Map Projection, Model Grid Setup and Grid Nesting

7.1. Introduction

This chapter describes the map projection options, the setup of the computational grid (including the options for vertical grid stretching), the adaptive grid refinement and grid nesting capability, and the model domain translation capabilities.

7.2. Map Projection

ARPS includes a set of map projection subroutines which transform the Cartesian computational coordinates to latitude and longitude locations on the Earth and vice versa. Three different map projections are currently supported. *The map projection factors are not included in the dynamic equations of ARPS version 4.0, but will be in a future release.* They are, however, already included in the pre- and post-processors. If one chooses the Lambert conformal projection and uses a relatively small model domain (less than 1000 km), then the effect of the map factor is negligible.

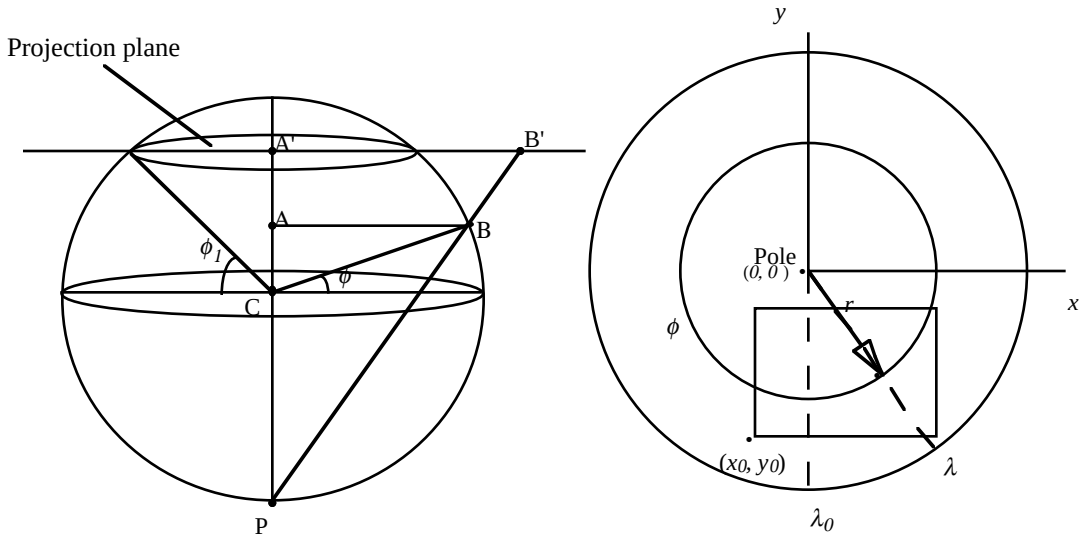
7.2.1. Map projection options

The three map projection options available in ARPS include 1) polar stereographic, which is most useful at high latitudes and for interfacing with many large-scale NWP models, 2) Lambert conformal, most useful at mid-latitudes, and 3) Mercator, which is best for low latitudes, generally within 20 degrees of the equator. Because some archived data are provided on latitude-longitude grids, a latitude-longitude coordinate option is also provided as part

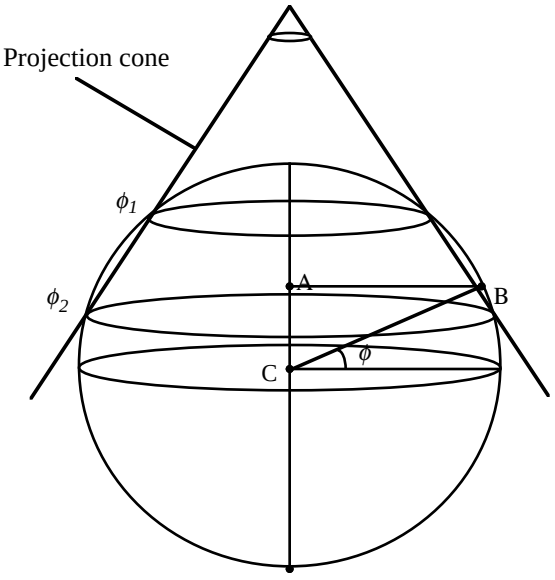
of the map projection utilities, but, because it is non-conformal, it is not recommended for use as a model coordinate.

