LAMPS 90

PROGRAMMER'S GUIDE

August 11, 1994

Robert Cohen

Numerical Model Development Facility

Drexel University

NOTICE

LAMPS 90 is written by and for the Numerical Model Development Facility at Drexel University. For further information contact Drs. Donald Perkey and Carl Kreitzberg at Drexel University, Department of Physics and Atmospheric Science, Philadelphia, PA, 19104.

Formatting Convention Used in This Document

word type	style	examples
files and directories	**bold**	**src/ case.cfg merge.F**
UNIX or FORTRAN commands	`typeface`	`make #include read`
parameters	*emphasis*	*prog case*

Contents

1 INTRODUCTION **7**
- 1.1 Short description of LAMPS 90 . . . 7
- 1.2 Short description of this documentation . . . 7
- 1.3 Prerequisites . . . 7
- 1.4 Using this documentation . . . 7
- 1.5 Program dependencies . . . 8
  - 1.5.1 Source codes . . . 8
  - 1.5.2 Input/Output . . . 8

I Initialization **10**

2 analytic **11**
- 2.1 Description . . . 11
- 2.2 Equations . . . 11
  - 2.2.1 Stable Atmosphere . . . 11
  - 2.2.2 Standard Atmosphere . . . 14

3 begin **16**
- 3.1 Description . . . 16
- 3.2 Input/Output . . . 16

4 fgsa3d **17**
- 4.1 Description . . . 17
- 4.2 Input/Output . . . 17

5 frstg25 **18**
- 5.1 Description . . . 18
- 5.2 Input/Output . . . 18
- 5.3 Procedure . . . 18

6 frstg381 **20**
- 6.1 Description . . . 20
- 6.2 Input/Output . . . 20
- 6.3 Procedure . . . 20

7 gdinterp **21**
- 7.1 Description . . . 21
- 7.2 Input/Output . . . 21

8 isenv2p2 **22**
8.1 Description . . . 22
8.2 Input/Output . . . 22
8.3 Procedure . . . 22
8.4 References . . . 22

9 lyrmean **23**
9.1 Description . . . 23
9.2 Input/Output . . . 23

10 merge **24**
10.1 Description . . . 24
10.2 Input/Output . . . 24

11 masa3d **25**
11.1 Description . . . 25
11.2 Input/Output . . . 25

12 reha3d **26**
12.1 Description . . . 26
12.2 Input/Output . . . 26

13 rhanal **27**
13.1 Description . . . 27
13.2 Input/Output . . . 27
13.3 Procedure . . . 27
13.3.1 Method . . . 27
13.3.2 Analysis Equation . . . 27
13.3.3 Correction Weights . . . 28
13.3.4 Correlation Model . . . 29
13.3.5 Data . . . 29
13.3.6 Reference . . . 29

14 sound **30**
14.1 Description . . . 30
14.2 Input/Output . . . 30
14.3 Procedure . . . 30

15 tbound **32**
15.1 Description . . . 32
15.2 Input/Output . . . 32

16 terrain **33**
16.1 Description . . . 33
16.2 Input/Output . . . 33

17 vertran **34**
17.1 Description . . . 34
17.2 Input/Output . . . 34

18 wdadjust **35**
18.1 Description . . . 35
18.2 Input/Output . . . 37

19 wina3d **38**
19.1 Description . . . 38
19.2 Input/Output . . . 38

II Dynamic Models 39

20 model **40**
20.1 Description . . . 40
20.1.1 variables . . . 40
20.1.2 structure . . . 41
20.2 Input . . . 42
20.2.1 Input Cards . . . 42
20.2.2 Initialization Data File . . . 43
20.3 Output . . . 43
20.4 Standard Output . . . 43
20.4.1 History . . . 44
20.5 Array Dump . . . 44
20.6 Equations . . . 45

21 modeldry **46**
21.1 Description . . . 46
21.2 Input/Output . . . 47
21.3 Equations . . . 47

III Post-processing 48

22 dvutil **49**
22.1 Description . . . 49

23 pltmout **50**

24 reformat **51**

25 trajec **52**

26 trajplt **53**

IV Libraries 54

27 libdvsubs **55**
27.1 History . . . 55
27.2 Notes . . . 56
27.3 Introduction . . . 56
27.4 Design Structure . . . 57
27.4.1 Data Volume Header Record (record 1) . . . 57
27.4.2 I/O Routines (records `iorec` to `iorec+niosub/dvrecl`) . . . 58
27.4.3 Internal Files . . . 58

28 libgraph **60**

29 liblamps **61**
29.1 List of routines 61
29.2 Saturation Vapor Pressure 62
29.3 References 66

30 libmdlio **68**

31 postlib **69**

32 Object files used as libraries **70**
32.1 gsbytes 70
32.2 maptran 70
32.3 plotmap 70
32.4 uexit 70

Chapter 1

INTRODUCTION

1.1 Short description of LAMPS 90

LAMPS 90 is a set of programs that take an initial state of the atmosphere (derived from observations or analytic conditions), simulate the evolution of the atmosphere from the initial state and graph and analyze the results. Each program performs one step of the process.

The programs and all of the files associated with the compilation and running of the programs are organized and arranged in a structure that takes advantage of the capabilities of a UNIX machine and makes the process of installing, compiling and running the programs relatively straightforward.

1.2 Short description of this documentation

This documentation describes the various LAMPS 90 programs. For information on implementing LAMPS 90 on your machine or methods of compiling and running LAMPS 90 programs, see the *LAMPS 90 User's Guide*.

1.3 Prerequisites

If you are interested in using the programs discussed in this guide, it is strongly suggested that you first read the *LAMPS 90 User's Guide*, which reviews the basic components of LAMPS 90 and also introduces the terminology. This guide assumes that the reader is already familiar with the information in the User's Guide.

1.4 Using this documentation

Each chapter in this guide documents one of the programs available within LAMPS 90. Three types of programs are available, corresponding to the three parts of the process required to make a simulation.

(1) INITIALIZATION - get a consistent dataset for the dynamic model
(2) DYNAMIC MODEL - use the initialization data to create a forecast

(3) POST-PROCESSING - examine forecast fields (e.g., plots)

This guide is organized according to the three types of programs. Part I documents all of the initialization programs. Part II documents the two dynamic model programs. Part III documents the post-processing programs.

Each program usually requires routines from one or more of the LAMPS 90 libraries. Part IV documents the various LAMPS 90 libraries.

Each program is built from files that contain various routines. Each initialization program is stored in a sub-directory of **src/init**. Each dynamic model program is stored in a sub-directory of **src/model**. Each post-processing program is stored in a sub-directory of **src/post**. Each library is stored in a sub-directory of **src/rlib**.

For each program, the corresponding main routine is stored in the appropriately named subdirectory of the above directories. For example, **src/init/lyrmean** for the **lyrmean** codes, **src/rlib/liblamps** for the liblamps library codes.

1.5 Program dependencies

1.5.1 Source codes

One can find out what source codes are needed to build each program by referring to the individual program makefiles. These makefiles, one for each program, are stored in the **config/mks** sub-directory of the main LAMPS 90 directory. By setting up one's case directory with **Setup**, these makefiles are copied into a **mks/** sub-directory of the case directory.

Each individual program makefile lists the entire dependency list for the particular program, including the libraries and any included files.

Note: In order to maintain a consistency between the various programs, all of the codes have all of the established geophysical constants set via an include file (i.e., **incl/const.h**) or a call to the general purpose **const** (within the liblamps library). All of the machine-dependent constants are set via an include file (either **incl/generic.cfg** or **incl/mach.h**).

1.5.2 Input/Output

There are two types of input normally required for each program. The run-time instructions are usually stored in an input file. Information on the input card structure needed/possible for each program is given in a series of default input files in the **config/inp** subdirectory of the main LAMPS 90 directory. By setting up one's case directory with **Setup**, these input files are copied into an input sub-directory of the case directory. See the *LAMPS 90 User's Guide* for more information.

The program may also require some data, possibly produced by a prior program. The dependency of the initialization programs are shown in figure 1.1. All of the initialization programs typically read and write to the same file, called the initialization data volume. The final product of **wdadjust** (or **vertran**, **gdinterp** or **analytic**) is then read by **merge** which, in turn, produces an initial data file. This data file is read by the dynamic model programs which, in turn, produce a model history data volume. This data volume can then be processed by the post-processing programs.

