read staggered wind
call read_values(ncid, wrfVar(VStag)%wrfName, vs)
! Unstagger it
v(:, 1:ny, :, :) = .5 * (vs(:, 1:ny, :, :) + vs(:, 2:ny+1, :, :))
end function diag_VUnstag
! =====
! diag_CWtr diagnoses the cloud water mixing ratio (kg/kg), including ice.
function diag_CWtr(ncid, nx, ny, nz, nt) result (cw)
integer, intent(in) :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: cw
real, dimension(nx,ny,nz,nt) :: cloud, ice
call read_values(ncid, wrfVar(MixRCloud)%wrfName, cloud)
call read_values(ncid, wrfVar(MixRIce)%wrfName, ice)
cw = cloud + ice
end function diag_CWtr
! =====
! diag_WUnstag diagnoses the wind component (m/s) in the vertical direction

```

```

! at the mass points (grid box centers).
function diag_WUnstag(ncid, nx, ny, nz, nt) result (w)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: w
  real, dimension(nx,ny,nz+1,nt) :: ws
  ! Read staggered wind
  call read_values(ncid, wrfVar(WStag)%wrfName, ws)
  ! Unstagger it
  w(:,:,1:nz,:) = .5 * (ws(:,:,1:nz,:) + ws(:,:,2:nz+1,:))
end function diag_WUnstag
! =====
! diag_GeopHghtS diagnoses geopotential height (m) on w (full) levels.
function diag_GeopHghtS(ncid, nx, ny, nz, nt) result (height)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz+1,nt) :: height
  real, dimension(nx,ny,nz+1,nt) :: gpb, gpp
  call read_values(ncid, wrfVar(GeopBase)%wrfName, gpb)
  call read_values(ncid, wrfVar(GeopPert)%wrfName, gpp)
  height = (gpb + gpp) / Grav
end function diag_GeopHghtS
! =====
! diag_SfcTempF0 diagnoses 2-m temperature (K), including the initial time.
function diag_SfcTempF0(ncid, nx, ny, nz, nt) result (temp)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nt) :: temp
  real, dimension(nx,ny,nz,nt) :: potTemp, p
  real, dimension(nx,ny)      :: tk1
  ! For initial time, calculate temperature in lowest grid box.
  potTemp = diag_Theta(ncid, nx, ny, nz, nt)
  p = diag_Pressure(ncid, nx, ny, nz, nt, si = .true.) ! p in Pa
  tk1 = temp_from_theta_p(potTemp(:,:,1,1), p(:,:,1,1))
  ! Read in WRF 2-m temperature, but WRF doesn't include this variable at
  ! initial time, so assign the temperature in the lowest grid box to it.
  call read_values(ncid, wrfVar(SfcTemp)%wrfName, temp)
  temp(:,:,1) = tk1
end function diag_SfcTempF0
! =====
! diag_SfcMixRF0 diagnoses 2-m mixing ratio (kg/kg), including the initial
! time.
function diag_SfcMixRF0(ncid, nx, ny, nz, nt) result (w2m)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nt) :: w2m
  real, dimension(nx,ny,nz,nt) :: w
  ! Read in WRF 2-m mixing ratio, but WRF doesn't include this variable at
  ! initial time, so assign the mixing ratio in the lowest grid box to it.
  call read_values(ncid, wrfVar(MixR)%wrfName, w)
  call read_values(ncid, wrfVar(SfcMixR)%wrfName, w2m)
  w2m(:,:,1) = w(:,:,1,1)
end function diag_SfcMixRF0
! =====

```

```

! diag_SfcUf0 diagnoses the 2-m wind component (m/s) in the x-coordinate
! direction, including the initial time.
function diag_SfcUf0(ncid, nx, ny, nz, nt) result (u2m)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nt) :: u2m
    real, dimension(nx,ny,nz,nt) :: u
    ! Read in WRF 2-m u-component wind, but WRF doesn't include this variable
    ! at initial time, so assign the u-component wind in the lowest grid box to
    ! it.
    u = diag_UUnstag(ncid, nx, ny, nz, nt)
    call read_values(ncid, wrfVar(SfcU)%wrfName, u2m)
    u2m(:,:,1) = u(:,:,1,1)
end function diag_SfcUf0
! =====
! diag_SfcVf0 diagnoses the 2-m wind component (m/s) in the y-coordinate
! direction, including the initial time.
function diag_SfcVf0(ncid, nx, ny, nz, nt) result (v2m)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nt) :: v2m
    real, dimension(nx,ny,nz,nt) :: v
    ! Read in WRF 2-m v-component wind, but WRF doesn't include this variable
    ! at initial time, so assign the v-component wind in the lowest grid box to
    ! it.
    v = diag_VUnstag(ncid, nx, ny, nz, nt)
    call read_values(ncid, wrfVar(SfcV)%wrfName, v2m)
    v2m(:,:,1) = v(:,:,1,1)
end function diag_SfcVf0
! =====
! diag_LiftIdx diagnoses the Lifted Index (K).
function diag_LiftIdx(ncid, nx, ny, nz, nt) result (li)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nt) :: li
    real, dimension(nx,ny,nz,nt) :: th, p, w
    integer :: i, j, t
    ! Read in the necessary data from WRF output.
    th = diag_Theta(ncid, nx, ny, nz, nt)
    p = diag_Pressure(ncid, nx, ny, nz, nt) ! p in hPa
    call read_values(ncid, wrfVar(MixR)%wrfName, w)
    ! Calculate lifted index, one profile at a time.
    do t = 1, nt
        do j = 1, ny
            do i = 1, nx
                li(i,j,t) = lifted_index(th(i,j,:,t), p(i,j,:,t), w(i,j,:,t))
            end do
        end do
    end do
end function diag_LiftIdx
! =====
! diag_MeanRH diagnoses the mean relative humidity (%) in the 850-500 mb
! layer.

```

```

function diag_MeanRH(ncid, nx, ny, nz, nt) result (mrh)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nt) :: mrh
  real, dimension(nx,ny,nz,nt) :: th, p, w
  real, dimension(nx,ny,nt)   :: ps
  integer :: i, j, t
  ! Read in the necessary data from WRF output.
  th = diag_Theta(ncid, nx, ny, nz, nt)
  p = diag_Pressure(ncid, nx, ny, nz, nt) ! p in hPa
  call read_values(ncid, wrfVar(MixR)%wrfName, w)
  ps = diag_SfcP(ncid, nx, ny, nz, nt) ! ps in hPa
  ! Calculate mean RH, one profile at a time.
  do t = 1, nt
    do j = 1, ny
      do i = 1, nx
        mrh(i,j,t) = mean_rh(th(i,j,:,t), p(i,j,:,t), w(i,j,:,t), &
                             ps(i,j,t), 850., 500.)
      end do
    end do
  end do
end function diag_MeanRH
! =====
! diag_SfcSpHum diagnoses 2-m specific humidity (kg/kg).
function diag_SfcSpHum(ncid, nx, ny, nz, nt) result (q2m)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nt) :: q2m
  q2m = mix_ratio_to_spec_hum(diag_SfcMixRF0(ncid, nx, ny, nz, nt))
end function diag_SfcSpHum
! =====
! diag_SpHum diagnoses specific humidity (kg/kg).
function diag_SpHum(ncid, nx, ny, nz, nt) result (q)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: q
  real, dimension(nx,ny,nz,nt) :: w
  call read_values(ncid, wrfVar(MixR)%wrfName, w)
  q = mix_ratio_to_spec_hum(w)
end function diag_SpHum
! =====
! diag_TmpK diagnoses temperature (K).
function diag_TmpK(ncid, nx, ny, nz, nt) result (tk)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: tk
  tk = temp_from_theta_p(diag_Theta(ncid, nx, ny, nz, nt), &
                        diag_Pressure(ncid, nx, ny, nz, nt, si=.true.))
end function diag_TmpK
! =====
! diag_GeopHght diagnoses geopotential height (m) on half (theta) levels.
function diag_GeopHght(ncid, nx, ny, nz, nt) result (hght)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: hght

```

```

real, dimension(:,:), allocatable :: etaTemp
real, dimension(nx,ny,nz+1,nt)    :: hghtStag
real, dimension(nx,ny,nt)         :: mu
real, dimension(1,nt)             :: pt ! really a constant
real, dimension(nz+1)             :: znw
real, dimension(nz)               :: znu
integer :: status, i, j, t
hghtStag = diag_GeopHghtS(ncid, nx, ny, nz, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
allocate(etaTemp(nz+1,nt), stat=status)
call mem_error(status, 1, "diag_GeopHght")
call read_values(ncid, wrfVar(EtaW)%wrfName, etaTemp)
znw = etaTemp(:,1)
deallocate(etaTemp, stat=status)
call mem_error(status, 2, "diag_GeopHght")
allocate(etaTemp(nz,nt), stat=status)
call mem_error(status, 1, "diag_GeopHght")
call read_values(ncid, wrfVar(EtaMass)%wrfName, etaTemp)
znu = etaTemp(:,1)
deallocate(etaTemp, stat=status)
call mem_error(status, 2, "diag_GeopHght")
call read_values(ncid, "P_TOP", pt)
! Unstagger height by column.
do t = 1, nt
  do j = 1, ny
    do i = 1, nx
      hght(i,j,:,t) = &
        unstag_hght(hghtStag(i,j,:,t), znw, znu, mu(i,j,t), pt(1,1))
    end do
  end do
end do
end function diag_GeopHght
! =====
! diag_DryAir diagnoses the dry air mass in the column (Pa).
function diag_DryAir(ncid, nx, ny, nt) result (mu)
  integer, intent(in)      :: ncid, nx, ny, nt
  real, dimension(nx,ny,nt) :: mu
  real, dimension(nx,ny,nt) :: mub
  call read_values(ncid, wrfVar(MuBase)%wrfName, mub)
  call read_values(ncid, wrfVar(MuPert)%wrfName, mu)
  mu = mub + mu
end function diag_dryAir
! =====
function diag_MCap(ncid, nx, ny, nt) result(mCap)
  integer, intent(in)      :: ncid, nx, ny, nt
  real, dimension(nx,ny,nt) :: mCap
  real, parameter :: p0 = 100000.
  real :: pt
  call read_values_const(ncid, "P_TOP", pt)
  mCap = diag_dryAir(ncid, nx, ny, nt) / (p0 - pt)

```