Figure 7.1 shows schematics of the map projections used in ARPS, including some of the key angles that are used in specifying the projection and

a) Polar Stereographic Projection



b) Lambert Conformal Projection



c) Mercator Projection

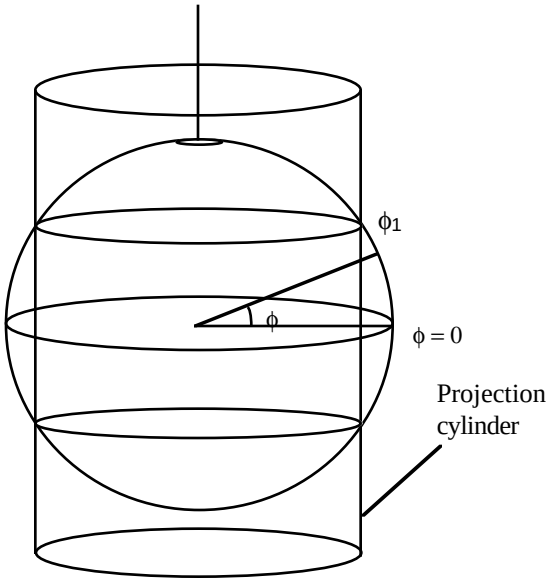


Figure 7.1. Schematics of the a) Polar stereographic projection, b) Lambert conformal projection and c) Mercator projection.

deriving the transformation equations. Figure 7.2 shows examples of the US state map plotted with each of the three map projections. Mathematical details of the projection transformations are presented in Section 7.2.2. Saucier (1989) discusses the appropriate selection of map projections.

In ARPS, the map projection is specified using four parameters in the input file, *arps40.input*. These parameters include: *maproj,* the map projection type, *trulat1* and *trulat2,* the standard latitudes, *trulon,* the standard longitude, and *sclfct,* the scaling factor of the map projection. The choices for their values are described in Chapter 4.

The Lambert conformal projection requires that two standard latitudes be specified. These two parameters are order independent and must be consistent with the hemisphere selected (location of cone apex). In general a safe procedure is to specify the two latitudes in the same hemisphere as the cone apex. An error message will be printed and the program will stop if improper parameter values are specified.

Those writing software to use the map projection software should note that the map projection is set up by calling subroutine SETMAPR. The values

of the map projection parameters being used can be retrieved by calling subroutine GETMAPR. Only one map projection can be active at a time.

Figure 7.2. Examples of US political boundaries plotted with a) Polar stereographic, b) Lambert conformal and c) Mercator projections.

7.2.2. Map transformations

Transformation equations are used to map the earth coordinates to the projection plane. This section describes the transformation equations and the magnification of distances from the earth's surface to the map projection surface for each of the map projections. All three primary map projections are conformal, that is the projection magnification, known as the image scale, is the same in all directions.

a) Polar stereographic projection

The polar stereographic projection is created by projection rays beginning at the opposite pole and running through each point on the earth's surface to the projection plane, which is a plane parallel to the equator at the latitude specified by the input parameter *trulat1*, ϕ_1, as illustrated in Fig. 7.1a.

Considering the earth angles as projected on the plane (see Fig. 7.1b), the transformation equation from earth to map coordinates is

$$\begin{aligned} x &= r\sin(\lambda - \lambda_0) \\ y &= -r\cos(\lambda - \lambda_0) \\ r &= \sigma r_e \cos\phi \end{aligned} \tag{7.2.1}$$

where λ is the longitude, ϕ is the latitude, and r_e is the radius of the earth. The variable λ_0 is the input parameter *trulon*. That meridian defines the y coordinate on the plane. Unless one is trying to match the orientation of an external data grid, a λ near the center of the grid should be specified as λ_o.

The polar stereographic image scale, σ in Eq. (7.2.1), can be derived from the geometry of the triangles in Fig 7.1a and is written

$$\sigma = \frac{1+\sin\phi_1}{1+\sin\phi}. \tag{7.2.2}$$

Clearly ϕ_1, the input parameter *trulat1*, should be near the grid center to keep σ near unity. The input parameter named *trulat2* is not used in this projection.

b) Lambert conformal projection

In the Lambert conformal projection, projection rays begin at the center of the earth and pass through points on the spherical surface to a conical surface which intersects the spherical surface at two latitude circles specified by *trulat1 and trulat2*, written here as ϕ_1 and ϕ_2, respectively (see Fig. 7.1b). When ϕ_1 and ϕ_2, are equal, the cone is tangent to the sphere, and the projection is known as a Lambert tangent projection, otherwise it is known as a secant projection.

From the equations presented by Saucier (1989), the transformation equations for the Lambert secant projection can be shown to be:

$$x = r\sin\left[n\left(\lambda - \lambda_0\right)\right]$$

$$y = -r\cos\left[n\left(\lambda - \lambda_0\right)\right]$$

$$r = \frac{r_e \cos\phi_1}{n}\left[\frac{\tan\left(\frac{(\pi/2)-\phi}{2}\right)}{\tan\left(\frac{(\pi/2)-\phi_1}{2}\right)}\right]^n \tag{7.2.3}$$

where n is the cone constant. It is related to the shape of the projection cone,

$$n = \frac{\left(\ln\cos\phi_1 - \ln\cos\phi_2\right)}{\left(\ln\tan\left(\frac{\pi/2-\phi_1}{2}\right) - \ln\tan\left(\frac{\pi/2-\phi_2}{2}\right)\right)}. \tag{7.2.4}$$

The image scale is described by

$$\sigma = \frac{\cos\phi_1}{\cos\phi}\left[\frac{\tan\left(\frac{\pi/2-\phi}{2}\right)}{\tan\left(\frac{\pi/2-\phi_1}{2}\right)}\right]^n, \tag{7.2.5}$$

and σ is unity at ϕ_1 and ϕ_2.