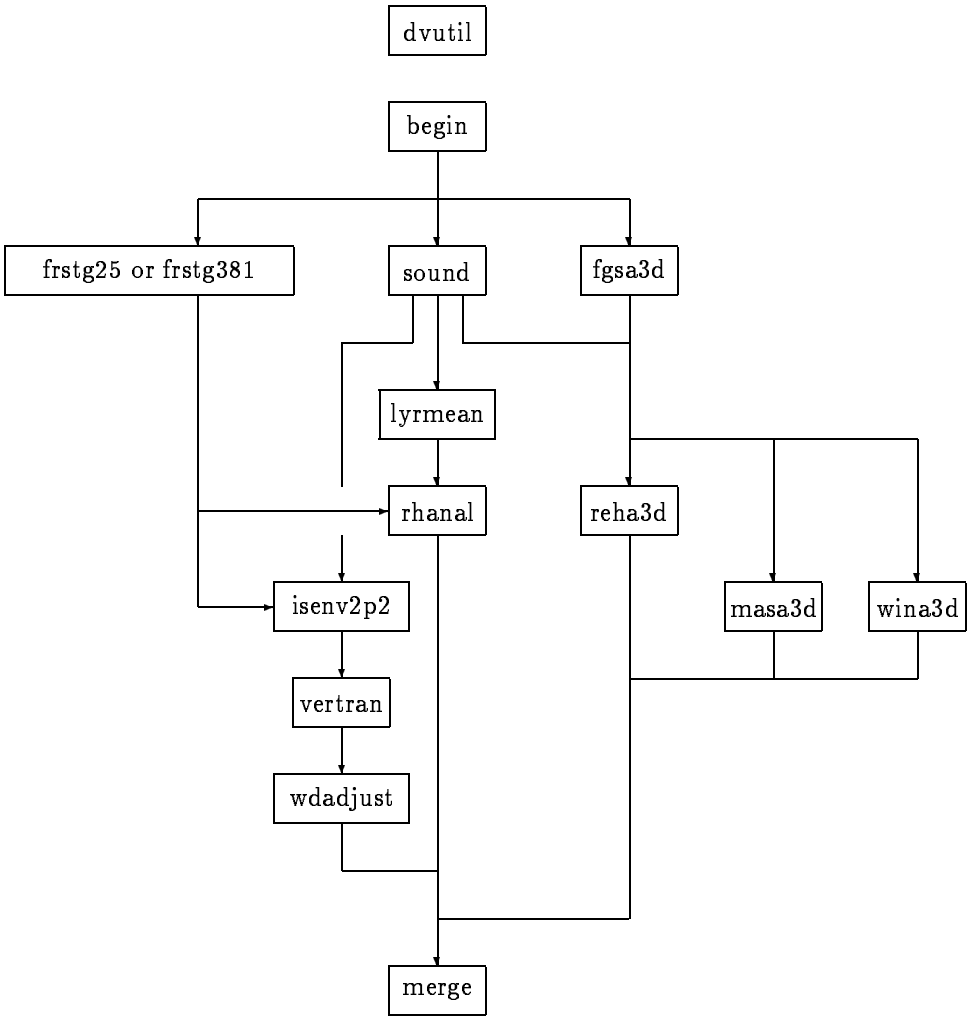

Figure 1.1: Flowchart depicting the sequence of programs needed to create an initial data set with real data. The **dvutil** program is normally used although it is not necessary. All of the programs after **begin** require the output from **begin**. It is also possible to by-pass the **reha3d**, **masa3d** and/or **wina3d** programs and have the **merge** program use the **fgsa3d** output directly.

Part I

Initialization

Chapter 2

analytic

2.1 Description

The **analytic** routines provide a simple way of producing data to test the dynamic model without having to go through the complicated initialization process using real data. This program does not create an initialization data volume. Instead, it creates a model initial data file for direct input into the dynamic model (without needing **merge**).

Basically, the **analytic** program uses analytic equations to simulate a certain type of atmospheric structure. See table 2.1 for the options available.

2.2 Equations

2.2.1 Stable Atmosphere

A stable atmosphere assumes

1. $d\theta/dz$ is constant
2. hydrostatic balance
3. ideal gas
4. no tropopause
5. potential temperature is constant at sea level

Assuming hyrdrostatic balance gives

$$d\pi = -\frac{g}{\theta} dz$$

Replacing dz with

- standard atmosphere
  specify : tropopause height,
  lapse rate in troposphere,
  potential temperature at the surface,
  U-component wind speed
- hills
  specify : position (lat/long),
  horizontal wave numbers,
  height
- stable atmosphere
  specify : vertical potential temperature gradient,
  potential temperature at the surface,
  U-component wind speed
- lamb wave
  specify : horizontal wavelength,
  U-component wind speed amplitude,
  Average temperature
- gravity wave
  specify : horizontal wavelength,
  vertical wavelength,
  temperature amplitude,
  Average temperature
- manual wave
  specify : horizontal wavelength,
  U-component wind speed amplitude,
  Average temperature

Table 2.1: Input options for analytic initialization

$$d\theta \left(\frac{d\theta}{dz}\right)^{-1}$$

gives

$$d\pi = -g \left(\frac{d\theta}{dz}\right)^{-1} d\ln\theta$$

Integrating this expression gives

$$\pi - \pi_s = -g \left(\frac{d\theta}{dz}\right)^{-1} d\ln\frac{\theta}{\theta_s}$$

or

$$\pi = \pi_s - g \left(\frac{d\theta}{dz}\right)^{-1} d\ln\frac{\theta}{\theta_s}$$

where

$$\theta = \theta_s + \frac{d\theta}{dz}$$

For wind in geostrophic balance,

$$\theta\frac{d\pi}{dy} = -fu$$

and

$$\theta\frac{d\pi}{dx} = +fv$$

If the wind and potential temperature remain constant then, replacing f by $2\Omega\sin(\phi)$ and dy by $R_e d\phi$, we get

$$\theta d\pi = -2\Omega\sin(\phi)uR_e d\phi$$

Integrating, we get

$$\theta(\pi_2 - \pi_1) = 2\Omega uR_e(\cos\phi_2 - \cos\phi_1)$$

Note: We could also keep T constant, but that would lead to a decrease of θ to the north.

2.2.2 Standard Atmosphere

A standard atmosphere assumes

1. dT/dz is constant
2. hydrostatic balance
3. ideal gas
4. at some tropopause height, dT/dz becomes zero
5. potential temperature is constant at sea level

Hyrdrostatic balance gives

$$d\pi = -\frac{g}{\theta} dz$$

The ideal gas law gives

$$\pi = c_p \frac{T}{\theta}$$

The ratio gives

$$d \ln \pi = -\frac{g}{c_p T} dz$$

Replacing dz with

$$d\theta \left(\frac{d\theta}{dz} \right)^{-1}$$

gives

$$d \ln \pi = -\frac{g}{c_p} \left(\frac{dT}{dz} \right)^{-1} d \ln T$$

Integrating this expression gives

$$\frac{\pi}{\pi_s} = \left(\frac{T}{T_s} \right)^{-g/(dT/dz)/c_p}$$

or

$$\pi = \pi_s \left(\frac{T}{T_s} \right)^{-g/(dT/dz)/c_p}$$

Above the tropopause, T is constant so that the ratio

$$d \ln \pi = -\frac{g}{c_p T} dz$$

can be integrated to give

$$\ln \frac{\pi}{\pi_{\text{trop}}} = -\frac{g}{c_p T_{\text{trop}}} (z - z_{\text{trop}})$$

$$\frac{\pi}{\pi_{\text{trop}}} = \exp \left[\frac{g}{c_p T_{\text{trop}}} (z_{\text{trop}} - z) \right]$$

Chapter 3

begin

3.1 Description

begin adds grid, terrain, surface type and SST to the data volume. More specifically, it outputs the latitudes, longitudes and vertical coordinates of the grid to be initialized. Smoothed terrain tables are also determined and output along with information on the sea-surface temperature and surface type.

3.2 Input/Output

begin needs the terrain data volume (output from **terrain**) and the SST climatology. The ERICA 50 km SST and ERICA 14 km SST data sets are optional.

Case-independent data files (e.g., sea-surface temperature from climatology and the terrain data) should be in the **dbs/** sub-directory of the main LAMPS 90 directory.

The output from **begin** is accessed by all of the initialization programs that deal with real data. The output is also needed by **gdinterp**.

Chapter 4

fgsa3d

4.1 Description

fgsa3d creates a first guess (model levels) from NMC 2.5 deg. data. As such, it needs the NMC 2.5 degree gridded data.

For more information, see Dr. Jing Guo at Drexel University.

4.2 Input/Output

fgsa3d needs the output from **begin**.

Chapter 5

frstg25

5.1 Description

frstg25 creates a first guess (θ levels) from NMC 2.5 deg. data. The NMC analysis is interpolated onto the latitude/longitude grid specified by the **begin** output. Vertically, the data is interpolated onto θ_v levels at 2.5 K intervals. The variables that are output are the modified Exner function (π), the partial derivative of pi with respect to θ_v, the montgomery stream function (M) and the u and v winds. In addition, relative humidity is provude on model levels below approximately 490 mb.

The following information was provided in the old LAMPS documentation by D. Miller (7/79).

The "first guess" fields are taken from the NMC analyses for the initial time, which utilize both sounding observations taken at that time and the forecast results of the previous, 12-hour run of the NMC model.

5.2 Input/Output

frstg25 needs the output from **begin** along with the NMC 2.5 degree gridded data.

5.3 Procedure

The **frstg25** program was written mainly by Jing Guo at Drexel University.

First the fields are interpolated horizontally (along a pressure surface) to the model grid points via a two-dimensional cubic spline. Then the mass fields T_v, z and p are converted to θ_v, π and the Montgomery stream function ψ as follows.

$$\theta_v = T_v \left(\frac{1000}{p} \right)^{R_d/c_p}$$

$$\pi = c_p \left(\frac{T_v}{\theta_v} \right)$$

$$\psi = \pi\theta_v + gz$$

where R_d is the gas constant for dry air, c_p is the specific heat for dry air at constant pressure and g is the acceleration due to gravity.

The final step is to vertically interpolate the fields onto θ_v levels. A first-order scheme is used to interpolate π, $\mathbf{v}$ and q. A second-order scheme is used to interpolate ψ. Values of stability $-\partial\pi/\partial\theta_v$ are also calculated and interpolated onto the θ_v grid in order to maintain stability during the optimal interpolation. Values of stability are obtained by assuming a constant stability within the layer defined by the two adjacent pressure levels.

Chapter 6

frstg381

6.1 Description

frstg381 creates a first guess (model levels) from NMC 381 km data. As such, it needs the NMC 381 km gridded data.

6.2 Input/Output

frstg381 needs the output from **begin**.

6.3 Procedure

The way to run **frstg381** is similar to that for **frstg25** while the algorithm is more interesting than **frstg25**. See Dr. Jing Guo at Drexel University for more information.

Chapter 7

gdinterp

7.1 Description

gdinterp takes a three-dimensional grid of data and interpolates it to a different grid. Data are interpolated horizontally and vertically using bilinear interpolation. **gdinterp** is usually used to take a data set that has already been run through the initialization (or simulation) process and interpolate it to a finer grid.