```

end function diag_MCap
! =====
function diag_MScript1(ncid, nx, ny, nz, nt) result(ms1)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx+1,ny,nz,nt) :: ms1
  real, dimension(nx+1,ny,nz,nt) :: us
  real, dimension(nx,ny,nt)     :: mCap
  call read_values(ncid, wrfVar(USTag)%wrfName, us)
  mCap = diag_MCap(ncid, nx, ny, nt)
  ms1(2:nx, :, :, :) = us(2:nx, :, :, :) * Half * &
    spread(mCap(1:nx-1, :, :, :) + mCap(2:nx, :, :, :), 3, nz)
  ms1((/1,nx+1/), :, :, :) = us((/1,nx+1/), :, :, :) * &
    spread(mCap((/1,nx/), :, :, :), 3, nz)
end function diag_MScript1
! =====
function diag_MScript2(ncid, nx, ny, nz, nt) result(ms2)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny+1,nz,nt) :: ms2
  real, dimension(nx,ny+1,nz,nt) :: vs
  real, dimension(nx,ny,nt)     :: mCap
  call read_values(ncid, wrfVar(VStag)%wrfName, vs)
  mCap = diag_MCap(ncid, nx, ny, nt)
  ms2(:, 2:ny, :, :) = vs(:, 2:ny, :, :) * Half * &
    spread(mCap(:, 1:ny-1, :, :) + mCap(:, 2:ny, :, :), 3, nz)
  ms2(:, (/1,ny+1/), :, :) = vs(:, (/1,ny+1/), :, :) * &
    spread(mCap(:, (/1,ny/), :, :), 3, nz)
end function diag_MScript2
! =====
function diag_MS1Unstag(ncid, nx, ny, nz, nt) result (ms1u)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: ms1u
  real, dimension(nx+1,ny,nz,nt) :: ms1
  ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
  ms1u = Half * (ms1(:,nx, :, :, :) + ms1(2:, :, :, :))
end function diag_MS1Unstag
! =====
function diag_MS2Unstag(ncid, nx, ny, nz, nt) result (ms2u)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: ms2u
  real, dimension(nx,ny+1,nz,nt) :: ms2
  ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
  ms2u = Half * (ms2(:, :,ny, :, :) + ms2(:, 2:, :, :))
end function diag_MS2Unstag
! =====
function diag_Diverg(ncid, nx, ny, nz, nt) result(div)
  integer, intent(in)          :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: div
  real, dimension(nx,ny,nz,nt) :: u, v
  real, dimension(nx,ny)      :: dudx, dvdy, mfm
  real      :: rdx

```



```

integer :: t, k
u = diag_UUnstag(ncid, nx, ny, nz, nt)
v = diag_VUnstag(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  do k = 1, nz
    dudx(2:nx-1,:) = rdx * Half * mfm(2:nx-1,:) * &
      ((u(3:nx,:,k,t) - u(1:nx-2,:,k,t)) - &
       u(2:nx-1,:,k,t) * (mfm(3:nx,:) - mfm(1:nx-2,:)))
    dudx((/1,nx/),:) = rdx * mfm((/1,nx/),:) * &
      ((u((/2,nx/),:,k,t) - u((/1,nx-1/),:,k,t)) - &
       u((/1,nx/),:,k,t) * (mfm((/2,nx/),:) - mfm((/1,nx-1/),:)))
    dvdy(:,2:ny-1) = rdx * Half * mfm(:,2:ny-1) * &
      ((v(:,3:ny,k,t) - v(:,1:ny-2,k,t)) - &
       v(:,2:ny-1,k,t) * (mfm(:,3:ny) - mfm(:,1:ny-2)))
    dvdy(:,(/1,ny/)) = rdx * mfm(:,(/1,ny/)) * &
      ((v(:,(/2,ny/),k,t) - v(:,(/1,ny-1/),k,t)) - &
       v(:,(/1,ny/),k,t) * (mfm(:,(/2,ny/)) - mfm(:,(/1,ny-1/)))))
    div(:, :, k, t) = dudx + dvdy
  end do
end do
end function diag_Diverg
! =====
function diag_DScript(ncid, nx, ny, nz, nt) result(dScr)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: dScr
  real, dimension(nx+1,ny,nz,nt) :: mScr1
  real, dimension(nx,ny+1,nz,nt) :: mScr2
  real, dimension(nx+1,ny) :: mfu
  real, dimension(nx,ny+1) :: mfv
  real, dimension(nx,ny) :: mfm
  real :: rdx
  integer :: t, k
  mScr1 = diag_MScript1(ncid, nx, ny, nz, nt)
  mScr2 = diag_MScript2(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, "RDX", rdx)
  do t = 1, nt
    do k = 1, nz
      dScr(:, :, k, t) = mfm**2 * ( &
        cen_diff_stag(mScr1(:, :, k, t)/mfu, rdx, 1) + &
        cen_diff_stag(mScr2(:, :, k, t)/mfv, rdx, 2))
    end do
  end do
end function diag_DScript
! =====
function diag_DDDt(ncid, nx, ny, nz, nt) result(dd)

```

```

integer, intent(in)          :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: dd
real, parameter :: dtr = One / 3600.
real, dimension(nx,ny,nz,nt) :: dScr
dScr = diag_DScrip(ncid, nx, ny, nz, nt)
dd = cen_diff(dScr, dtr, 4)
print *, maxval(abs(dd(:,:,18,40))), maxloc(abs(dd(:,:,18,40)))
print *, mean(abs(dd(:,:,18,40)))
end function diag_DDDt
! =====
function diag_EtaDot(ncid, nx, ny, nz, nt) result(ed)
integer, intent(in)          :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: ed
real, dimension(nx,ny,nz+1,nt) :: z
real, dimension(nz+1)         :: eta
z = diag_GeopHghtS(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, eta)
ed = diag_WUnStag(ncid, nx, ny, nz, nt) * &
    spread(spread(spread(eta(2:)-eta(:nz),1,nx),2,ny),4,nt) / &
    (z(:,:,2:,:) - z(:,:,1:nz,:))
end function diag_EtaDot
! =====
function diag_TermA2(ncid, nx, ny, nz, nt) result(a2)
integer, intent(in)          :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: a2
real, dimension(nz)          :: eta
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, eta)
a2 = diag_EtaDot(ncid, nx, ny, nz, nt) * &
    uneven_deriv(diag_DScrip(ncid, nx, ny, nz, nt), eta, 3)
print *, maxval(abs(a2(:,:,18,40))), maxloc(abs(a2(:,:,18,40)))
print *, mean(abs(a2(:,:,18,40)))
end function diag_TermA2
! =====
function diag_TermA3(ncid, nx, ny, nz, nt) result(a3)
integer, intent(in)          :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: a3
real, dimension(nx+1,ny,nz,nt) :: ms1
real, dimension(nx,ny+1,nz,nt) :: ms2
real, dimension(nx,ny,nz,nt)   :: eDot
real, dimension(nx,ny)         :: mfm
real, dimension(nz)            :: eta
real                           :: rdx
ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
eDot = diag_EtaDot(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, eta)
call read_values_const(ncid, "RDX", rdx)
a3 = spread(spread(mfm,3,nz),4,nt) * Half * ( &
    cen_diff(eDot, rdx, 1) * &

```

```

        uneven_deriv(ms1(:,nx,::,::) + ms1(2:,:,::,::), eta, 3) + &
        cen_diff(eDot, rdx, 2) * &
        uneven_deriv(ms2(:,ny,::,::) + ms2(:,2:,:,::), eta, 3) )
    print *, maxval(abs(a3(:,::,18,40))), maxloc(abs(a3(:,::,18,40)))
    print *, mean(abs(a3(:,::,18,40)))
end function diag_TermA3
! =====
function diag_TermA4(ncid, nx, ny, nz, nt) result(a4)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: a4
    real, parameter :: dtr = One / 3600.
    a4 = -diag_DScript(ncid, nx, ny, nz, nt) * &
        cen_diff(spread(log(diag_MCap(ncid, nx, ny, nt)),3,nz), dtr, 4)
    print *, maxval(abs(a4(:,::,18,40))), maxloc(abs(a4(:,::,18,40)))
    print *, mean(abs(a4(:,::,18,40)))
end function diag_TermA4
! =====
function diag_TermA5(ncid, nx, ny, nz, nt) result(a5)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: a5
    real, parameter :: dtr = One / 3600.
    real, dimension(nx+1,ny,nz,nt) :: ms1
    real, dimension(nx,ny+1,nz,nt) :: ms2
    real, dimension(nx,ny,nz,nt)   :: dlnmdt
    real, dimension(nx,ny)         :: mfm
    real                           :: rdx
    ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
    ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, "RDX", rdx)
    dlnmdt = cen_diff(spread(log(diag_MCap(ncid, nx, ny, nt)),3,nz), dtr, 4)
    a5 = spread(spread(mfm,3,nz),4,nt) * Half * &
        (-ms1(:,nx,::,::) + ms1(2:,:,::,::)) * cen_diff(dlnmdt, rdx, 1) - &
        (ms2(:,ny,::,::) + ms2(:,2:,:,::)) * cen_diff(dlnmdt, rdx, 2))
    print *, maxval(abs(a5(:,::,18,40))), maxloc(abs(a5(:,::,18,40)))
    print *, mean(abs(a5(:,::,18,40)))
end function diag_TermA5
! =====
function diag_TermB1(ncid, nx, ny, nz, nt) result(b1)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: b1
    real, dimension(nx+1,ny,nz,nt) :: us, ms1
    real, dimension(nx,ny+1,nz,nt) :: vs, ms2
    real, dimension(nx,ny,nz,nt)   :: dvdx, dudy, dms2dx, dms1dy
    real, dimension(nx,ny)         :: mfm
    real                           :: rdx
    integer :: k, t
    call read_values(ncid, wrfVar(USTag)%wrfName, us)
    ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
    call read_values(ncid, wrfVar(VSTag)%wrfName, vs)

```