Although the tangent projection is a special case of the secant projection, the quotient in the equation for the cone constant is not defined when $\phi_1 = \phi_2$. The cone constant equation instead simplifies to

$$n = \sin\phi_1. \tag{7.2.6}$$

ARPS software will recognize the specification of a tangent projection and use the proper set of equations. There are certain combinations of *trulat1* and *trulat2* that will create a badly-formed or undefined projection cone. ARPS will print notification messages if such cases are detected during the grid-setup.

Specifying *trulat1* and *trulat2* within the model domain will help keep the image scale factor near unity across the domain, resulting in low distortion.

c) Mercator projection

The Mercator projection maps each point on the spherical surface to a cylinder which intersects the surface at the latitude circle specified by the input parameter *trulat1*, ϕ_1. From Figure 7.1c) we see that the radius of any latitude circle projected on the map cylinder is

$$r = r_e \cos\phi_1 \tag{7.2.7}$$

and equations for x and y on the cylinder that create a conformal map are written

$$x = -r(\lambda - \lambda_0)$$

$$y = r\ln\left(\tan\left(\tfrac{\pi}{4} + \tfrac{\phi}{2}\right)\right) \tag{7.2.8}$$

The image scale for the Mercator projection follows from Eq. 7.2.7:

$$\sigma = \frac{r}{r_e \cos\phi} = \frac{\cos\phi_1}{\cos\phi}. \tag{7.2.9}$$

It is evident that the image scale will become quite large and increase rapidly with latitude outside of the tropics, and hence the Mercator projection is only recommended for the tropics.

7.3. Vertical Coordinate Transformation

7.3.1. The computational coordinate transformation

A general curvilinear coordinate system is used by ARPS (see Appendix B). The irregular grid associated with the curvilinear coordinate in the physical space is mapped to a regular, rectangular computational grid through a coordinate transformation. All calculations are then carried out in the computational space, using standard algorithms that are developed for Cartesian systems. The regular shape of computational boundaries also simplifies implementation of boundary conditions.

In ARPS, the transformation Jacobians are calculated numerically on the computational grid, so that the vertical transformed coordinate can be generally defined.

There are two steps involved in the grid transformation. The first is the terrain-following coordinate transformation and the second is the vertical grid stretching. The terrain-following coordinate transformation is given by

$$\mu = (z_{flat} - z_0)\frac{z - h}{z_{flat} - h} + z_0 \qquad \text{for } z_0 \le z \le z_{flat}$$

$$\mu = z \quad \text{for } z_{flat} < z \le D + z_0 \tag{7.3.1}$$

where z is the height in the physical space and μ the height of the terrain-following coordinate. D is the depth of model physical domain, h the surface terrain height, z_0 the reference height for the model lower boundary and z_{flat} a height at which the terrain-following coordinate changes back to the physical coordinate. It is clear that $\mu = z_0$ at the lower boundary, $\mu = D$ at the top and $z_0 \le \mu \le D + z_0$.

In APRS, vertical grid stretching is defined by the mapping function

$$\mu = \mu(\zeta). \tag{7.3.2}$$

where ζ is the vertical computational coordinate. The grid stretching is discussed in the following section.

7.3.2. Vertical grid stretching

Two grid stretching options are presently available in ARPS. One uses a cubic function and the other uses a hyperbolic tangent function. Both have the prevision for defining layers of uniform grid near the ground and near the top boundary. The details are given in next subsection.

As illustrated in Fig. 7.3, the vertical extent of the model is divided into three layers. The lowest layer (layer 1) has a uniform resolution of $\Delta\mu_l = \Delta\mu_{min}$. The middle layer (layer 2) has a grid spacing that increases from $\Delta\mu_m$ at its lowest level to $\Delta\mu_u = 2\Delta\mu_m - \Delta\mu_{min}$ at its top, where $\Delta\mu_m$ is the average grid spacing of this layer. Above, in layer 3, the spacing is constant and is approximately equal to $\Delta\mu_u$ (an adjustment to $\Delta\mu_u$ may be required to insure that the specified depth 3 contains an integral number of levels).