7.2 Input/Output

gdinterp can interpolate data from either an initialization data volume (with output from **begin**, **rhanal** and either **vertran** or **wdadjust**), the initialization file (i.e., **merge** output) or a volume-format model history data volume (i.e., output from **reformat**).

gdinterp also needs **begin** output that corresponds to the new grid.

The output will be similiar to the output from **rhanal** and **vertran**. One then needs to run **wdadjust** (optional) and **merge** to get the data ready for the dynamic model.

Chapter 8

isenv2p2

8.1 Description

isenv2p2 takes the observation information from **sound** and the first-guess information from either **frstg381** or **frstg25** and performs an analysis of the Montgomery stream function M, the modified Exner function pi, $-\partial\pi/\partial\theta_v$, u and v on θ_v levels.

8.2 Input/Output

isenv2p2 needs the output from **begin**, **sound** and either **frstg25** or **frstg381**.

8.3 Procedure

The **isenv2p2** program was written mainly by Jing Guo at Drexel University. Eventually, it should be replaced by the 3-d analysis programs of **wina3d** and **masa3d**.

The analysis procedure is as follows. The π, ψ and **v** fields are compared to the observational data and modified via a two-dimensional univariate optimal interpolation scheme that utilizes correlation functions represented by Bessel functions of zero order $J_o(\alpha_k x)$. The correlation functions are modeled on a case-by-case basis. See Guo and Kreitzberg 1993 for a complete discussion on the procedure for modeling the correlation functions.

8.4 References

Guo, J., and C. W. Kreitzberg, 1993: *A Three-dimensional Statistical Analysis System for a Limited Area and Mesoscale Prediction System*, 34 pp. (plus figures) [Manuscript available from authors, Dept. of Physics and Atmospheric Sciences, Drexel University, Philadelphia, PA 19104. e-mail: jguo@nimbus.physics.drexel.edu]

Chapter 9

lyrmean

9.1 Description

lyrmean computes the layer mean values of sounding data.

The following information was provided in the old LAMPS documentation by D. Miller (7/79).

lyrmean takes the sounding output (from **sound**) and prepares layer-mean values of u, v, T_v and RH for the **rhanal** program and for verification purposes.

9.2 Input/Output

lyrmean needs the output from **begin** and **sound**.

Chapter 10

merge

10.1 Description

merge reformats the **vertran** output to the model slab format.

10.2 Input/Output

merge needs the output from **begin**, either **rhanal** or **reha3d** and either **wdadjust** or both **masa3d** and **wina3d**.

Chapter 11

masa3d

11.1 Description

masa3d does a 3-D analysis of the mass field.

For more information, see Dr. Jing Guo at Drexel University.

11.2 Input/Output

masa3d needs the output from **begin**, **fgsa3d** and **sound**.

Chapter 12

reha3d

12.1 Description

reha3d does a 3-D analysis of relative humidity field.

For more information, see Dr. Jing Guo at Drexel University.

12.2 Input/Output

reha3d needs the output from **begin**, **fgsa3d** and **sound**.

Chapter 13

rhanal

13.1 Description

rhanal takes the observation information from **lyrmean** and the first-guess information from either **frstg381** or **frstg25** and performs an analysis of the relative humidity field on the model h-surfaces.

13.2 Input/Output

rhanal needs the output from **begin**, **lyrmean** and either **frstg25** or **frstg381**.

13.3 Procedure

The **rhanal** should eventually be replaced with the 3-d analysis program **reha3d**. The following information was provided in the old LAMPS documentation by R. G. Rasmussen (7/78).

13.3.1 Method

Relative humidity is analyzed on a horizontal plane using an optimal interpolation scheme in a single variable. A model forecast or external analysis is used as a first guess for the first pass. For subsequent passes, the analysis obtained from the previous pass is used as the first guess. Also in each subsequent pass the radius of influence is decreased. Only data from within the circle of influence about a grid point is used in the analysis at that grid point. The analysis is repeated at each model level.

13.3.2 Analysis Equation

The analyzed humidity at a grid point is given by the first guess at the grid point plus a series of corrections to the first guess. There is one correction term for each observation within the circle of influence.

$$r^a = r^g + w_n(r_n^o - r_n^g) \tag{13.1}$$

Summation is implied by the repeated indices. Here, r^a is the analyzed value and r^g is the guess value of the humidity at the grid point. THe subscript indicates values at the observations points: r_n^o is the observed value, r_n^g is the guess value and w_n is a weight for the correction factor at point n.

13.3.3 Correction Weights

The weights in equation 13.1 are determined by requiring that the mean square analysis error (i.e., the analysis error variance) be minimal.

$$(\sigma^2)^a \equiv \overline{(r^a - r^t)^2} = \overline{[(r^g - r^t) + w_n(r_n^o - r_n^g)]^2}$$

Here r^t isthe "true" relative humidity at the grid point. The criterion that $(\sigma^2)^a$ must satisfy is:

$$\frac{\partial(\sigma^2)^a}{\partial w_l} = 0 \tag{13.2}$$

Before evaluating the weights, we simplify the notation:

$$f^t \equiv r^t - r^g \qquad\qquad f_n^o \equiv r_n^o - r_n^g$$

Thus equation 13.2 may be written:

$$\frac{\partial}{\partial w_l}\left[\overline{(f^t)^2} + w_m w_n \overline{f_m^o f_n^o} - 2w_n \overline{f^t f_n^o}\right] = 0$$

When simplified, we have

$$\overline{f^t f_l^o} = \overline{f_l^o f_n^o} w_n \tag{13.3}$$

The term on the left may be identified as the covariance of the true humidity at the grid with the observed humidity at the point l when both are referenced to the guess field values at the respective points.

At this point, we assume that:

$$\sigma^2 = \overline{(f^t)^2} = \overline{(f_l^o)^2} = \overline{(f_n^o)^2}$$

where l and n are arbitrary observation points. Using σ^2 to divide both sides of equation 13.3, we obtain an expression for the weights in terms of the correlation of the true humidity at the grid with the observed humidity at point l and also the correlations of the observed humidities.

$$\rho_l^{to} = \rho_{ln}^{oo} w_n$$

This equation may be expressed in vector and matrix notation and the resulting equation may be solved for the weights:

$$\mathbf{W} = (\mathbf{R}^{oo})^{-1}\mathbf{R}^{to}$$

Thus knowledge of the various correlations allows us to compute the weights and perform the analysis.

13.3.4 Correlation Model

The correlations are modelled with a Gaussian function of only separation. Ideally, the functional form of the correlation function should be obtained objectively from real data. In our case, we subjectively assigned a correlation function

$$\rho(d) = \exp(ad^2)$$

We determined the paramter a by requiring that

$$\rho(d_o) = \rho_o \qquad \text{or} \qquad a = \frac{\ln \rho_o}{d_o^2}$$

where d_o and ρ_o were subjectively choses to be 500 km and 0.04 respectively. WIth this choice, a turns out to be -1.288×10^-5 km^{-2}.

For the first pass through the data, the radius of influence is set equal to 500 km. After each pass the radius is halved. The iteration is terminated if either of the two following criteria are satisfied:

(a) the radius of influence drops below 20 km

(b) the maximum deviation f_n^o over the entire set of observations is less than or equal to 5%.

Usually the iteration is terminated by criteria (b) being satisfied after three or four passes.

13.3.5 Data

Currently, only synoptic upper air data is used. The raw rawinsonde data occurs on mandatory and significant levels. Prior to the analysis, layer mean humidities are computed at the model levels. Future plans call for the use of surface radar, and satellite data in addition to the upper air data.

13.3.6 Reference

The analysis scheme was previously documented in the following reference:

Perkey, D. J., 1976: Predicition of convective activity using a system of parasitic-nested numerical models. *NASA contractor report*, CR-2761, National Aeronautics and Space Administration, Washington, DC.

Chapter 14

sound

14.1 Description

sound adds sounding data to data volume.

14.2 Input/Output

sound needs the output from **begin**. In addition, it needs sounding files. Currently, it can handle NMC or GALE/ERICA format soundings.

14.3 Procedure

The following information was provided in the old LAMPS documentation by D. Miller (7/79).

The soundings which provide the initial data for the LAMPS model are first checked for errors and smoothed. After the mandatory and significant level data have been ordered sequentially, for each sounding, a check is made for "spikes" in the temperature data (i.e., levels at which the temperature deviates markedly from the rest of the trace, such that a dry unstable layer, and an extremely stable layer, exist in succession). If such a spike can be eliminated by experimentally changing the sign of the dubious temperature (which is reported in ∘C) that change is made permanent. THe vertical lapse rate of temperature is then computed between successive levels of each sounding, beginning at the surface. If a dry, unstable layer is encountered, the sounding is entirely eliminated from the data set. (In general, a potential temperature decrease with height of up to 2∘C is permitted between any two successive levels, and in the first two layers above the surface, a potential temperature decrease with height of up to 5∘C is deemed acceptable.)

During instances in which a temperature sign is changed in order to eliminate a "spike", the dew-point temeprature is changed by the same magnitude, so as to preserve the dew-point depression. At levels where the dew-point temeprature comes in "missing", a default value of relative humidity is assigned, so as to best fit a typical, mid-latitude U.S. Standard Atmosphere moisture profile, according to the formulas:

$$\mathrm{RH} = \begin{cases} .2P - 125 & \text{for } P \geq 850 \\ .033P + 16.7 & \text{for } p < 850 \end{cases}$$

where P is the pressure in mb and RH is the relative humidity is percent.