```

ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, "RDX", rdx)
dvdv = Half * cen_diff(vs(:,ny,,:,:) + vs(:,2,,:,:), rdx, 1)
dudy = Half * cen_diff(us(:,nx,,:,:) + us(2,,:,:), rdx, 2)
dms2dx = Half * cen_diff(ms2(:,ny,,:,:) + ms2(:,2,,:,:), rdx, 1)
dms1dy = Half * cen_diff(ms1(:,nx,,:,:) + ms1(2,,:,:), rdx, 2)
do t = 1, nt
  do k = 1, nz
    b1(:,k,t) = mfm**2 * (cen_diff_stag(us(:,k,t), rdx, 1) * &
      cen_diff_stag(ms1(:,k,t), rdx, 1) + dvdv(:,k,t) * &
      dms1dy(:,k,t) + dudy(:,k,t) * dms2dx(:,k,t) + &
      cen_diff_stag(vs(:,k,t), rdx, 2) * &
      cen_diff_stag(ms2(:,k,t), rdx, 2))
  end do
end do
print *, maxval(abs(b1(:,18,40))), maxloc(abs(b1(:,18,40)))
print *, mean(abs(b1(:,18,40)))
end function diag_TermB1
! =====
function diag_TermB2(ncid, nx, ny, nz, nt) result(b2)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: b2
  real, dimension(nx+1,ny,nz,nt) :: ms1
  real, dimension(nx,ny+1,nz,nt) :: ms2
  real, dimension(nx,ny,nz,nt) :: u, v, lnm, vdglm, mfm4d
  real, dimension(nx,ny) :: mfm
  real :: rdx
  ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
  ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
  u = diag_UUnstag(ncid, nx, ny, nz, nt)
  v = diag_VUnstag(ncid, nx, ny, nz, nt)
  lnm = spread(log(diag_MCap(ncid, nx, ny, nt)),3,nz)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, "RDX", rdx)
  mfm4d = spread(spread(mfm,3,nz),4,nt)
  vdglm = mfm4d * (u * cen_diff(lnm, rdx, 1) + v * cen_diff(lnm, rdx, 2))
  b2 = -Half * mfm4d * ( &
    (ms1(:,nx,,:,:) + ms1(2,,:,:),) * cen_diff(vdglm, rdx, 1) + &
    (ms2(:,ny,,:,:) + ms2(:,2,,:,:),) * cen_diff(vdglm, rdx, 2) )
  print *, maxval(abs(b2(:,18,40))), maxloc(abs(b2(:,18,40)))
  print *, mean(abs(b2(:,18,40)))
end function diag_TermB2
! =====
function diag_TermB3(ncid, nx, ny, nz, nt) result(b3)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: b3
  real, dimension(nx,ny,nz,nt) :: u, v, dScr, lnm
  real, dimension(nx,ny) :: mfm
  real :: rdx

```

```

u = diag_UUnstag(ncid, nx, ny, nz, nt)
v = diag_VUnstag(ncid, nx, ny, nz, nt)
dScr = diag_DScript(ncid, nx, ny, nz, nt)
lnm = spread(log(diag_MCap(ncid, nx, ny, nt)),3,nz)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, "RDX", rdx)
b3 = spread(spread(mfm,3,nz),4,nt) * &
      (u * cen_diff(dScr, rdx, 1) + v * cen_diff(dScr, rdx, 2) - &
      dScr * (u * cen_diff(lnm, rdx, 1) + v * cen_diff(lnm, rdx, 2)))
print *, maxval(abs(b3(:,:,18,40))), maxloc(abs(b3(:,:,18,40)))
print *, mean(abs(b3(:,:,18,40)))
end function diag_TermB3
! =====
function diag_TermC(ncid, nx, ny, nz, nt) result(c)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: c
  real, dimension(nx+1,ny,nz,nt) :: ms1
  real, dimension(nx,ny+1,nz,nt) :: ms2
  real, dimension(nx,ny,nz,nt)   :: mfm4d, ms1u, ms2u
  real, dimension(nx,ny)         :: mfm, f
  real :: rdx
  ms1 = diag_MScript1(ncid, nx, ny, nz, nt)
  ms2 = diag_MScript2(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
  call read_values_const(ncid, "RDX", rdx)
  mfm4d = spread(spread(mfm,3,nz),4,nt)
  ms1u = Half * (ms1(:nx,:,:,) + ms1(2:,:,:,))
  ms2u = Half * (ms2(:,ny:,:,:) + ms2(:,2:,:,:,))
  c = -spread(spread(f,3,nz),4,nt) * &
      (mfm4d * cen_diff(ms2u, rdx, 1) - mfm4d * cen_diff(ms1u, rdx, 2) - &
      ms2u * cen_diff(mfm4d, rdx, 1) + ms1u * cen_diff(mfm4d, rdx, 2))
  print *, maxval(abs(c(:,:,18,40))), maxloc(abs(c(:,:,18,40)))
  print *, mean(abs(c(:,:,18,40)))
end function diag_TermC
! =====
function diag_TermD(ncid, nx, ny, nz, nt) result(d)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: d
  real, dimension(nx,ny,nz,nt) :: f4d
  real, dimension(nx,ny)       :: mfm, f
  real :: rdx
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
  call read_values_const(ncid, "RDX", rdx)
  f4d = spread(spread(f,3,nz),4,nt)
  d = spread(spread(mfm,3,nz),4,nt) * ( &
      diag_MS1Unstag(ncid, nx, ny, nz, nt) * cen_diff(f4d, rdx, 2) - &
      diag_MS2Unstag(ncid, nx, ny, nz, nt) * cen_diff(f4d, rdx, 1) )
  print *, maxval(abs(d(:,:,18,40))), maxloc(abs(d(:,:,18,40)))

```

```

    print *, mean(abs(d(:,:,18,40)))
end function diag_TermD
! =====
function diag_WaterInv(ncid, nx, ny, nz, nt) result(chi)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: chi
    real, dimension(nx,ny,nz,nt) :: qv, qc, qr
    call read_values(ncid, wrfVar(MixR)%wrfName, qv)
    call read_values(ncid, wrfVar(MixRCloud)%wrfName, qc)
    call read_values(ncid, wrfVar(MixRRain)%wrfName, qr)
    chi = One / (qr + qc + qv + One)
end function diag_WaterInv
! =====
function diag_VapPres(ncid, nx, ny, nz, nt) result(e)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: e
    real, dimension(nx,ny,nz,nt) :: p, qv
    p = diag_Pressure(ncid, nx, ny, nz, nt, si=.true.)
    call read_values(ncid, wrfVar(MixR)%wrfName, qv)
    e = p / (One + Eps/qv)
end function diag_VapPres
! =====
function diag_GeopS(ncid, nx, ny, nz, nt) result(phi)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz+1,nt) :: phi
    real, dimension(nx,ny,nz+1,nt) :: gpb, gpp
    call read_values(ncid, wrfVar(GeopBase)%wrfName, gpb)
    call read_values(ncid, wrfVar(GeopPert)%wrfName, gpp)
    phi = gpb + gpp
end function diag_GeopS
! =====
function diag_TermE(ncid, nx, ny, nz, nt) result(e)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: e
    real, dimension(nx,ny,nz+1,nt) :: phis
    real, dimension(nx,ny,nz,nt)   :: chi, pv, phi, eta4d, mur4d, m4d, &
                                     dphideta, mfm4d

    real, dimension(nx,ny,nt)      :: mu, m
    real, dimension(nx,ny)         :: mfm
    real, dimension(nz+1)          :: etaz
    real, dimension(nz)            :: etam
    real    :: rdx, pt
    integer :: i, j, t
    print *, 1
    phis = diag_GeopS(ncid, nx, ny, nz, nt)
    print *, 2
    chi = One ! For ideal case
!   chi = diag_WaterInv(ncid, nx, ny, nz, nt)
    print *, 3
    pv = diag_VapPres(ncid, nx, ny, nz, nt)

```

```

print *, 4
mu = diag_DryAir(ncid, nx, ny, nt)
print *, 5
m = diag_MCap(ncid, nx, ny, nt)
print *, 6
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "RDX", rdx)
call read_values_const(ncid, "P_TOP", pt)
do t = 1, nt
  do j = 1, ny
    do i = 1, nx
      phi(i,j,:,t) = &
        unstag_hght(phis(i,j,:,t), etaz, etam, mu(i,j,t), pt)
    end do
  end do
end do
eta4d = spread(spread(spread(etam,1,nx),2,ny),4,nt)
mur4d = spread(One/mu, 3, nz)
m4d = spread(m,3,nz)
dphideta = (phis(:, :, 2:,:) - phis(:, :, :nz,:)) / &
  spread(spread(spread(etaz(2:) - etaz(:nz),1,nx),2,ny),4,nt)
mfm4d = spread(spread(mfm,3,nz),4,nt)
e = -mfm4d * ( &
  cen_diff(chi * mfm4d * ((One + mur4d * uneven_deriv(pv,etam,3)) * &
  m4d * cen_diff(phi,rdx,1) - (eta4d * cen_diff(m4d,rdx,1) + m4d * &
  mur4d * cen_diff(pv,rdx,1)) * dphideta), rdx, 1) + &
  cen_diff(chi * mfm4d * ((One + mur4d * uneven_deriv(pv,etam,3)) * &
  m4d * cen_diff(phi,rdx,2) - (eta4d * cen_diff(m4d,rdx,2) + m4d * &
  mur4d * cen_diff(pv,rdx,2)) * dphideta), rdx, 2) )
print *, maxval(abs(e(:, :, 18,40))), maxloc(abs(e(:, :, 18,40)))
print *, mean(abs(e(:, :, 18,40)))
end function diag_TermE
! =====
function diag_B1B2CD(ncid, nx, ny, nz, nt) result(total)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: total
  total = diag_TermB1(ncid, nx, ny, nz, nt) + &
    diag_TermB2(ncid, nx, ny, nz, nt) + &
    diag_TermC(ncid, nx, ny, nz, nt) + &
    diag_TermD(ncid, nx, ny, nz, nt)
end function diag_B1B2CD
! =====
function diag_ABCD(ncid, nx, ny, nz, nt) result(total)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: total
  total = diag_DDDt(ncid,nx,ny,nz,nt) + diag_TermA2(ncid,nx,ny,nz,nt) + &
    diag_TermA3(ncid,nx,ny,nz,nt) + diag_TermA4(ncid,nx,ny,nz,nt) + &
    diag_TermA5(ncid,nx,ny,nz,nt) + diag_TermB1(ncid,nx,ny,nz,nt) + &

```

```

        diag_TermB2(ncid,nx,ny,nz,nt) + diag_TermB3(ncid,nx,ny,nz,nt) + &
        diag_TermC(ncid,nx,ny,nz,nt) + diag_TermD(ncid,nx,ny,nz,nt)
end function diag_ABCD
! =====
function diag_MVort(ncid, nx, ny, nz, nt) result(vort)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx-1,ny-1,nz,nt) :: vort
    real, dimension(0:nx,ny,nz,nt) :: m1s
    real, dimension(nx,0:ny,nz,nt) :: m2s
    real, dimension(0:nx,ny) :: mfu
    real, dimension(nx,0:ny) :: mfv
    real, dimension(nx-1,ny-1) :: mfpsi
    real :: rdx
    integer :: i, j, k, t
    m1s = diag_MScript1(ncid, nx, ny, nz, nt)
    m2s = diag_MScript2(ncid, nx, ny, nz, nt)
    call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
    call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
    mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        do k = 1, nz
            vort(:, :, k, t) = mfpsi**2 * ( &
                cen_diff_stag(m2s(:,1:ny-1,k,t)/mfv(:,1:ny-1), rdx, 1) - &
                cen_diff_stag(m1s(1:nx-1,:,k,t)/mfu(1:nx-1,:), rdx, 2) )
        end do
    end do
end function diag_MVort
! =====
function diag_Stream(ncid, nx, ny, nz, nt) result(psi)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: psi
    real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
    real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
    real, dimension(nx,ny,0:nz,nt) :: ph
    real, dimension(nx,ny,nz,nt) :: div
    real, dimension(nx-1,ny-1,nz,nt) :: vort
    real(cp), dimension(0:nx+1,0:ny+1) :: ch
    real(cp), dimension(0:nx,0:ny) :: ps
    real, dimension(0:nx,ny) :: mfu
    real, dimension(nx,0:ny) :: mfv
    real, dimension(nx,ny) :: mfm
    real, dimension(nx-1,ny-1) :: mfpsi
    real(cp) :: phill
    real :: rdx
    integer :: i, j, k, t
    ph = diag_GeopS(ncid, nx, ny, nz, nt)
    us = diag_MScript1(ncid, nx, ny, nz, nt)
    vs = diag_MScript2(ncid, nx, ny, nz, nt)
    div = diag_DScript(ncid, nx, ny, nz, nt)

```