According to Figure 7.3,

$$\begin{aligned} &n_1 = D_1/\Delta\mu_{min},\ n_2 = D_2/\Delta\mu_m \text{ and } n_3 = D_3/\Delta\mu_u, \\ &n_1 + n_2 + n_3 = nz - 3, \\ &D_1 + D_2 + D_3 = D. \end{aligned} \tag{7.3.3}$$

In Eq. (7.3.3), all parameters are known except for $\Delta\mu_m$. It is required that $\Delta\mu_m$ satisfy the following relation:

$$\frac{D_1}{\Delta\mu_{\min}} + \frac{D_2}{\Delta\mu_m} + \frac{D_3}{2\Delta\mu_m - \Delta\mu_{\min}} = \frac{D}{\Delta\zeta} = nz - 3. \tag{7.3.4}$$

where $nz - 3$ is the total number of grid levels between the bottom and top boundaries. $\Delta\zeta$ is the grid spacing in the computation space, which is the average grid spacing in the entire domain.

Equation (7.3.4) is solved for $\Delta\mu_m$. The grid spacing in layers 1 and 3 is then known. The spacing in layer 2 will be obtained from certain stretching functions under the constraint that the average grid spacing is $\Delta\mu_m$ and the spacing at the bottom and top of this layer are respectively $\Delta\mu_{min}$ and $2\Delta\mu_m - \Delta\mu_{min}$. These constraints can be satisfied by any odd function.

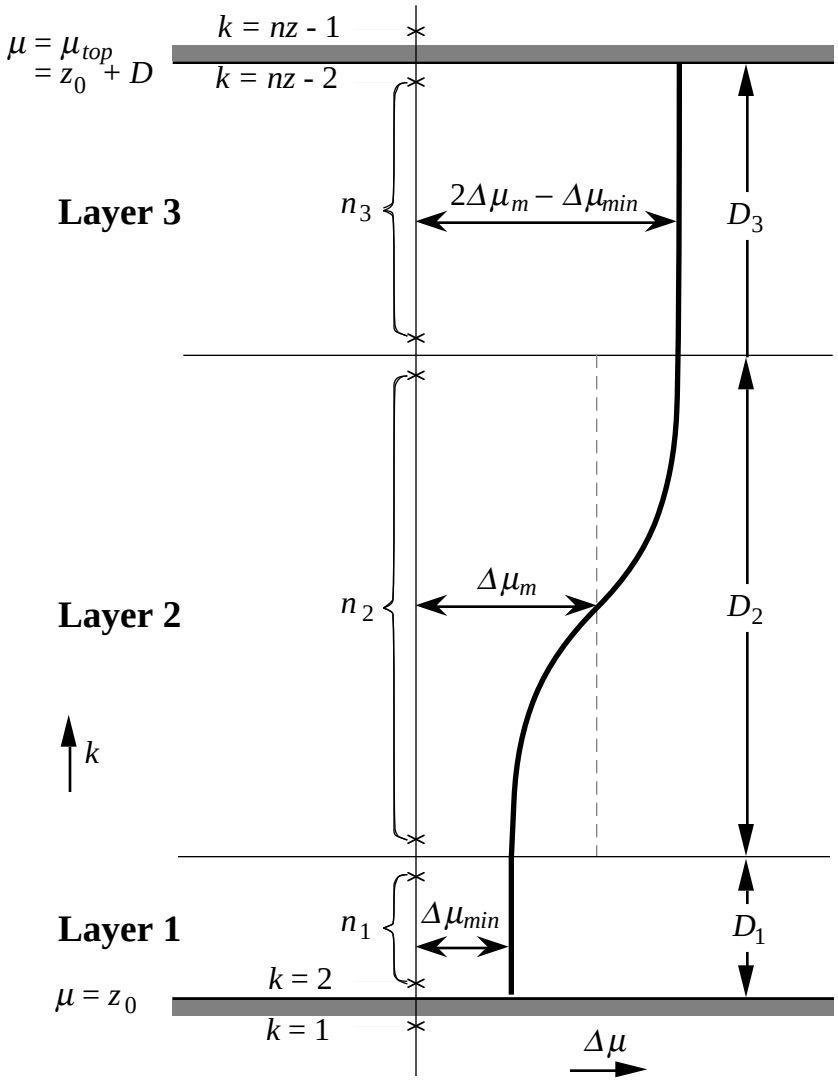


Figure 7.3. An illustration of a vertically stretched grid. D_1, D_2 and D_3 are the depths of layer 1, 2 and 3 respectively. D_1, and D_2 are specified by the user as control parameters. $D_3 = D - D_1 - D_2$, where D is the depth of the entire domain. n_1, n_2 and n_3 are, respectively, the number of (scalar) grid levels in each of the three layers.