With a now complete vertical moisture profile, the virtual temperature can be computed at each level of the sounding. THe "thickness", or geopotential height-difference, is then calculated hydrostatically between each pair of successive levels, beginning at the surface and proceeding upwards; the computed height at each level is thus the sum of the thickness of all the layers below. These hydrostatically computed heights are then compared, where possible (i.e., only at the "mandatory" levels), with the height values which were reported in the original radiosonde code. If the difference between the computed and reported thicknesses ($\delta\Delta z$) in the first layer above the surface is 25 m or more, and if $\delta\Delta z$ in the second layer above the surface is less than 30% of the $\delta\Delta z$ in the first layer, it is assumed that the radiosonde station in question has been moved to a different location, and is now at a different surface elevation than that which was previously reported and was used in the hydrostatic computation of the heights. In such an instance, the value of the $\delta\Delta z$ at the first mandatory level above the surface is added to the previously assumed station elevation to arrive at a corrected elevation, and all the heights are recomputed hydrostatically upwards, in the same manner as before. In either the event that a correction is or is not made to the station elevation, the final height values of a sounding which are input into the LAMPS model are the hydrostatically computed, rather than the originally code, heights.

The wind data of each sounding (which is coded separated from the height, pressure, temperature and dew point temperature data) is also checked for spikes. The speed and direction information are considered individually, and a spike is considered to be present if the wind speed at a given level differes from the average value of the wind speeds of the levels one-above and one-below by 50 knots or more; or if the product of the wind speed at a given level (in knots) and the difference between the reported and the average (of the levels above and below) values of the wind direction at that level (in degrees) is of the magnitude 3600 or greater. If the first criterionfor a spike is met, integer multiples of 100 knots are added to, or subtracted from, the dubious reported wind speed, and if one of these experimental corrections can bring the speed to within 20 knots of the average wind speed value of the layers above and below, this corrected value is inserted permanently into the wind sounding. If the second criterion for a spike is met, the same procedures and tests as described above are applied to the wind direction data, with the units "degrees (radial)" replacing the "knots" in the description. If a spike is not eradicated by either of these procedures, the wind velocity vector at the level in question is removed and replaced by a "missing data" notation – the missing data not being used in any further analyses. The remainder of the wind sounding data, however, is retained.

Chapter 15

tbound

15.1 Description

tbound generates time-dependent boundary conditions from a time-series of **merge** output.

15.2 Input/Output

tbound needs a time-series (at least two) of **merge** output. The output of **tbound** can then be used in conjunction with an initial data file as input to the dynamic model.

Chapter 16

terrain

16.1 Description

terrain generates a terrain data volume from the global terrain information stored in the Navy terrain. This is done to save space because the Navy terrain file is very large (about 19 MB) and the output from **terrain** is only about 2 MB. The output contains all of the information for the Northern Hemisphere. Thus, as long as one is running simulations with grid solely in the Northern hemisphere, one doesn't need to rerun **terrain**.

16.2 Input/Output

terrain needs the Navy terrain data file. The output from **terrain** is used by **begin**.

Chapter 17

vertran

17.1 Description

vertran takes the grid information from **begin** and the data fields on θ_v surfaces from **isenv2p2** and interpolates the θ data vertically to the model levels (sigma height surfaces). The values of π, θ_v, u and v are output on h-surfaces.

17.2 Input/Output

vertran needs the output from **begin** and **isenv2p2**.

Chapter 18

wdadjust

18.1 Description

wdadjust adjusts the winds to the terrain. Plans are to replace the current **wdadjust** with a digital filter (see Dr. Jing Guo at Drexel University for more information).

The following information was provided in the old LAMPS documentation by C. Chang (7/79).

The primitive equation model is very sensitive to the initial state. An initial state containing a large imbalance between the velocity and the mass fields can lead to large external and internal gravity wave oscillations which may grow during the forecast and, hence, ruin the large-scale atmosphere. Unfortunately, the analyzed wind and pressure on the model levels often contain large imbalances which are unlikely to be representative of the true state of larger-scale motions. The major purpose of this program is to suppress the amplitude of the external gravity wave in the analyzed wind fields. The technique used here is to adjust the divergence part of the wind such that the vertical velocity at the model top becomes very small. This is consistent with the upper boundary condition ($w = 0$) in the prediction model[1].

The diagnostic vertical velocity equation used in this program is

$$\frac{\partial}{\partial h}\frac{P}{\pi}\dot{h} = -\nabla \cdot \frac{P}{\pi}\mathbf{v} + \frac{P}{\pi}\left(v\frac{\tan\phi}{a} + \frac{\mathbf{v}\cdot\nabla E}{H-E}\delta\right) \tag{18.1}$$

where P, ϕ, a and E are pressure, latitude, radius of the earth, and height of the smoothed terrain, respectively, and δ is 1 for $h < H$ ($= 5.25$ km)[2] and 0 for $h \geq H$. The vertical coordinate h is defined as

$$h = H\frac{z-E}{H-E}$$

At $h = 0$, $\dot{h}$ is identical to zero because of the kinematic boundary condition ($w = \mathbf{v}\cdot\nabla E$).

The adjustment procedures are carried out as follows.

[1] Ed. note: The actual upper boundary condition is $dp/dt = 0$.
[2] Ed. note: The actual value H should be variable and set in the input to **begin**.

1. Integrating equation 18.1 upwards from $h = 0$ to $h = h^*$, we have

$$\int_0^{h^*} \frac{\partial}{\partial h}\frac{P}{\pi}\dot{h}dh = \int_0^{h^*} \text{RHS}dh$$

where h^* is the first model level above H and the RHS represents the sum of those terms on the right hand side of equation 18.1. The upward integration yields

$$\frac{P}{\pi}\dot{h}_{\uparrow}(h^*) = \int_0^{h^*} \text{RHS}dh \tag{18.2}$$

Integrating equation 18.1 downward form the model top ($h = z_t$) to $h = h^*$, assuming $\dot{h} = 0$ at z_t, we have

$$\frac{P}{\pi}\dot{h}_{\downarrow}(h^*) = -\int_{z_t}^{h^*} \text{RHS}dh \tag{18.3}$$

2. The difference between the left hand sides of equations18.2 and 18.3 is computed.

$$\Delta h^* = (\dot{h}_{\uparrow} - \dot{h}_{\downarrow})\frac{P}{\pi} \tag{18.4}$$

Now we have to adjust the divergence part of the wind to achieve a condition of very small $\Delta h^* P/\pi$ at z_t.

The amounts of divergence to be adjusted at the model levels is determined by the formula

$$F(h) = \sigma(h)\frac{P}{\pi}\Delta h/(\Sigma_1^N \sigma(h)\Delta h) \tag{18.5}$$

where $\sigma(h) = (1 - h/H)\delta$, N is the number of model levels, and Δh is the thickness between two adjoining levels.

3. Now, we define a scalar quantity called the velocity potential and denoted by ξ as follows.

$$\frac{\partial \xi}{\partial x} = \frac{P}{\pi}u_D$$

$$\frac{\partial \xi}{\partial y} = \frac{P}{\pi}v_D$$

where u_D, v_D are the divergent part of the wind. Because the non-divergent part of the wind does not contribute to the vertical motion, equation 18.1 can be rewritten in terms of the velocity potential.

$$\frac{\partial}{\partial h}\frac{P}{\pi}\dot{h} = -\nabla^2\xi + \frac{\tan\phi}{a}\frac{\partial \xi}{\partial y} + \delta\frac{\nabla E}{H - E}\cdot\nabla\xi$$

If we replace the left hand side of this equation be $F(h)$ of equation 18.5, then solve for ξ by the relaxation method, we will obtain a ξ field corresponding to the amount of divergence needed to be adjusted at a given level. Thus, the working equation for wind adjustment can be expressed as

$$F(h) = -\nabla^2\xi + \frac{\tan\phi}{a}\frac{\partial \xi}{\partial y} + \delta\frac{\nabla E}{H - E}\cdot\nabla\xi \tag{18.6}$$

The boundary condition used for solving equation 18.6 is $\xi = 0$.

4. Compute the adjusted winds by

$$\mathbf{v}_{\text{adj}} = \mathbf{v} - \frac{\nabla \xi}{P/\pi}$$

This computed $\mathbf{v}_{\text{adj}}$ will satisfy the upper boundary condition of very small $\dot{h}$.

Note: $\sigma(h)$ of equation 18.5 is 1 at $h = 0$ and 0 for $h > H$. Therefore, adjustments are performed only for model levels below H under the assumption that the external gravity waves are mainly excited by introducing terrain. However, equation 18.6 is a general form of working equation which is suitable for any vertical profile of $\sigma(h)$.

18.2 Input/Output

wdadjust needs the output from **begin** and **vertran**.

Chapter 19

wina3d

19.1 Description

wina3d does a 3-D analysis of wind field.

For more information, see Dr. Jing Guo at Drexel University.

19.2 Input/Output

wina3d needs the output from **begin**, **fgsa3d** and **sound**.

Part II

Dynamic Models

Chapter 20

model

FILE	PROCESS
model.pre	main control
initial.pre	user input
start.pre	variable initialization
output.pre	variable output to standard output
history.F	variable output to data volume format
physics1.pre	physics control
physics2.pre	physics library
difdrvs.pre	diffusion and derivative calculations
pblphys.pre	PBL parameterization
flow3.F	array flow handling
flowcopy.F	array flow copying
flowmach.F	I/O handling
modconst.F	constants
cumulus.pre	cumulus parameterization
cloud.F	cumulus library

20.1 Description

20.1.1 variables

The **model** program simulates the evolution of the atmosphere in terms of four basic variables : virtual potential temperature, momentum (u and v components of the wind), the Exner function (related to pressure) and moisture. There are others, of course, but these are the principle ones. In fact, these are the ones that are predicted (although the Exner function is only predicted at the top). These variables are called prognostic variables. Other variables, like the Exner function below the top, can be derived from the others and are called diagnostic variables.

The dynamic model is also a three-dimensional model[1]. This means that it calculates the atmospheric variables at specified points in the x-direction (longitude), y-direction (latitude) and z-direction (σ_z height). The grid spacing is specified in the following manner via **begin** or **analytic**.