```

vort = diag_MVort(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
call read_values_const(ncid, "RDX", rdx)
open(33, file="ps.dat", status="new", form="unformatted", action="write")
write (33) nx, ny, nz, nt
do t = 1, nt
  do k = 1, nz
    write (*,"(2(a,i0))") "Time: ", t, " Level: ", k
    phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) &
      + ph(1,2,k,t))
    call partition(real(us(:, :, k, t), cp), real(vs(:, :, k, t), cp), &
      real(div(:, :, k, t), cp), real(vort(:, :, k, t), cp), real(mfm, cp), &
      real(mfpsi, cp), real(mfu, cp), real(mfv, cp), phill, real(rdx, cp), &
      ch, ps)
    write (33) ps
    psi(:, :, k, t) = .25 * &
      (ps(:, nx-1, :, t) + ps(:, nx-1, 1, t) + ps(1, :, ny-1, t) + ps(1, 1, t))
  end do
end do
close(33)
! Correction for 0 in corners
psi(1,1, :, t) = psi(1,1, :, t) * 1.3333333
psi(nx,1, :, t) = psi(nx,1, :, t) * 1.3333333
end function diag_Stream
! =====
function diag_VelocPot(ncid, nx, ny, nz, nt) result(chi)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: chi
  integer, parameter :: k = 18, t = 40
  real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
  real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: div
  real, dimension(nx-1,ny-1,nz,nt) :: vort
  real(cp), dimension(0:nx+1,0:ny+1) :: ch
  real(cp), dimension(0:nx,0:ny) :: ps
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm
  real, dimension(nx-1,ny-1) :: mfpsi
  real(cp) :: phill
  real :: rdx
  integer :: i, j
  ph = diag_GeopS(ncid, nx, ny, nz, nt)
  us = diag_MScript1(ncid, nx, ny, nz, nt)
  vs = diag_MScript2(ncid, nx, ny, nz, nt)
  div = diag_DScript(ncid, nx, ny, nz, nt)

```

```

vort = diag_MVort(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
call read_values_const(ncid, "RDX", rdx)
phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) + ph(1,2,k,t))
call partition(real(us(:, :, k, t), cp), real(vs(:, :, k, t), cp), &
               real(div(:, :, k, t), cp), real(vort(:, :, k, t), cp), real(mfm, cp), &
               real(mfpsi, cp), real(mfu, cp), real(mfv, cp), phill, real(rdx, cp), ch, ps)
do j = ny+1, ny-1, -1
  print *, ch(nx-1:nx+1,j)
end do
chi = spread(spread(ch(1:nx,1:ny),3,nz),4,nt)
print *, maxval(abs(chi(:, :, 10, 13)))
print *, mean(abs(chi(:, :, 10, 13)))
end function diag_VelocPot
! =====
function diag_MS1Psi(ncid, nx, ny, nz, nt) result(ms1)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(0:nx,ny) :: ms1
  integer, parameter :: k = 18, t = 40
  real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
  real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: div
  real, dimension(nx-1,ny-1,nz,nt) :: vort
  real(cp), dimension(0:nx+1,0:ny+1) :: ch
  real(cp), dimension(0:nx,0:ny) :: ps
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm
  real, dimension(nx-1,ny-1) :: mfpsi
  real(cp) :: phill
  real :: rdx
  integer :: i, j
  ph = diag_GeopS(ncid, nx, ny, nz, nt)
  us = diag_MScript1(ncid, nx, ny, nz, nt)
  vs = diag_MScript2(ncid, nx, ny, nz, nt)
  div = diag_DScript(ncid, nx, ny, nz, nt)
  vort = diag_MVort(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
  call read_values_const(ncid, "RDX", rdx)
  phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) + ph(1,2,k,t))
  call partition(real(us(:, :, k, t), cp), real(vs(:, :, k, t), cp), &
               real(div(:, :, k, t), cp), real(vort(:, :, k, t), cp), real(mfm, cp), &
               real(mfpsi, cp), real(mfu, cp), real(mfv, cp), phill, real(rdx, cp), ch, ps)

```

```

    ms1 = -mfu * rdx * (ps(:,1:) - ps(:,ny-1))
end function diag_MS1Psi
! =====
function diag_MS2Psi(ncid, nx, ny, nz, nt) result(ms2)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,0:ny) :: ms2
    integer, parameter :: k = 18, t = 40
    real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
    real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
    real, dimension(nx,ny,0:nz,nt) :: ph
    real, dimension(nx,ny,nz,nt) :: div
    real, dimension(nx-1,ny-1,nz,nt) :: vort
    real(cp), dimension(0:nx+1,0:ny+1) :: ch
    real(cp), dimension(0:nx,0:ny) :: ps
    real, dimension(0:nx,ny) :: mfu
    real, dimension(nx,0:ny) :: mfv
    real, dimension(nx,ny) :: mfm
    real, dimension(nx-1,ny-1) :: mfpsi
    real(cp) :: phill
    real :: rdx
    integer :: i, j
    ph = diag_GeopS(ncid, nx, ny, nz, nt)
    us = diag_MScript1(ncid, nx, ny, nz, nt)
    vs = diag_MScript2(ncid, nx, ny, nz, nt)
    div = diag_DScript(ncid, nx, ny, nz, nt)
    vort = diag_MVort(ncid, nx, ny, nz, nt)
    call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
    call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
    call read_values_const(ncid, "RDX", rdx)
    phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) + ph(1,2,k,t))
    call partition(real(us(:, :, k, t), cp), real(vs(:, :, k, t), cp), &
        real(div(:, :, k, t), cp), real(vort(:, :, k, t), cp), real(mfm, cp), &
        real(mfpsi, cp), real(mfu, cp), real(mfv, cp), phill, real(rdx, cp), ch, ps)
    do j = 0, ny
        do i = 1, nx
            ms2(i,j) = mfv(i,j) * rdx * (ps(i,j) - ps(i-1,j))
        end do
    end do
end function diag_MS2Psi
! =====
function diag_MS1PsiUnstag(ncid, nx, ny, nz, nt) result (ms1u)
    integer, intent(in)      :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: ms1u
    real, dimension(nx+1,ny,nz,nt) :: ms1
    ms1 = spread(spread(diag_MS1Psi(ncid, nx, ny, nz, nt),3,nz),4,nt)
    ms1u = Half * (ms1(:,nx,:,:) + ms1(2:,:,:))
end function diag_MS1PsiUnstag
! =====

```

```

function diag_MS2PsiUnstag(ncid, nx, ny, nz, nt) result (ms2u)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: ms2u
  real, dimension(nx,ny+1,nz,nt) :: ms2
  ms2 = spread(spread(diag_MS2Psi(ncid, nx, ny, nz, nt),3,nz),4,nt)
  ms2u = Half * (ms2(:,ny,:,:) + ms2(:,2:,:,:))
end function diag_MS2PsiUnstag
! =====
function diag_DivPsi(ncid, nx, ny, nz, nt) result(div)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: div
  integer, parameter :: k = 18, t = 40
  real, dimension(0:nx,ny) :: mfu, ms1
  real, dimension(nx,0:ny) :: mfv, ms2
  real, dimension(nx,ny) :: mfm
  real :: rdx
  integer :: i, j
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  ms1 = diag_MS1Psi(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  ms2 = diag_MS2Psi(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, "RDX", rdx)
  do j = 1, ny
    do i = 1, nx
      div(i,j,:) = mfm(i,j)**2 * rdx * ( &
        ms1(i,j)/mfu(i,j) - ms1(i-1,j)/mfu(i-1,j) + &
        ms2(i,j)/mfv(i,j) - ms2(i,j-1)/mfv(i,j-1) )
    end do
  end do
  print *, maxval(abs(div(:,:k,t)))
  print *, mean(abs(div(:,:k,t)))
end function diag_DivPsi
! =====
function diag_MS1Chi(ncid, nx, ny, nz, nt) result(ms1)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(0:nx,ny) :: ms1
  integer, parameter :: k = 18, t = 40
  real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
  real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: div
  real, dimension(nx-1,ny-1,nz,nt) :: vort
  real(cp), dimension(0:nx+1,0:ny+1) :: ch
  real(cp), dimension(0:nx,0:ny) :: ps
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm
  real, dimension(nx-1,ny-1) :: mfpsi
  real(cp) :: phill

```

```

real    :: rdx
integer :: i, j
ph = diag_GeopS(ncid, nx, ny, nz, nt)
us = diag_MScript1(ncid, nx, ny, nz, nt)
vs = diag_MScript2(ncid, nx, ny, nz, nt)
div = diag_DScript(ncid, nx, ny, nz, nt)
vort = diag_MVort(ncid, nx, ny, nz, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))
call read_values_const(ncid, "RDX", rdx)
phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) + ph(1,2,k,t))
call partition(real(us(:, :, k, t), cp), real(vs(:, :, k, t), cp), &
               real(div(:, :, k, t), cp), real(vort(:, :, k, t), cp), real(mfm, cp), &
               real(mfpsi, cp), real(mfu, cp), real(mfv, cp), phill, real(rdx, cp), ch, ps)
do j = 1, ny
  do i = 0, nx
    ms1(i,j) = mfu(i,j) * rdx * (ch(i+1,j) - ch(i,j))
  end do
end do
end function diag_MS1Chi
! =====
function diag_MS2Chi(ncid, nx, ny, nz, nt) result(ms2)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,0:ny) :: ms2
  integer, parameter :: k = 18, t = 40
  real, dimension(0:nx,ny,nz,nt) :: us ! Really the x-component of MScr
  real, dimension(nx,0:ny,nz,nt) :: vs ! Really the y-component of MScr
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: div
  real, dimension(nx-1,ny-1,nz,nt) :: vort
  real(cp), dimension(0:nx+1,0:ny+1) :: ch
  real(cp), dimension(0:nx,0:ny) :: ps
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm
  real, dimension(nx-1,ny-1) :: mfpsi
  real(cp) :: phill
  real :: rdx
  integer :: i, j
  ph = diag_GeopS(ncid, nx, ny, nz, nt)
  us = diag_MScript1(ncid, nx, ny, nz, nt)
  vs = diag_MScript2(ncid, nx, ny, nz, nt)
  div = diag_DScript(ncid, nx, ny, nz, nt)
  vort = diag_MVort(ncid, nx, ny, nz, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  mfpsi = calc_mfpsi(mfu(1:nx-1,:), mfv(:,1:ny-1))

```