For stretching option 1, the vertical grid spacing is defined by a cubic function:

$$\Delta\mu_i = \Delta\mu_m + (\Delta\mu_m - \Delta\mu_{min}) \left[\frac{2(i-1)}{n_2-1} - 1 \right]^3 \quad \text{for} \quad i = 1, n_2, \qquad (7.3.5)$$

for the mid-layer (layer 2), where i is the index for the vertical grid levels in this layer and is directly related to the computational coordinate ζ. n_2 is the number of levels.

For stretching option 2, the vertical grid spacing is defined by a hyperbolic tangent function:

$$\Delta \mu_i = \Delta\mu_m + \frac{\Delta\mu_{\min} - \Delta\mu_m}{\tanh(2\alpha)} \tanh\left[\frac{2\alpha}{1-a}(\mathrm{i}-\mathrm{a})\right] \quad \text{for} \quad i=1, n_2, \qquad (7.3.6)$$

where $a = (1+n_2)/2$ and α is a tuning parameter which can take a value between 0.2 and 5.0. The tuning parameter is set in the source code; ARPS 4.0 is delivered with α = 1.0. For larger values of α, $\Delta\mu_i$ tends to vary more linearly with increasing i while for a smaller value of α, $\Delta\mu_i$ changes more abruptly at around $i = n_2/2$ (see examples in Fig. 7.5). When $\Delta\mu_i$ is known, the height μ of the coordinate surface (defined as a w point) is obtained by vertical summation of μ_i . When the vertical grid is uniform, $\Delta \mu_i = \Delta\zeta$ and $\mu = \zeta$.

Figures 7.4 and 7.5 show several examples of the vertical grid stretching.

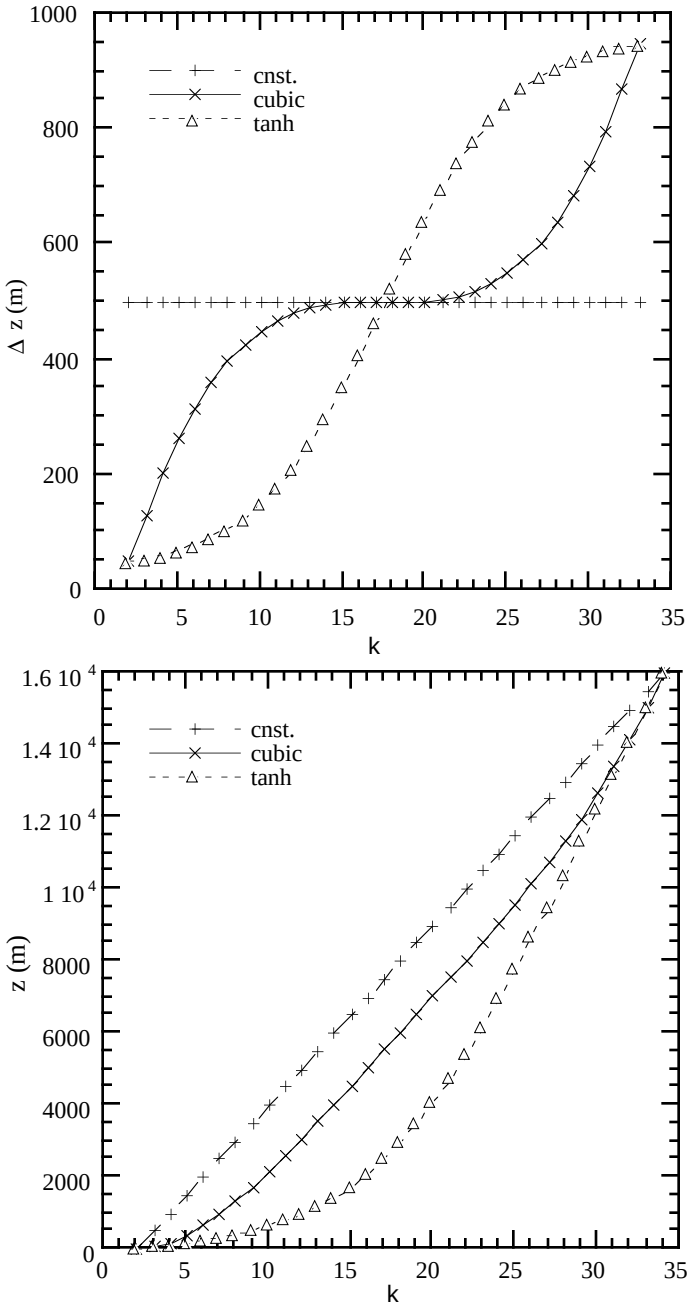


Figure 7.4. The vertical grid spacing Δz (upper panel) and the physical height z (lower panel) of the vertical grid levels as functions of the level index k, non-stretching grid and for stretching options 1 and 2. For these cases, nz=35, D=16 km, D_1=0 m, D_2=16 km and Δz_{min}=50m. It can be seen that for stretching option 1, Δz increases rapidly near the top and bottom of the vertical domain, while option 2 has the fastest increase in Δz at k~nz/2.