Latitude... The increment in the latitude direction is specified in degrees.

[1] The model can also be used as a two-dimensional model by using a y-dimension of 1.

Longitude... The increment in the longitude direction is obtained from the latitude increment by converting the latitude increment to units of length and then converting this length to the equivalent number of longitude degrees at some reference latitude. If the reference latitude is not specified by the user during **begin**, it is assumed to be the average latitude.

σ_z **Height**... Since the desired vertical resolution usually changes with height, the user needs to specify each height of the vertical grid. Near the surface the resolution is usually much finer that at higher levels. In addition, the surface parameterization scheme (in the full, physics model) may require a particular vertical grid near the surface.

The vertical coordinate is σ_z height. This is similar to geopotential height, but it follows the terrain near the ground according to the following relationship.

$$\sigma_z = \begin{cases} (z - E)\frac{H}{H-E} & \text{for } h < H \\ z & \text{for } h \geq H \end{cases}$$

where σ_z is the vertical coordinate used in the model, z is the geopotential height, E is the elevation height and H is a height specified during the **begin** step.

The grid lengths, as well as time, are scaled in order to keep the magnitude of the model variables close to unity (see *LAMPS 90 Resource Manual No. 1, Dynamic Model Equations.*

model variable	time	speed	hor. grid length	vert. grid length
scale	10,000 s	10 m s^{-1}	100,000 m	1000 m

In order to save space, all of the two and three dimensional variables in the model are usually stored as part of the workspace array (i.e., `wksp` in the code). The workspace array contains information for all of the variables and for all of the x (longitude) and z (height) values but only for a couple of y (latitude) values. That is, only a few slabs are stored in the workspace, a slab containing all of the x and z values of all of the variables associated with one y value.

The information for the slabs not stored in the `wksp` array are instead stored in one of two files called flow files. Each flow file contains all the information for all of the workspace variables for one time step. The current time step alternates between the two flow files. Each time a new variable is predicted, the new information goes into a temporary workspace array (`wksp_new` in the code). Once that slab is processed, the temporary information is then saved to the flow file.

This procedure of copying and saving is slow but it minimizes the amount of memory that is needed. Furthermore, it makes outputting variables simpler because one can specify the list of variables to be saved just by giving the relative order of the variable in the workspace array.

20.1.2 structure

Before the integration or forecast is started, the dynamic model does two things.

1. First, the parameters which determine the type of simulation need to be set. This process is controlled by routine `init` in **src/model/model/initial.pre**. See section 20.2.1 for more information.
2. Second, the initial meteorological conditions are set by reading in the initial state from somewhere. This process is controlled by routine `begin` or `restart` in **src/model/model/start.pre**. See section 20.2.2 for more information.

Once this initialization is complete, the dynamic model can start stepping forward in time as it makes the forecast. At each step, the dynamic model does two checks.

1. First, any requested output is created. This can be in three different forms, each controlled by a different routine. Two (`history` and `save`) are in **src/model/model/history.F** and the other (`output`) is in **src/model/model/output.pre**. See section 20.3 for more information.
2. Then, any updates are done. These updates are for optional physics or physics which don't need to be updated at every time step, e.g., time-dependent boundaries, radiation, cumulus parameterization, surface parameterization and planetary boundary layer parameterization.

After these checks, the dynamic model can perform its physics in the following order. These processes are all called by the routine `physics` in **src/model/model/physics1.pre**.

1. cumulus and surface energy calculations are updated, if necessary
2. horizontal and vertical derivative quantities are calculated
3. vertical velocity profile is calculated
4. horizontal diffusion quantities are calculated
5. diffusion coefficients are updated, if necessary
6. vertical diffusion quantities are calculated
7. radiation calculations are updated, if necessary
8. time tendencies are calculated and adjusted
9. the prognostic variables are marched forward in time using a leap-frog scheme
10. temperature and moisture field is adjusted
11. surface values are diagnosed
12. the new thermodynamic field of π, etc. is diagnosed

For information on how these calculations are made, see the *LAMPS 90 Resource Manual No. 1, Dynamic Model Equations.*

20.2 Input

To run the dynamic model, two types of input are required. One is the command list which tells the model how to run the forecast (i.e., the input cards) and the other is the initialization data file.

20.2.1 Input Cards

Before the model can begin its forecast, it must have some instructions from the user telling it how long to forecast, the time increment, etc. These instructions can be typed in interactively at the start of the run or they can be stored in a separate file that is redirected as input to the program (see *LAMPS 90 User's Guide* for more information).

The types of instructions available are listed here.

bf Command	bf Explanation
`.title`	title of run
`.time`	start and end time of simulation
`.flow`	names of flow scratch files (for storage of variables)
`.start`	how model will be started (i.e., where initial data is from)
`.history`	frequency of data output and name of the file for history
`.hisout3`	list of 3-D variables to save to history file
`.hisout2`	list of 2-D variables to save to history file
`.save`	frequency of data output and name of the file for "save" dump
`.output`	frequency and type of ascii output
`.timbnd`	file containing tendencies around the boundaries (if needed)
`.vartit`	name of file containing titles of the variables in the model
`.run`	start forecast (last input card)

20.2.2 Initialization Data File

There are currently three ways to initialize the model:

1. from some external initialization process (e.g., from the LAMPS 90 initialization process)
2. from a model dump
3. from a model history file

For each case, all the data that is present is read into the model variables. In addition, all of the following is done.

1. grid parameters are initialized (latitude and longitude boundaries, latitude step and longitude step)
2. sigma parameters are initialized (levels which are affected by terrain)
3. frequency counters are initialized
4. all other variables are set (i.e., those not specified by initialization data but which are still needed in the model)

Note: If values from the previous time step are not read in, then they are set to the current values.

20.3 Output

20.4 Standard Output

During the process of making a forecast, the model will generate messages for standard output. For example, whenever a history record is written, a message will be generated. In addition, a message stating the forecast time will be printed every time step. Any problems encountered during the forecast (e.g., difficulties finding a solution to integrations) will also be printed.

Besides these general comments, the user can also request that specific information be printed regarding the values of specified variables at specified grid points. The user specifies these requests via the `.output` card. The types allowed are as follows.

Command	Explanation
`secvar`	specify variables to print y-sections (X/Z cross sections)
`slabs`	y-values to print for X/Z cross sections
`ptvar`	specify variables to print point values
`point`	grid points to print for point value output
`horiz3d`	3-D variables and z-sections to print for X/Y cross sections
`horiz2d`	2-D variables to print
`endout`	must always end the `.output` option list

When the `point`/`ptvar` option is used, the value of the specify variable(s) is printed for the specified grid point and the surrounding grid points (± 2 grid points in each horizontal direction). The x, y and z derivatives and the x, y and z diffusions are also printed. When the other options are used, not all the grid point values are printed in order to get the output to be under 132 characters wide.

When using the `.output` options, it is important to use the `.vartit` command so that the output is identified according to variable name.

20.4.1 History

During the forecast process, the model will occasionally save some model data for the user to examine. This data set is called the history because it contains a history of the forecast.

Any of the workspace variables (i.e., `wksp`) can be saved to the history (see section 20.1.1). One specifies which variables to be saved by use of the `.hisout3` and `.hisout2` input cards.

The history file is saved in the data volume format. This means that it requires special routines to read and write the data. Even though the data volume consists of one physical file, it can be thought of having several internal files.

First internal file	initialization date
	title of run
	dimensions of run (x,y,z)
	grid (lat, long, σ_z)
	variables which are stored
other internal files	dimensions of run (x,y,z)
	grid (lat, long, σ_z)
	specified data

Note: The data is saved in the "slab" structure.

Note: The velocity variables are rescaled so that they are stored in units of m s^{-1} (u and v) and cm s^{-1} (vertical velocity).

20.5 Array Dump

In order to prevent a total loss of a forecast when the model bombs, an option is available in which the entire work space array will be saved at a certain frequency, over-writing itself for each dump.

All the variables which are part of the workspace (i.e., `wksp`) will be dumped.

The save file is saved in the FORTRAN, sequential, unformatted format. This format may vary from machine to machine but the internal format will be as follows.

First record	initialization date
	dimensions of run (x,y,z)
	frequency counters
Second record	grid (lat, long, σ_z)
Other records	slabs of data

20.6 Equations

For information on the equations used in **model**, see the *LAMPS 90 Resource Manual No. 1, Dynamic Model Equations.*

Chapter 21

modeldry

FILE	PROCESS
modeldry.pre	main control
initidry.pre	user input
startdry.pre	variable initialization
otputdry.pre	variable output to standard output
histodry.F	variable output to data volume format
phys1dry.pre	physics control
phys2dry.F	physics library
difdrvdry.pre	diffusion and derivative calculations
pbldry.pre	PBL parameterizations
flow3dry.F	array flow handling
flwcpydry.F	array flow copying
flowmdry.F	I/O handling
modcstdry.F	constants

21.1 Description

The **modeldry** program makes a simulation of the atmosphere in much the same way as **model**. Only those things that differ from **model** will be discussed here. See chapter 20 for information concerning the general properties of the model.

The **modeldry** program differs from the **model** program in the following ways.

1. **modeldry** has no moisture variables (and thus no cloud parameterizations)
2. **modeldry** has no friction nor interactions with the surface[1]
3. **modeldry** has no radiation nor surface energy calculations
4. **modeldry** does not differentiate between surface types

This makes the **modeldry** program much simpler than the **model** program. Since the processing and structure are the same, the **modeldry** program can be used as a tool to introduce numerical modeling to novice users. This was, in fact, its original purpose although it can be used to make tests of new calculations.

[1] Boundary layer calculations can be added by defining the macro use_pbl within **case.cfg**.

21.2 Input/Output

The input and output specifications for the **modeldry** program are the same as the **model** program (see chapter 20).