```

call read_values_const(ncid, "RDX", rdx)
phill = Quarter * (ph(1,1,k-1,t) + ph(1,2,k-1,t) + ph(1,1,k,t) + ph(1,2,k,t))
call partition(real(us(:,:,k,t),cp), real(vs(:,:,k,t),cp), &
               real(div(:,:,k,t),cp), real(vort(:,:,k,t),cp), real(mfm,cp), &
               real(mfpsi,cp), real(mfu,cp), real(mfv,cp), phill, real(rdx,cp), ch, ps)
do j = 0, ny
  do i = 1, nx
    ms2(i,j) = mfv(i,j) * rdx * (ch(i,j+1) - ch(i,j))
  end do
end do
end function diag_MS2Chi
! =====
function diag_MS1ChiUnstag(ncid, nx, ny, nz, nt) result (ms1u)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: ms1u
  real, dimension(nx+1,ny,nz,nt) :: ms1
  ms1 = spread(spread(diag_MS1Chi(ncid, nx, ny, nz, nt),3,nz),4,nt)
  ms1u = Half * (ms1(:,ny,:,:) + ms1(2:,:,:))
end function diag_MS1ChiUnstag
! =====
function diag_MS2ChiUnstag(ncid, nx, ny, nz, nt) result (ms2u)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: ms2u
  real, dimension(nx,ny+1,nz,nt) :: ms2
  ms2 = spread(spread(diag_MS2Chi(ncid, nx, ny, nz, nt),3,nz),4,nt)
  ms2u = Half * (ms2(:,ny,:,:) + ms2(:,2:,:,:))
end function diag_MS2ChiUnstag
! =====
function diag_ballLHS(ncid, nx, ny, nz, nt) result (lhs)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: lhs
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
  real, dimension(nx,ny,nt) :: mCap
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm, f
  real :: rdx
  integer :: i, j, k, t
  open(33, file="ps.dat", status="old", form="unformatted", action="read")
  read (33) i, j, k, t
  print *, i, j, k, t
  do t = 1, nt
    do k = 1, nz
      read (33) ps(0:nx,0:ny,k,t)
    end do
  end do
  close(33)
  mCap = diag_MCap(ncid, nx, ny, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)

```

```

call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  print *, t
  call balance_lhs(ps(:,:,:t), real(f,cp), real(mCap(:,:t),cp), &
    real(mfm,cp), real(mfu,cp), real(mfv,cp), real(rdx,cp), &
    lhs(:,:,:t))
end do
end function diag_ballLHS
! =====
function diag_ballLHSExp(ncid, nx, ny, nz, nt) result (lhs)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: lhs
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
  real, dimension(nx,ny,nt) :: mCap
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm, f
  real :: rdx
  integer :: i, j, k, t
  open(33, file="ps.dat", status="old", form="unformatted", action="read")
  read (33) i, j, k, t
  print *, i, j, k, t
  do t = 1, nt
    do k = 1, nz
      read (33) ps(0:nx,0:ny,k,t)
    end do
  end do
  close(33)
  mCap = diag_MCap(ncid, nx, ny, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
  call read_values_const(ncid, "RDX", rdx)
  do t = 1, nt
    print *, t
    call balance_lhs_exp(ps(:,:,:t), real(f,cp), real(mCap(:,:t),cp), &
      real(mfm,cp), real(mfu,cp), real(mfv,cp), real(rdx,cp), &
      lhs(:,:,:t))
  end do
  print *, lhs(10,11,9,9)
end function diag_ballLHSExp
! =====
function diag_ballLHSPert(ncid, nx, ny, nz, nt) result (lhs)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: lhs
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
  real, dimension(nx,ny,nt) :: mCap

```

```

real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny)   :: mfm, f
real   :: rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
  do k = 1, nz
    read (33) ps(0:nx,0:ny,k,t)
  end do
end do
close(33)
ps(5,6,:,:) = ps(5,6,:,:) + 5e6
mCap = diag_MCap(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  call balance_lhs_exp(ps(:,:,:), real(f,cp), real(mCap(:,:),t),cp), &
    real(mfm,cp), real(mfu,cp), real(mfv,cp), real(rdx,cp), &
    lhs(:,:,:),t))
end do
print *, lhs(10,11,9,9)
end function diag_ballHSPert
! =====
function diag_ballHSTLM(ncid, nx, ny, nz, nt) result (lhs)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: lhs
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps
  real, dimension(nx,ny,nt) :: mCap
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny)   :: mfm, f
  real   :: rdx
  integer :: i, j, k, t
  open(33, file="ps.dat", status="old", form="unformatted", action="read")
  read (33) i, j, k, t
  print *, i, j, k, t
  do t = 1, nt
    do k = 1, nz
      read (33) ps(0:nx,0:ny,k,t)
    end do
  end do
  close(33)
  dps = 0.
  dps(5,6,:,:) = 5e6

```



```

mCap = diag_MCap(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
    call balance_lhs_tlm(dps(:,:,:),t), ps(:,:,:),t), real(f,cp), &
        real(mCap(:,:,:),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
        real(rdx,cp), lhs(:,:,:),t))
end do
print *, lhs(10,11,9,9)
end function diag_ballHSTLM
! =====
function diag_ballHSAadj(ncid, nx, ny, nz, nt) result (adj)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: adj
    real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps, ataq
    real, dimension(nx,ny,nz,nt) :: dlhs
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(0:nx,ny) :: mfu
    real, dimension(nx,0:ny) :: mfv
    real, dimension(nx,ny) :: mfm, f
    real :: rdx
    integer :: i, j, k, t
    open(33, file="ps.dat", status="old", form="unformatted", action="read")
    read (33) i, j, k, t
    print *, i, j, k, t
    do t = 1, nt
        do k = 1, nz
            read (33) ps(0:nx,0:ny,k,t)
        end do
    end do
    close(33)
    dps = 0.
    dps(5,6,:,:) = 5e6
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
    call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        call balance_lhs_tlm(dps(:,:,:),t), ps(:,:,:),t), real(f,cp), &
            real(mCap(:,:,:),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
            real(rdx,cp), dlhs(:,:,:),t))
    end do
    t = 13
    print *, real(sum(real(dlhs,cp)**2))
    do t = 1, nt

```

```

        call balance_lhs_adj(real(dlhs(:,:,:),t),cp), ps(:,:,:), real(f,cp), &
            real(mCap(:,:),t),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
            real(rdx,cp), ataq(:,:,:),t))
    end do
    print *, real(sum(dps*real(ataq,cp)))
    adj = 0
end function diag_ballHSAAdj
! =====
function diag_balRHSSimp(ncid, nx, ny, nz, nt) result (rhs)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: rhs
    real, dimension(nx,ny,0:nz,nt) :: ph
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(nx,ny) :: mfm
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: rdx
    integer :: t
    ph = diag_GeopS(ncid, nx, ny, nz, nt)
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        print *, t
        call balance_rhs_simp(real(ph(:,:,:),t),cp), real(mCap(:,:),t),cp), &
            real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
            rhs(:,:,:),t))
    end do
end function diag_balRHSSimp
! =====
function diag_balRHSSimpExp(ncid, nx, ny, nz, nt) result (rhs)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: rhs
    real, dimension(nx,ny,0:nz,nt) :: ph
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(nx,ny) :: mfm
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: rdx
    integer :: t
    ph = diag_GeopS(ncid, nx, ny, nz, nt)
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        print *, t

```

```

        call balance_rhs_simp_exp(real(ph(:,:,:),t),cp), real(mCap(:,:,:),cp), &
            real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
            rhs(:,:,:),t))
    end do
end function diag_balRHSSimpExp
! =====
function diag_balRHSSimpPert(ncid, nx, ny, nz, nt) result (rhs)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: rhs
    real, dimension(nx,ny,0:nz,nt) :: ph
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(nx,ny) :: mfm
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: rdx
    integer :: t
    ph = diag_GeopS(ncid, nx, ny, nz, nt)
    ph(5,6,:,:) = ph(5,6,:,:) + 50.
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        print *, t
        call balance_rhs_simp_exp(real(ph(:,:,:),t),cp), real(mCap(:,:,:),cp), &
            real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
            rhs(:,:,:),t))
    end do
end function diag_balRHSSimpPert
! =====
function diag_balRHSSimpTLM(ncid, nx, ny, nz, nt) result (rhs)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: rhs
    real(cp), dimension(nx,ny,0:nz,nt) :: dph
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(nx,ny) :: mfm
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: rdx
    integer :: i, j, k, t
    dph = 0.
    dph(2,6,:,:) = 50.
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        call balance_rhs_simp_tlm(dph(:,:,:),t), real(mCap(:,:,:),cp), &

```

```

        real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
        rhs(:, :, :, t))
    end do
    print *, rhs(10,11,9,9)
end function diag_balRHSSimpTLM
! =====
function diag_balRHSSimpAdj(ncid, nx, ny, nz, nt) result (adj)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: adj
    real(cp), dimension(nx,ny,0:nz,nt) :: dph, ataq
    real, dimension(nx,ny,nz,nt) :: drhs
    real, dimension(nx,ny,nt) :: mCap
    real, dimension(nx,ny) :: mfm
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: rdx
    integer :: i, j, k, t
    dph = 0.
    dph(5,6, :, :) = 50.
    mCap = diag_MCap(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "RDX", rdx)
    do t = 1, nt
        call balance_rhs_simp_tlm(dph(:, :, :, t), real(mCap(:, :, t), cp), &
            real(mfm, cp), real(etaz, cp), real(etam, cp), real(rdx, cp), &
            drhs(:, :, :, t))
    end do
    print *, drhs(10,11,9,9)
    print *, real(sum(real(drhs, cp)**2))
    do t = 1, nt
        call balance_rhs_simp_adj(real(drhs(:, :, :, t), cp), &
            real(mCap(:, :, t), cp), real(mfm, cp), real(etaz, cp), real(etam, cp), &
            real(rdx, cp), ataq(:, :, :, t))
    end do
    print *, real(sum(dph*real(ataq, cp)))
    adj = 0
end function diag_balRHSSimpAdj
! =====
function diag_ImbalSimp(ncid, nx, ny, nz, nt) result (imb)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: imb
    real, dimension(nx,ny,nz,nt) :: lhs, rhs
    real :: norm
    integer :: t
    lhs = diag_balLHS(ncid, nx, ny, nz, nt)
    rhs = diag_balRHSSimp(ncid, nx, ny, nz, nt)
    imb = lhs - rhs
end function diag_ImbalSimp

```