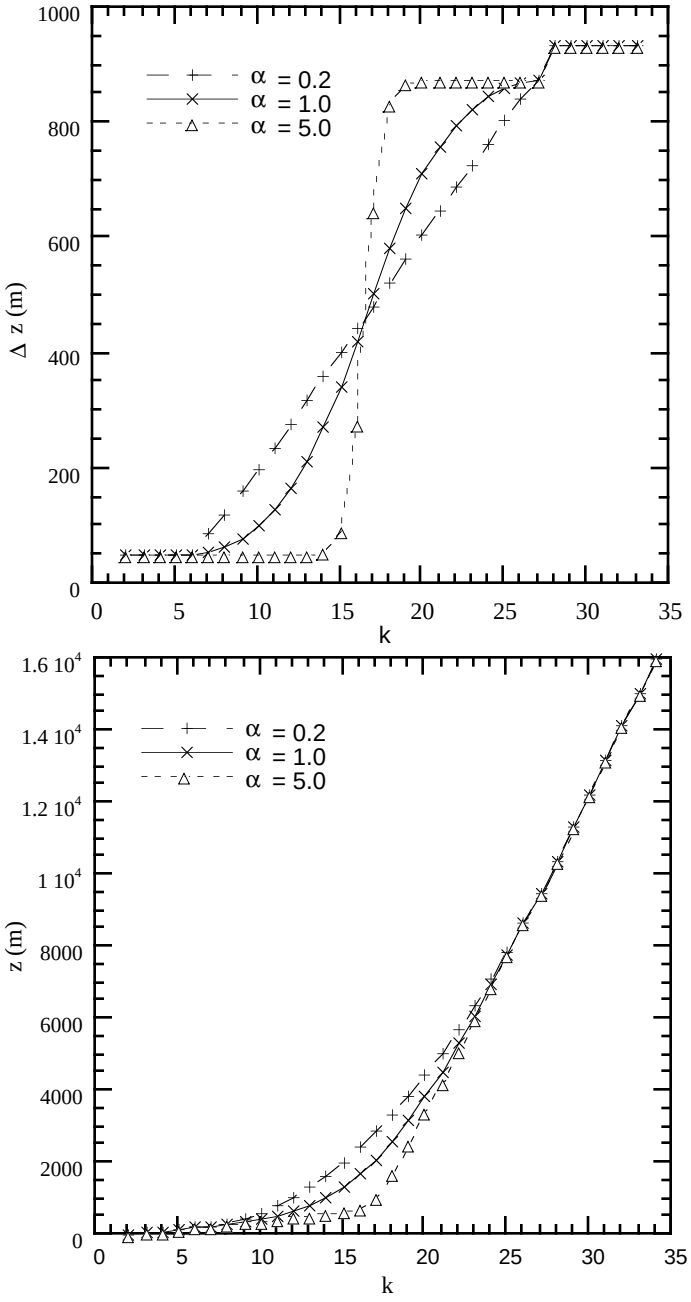


Figure 7.5. The vertical grid spacing Δz (upper panel) and the physical height z (lower panel) of the vertical grid levels as functions of the level index k, for the tuning factor α = 0.2, 1.0, 5.0 for stretching option 2. In these cases, nz=35, D=16 km, D_1=200 m, D_2=9800 m and Δz_{min}=50m. It can be seen that Δz remains constant in layer 1 and layer 3, and increases almost linearly with k for α=0.2 case but very rapidly at the mid-levels for α=5.0.

To conclude this section, we comment that the options for coordinate transformations available in ARPS cover only a few possibilities. A user can define his/her own coordinate transformation by editing the source code in subroutine INIGRD, where array *zp* (the vertical coordinate of *w*-points in physical space) is initialized. When doing so it is recommended, though not required, that the *zp*(*nz*-1)-*zp*(2) be equal to (*nz*-3) *dz* in the absence of terrain, where *dz* is the constant computational grid spacing. The Jacobians are then calculated from *zp* using finite differences. In Figure 7.6, an example of ARPS grid uses stretching option 2 and the transformation relation (7.3.1).