21.3 Equations

For information on the equations used in **modeldry**, see the *LAMPS 90 Resource Manual No. 1, Dynamic Model Equations.*

Part III

Post-processing

Chapter 22

dvutil

22.1 Description

The program **dvutil** allows the user to access the data volume information. The following types of access are currently available.

a. initialize a new data volume (i.e., set up header record)

b. list the contents of a data volume

c. add/extract I/O routines to/from a data volume

d. copy internal files from one data volume to another

e. list the contents of a history data volume header record

Typically, when starting an initialization, we have always started with the **dvutil** program in order to copy the data volume routines to the initialization data volume.

For more information, see the **src/rlib/libdvsubs/info/dvinfo.i** file under the main LAMPS 90 directory.

Chapter 23

pltmout

The **pltmout** program creates horizontal, x-section and difference maps. In addition, it can create vertical profiles of dry temperature and dew point (Skew-T Log-P). The process uses the **pltmout.inp** input file, which has the specifications regarding the history file name, time to be plotted, variables to be plotted, etc. If the variable being plotted is not in the history file, **pltmout** will try to derive it before giving up.

Note: All graphics produce a `.gmeta` file (see *LAMPS 90 User's Guide*).

Note: **pltmout** can read from either slab or volume-format history data volumes.

Note: To plot the model data within an graphics system external to the LAMPS 90/NCAR graphics system, use the **reformat** program to first reformat the data to the specific format required by the external system. Currently, only output to the AVS (Advanced Visual Systems) field file format is available.

Chapter 24

reformat

Each internal record of the history data volume, as produced by the dynamic model, contains one "slab" of data for all the variables. A slab contains all of the information from one y-directory grid point (or longitude). That is, each record contains data from an X-Z slab in space of all the variables. This may seem like a strange way to store information but it is used because that is the way the dynamic model stores the information and it is more efficient to write the history data volume with the same format.

Besides appearing a little strange, there are other disadvantages to this formatting scheme. One problem is that to extract an entire 3-D volume for one variable one needs to read through all of the records (since a portion of each variable is stored on each record). Another problem is that the packing scheme works best when the numbers stored on the record are about the same order of magnitude.

In order to save space and time, the **reformat** program can be used. This program can write out a history data volume that has one variable per record. This format is called the volume format, as opposed to the slab format produced by the dynamic models.

Usually, it is not necessary to rewrite the history data volume. The amount of space and time saved will usually be insignificant unless one is doing a lot of post-processing. However, if you are doing a lot of work with only a few variables, you may want to extract (or derive) only those variables from the history data volume, save them to another history data volume and remove the original history data volume from the disk[1]. This is what the **reformat** program is typically used for. In the process, **reformat** will automatically convert the data to the volume format (if it wasn't already in that format to start with).

For most purposes, the user can remain blissfully unaware of the particular format employed by his/her history data volume. For more information on the data volume, see the *LAMPS 90 User's Guide*.

[1] It might be wise to save the original history data volume on tape before removing it from the disk.

Chapter 25

trajec

The **trajec** program reads in the data from the history data volume and creates trajectory tracks. The process uses the **trajec.inp** input file, which has the specifications regarding the data volume file name(s), beginning and ending times of trajectories, starting positions, time step to be used, etc.

The output can then be plotted by **trajplt**.

Note: For calculations of trajectories on a three-dimensional grid, one needs to send the **trajec** output in the data volume format. If this option is taken, one must use **pltmout** instead of **trajplt** to plot the results.

Chapter 26

trajplt

The **trajplt** program reads in the **trajec** output and creates maps. See table 26.1 for the type of maps which are currently available.

type	*description*
3-D plots	3-D representation of one trajectory track
2-D plots	horizontal representation of multiple trajectory tracks (in absolute space or relative to a trajectory)
traces	plots of atmospheric parameters over the course of specified trajectory tracks
fronts	draw air stream boundaries
reconstruct	reconstruct fields using values obtained from trajectory starting/ending locations
divergence	plots isopleths of the divergence of the trajectories

Table 26.1: Plots available from **trajplt**

Part IV

Libraries

Chapter 27

libdvsubs

- **libdvsubs.a** contains...

  dviosubs.F : deals with reading from / writing to DATA VOLUME — contains "user"-type data volume handling routines

`newdv`	opens a new DATA VOLUME
`opendv`	opens a DATA VOLUME
`posdv`	positions the DATA VOLUME at the requested data or at end of DATA VOLUME
`pos2dv`	positions the DATA VOLUME via a specified position of the internal file within the DATA VOLUME
`closedv`	closes a DATA VOLUME
`readdv`	sequentially reads in one record from DATA VOLUME and unpacks it as reals (use ireaddv for integers, creaddv for character strings)
`writedv`	sequentially writes out one record to DATA VOLUME (use iwritedv for integers, cwritedv for character strings)

  dvlib.F : deals with I/O processing within DATA VOLUME — contains "programmer"-type data volume routines

  dvsubs.cfg : configuration file (parameters)

  dvsubs.h : common blocks

27.1 History

August	1992	...	LAMPS902
May	1992	...	LAMPS901
December	1991	...	LAMPS900
November	1991	...	Temporary scheme to permit portability
?		...	LAMPS version (version 2)
?		...	NCAR version

The following list summarizes the type of changes which have been made from the LAMPS version 2 I/O routines.

1. All values in the data volume are packed which allows portability.

2. Records within internal files are packed.
3. The additive constant and scaling factor are no longer used. Instead, the max and min values are stored and an algorithm is used to pack the values so that the number of bits used for storage is used most efficiently.
4. The number of bits per value are determined automatically given the desired precision.

27.2 Notes

Note: See the **dvutil** program for a utility that allows the user to create and rearrange data volumes.

Note: In some cases, only part a record will be used. In these cases, the computer may fill the unused portion with garbage. Don't let this bother you. It's normal.

Note: To pack the data, the routines gbytes and sbytes are used. These are routines within the NCAR graphics package. One problem I have encountered is with the passing of arguments to these routines. Passing an argument like [lcat+1] does not seem to work. Instead, one must use two calls; one with [lcat] and one with [1].

Note: Eventually, I would like to change getdateid so that the variable returned is iyymmdd, not immddyy.

Note: Eventually, parameters like variable type, format type and grid type should be included in the headers. However, at this time, it is not clear how the data volume should be structured. When this is finally determined, these routines should be altered and conversion routines should be made to convert the current structure to the new structure (use dvutil). The tables describing the various variable, format and grid types should be included in the data volume somewhere as well.

27.3 Introduction

DATA VOLUME is the word used to describe the actual physical file containing all the data these routines will read and/or write.

From the USER's point of view, the data volume is like a directory which contains many files. The files, however, are all stored within the data volume file. Thus, these "files" are called INTERNAL FILES. By specifying a particular FILE ID, the user positions the data volume at the particular point within the data volume file where the desired internal file resides.

Just like files, the internal files appear (to the user) to have records. These records, however, are all packed one after another within the internal files. Thus, these "records" are called INTERNAL RECORDS.

From the PROGRAMMER's point of view, the data volume consists of internal files which start on a byte which is an integer multiple of some number. This number is `dvrecl`. In this way, the location of each internal file can be specified as some integer multiple of `dvrecl`.

`dvrecl` is usually defined in **dvsubs.cfg**, but can be any number. That is, if it is different than what is defined in **dvsubs.cfg**, the computer will just adjust its calculations accordingly. Since `dvrecl` only tells the computer where each internal file begins, it really doesn't matter what it is. Still, a large `dvrecl` may waste space and a small `dvrecl` may require unneeded I/O operations.

The length of each internal file is in no way limited by `dvrecl` because each internal file consists of a continuous string of bits in which the internal records are packed. Each internal record contains information

on its length, allowing the programmer to move from one internal record to the next.

Each internal file is identified by various parameters, one of which is the BLOCK NUMBER. The block number identifies the program that produced the internal file. The location of the latest internal file written by each program is referred to as the CATALOG NUMBER. Thus, there will be one catalog number per block number. These numbers are stored in the array `lcatalog`. `lcatalog` is stored in the first record of the data volume file (the DATA VOLUME HEADER RECORD). In this way, the position of the latest internal file produced by each program is known. To find the other internal files, the parameter `prev` is stored with each internal file. `prev` specifies the location of the previous internal file written by the corresponding program.

From the COMPUTER's point of view, the data volume consists of just a string of bits. By placing all the responsibility of interpreting the file structure information within these routines (i.e., the programmer), the use of the data volume becomes completely machine independent.

27.4 Design Structure

The DATA VOLUME is a binary, direct-access file. It contains a DATA VOLUME HEADER RECORD, followed by I/O routines (optional), followed by a series of internal files.

27.4.1 Data Volume Header Record (record 1)

Variable name	Description	bytes/bits
`dvid`	data volume id (ASCII)	8/64
`dvstime`	data volume creation time, YYMMDDHHMMSS (ASCII)	6/48
`dvrecl`	record size (int)	4/32
`niosub`	number of bytes containing I/O routines (int)	4/32
`iorec`	record number for beginning of I/O routines (int)	4/32
`nrdv`	number of `dvrecl`-size records in data volume (int)	4/32
`catalog(lcat)`	latest internal file header record position for each block number (int)	4/32
`ifn(lcat)`	number of internal files for each block number (int)	2/16

Note: Since `nrdv` is stored in 32 bits, the maximum value of `nrdv` is about 4×10^9. Therefore, the maximum size of the data volume (in bytes) will be $4 \times 10^9 \times$ `dvrecl` which is about 4 terabytes.

Note: A previous version of the data volume (i.e., LAMPS901) had the variables `iorec`, `nrdv` and `catalog` stored in 16 bits each (`ifn` was stored with 8 bits each), which limited the size of the data volume to about 65 Megabytes.