```

! =====
function diag_ImbalSimpPert(ncid, nx, ny, nz, nt) result (imb)
  integer, intent(in)      :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: imb
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: lhs, rhs
  real, dimension(nx,ny,nt) :: mCap
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: mfm, f
  real, dimension(0:nz) :: etaz
  real, dimension(nz) :: etam
  real :: rdx, norm
  integer :: i, j, k, t
  open(33, file="ps.dat", status="old", form="unformatted", action="read")
  read (33) i, j, k, t
  print *, i, j, k, t
  do t = 1, nt
    do k = 1, nz
      read (33) ps(0:nx,0:ny,k,t)
    end do
  end do
  close(33)
  ps(5,6,:,:) = ps(5,6,:,:) + 5e6
  mCap = diag_MCap(ncid, nx, ny, nt)
  call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
  call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
  call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
  call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
  call read_values_const(ncid, "RDX", rdx)
  do t = 1, nt
    call balance_lhs_exp(ps(:,:,:,t), real(f,cp), real(mCap(:,:,:),cp), &
      real(mfm,cp), real(mfu,cp), real(mfv,cp), real(rdx,cp), &
      lhs(:,:,:,t))
  end do
  ph = diag_GeopS(ncid, nx, ny, nz, nt)
  ph(5,6,:,:) = ph(5,6,:,:) + 50.
  call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
  call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
  do t = 1, nt
    call balance_rhs_simp_exp(real(ph(:,:,:,t),cp), real(mCap(:,:,:),cp), &
      real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
      rhs(:,:,:,t))
  end do
  imb = lhs - rhs
end function diag_ImbalSimpPert
! =====
function diag_ImbalSimpTLM(ncid, nx, ny, nz, nt) result (dimb)
  integer, intent(in)      :: ncid, nx, ny, nz, nt

```

```

real, dimension(nx,ny,nz,nt) :: dimb
real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps
real(cp), dimension(nx,ny,0:nz,nt) :: dph
real, dimension(nx,ny,nz,nt) :: dlhs, drhs
real, dimension(nx,ny,nt) :: mCap
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: mfm, f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real :: rdx, norm
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
    do k = 1, nz
        read (33) ps(0:nx,0:ny,k,t)
    end do
end do
close(33)
dps = 0.
dps(5,6,::) = 5e6
mCap = diag_MCap(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
    call balance_lhs_tlm(dps(:,:,:,t), ps(:,:,:,t), real(f,cp), &
        real(mCap(:,:,:,t),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
        real(rdx,cp), dlhs(:,:,:,t))
end do
dph = 0.
dph(5,6,::) = 50.
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
do t = 1, nt
    call balance_rhs_simp_tlm(dph(:,:,:,t), real(mCap(:,:,:,t),cp), &
        real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
        drhs(:,:,:,t))
end do
dimb = dlhs - drhs
end function diag_ImbalSimpTLM
! =====
function diag_ImbalSimpAdj(ncid, nx, ny, nz, nt) result (adj)
integer, intent(in) :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: adj
real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps, adjpsi

```

```

real(cp), dimension(nx,ny,0:nz,nt) :: dph, adjphi
real(cp), dimension(nx,ny,nz,nt) :: dimb
real, dimension(nx,ny,nz,nt) :: dlhs, drhs
real, dimension(nx,ny,nt) :: mCap
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: mfm, f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real :: rdx, norm
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
    do k = 1, nz
        read (33) ps(0:nx,0:ny,k,t)
    end do
end do
close(33)
dps = 0.
dps(5,6,,:) = 5e6
mCap = diag_MCap(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
    call balance_lhs_tlm(dps(:,:,:),t), ps(:,:,:),t), real(f,cp), &
        real(mCap(:,:),t,cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
        real(rdx,cp), dlhs(:,:,:),t))
end do
dph = 0.
dph(5,6,,:) = 50.
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
do t = 1, nt
    call balance_rhs_simp_tlm(dph(:,:,:),t), real(mCap(:,:),t,cp), &
        real(mfm,cp), real(etaz,cp), real(etam,cp), real(rdx,cp), &
        drhs(:,:,:),t))
end do
dimb = dlhs - drhs
print *, real(sum(real(dimb,cp)**2))
do t = 1, nt
    call imbalance_simp_adj(dimb(:,:,:),t), ps(:,:,:),t), real(f,cp), &
        real(mCap(:,:),t,cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
        real(etaz,cp), real(etam,cp), real(rdx,cp), adjpsi(:,:,:),t), &
        adjphi(:,:,:),t))
end do

```

```

    print *, real(sum(dps*real(adjpsi,cp))+sum(dph*real(adjphi,cp)))
    adj = 0
end function diag_ImbalSimpAdj
! =====
function diag_thetaFromPhi(ncid, nx, ny, nz, nt) result (th)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: th
    real, dimension(nx,ny,0:nz,nt) :: phi
    real, dimension(nx,ny,nz,nt) :: potTemp
    real, dimension(nx,ny,nt) :: mu
    real, dimension(0:nz) :: etaz
    real, dimension(nz) :: etam
    real :: pt
    integer :: i, j, k, t
    potTemp = diag_Theta(ncid, nx, ny, nz, nt)
    phi = diag_GeopS(ncid, nx, ny, nz, nt)
    mu = diag_DryAir(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
    call read_values_const(ncid, "P_TOP", pt)
    call theta_phi(phi, mu, etaz, etam, pt, th)
    i = 5
    j = 6
    k = 11
    t = 13
    print *, phi(i,j,k,t), phi(i,j,k-1,t)
    print *, etaz(k-1), etam(k), etaz(k)
    print *, mu(i,j,t)
    print *, potTemp(i,j,k,t), th(i,j,k,t)
end function diag_thetaFromPhi
! =====
function diag_PotVort(ncid, nx, ny, nz, nt) result (q)
    integer, intent(in)          :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: q
    real, dimension(nx+1,ny,nz,nt) :: u
    real, dimension(nx,ny+1,nz,nt) :: v
    real, dimension(nx,ny,nz+1,nt) :: ht
    real, dimension(nx,ny,nz,nt) :: p, th, qh2o
    real, dimension(nx,ny) :: mf, f
    real, dimension(nz+1) :: etaz
    real, dimension(nz) :: etam
    real :: rdx, dx
    call read_values(ncid, wrfVar(UStag)%wrfName, u)
    call read_values(ncid, wrfVar(VStag)%wrfName, v)
    ht = diag_GeopHghtS(ncid, nx, ny, nz, nt)
    p = diag_Pressure(ncid, nx, ny, nz, nt, si=.true.)
    th = diag_Theta(ncid, nx, ny, nz, nt)
    call read_values(ncid, wrfVar(MixR)%wrfName, qh2o)
    call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mf)
    call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)

```



```

call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "RDX", rdx)
dx = 1./rdx
call pvor(u, v, ht, th, density(p, temp_from_theta_p(th, p), qh2o), f, mf, &
    etam, etaz, dx, q)
end function diag_PotVort
! =====
function diag_PVExp(ncid, nx, ny, nz, nt) result (pv)
integer, intent(in)      :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: pv
real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
real, dimension(nx,ny,0:nz,nt) :: ph
real, dimension(nx,ny,nt) :: mCap, mu
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real :: pt, rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
    do k = 1, nz
        read (33) ps(0:nx,0:ny,k,t)
    end do
end do
close(33)
ph = diag_GeopS(ncid, nx, ny, nz, nt)
mCap = diag_MCap(ncid, nx, ny, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "P_TOP", pt)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
    print *, t
    call pv_exp(ps(:, :, :, t), real(ph(:, :, :, t), cp), real(f, cp), &
        real(mCap(:, :, t), cp), real(mu(:, :, t), cp), real(mfu, cp), real(mfv, cp), &
        real(etaz, cp), real(etam, cp), real(pt, cp), real(rdx, cp), pv(:, :, :, t))
end do
end function diag_PVExp
! =====
function diag_PVPert(ncid, nx, ny, nz, nt) result (pv)
integer, intent(in)      :: ncid, nx, ny, nz, nt

```

```

real, dimension(nx,ny,nz,nt) :: pv
real(cp), dimension(0:nx,0:ny,nz,nt) :: ps
real, dimension(nx,ny,0:nz,nt) :: ph
real, dimension(nx,ny,nt) :: mCap, mu
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real :: pt, rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
  do k = 1, nz
    read (33) ps(0:nx,0:ny,k,t)
  end do
end do
close(33)
ps(5,6,10,:) = ps(5,6,10,:) + 5e6
ph = diag_GeopS(ncid, nx, ny, nz, nt)
ph(5,6,10,:) = ph(5,6,10,:) + 50.
mCap = diag_MCap(ncid, nx, ny, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "P_TOP", pt)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  call pv_exp(ps(:,:,t), real(ph(:,:,t),cp), real(f,cp), &
    real(mCap(:,:,t),cp), real(mu(:,:,t),cp), real(mfu,cp), real(mfv,cp), &
    real(etaz,cp), real(etam,cp), real(pt,cp), real(rdx,cp), pv(:,:,t))
end do
end function diag_PVPert
! =====
function diag_PVTLM(ncid, nx, ny, nz, nt) result (dpv)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,nz,nt) :: dpv
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps
  real(cp), dimension(nx,ny,0:nz,nt) :: dph
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nt) :: mCap, mu
  real, dimension(0:nx,ny) :: mfu
  real, dimension(nx,0:ny) :: mfv
  real, dimension(nx,ny) :: f
  real, dimension(0:nz) :: etaz

```

```

real, dimension(nz)          :: etam
real :: pt, rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
  do k = 1, nz
    read (33) ps(0:nx,0:ny,k,t)
  end do
end do
close(33)
dps = 0.
dps(5,6,10,:) = 5e6
ph = diag_GeopS(ncid, nx, ny, nz, nt)
dph = 0.
dph(5,6,10,:) = 50.
mCap = diag_MCap(ncid, nx, ny, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "P_TOP", pt)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  call pv_tlm(dps(:,:,:,t), dph(:,:,:,t), ps(:,:,:,t), real(ph(:,:,:,t),cp), &
    real(f,cp), real(mCap(:,:,:,t),cp), real(mu(:,:,:,t),cp), real(mfu,cp), &
    real(mfv,cp), real(etaz,cp), real(etam,cp), real(pt,cp), &
    real(rdx,cp), dpv(:,:,:,t))
end do
end function diag_PVTLM
! =====
function diag_PVAdj(ncid, nx, ny, nz, nt) result (adj)
integer, intent(in)          :: ncid, nx, ny, nz, nt
real, dimension(nx,ny,nz,nt) :: adj
real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, dps, adjpsi
real(cp), dimension(nx,ny,0:nz,nt) :: dph, adjphi
real, dimension(nx,ny,0:nz,nt) :: ph
real, dimension(nx,ny,nz,nt) :: dpv
real, dimension(nx,ny,nt) :: mCap, mu
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real :: pt, rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")

```