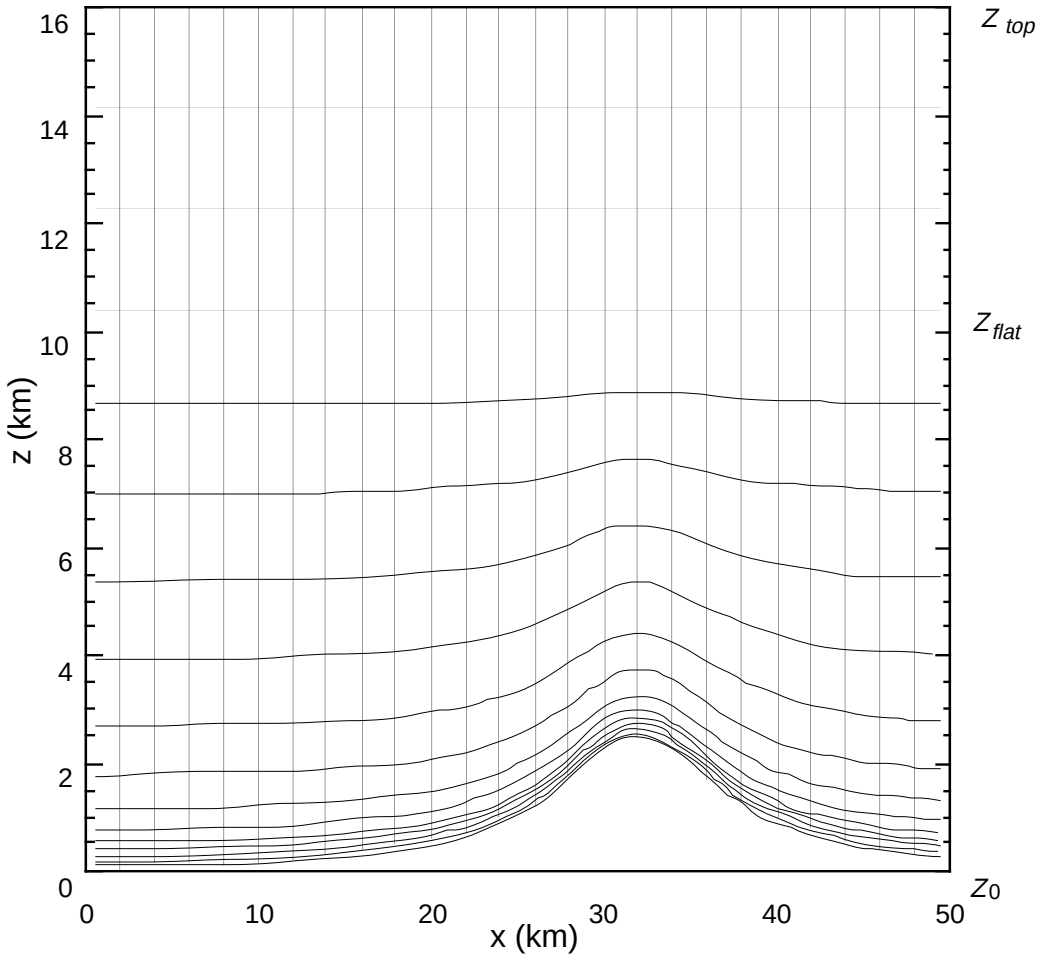


Figure 7.6. An illustration of ARPS computational domain using coordinate transformation relation (7.3.1) and grid stretching option 2 (Eq. (7.3.6)). z_{flat} =10 km, *nz*=35, *D*=16 km, D_1=200 m, D_2=9800 m, Δz_{min}=50 m and α=1.0. Only every other grid line is plotted in the vertical direction.

7.4. Adaptive Grid Refinement and Two-way Interactive Grid Nesting

The adaptive grid refinement (AGR) technique originally developed by Berger and Oliger (1984) and introduced to atmospheric applications by Skamarock *et al*. (1989) and Skamarock and Klemp (1993) is implemented in ARPS. It is one in a class of techniques aimed at automatically improving the accuracy in the time integration of a partial differential equation system. It is a local refinement method that adds grid points when and where they are needed, as opposed to global refinement where the same number of grid points are redistributed. This is achieved by adding fine grid rectangles in regions where the truncation error exceeds a tolerance, and by removing those grids that are no longer needed. In practice, they exhibit themselves as a number of multiple-nested grids that move with the developing systems. In simulation applications, the insertion and removal of nested grids are often performed manually, so that the limited computational resources are applied to areas of greatest interest.

The implementation of AGR in ARPS is based on a generalized adaptive grid refinement interface code (AGRI) developed by Skamarock and Xue. It is general in the sense that any models (solvers) that satisfy a set of code construction rules can utilize this piece of software and easily acquire the adaptive grid refinement capability, and therefore the multiple-level grid nesting capability.

AGRI requires a minimal change to the code that solves the equation system on a single grid. AGRI handles the addition and removal, initialization, time integration and information transfer (interpolation of boundary solutions from coarse grids to fine grids, and averaging of fine grid solutions to the coarse grids) between multiple-level nested grids. AGRI also manages the memory allocation of variables and arrays on each grid.