`catalog` is an array of size `lcat`. Each value of the array corresponds to a block number. At present, there are eleven different block numbers corresponding to eleven different procedures, as shown in the table below.

block number	program
1	begincase
2	firstguess
3	sounding
4	layermean
5	rhanalysis
6	isenanal
7	vertran
8	windadjust
9	3-D analysis
10	merge
11	model history (slabs)
12	model output (variables)

Since each program may write out more than one internal file, there may be many internal files which have the same block number. Internal files having the same block number can be differentiated by the contents of the INTERNAL FILE HEADER. Currently, `itime` (i.e., the initialization time) and `ifid` (i.e., the file id) are used.

The `catalog` array contains the header record numbers for the most recent internal file written by each program. Since each internal file header contains the position of the previous internal file written by each program, the user can search "backward" through the data volume until an adequate match is found.

The creation time for each internal file is saved to differentiate between different versions of the same file.

27.4.2 I/O Routines (records `iorec` to `iorec+niosub/dvrecl`)

The I/O routines are copies of the FORTRAN routines which were used to create the data volume. They are store in ASCII, 8 bits per character, stored sequentially with linefeeds (HEX: 0A) as line delimiters.

dvutil is used to create and extract the I/O routines.

27.4.3 Internal Files

Each internal file consists of internal records. The first internal record is the internal file header record. The remaining internal records are internal file data records.

he contents of the header record (internal record 1) are as follows.

Variable name	Description	bits
`it`	time of initialization data in YYMMDDHHMM (size 5 array)	32
`ifstime`	creation time in YYMMDDHHMMSS (size 6 array)	48
`iblkn`	block number	8
`ipver`	program ver number	8
`ifid`	internal file ID number	16
`prev`	header record # for previous file of same block type	32
`nr`	number of internal records in internal file not including header	32
`nrf`	# of actual records in internal file incl. header (`dvrecl` bytes/record)	32

Note: The oldest internal file (i.e., first) of each block will have `prev` = 0.

Note: A previous version of the data volume (i.e., LAMPS901) had the variables `prev`, `nr` and `nrf` stored

in 16 bits each, which limited the size of the data volume to about 65 Megabytes.

Each data record (internal records 2 to nr+1) has the following contents.

Variable name	Description	bits
inum	number of values stored in record	26
lbyte	number of bits per value	6
isignp	sign of precision	1
prec	precision	7
isignm	sign of minimum	1
imin	minimum value (multiplied by $10^{\texttt{prec}}$)	31
kbuf	values	inum × lbyte
inum	number of values stored in record	26
lbyte	number of bits per value	6

imin is the minimum value stored (multipled by $10^{\texttt{prec}}$). Along with the corresponding maximum value stored (multiplied by $10^{\texttt{prec}}$), the number of bits required to store to the desired precision was calculated via the following algorithm:

$$\texttt{lbyte} = \text{INT}\{\log_2[\text{REAL}(\texttt{imax} - \texttt{imin+1})]\}$$

where INT in this case is truncation toward positive infinity. Each value is then packed via the following algorithm:

$$\text{NINT}(\texttt{buf(i)}10^{\texttt{prec}}) - \texttt{imin}$$

Note: A previous version of the data volume (i.e., LAMPS900) had a different packing scheme which required the storage of the max and min values within the data record. Thus, in place of isignp, prec, isignm and imin (40 bits total), rmax and rmin (32 bits each) would be stored. rmax and rmin were stored and read via the programs **packIEEE** and **readIEEE**. These programs would use the IEEE standard (as on the SUN) as a basis for storage.

bit 1 = sign bit (s)
bits 2-9 = exponent bits (e)
bits 10-32 = fraction bits (f)
bias = 127 (b)

The real numbers are reconstructed via

$$\text{r} = (-1)^{\text{s}}2^{\text{e-b}}(1.\text{f})$$

These numbers were used to pack each value via the following algorithm:

$$(2^{\texttt{lbyte}} - 1)(\texttt{rmax} - \texttt{rmin}) + \text{rmin}$$

where lbyte was number of bits per value as determined via the following algorithm:

$$\text{INT}\left[\log_2\left(\frac{2^{\text{rmax-rmin}}}{\text{prec}} + 1\right)\right]$$

where INT in this case is truncation toward positive infinity.

LAMPS900 also had inum/lbyte stored as 16 bits each. Since this limited the size of inum to 65535 and since lbyte is never expected go above 64, let alone 32, inum/lbyte were changed to the LAMSP901 values of 26 and 6 bits.

Chapter 28

libgraph

- **libgraph.a** contains...

graflib.F : routines for use with NCAR GRAPHICS library

`mbkrnd`	creates a stereographic map background
`linear`	estimates a value via linear interpolation
`label`	labels the axes of a two-d horizontal map
`plt`	plots the contours of any two-d field
`windflag`	plots a single wind flag
`conmap`	plots the contours of any two-d field (like `range`/ `plt`)
`wndmap`	plots the vector winds (like `pltwnd`)
`eframe`	ends a frame and plots the date

skewtlib.f : routines for plotting skew-T log-P charts

plotskewt	plots the sounding data on a skew-T diagram
sbkrnd	creates the skew-T background

plotxlib.f : routines for plotting X-section maps

plotxmap	plots the contours of any 2-D field
xbkrnd	assembles the plotting instructions needed to plot the background of a latitude or longitude cross section
height	calculate height given terrain, Htop and sigma height

plotline.f : routines for plotting simple curves

pltxy	plots Y as a function of X
getfmt	assigns the proper format to handle axis labels
maxmin	calculates the appropriate max, min and increment for plotting purposes

Chapter 29

liblamps

29.1 List of routines

- **liblamps.a** contains...

gpur2.F : general purpose math routines and constants

common blocks	`const`	sets constant common blocks
	`siunit`	sets SI units common blocks
array sorts	`copy`	copy array x to array y
	`permute`	permute the elements of an array in the order specified by an index permutation array
	`range`	compute simple statistics for a 1-, 2-, or 3-d field
	`setip`	initialize an index permutation array
	`sift`	examine all elements in an array and retain only those which pass a range test
	`sort`	order the elements of an array by element value
	`initary`	assign a value to an entire array
physics	`static`	compute the hydrostatic pressure at levels above and below a reference level
	`wind`	transform u,v to ff,dd or vice versa
other	`lwr2upr`	convert lower-case letters in a string to upper-case
	`printmxmn`	write out the max and min of an array
exit control	`abend`	abort/end execution
	`egress`	terminate execution normally
	`exitx`	terminate execution with exit code
functions	`lspv`	.true. if value is the special value spval
	`igetivar`	position in array of given string
	`arctan`	inverse tangent given sine and cosine
	`lnblnk`	last non-blank character in a string

inpctrl.F : general purpose input control routines

`readcd`	reads in the next valid `lncard` from the input logical unit and returns the number of arguments
`chararg`	returns the character value of the i^{th} argument in `lncard`
`chararg8`	returns the 8-byte character value of the i^{th} argument in `lncard`
`intarg`	returns the integer value of the i^{th} argument in `lncard`
`realarg`	returns the real value of the i^{th} argument in `lncard`

outpctrl.F : general purpose output control routines

`eject`	causes the line printer to eject the current page
`print`	prints a variable's name and value
`slash`	causes the line printer to skip n lines
`string`	prints a character string of up to 128 characters with the option of being left justified or centered in the line of 128 characters
`warn`	prints a warning message but does not terminate execution
`table`	prints a table of real numbers complete with column labels
`tablep`	prints a table of real numbers in the order specified by an index permutation array (column labels are optional)

mathlib.F : math routines

bilocation	performs a binary search of a given value in an array
matrix solving	solves the matrix equation $ax = b$
values and derivatives	interpolates values and/or derivatives of a real function of one real variable, given only function values at a sequence of abscissae
curve and surface fitting	interpolates a curve or surface at a given point or coordinate pair using a spline under tension

vaplib.F : deals with saturation vapor pressure

`esx`	calculates the saturation vapor pressure (mb) over water for $T \geq T_f$ and over ice for $T < T_f$
`e_to_td`	determines the dew-point temperature (in K) associated with the input vapor pressure
`qt qt2`	iteratively computes the specific humidity and temperature corresponding to given values of relative humidity, virtual temperature, and pressure

getdateid.F : deals with date and time computation

getconst.F : sets the meteorological constants

Note: **math03.pre** (math routines) and **oldgpur.F** (general purpose routines) are also present in the **rlib** sub-directory. They used to be used, but are no longer.

29.2 Saturation Vapor Pressure

BLOCK
ROUTINE `esx`
FILE **vaplib.F**

The basic source is for the calculation of saturation vapor pressure as a function of temperature is given by Tetens (1930). However, different versions are known to be used by different people. Three are presented here.

A quick derivation is as follows.

From Clausius-Clapeyron, we have...

$$de = \frac{L}{Td\alpha}$$

where $d\alpha$ is the change in specific volume from water to vapor. Since α of vapor $\gg$ α of water, we can neglect α of water. In addition, for air,

$$\alpha = \frac{1}{\rho} = \frac{RT}{e}$$

where R is the universal gas constant divided by the molecular weight of water vapor (i.e., 8.314/0.18). Substituting this in, we get...

$$\frac{de}{e} = \frac{L/R}{T^2}$$

Since the equation...

$$\frac{d \ln e}{dT} = \frac{bc}{(c+T)^2}$$

has...

$$e = e_o \exp\left(\frac{bT}{c+T}\right)$$

as a solution, this implies that...

$c = 273.15$ and $b = (L/R)/c = 19.82$ [for water at 0°C]
$c = 273.15$ and $b = (L/R)/c = 22.46$ [for ice at 0°C]

the difference being in the L.