```

read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
  do k = 1, nz
    read (33) ps(0:nx,0:ny,k,t)
  end do
end do
close(33)
dps = 0.
dps(5,6,10,:) = 5e6
ph = diag_GeopS(ncid, nx, ny, nz, nt)
dph = 0.
dph(5,6,10,:) = 50.
mCap = diag_MCap(ncid, nx, ny, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "P_TOP", pt)
call read_values_const(ncid, "RDX", rdx)
do t = 1, nt
  call pv_tlm(dps(:,:,:),t), dph(:,:,:),t), ps(:,:,:),t), real(ph(:,:,:),t),cp), &
    real(f,cp), real(mCap(:,:,:),t),cp), real(mu(:,:,:),t),cp), real(mfu,cp), &
    real(mfv,cp), real(etaz,cp), real(etam,cp), real(pt,cp), &
    real(rdx,cp), dpv(:,:,:),t))
end do
print *, real(sum(real(dpv,cp)**2))
do t = 1, nt
  call pv_adj(real(dpv(:,:,:),t),cp), ps(:,:,:),t), real(ph(:,:,:),t),cp), &
    real(f,cp), real(mCap(:,:,:),t),cp), real(mu(:,:,:),t),cp), real(mfu,cp), &
    real(mfv,cp), real(etaz,cp), real(etam,cp), real(pt,cp), &
    real(rdx,cp), adjpsi(:,:,:),t), adjphi(:,:,:),t))
end do
print *, real(sum(dps*real(adjpsi,cp))+sum(dph*real(adjphi,cp)))
adj = 0
end function diag_PVAdj
! =====
function diag_CostGradPhi(ncid, nx, ny, nz, nt) result (grad)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(nx,ny,0:nz,nt) :: gradgem
  real(cp), parameter :: e = 5000.
  real(cp), dimension(0:nx,0:ny,nz,nt) :: ps, gradpsi1, gradpsi2
  real(cp), dimension(nx,ny,0:nz,nt) :: gradphi1, gradphi2, grad
  real(cp), dimension(nx,ny,nz,nt) :: imb
  real(cp), dimension(0:nx,0:ny,nz) :: ps3
  real(cp), dimension(nx,ny,0:nz) :: ph3
  real, dimension(nx,ny,0:nz,nt) :: ph
  real, dimension(nx,ny,nz,nt) :: pv, lhs, rhs, pvGiven

```

```

real, dimension(nx,ny,nt) :: mCap, mu
real, dimension(0:nx,ny) :: mfu
real, dimension(nx,0:ny) :: mfv
real, dimension(nx,ny) :: mfm, f
real, dimension(0:nz) :: etaz
real, dimension(nz) :: etam
real(cp) :: costa, costb, cost
real :: pt, rdx
integer :: i, j, k, t
open(33, file="ps.dat", status="old", form="unformatted", action="read")
read (33) i, j, k, t
print *, i, j, k, t
do t = 1, nt
  do k = 1, nz
    read (33) ps(0:nx,0:ny,k,t)
  end do
end do
close(33)
ph = diag_GeopS(ncid, nx, ny, nz, nt)
pvGiven = diag_PVExp(ncid, nx, ny, nz, nt)
mCap = diag_MCap(ncid, nx, ny, nt)
mu = diag_DryAir(ncid, nx, ny, nt)
call read_values_const(ncid, wrfVar(MapFacU)%wrfName, mfu)
call read_values_const(ncid, wrfVar(MapFacV)%wrfName, mfv)
call read_values_const(ncid, wrfVar(MapFacM)%wrfName, mfm)
call read_values_const(ncid, wrfVar(Coriolis)%wrfName, f)
call read_values_const(ncid, wrfVar(EtaW)%wrfName, etaz)
call read_values_const(ncid, wrfVar(EtaMass)%wrfName, etam)
call read_values_const(ncid, "P_TOP", pt)
call read_values_const(ncid, "RDX", rdx)
t = 13
call imbalance_exp(ps(:,:,:,t), real(ph(:,:,:,t),cp), real(f,cp), &
  real(mCap(:,:,:,t),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
  real(etaz,cp), real(etam,cp), real(rdx,cp), imb(:,:,:,t))
costb = Half * e * sum(imb**2)
call pv_exp(ps(:,:,:,t), real(ph(:,:,:,t),cp), real(f,cp), &
  real(mCap(:,:,:,t),cp), real(mu(:,:,:,t),cp), real(mfu,cp), real(mfv,cp), &
  real(etaz,cp), real(etam,cp), real(pt,cp), real(rdx,cp), pv(:,:,:,t))
costa = Half * sum((real(pv(:,:,:,t),cp)-real(pvGiven(:,:,:,t),cp))**2)
cost = costa + costb
write(*,*) "Cost A & B:", costa, costb
write(*,*) "Total cost:", cost
call pv_adj(real(pv(:,:,:,t),cp)-real(pvGiven(:,:,:,t),cp), ps(:,:,:,t), &
  real(ph(:,:,:,t),cp), real(f,cp), real(mCap(:,:,:,t),cp), &
  real(mu(:,:,:,t),cp), real(mfu,cp), real(mfv,cp), real(etaz,cp), &
  real(etam,cp), real(pt,cp), real(rdx,cp), gradpsi1(:,:,:,t), &
  gradphi1(:,:,:,t))
call imbalance_simp_adj(imb(:,:,:,t), ps(:,:,:,t), real(f,cp), &
  real(mCap(:,:,:,t),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
  real(etaz,cp), real(etam,cp), real(rdx,cp), gradpsi2(:,:,:,t), &

```

```

        gradphi2(:,:,:,t))
    grad(:,:,:,t) = gradphi1(:,:,:,t) + e * gradphi2(:,:,:,t)
    ! Check the gradient
    print *, "Gradient phi 1 & 2: ", gradphi1(5,6,10,t), e * gradphi2(5,6,10,t)
    print *, "Total phi gradient: ", grad(5,6,10,t)
    call imbalance_exp(ps(:,:,:,t), real(ph(:,:,:,t),cp), real(f,cp), &
        real(mCap(:,:,:,t),cp), real(mfm,cp), real(mfu,cp), real(mfv,cp), &
        real(etaz,cp), real(etam,cp), real(rdx,cp), imb(:,:,:,t))
    costb = Half * e * sum(imb**2)
    call pv_exp(ps(:,:,:,t), real(ph(:,:,:,t),cp), real(f,cp), &
        real(mCap(:,:,:,t),cp), real(mu(:,:,:,t),cp), real(mfu,cp), real(mfv,cp), &
        real(etaz,cp), real(etam,cp), real(pt,cp), real(rdx,cp), pv(:,:,:,t))
    costa = Half * sum((real(pv(:,:,:,t),cp)-real(pvGiven(:,:,:,t),cp))**2)
    cost = costa + costb
    print *, "After perturbation:"
    write(*,*) "Cost A & B:", costa, costb
    write(*,*) "Total cost:", cost
    gradgem = grad
    ps3 = ps(:,:,:,t)
    ph3 = ph(:,:,:,t)
    print *, real(ps3(nx-1,ny-1,11)), real(ph3(nx-1,ny-1,11))
    call solve_phi_psi(real(pvGiven(:,:,:,t),cp), ps3, ph3, real(f,cp), &
        real(mCap(:,:,:,t),cp), real(mu(:,:,:,t),cp), real(mfm,cp), real(mfu,cp), &
        real(mfv,cp), real(etaz,cp), real(etam,cp), real(pt,cp), real(rdx,cp))
    print *, real(ps3(nx-1,ny-1,11)), real(ph3(nx-1,ny-1,11))
end function diag_CostGradPhi
! =====
function diag_InvertPhi(ncid, nx, ny, nz, nt) result (phiout)
    integer, intent(in) :: ncid, nx, ny, nz, nt
    real, dimension(nx,ny,nz,nt) :: phiout
    integer :: i, j, k, t
    real(cp), dimension(0:nx,0:ny,nz) :: psi
    real(cp), dimension(nx,ny,0:nz) :: phi
    real(cp), dimension(nx,ny,0:nz,nt) :: phi4
    real, dimension(nx,ny,nt) :: mu
    real, dimension(nz+1) :: znw
    real, dimension(nz) :: znu
    real :: pt
    open(34, file="psphiinvert.dat", status="old", form="unformatted", &
        action="read")
    read (34) i, j, k
    print *, i, j, k
    read (34) psi
    read (34) phi
    close(34)
    mu = diag_DryAir(ncid, nx, ny, nt)
    call read_values_const(ncid, wrfVar(EtaW)%wrfName, znw)
    call read_values_const(ncid, wrfVar(EtaMass)%wrfName, znu)
    call read_values_const(ncid, "P_TOP", pt)
    phi4 = spread(phi, 4, nt)

```

```

do t = 1, nt
  do j = 1, ny
    do i = 1, nx
      phiout(i,j,:,t) = &
        unstag_hght(real(phi4(i,j,:,t)), znw, znu, mu(i,j,t), pt)
    end do
  end do
end do
end function diag_InvertPhi
! =====
function diag_InvertPsi(ncid, nx, ny, nz, nt) result (psiout)
  integer, intent(in) :: ncid, nx, ny, nz, nt
  real, dimension(0:nx,0:ny,nz,nt) :: psiout
  integer :: i, j, k
  real(cp), dimension(0:nx,0:ny,nz,nt) :: psi4
  real(cp), dimension(0:nx,0:ny,nz) :: psi
  real(cp), dimension(nx,ny,0:nz) :: phi
  open(34, file="psphiinvert.dat", status="old", form="unformatted", &
    action="read")
  read (34) i, j, k
  print *, i, j, k
  read (34) psi
  read (34) phi
  close(34)
  psi4 = spread(psi, 4, nt)
  psiout = Quarter * (psi4(:nx-1,:ny-1,:,:) + psi4(1,:ny-1,:,:) &
    + psi4(:nx-1,1,:,:) + psi4(1:1,:,:))
end function diag_InvertPsi
end module registry

```

## 8.12 Main program

The main program is essentially a driver that calls various subroutines to do the work.

```

! This program reads a WRF history file in netCDF format and outputs
! specified variables (from convert.nl) into a new GEMPAK file.
! Identifiers within this program use the following conventions:
! o Variables are typed like numOuts
! o Constants are typed like GasConstant
! o Functions and subroutines generally have underscores.
! By Steve Decker
! July 2006
program wrf2gem
  use wrf2gem_subs, only: get_out_fields, get_num_files, open_wrf_file, &
    get_grid_info, update_grid_info, mem_error, &
    get_times, close_wrf_file
  use gempak,          only: init_gem, create_gemfile, close_and_exit_gem

```

```

use registry,      only: init_reg, output_var
implicit none
!
! Initialization
!
real,    dimension(3)           :: ang
real,    dimension(2)           :: lat, lon
integer, dimension(:), pointer :: outList
integer :: numOuts, pb, pt, dp, numFiles, iCount, ncid, nx, ny, nz, &
        numTimes, proj, timesPer12hr, i, igdfln, status
character(len=80), dimension(:), pointer :: outFiles
character(len=15), dimension(:), allocatable :: times
logical :: ok
!
! Execution
!
nullify(outList, outFiles)
! Find out from namelist which fields we should output to GEMPAK
call get_out_fields(numOuts, outList, pb, pt, dp)
call get_num_files(numFiles, outFiles)
if (numOuts < 1 .or. numFiles < 1) stop "Nothing to output!"
! Do this for each WRF history file
do iCount = 1, numFiles
    write (*,"(3a)") "Processing ", trim(outFiles(iCount)), "..."
    ! Open WRF history file
    call open_wrf_file(outFiles(iCount), ncid)
    if (iCount == 1) then
        ! Determine grid size, map projection, and number of output times
        ! Note nx, ny are based on number of grid boxes, i.e., cross points
        call get_grid_info(ncid, nx, ny, nz, numTimes, proj, lat, lon, ang)
    else
        ! Verify that grid projection matches the first file, and get new
        ! number of times in file.
        call update_grid_info(ncid, nx, ny, nz, proj, lat, lon, ang, &
            numTimes, ok)
        if (.not. ok) then
            write (*,"(3a)") "Skipping ", trim(outFiles(iCount)), " because of&
                & grid navigation mismatch with previous netCDF files."
            call close_wrf_file(ncid)
            cycle
        end if
    end if
    allocate (times(numTimes), stat=status)
    call mem_error(status, 1, "wrf2gem.f90")
    call get_times(iCount, outFiles(1:numFiles), ncid, times, timesPer12hr)
    ! (Re)initialize registry, since timesPer12hr might have changed
    call init_reg(timesPer12hr, pb, pt, dp)
    ! Initialize GEMPAK and create or add to GEMPAK file
    call init_gem
    call create_gemfile(proj, nx, ny, lat, lon, ang, igdfln, ok)
end do

```



```

if (.not. ok) then
  write (*,"(3a)") "Skipping ", trim(outFiles(iCount)), " because of&
    & grid navigation mismatch with GEMPAK file."
else
  ! Main loop over output fields
  do i = 1, numOuts
    call output_var(ncid, igdfln, nx, ny, nz, outList(i), iCount, &
      outFiles(1:numFiles), times)
  end do
end if
! Clean up
call close_wrf_file(ncid)
call close_and_exit_gem(igdfln)
deallocate (times, stat=status)
call mem_error(status, 2, "wrf2gem.f90")
end do
print *, "Successful completion!"
end program wrf2gem

```

# References

- Alexander, G. D., J. A. Weinman, and J. L. Schols, 1998: The use of digital warping of microwave integrated water vapor imagery to improve forecasts of marine extratropical cyclones. *Mon. Wea. Rev.*, **126**, 1469–1496.
- Arbogast, P., and A. Joly, 1998: Potential vorticity inversion of a two-dimensional steady flow: Application to symmetric instability. *Quart. J. Roy. Meteor. Soc.*, **124**, 317–339.
- Beare, R. J., A. J. Thorpe, and A. A. White, 2003: The predictability of extratropical cyclones: Nonlinear sensitivity to localized potential-vorticity perturbations. *Quart. J. Roy. Meteor. Soc.*, **129**, 219–237.
- Beier, T., and S. Neely, 1992: Feature-based image metamorphosis. *Comput. Graph.*, **26** (2), 35–42.
- Bijlsma, S. J., L. M. Hafkenscheid, and P. Lynch, 1986: Computation of the streamfunction and velocity potential and reconstruction of the wind field. *Mon. Wea. Rev.*, **114**, 1547–1551.
- Buizza, R., and P. Chessa, 2002: Prediction of the U.S. storm of 2426 January 2000 with the ECMWF ensemble prediction system. *Mon. Wea. Rev.*, **130**, 1531–1551.
- Chang, C.-P., L. Yi, and G. T.-J. Chen, 2000: A numerical simulation of vortex development during the 1992 East Asian summer monsoon onset using the Navy's regional model. *Mon. Wea. Rev.*, **128**, 1604–1631.
- Davis, C. A., and K. A. Emanuel, 1991: Potential vorticity diagnostics of cyclogenesis. *Mon. Wea. Rev.*, **119**, 1929–1953.
- , S. Low-Nam, M. A. Shapiro, X. Zou, and A. J. Krueger, 1999: Direct retrieval of wind from Total Ozone Mapping Spectrometer (TOMS) data: Examples from FASTEX. *Quart. J. Roy. Meteor. Soc.*, **125**, 3375–3391.
- Decker, S. G., 2003: The local energetics perspective on the life cycles of midlatitude synoptic-scale disturbances: Case studies and real-time diagnostics. M.S. thesis, Dept. of Atmospheric and Oceanic Sciences, University of Wisconsin—Madison, 145 pp.

- Demirtas, M., and A. J. Thorpe, 1999: Sensitivity of short-range forecasts to local potential vorticity modifications. *Mon. Wea. Rev.*, **127**, 922–939.
- Fulton, S. R., P. E. Ciesielski, and W. H. Schubert, 1986: Multigrid methods for elliptic problems: A review. *Mon. Wea. Rev.*, **114**, 943–959.
- Grassotti, C., H. Iskenerian, and R. N. Hoffman, 1999: Fusion of surface radar and satellite rainfall data using feature calibration and alignment. *J. Appl. Meteor.*, **38**, 677–695.
- Hakim, G. J., D. Keyser, and L. F. Bosart, 1996: The Ohio Valley wave-merger cyclogenesis event of 2526 January 1978. Part II: Diagnosis using quasigeostrophic potential vorticity inversion. *Mon. Wea. Rev.*, **124**, 2176–2205.
- Hoffman, R. N., Z. Liu, J.-F. Louis, and C. Grassotti, 1995: Distortion representation of forecast errors. *Mon. Wea. Rev.*, **123**, 2758–2770.
- , and C. Grassotti, 1996: A technique for assimilating SSM/I observations of marine atmospheric storms: Tests with ECMWF analyses. *J. Appl. Meteor.*, **35**, 1177–1188.
- Hoskins, B. J., M. E. McIntyre, and A. W. Robertson, 1985: On the use and significance of isentropic potential vorticity maps. *Quart. J. Roy. Meteor. Soc.*, **111**, 877–946.
- Huo, Z., D.-L. Zhang, and J. Gyakum, 1998: An application of potential vorticity inversion to improving the numerical prediction of the March 1993 superstorm. *Mon. Wea. Rev.*, **126**, 424–436.
- Loughe, A. F., C.-C. Lai, and D. Keyser, 1995: A technique for diagnosing three-dimensional ageostrophic circulations in baroclinic disturbances on limited-area domains. *Mon. Wea. Rev.*, **123**, 1476–1504.
- Kalnay, E., 2003: *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, 341 pp.
- Kim, H. M., and M. C. Morgan, 2002: Dependence of singular vector structure and evolution on the choice of norm. *J. Atmos. Sci.*, **59**, 3099–3116.
- Kundu, P. K., 1990: *Fluid Mechanics*. Academic Press, 638 pp.
- Mallet, I., P. Arbogast, C. Baehr, J.-P. Cammas, and P. Mascart, 1999: Effects of a low-level precursor and frontal stability on cyclogenesis during FASTEX IOP17. *Quart. J. Roy. Meteor. Soc.*, **125**, 3415–3437.
- Martin, J. E., and J. A. Otkin, 2004: The rapid growth and decay of an extratropical cyclone over the Central Pacific Ocean. *Wea. Forecasting*, **19**, 358–376.
- McMurdie, L., and C. Mass, 2004: Major numerical forecast failures over the Northeast Pacific. *Wea. Forecasting*, **19**, 338–356.

- Nehrkorn, T., R. H. Hoffman, C. Grassotti, and J.-F. Louis, 2003: Feature calibration and alignment to represent model forecast errors: Empirical regularization. *Quart. J. Roy. Meteor. Soc.*, **129**, 195–218.
- Purser, R. J., and D. F. Parrish, 2003: A Bayesian technique for estimating continuously varying statistical parameters of a variational assimilation. *Meteor. Atmos. Phys.*, **82**, 209–226.
- Raymond, D. J., 1992: Nonlinear balance and potential-vorticity thinking at large Rossby number. *Quart. J. Roy. Meteor. Soc.*, **118**, 987–1015.
- Reynolds, C. A., P. J. Webster, and E. Kalnay, 1994: Random error growth in NMC's global forecasts. *Mon. Wea. Rev.*, **122**, 1281–1305.
- Roebber, P. J., D. M. Schultz, and R. Romero, 2002: Synoptic regulation of the 3 May 1999 Tornado Outbreak. *Wea. Forecasting*, **17**, 399–429.
- Schneider, T., I. M. Held, and S. T. Garner, 2003: Boundary effects in potential vorticity dynamics. *J. Atmos. Sci.*, **60**, 1024–1040.
- Skamarock, W. C., J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, W. Wang, and J. G. Powers, 2005: A description of the Advanced Research WRF Version 2. NCAR Tech. Note NCAR/TN-468+STR, 88 pp.
- Sundqvist, H., 1975: Initialization for models using sigma as the vertical coordinate. *J. Appl. Meteor.*, **14**, 153–158.
- Swarbrick, S. J., 2001: Applying the relationship between potential vorticity fields and water vapor imagery to adjust initial conditions in NWP. *Meteor. Appl.*, **8**, 221–228.
- Vallis, G. K., G. J. Shutts, and M. E. B. Gray, 1997: Balanced mesoscale motion and stratified turbulence forced by convection. *Quart. J. Roy. Meteor. Soc.*, **123**, 1621–1652.
- Wimmers, A. J., and J. L. Moody, 2001: A fixed-layer estimation of upper tropospheric specific humidity from the GOES water vapor channel: Parameterization and validation of the altered brightness temperature product. *J. Geophys. Res. (D. Atmos.)*, **106**, 17 115–17 132.
- Wolberg, G., 1998: Image morphing: A survey. *Visual Comput.*, **14**, 360–372.
- Zhang, F., C. Snyder, and R. Rotunno, 2002: Mesoscale predictability of the Surprise Snowstorm of 2425 January 2000. *Mon. Wea. Rev.*, **130**, 1617–1632.
- Zupanski, M., D. Zupanski, D. F. Parrish, E. Rogers, and G. DiMego, 2002: Four-dimensional variational data assimilation for the Blizzard of 2000. *Mon. Wea. Rev.*, **130**, 1967–1988.