Note: If we use List's (1951) $L = L_o + aT$, where...

$$L_o = 2.501E6$$

and

$$a = -0.00237E6$$

then we need to include this before integrating. This will give...

$$e = e_o \exp\left(\frac{bT}{c+T}\right)\left(\frac{T+c}{c}\right)^d$$

where...

$$c = 273.15, \qquad b = \frac{L/R}{c} \qquad \text{and} \qquad d = \frac{a}{R}$$

However, to be even more accurate, one needs to include the density of water. And, even more, one can take into account the change of density with temperature. Either one further complicates the math.

In Tetens' formula, the simple exponential is used with the constants set to fit the saturation vapor pressure curve.

$$e = e_o \exp\left(\frac{bT}{c+T}\right)$$

LAMPS 90 uses

e_o	b	c	**state**
6.1078	17.2693882	237.29	for water
6.1078	21.8745584	265.49	for ice

Other values can be obtained by various sources.

- from Tetens (1930) as transformed in Murray (1967) and obtained from Donaldson (1990):

e_o	b	c	**state**
6.1078	17.2694	237.3	for water
6.1078	21.8746	265.5	for ice

- from the original Tetens formula (as included in Murray (1967) and obtained Donaldson:

e_o	b	c	**state**
6.107	$7.5 \ln 10$	237.3	for water
6.107	$9.5 \ln 10$	265.5	for ice

  Note: b is multiplied by $\ln 10$ because the original equation was in the form of powers of ten rather than powers of e.

- Merceret and Davis (1981)

e_o	b	c	**state**
6.1078	17.6498	243.17	for water
6.1078	22.4716	272.722	for ice

- from Bolton (1980), fitted to Wexler's results (copied from Donaldson):

e_o	b	c	**state**
6.112	17.67	243.5	for water

Other formulations are possible besides the Tetens exponential formula.

The basic formulation from Goff and Gratch (1946), as described in Miller and Friesen (1985), is:

$$\log e_s = \begin{cases} a + b\log T + c10^{d+eT} + f\left(10^{g+h/T} + \frac{i}{T}\right) & \text{for water} \\ a\log T - bT - \frac{c}{T} + d & \text{for ice} \end{cases}$$

where T is the temperature in Kelvin and the constants are as follows.

constant	value for water	value for ice
a	2.3832241×10^{1}	3.56654×10^{0}
b	-5.02808×10^{0}	3.2098×10^{-3}
c	-1.3816×10^{-7}	2.484956×10^{3}
d	1.1334×10^{1}	2.0702294×10^{0}
e	-3.03998×10^{-2}	
f	8.1328×10^{-3}	
g	3.49149×10^{0}	
h	-1.3028844×10^{3}	
i	-2.949076×10^{3}	

However, the basic equation can be rearranged depending on how one wants to scale T. For example,

- from Donaldson (1990), using a formulation as quoted in Murray (1967) and originally extracted from Strong (1982)[1].

$$\log e_s = \begin{cases} \log a - b\left(1 - \frac{1}{T}\right) + c\log\frac{1}{T} + d\left(10^{e(1-T)} - 1\right) + f\left(10^{g[1-(1/T)]} - 1\right) & \text{for water} \\ \log a - b\left(1 - \frac{1}{T}\right) + c\log\frac{1}{T} + d(1-T) & \text{for ice} \end{cases}$$

where T is the temperature in Kelvin scaled by the boiling point of water (for water) and by the freezing point of water (for ice). The constants are as follows.

constant	value for water	value for ice
a	1.013246×10^3	6.1071×10^0
b	-7.90298×10^0	-9.0918×10^0
c	5.02808×10^0	-3.56654×10^0
d	-1.3816×10^{-7}	8.76793×10^{-1}
e	1.1344×10^1	
f	8.1328×10^{-3}	
g	3.49149×10^0	

- from Donaldson (1990) (used in some ARC/AES dew point routine):

$$\log e_s = \log a - b\left(1 - \frac{1}{T}\right) + c\log\frac{1}{T} + d\left(10^{e(1-T)} - 1\right) + f\left(10^{g[1-(1/T)]} - 1\right) \text{(for water)}$$

where T is the temperature in Kelvin scaled by the freezing point of water. The constants are as follows.

constant	value for water
a	6.11139×10^0
b	1.079574×10^1
c	-5.02800×10^0
d	-1.50475×10^{-4}
e	8.2969×10^0
f	4.2873×10^{-4}
g	4.76955×10^0

- from some ARC/AES dew pt routine:

$$\log e_s = a + \frac{b}{T} + c\log T \text{(for water)}$$

where T is the temperature in Kelvin and the constants are as follows.

constant	value for water
a	2.3547×10^1
b	-2.9374×10^3
c	-4.928×10^0

- From Buck (1981), the following enhancement factor to the above calculations is given as:

$$e_s = e_s(a + bP)$$

where P is pressure (in mb) and the constants are as follows.

constant	value for water	value for ice
a	1.0007×10^0	1.0003×10^0
b	3.46×10^{-6}	4.18×10^{-6}

[1] Donaldson also references this as copied from a program of Bill Hall's at NCAR by someone at AES.

Note: The same expression was found in a program of Bill Hall's at NCAR, obtained via AES, but with $a = 1.0$ and $b = 5 \times 10^{-6}$ for water.

Approximations by polynomials are also available. They use a polynomial of the form...

$$e_s = a + T(b + T(c + T(d + T(e + T(f + Tg)))))$$

where T is in degrees Celsius.

From Rasmussen (1978), the constants are as follows.

constant	value for water
a	6.1070422×10^{0}
b	4.4411566×10^{-1}
c	1.4320982×10^{-2}
d	2.6513961×10^{-4}
e	3.0099985×10^{-6}
f	2.0088796×10^{-8}
g	$6.1926232 \times 10^{-11}$

Valid range is -60 to $+50°$C.

From Lowe and Ficke (1974) via page 625 of Pruppacher and Klett (1980) as gotten from Donaldson (1990). the constants are as follows.

constant	value for water	value for ice
a	$6.107799961 \times 10^{0}$	$6.109177956 \times 10^{0}$
b	$4.436518521 \times 10^{-1}$	$5.034698970 \times 10^{-1}$
c	$1.428945805 \times 10^{-2}$	$1.886013408 \times 10^{-2}$
d	$2.650648471 \times 10^{-4}$	$4.176223716 \times 10^{-4}$
e	$3.031240396 \times 10^{-6}$	$5.824720280 \times 10^{-6}$
f	$2.034080948 \times 10^{-8}$	$4.838803174 \times 10^{-8}$
g	$6.136820929 \times 10^{-11}$	$1.838826904 \times 10^{-10}$

Valid range is $T > -50°$C.

29.3 References

Bolton, 1980: *Mon. Wea. Rev.*

Buck, A. L., 1981: New equations for computing vapor pressure and enhancement factor, *J. Appl. Meteo.*, **20**, 1527-1532.

Donaldson, Norman, 1990: Personal communication. (AES)

Goff, J. A., and S. Gratch, 1946: Low-pressure properties of water from -160 to 212F. *Tran. A. Soc. Heat. Vent. Eng.*, **52**, 95-121.

List, 1950:

Lowe, P. R., and J. M. Ficke, 1974: *Techn. Paper*, No. 4-74, Environmental Prediction Res. Facility, Naval Post Grad. School, Monterey, Calif.

Miller, E. R., and R. B. Friesen, 1985: Standard Output Data Products from the NCAR Research Aviation Facility, *NCAR/RAF Bulletin No. 9*, October 31.

Merceret, Francis J., and Harlan W. Davis, 1981: The Determination of Navigaional and Meteorological Variables Measured by NOAA/RRFC WP3D Aircraft. *NOAA Technical Memorandum ERL RFC-7*, April.

Murray, 1967: *J. Applied Meteo.*

Pruppacher, H. R., and J. D. Klett, 1980: *Microphysics of Clouds and Precipitation*, D. Reidel, Holland, 714pp.

Rasmussen, L. A., 1978: *J. Appl. Meteor.*, **17**, 1564-1565.

Strong, G., 1982: Research Aircraft (RSL) software, ASD/ARC, June.

Tetens, 1930: *J.Geophys*, **6**.

Chapter 30

libmdlio

- **libmdlio.a** contains...

  modelio.F : deals with reading from / writing to MODEL HISTORY and MODEL OUTPUT DATA VOLUMES

`readvar`	calls either read12var or read11var
`read12var`	reads variable from a volume-format MODEL HISTORY DATA VOLUME
`read11var`	reads variable from a slab-format MODEL HISTORY DATA VOLUME
`readuvw`	reads winds from MODEL OUTPUT DATA VOLUME
`read123`	reads in any three variables from a volume-format MODEL HISTORY DATA VOLUME
`rdmhishdr`	reads MODEL HISTORY DATA VOLUME header record
`wtmhishdr`	writes MODEL HISTORY DATA VOLUME header record

Chapter 31

postlib

- **postlib.a** contains...

  postlib.f : POST-PROCESSING routines

`update`	add idtime minutes to the time array
`intertime`	calculates an array by interpolating between two arrays with respect to time
`interp3d`	extract a value via 3-D linear interpolation
`interp2d`	extract a value via 2-D linear interpolation
`julian`	calculates the number of minutes between two times

Chapter 32

Object files used as libraries

32.1 gsbytes

The **gsbytes** library, `gsbytes.o`, contains the C routines of the file `gsbytes.c`. These routines control the bit handling required for the data volume programs.

32.2 maptran

The **maptran** library, `maptran.o`, contains the routines of the file `maptran.f`. These routines control the map transformation required for the NCAR graphics programs.

32.3 plotmap

The **plotmap** library, `plotmap.o`, contains the routines of the file `plotmap.f`. These routines make the stereographic contour maps. It is case-dependent because it uses an array whose size is set by the size of the grid.

32.4 uexit

The **uexit** library, `uexit.o`, contains the C routine `uexit` of the file `uexit.c`. This routine is the same as `exit` and is used on those machines that don't have `exit` or whose `exit` function differs from that expected (e.g., hpux machines).