AGRI is designed especially for 3-D atmospheric models. Given the fact that horizontal refinement is, in most cases, more important and more effective than refinement in the vertical, AGRI refines the grids in the horizontal direction only. In the vertical, best results can be achieved by appropriately distributing the vertical grid levels for the entire horizontal domain, for example, by using a stretched grid.

AGRI code can handle multi-level grid nesting and inter-nesting of grids at the same level. These grids do not require alignment with the base grid, and can be rotated to best fit the solution error distribution. For most applications, the non-rotated grids are often preferred because they are easier

to handle for post-processing purposes. The aligned grids also facilitate the use of conservative interpolation schemes between coarse and fine grid solutions. In theory, AGRI allows for any level of grid nesting. In practice, 2 to 4 levels of nesting are most commonly used. An example of the application of AGRI in ARPS can be found in Xue *et al.*, (1993). In that paper, a simulation of small tornado vortices within a supercell storm is presented which used four levels of grid nesting. The finest grids have a horizontal resolution of about 55 m, while the base grid resolution is 1 km.

The source code for AGRI is usually contained in a separate directory from that of ARPS. The compilation and linking of ARPS subroutines are handled automatically by makefiles. The AGRI code is written for conventional vector computers, and does not work well on MPP machines. The AGRI source code is not currently distributed with the current official release of ARPS. Users who are interested in this capability should contact *arpsuser@tornado.gcn.uoknor.edu* or *mxue@uoknor.edu*.

7.5. One Way Interactive Self-Nesting

One way interactive self-nesting between successive runs of different resolutions can be easily performed with ARPS using the utility program, ARPSR2H (see Section 10.5). ARPSR2H interpolates the output from a larger domain run in either restart file or history dump format onto a smaller sub-domain of specified resolution, then writes out data files that can be used to initialize the smaller domain and provide the boundary forcing. ARPS standard input file, *arps40.input*, is used by ARPSR2H, *arps40.input* has an extra namelist block called *&gridinit* that is used only by ARPSR2H to define the smaller grid. When the external boundary condition data dump flag *exbcdmp* is turned on, ARPSR2H writes out external boundary forcing files for the fine grid at the same time history data are written out. The fine grid can either be initialized using interpolated coarse fields (from history format data created by ARPSR2H) or using analysis fields.

The command to compile and link ARPSR2H is

makearps *arpsr2h*

and the command to execute is

ext2arps < *arps40.input*.

For further information on initializing ARPS with a 3-D data set and on forcing ARPS using external data sets, the reader is referred to Chapter 8.

7.6. Grid Translation Features

Because of computer memory limitations, numerical modelers must decide for each experiment how their computational domain extent should be sacrificed in favor of grid resolution (and vice versa). Ideally, one would like a grid spacing fine enough to resolve the phenomena of interest and a domain size large enough that the placement of the boundaries would not unduly contaminate the solution, at least for the time scales of interest. One approach is to make use of nested or adaptive grids (see previous section). Another approach, described in this section, is to use a moving reference frame to keep the features of interest within the computational grid.

7.6.1. User-specified grid motion

When creating thunderstorm simulations, the user can reduce the total number of grid points needed by specifying a storm-following grid. The motion of the grid can be explicitly controlled by setting *grdtrns* = 1 in the input file and specifying input parameters *umove* and *vmove* which represent the components of grid motion during the simulation. However, since it is often not possible to know in advance what the storm motion will be, it may be preferable to apply one of the automatic grid-translation procedures described in the next few sub-sections (namely, cell tracking or optimal pattern translation techniques).

7.6.2. Cell tracking algorithm

ARPS includes a cell-tracking algorithm based, in part, on the radar cell tracking technique of Witt and Johnson (1993). This algorithm identifies convective cells and tracks their locations during a simulation. The location, size, direction and speed of motion of each cell are stored in a file for analysis and display. The cell tracking algorithm can be used as a "stand-alone" procedure or as the basis for a moving reference frame. The cell tracking algorithm is turned on by setting *cltkopt* = 1. To define a moving reference frame in terms of the motion of the vertical velocity centroid of these convective cells, *i.e.*, to use the above algorithm to define the domain translation, one should specify *grdtrns* = 2 in the input file. In this case, *cltkopt* is reset to 1 if not already so.

If *grdtrns* = 2, the simulation begins with a *umove* and *vmove* based on the mean wind in the lowest 5 km. As vertical velocity develops, the center of mass of updrafts exceeding 5 ms^{-1} is computed. The center of mass calculation uses data only at grid points below 12 km. After a sequence of updraft centroids has been identified, a least-squares technique finds the recent movement of the centroid and *umove* and *vmove* are modified to correct for the actual storm motion or development.

The cell tracking process produces a file named *runname.track*. This file contains a record of the cells identified during each call to the cell tracking subroutine, CELTRK. The cell absolute positions, their volumes (measured as number of grid cells having vertical velocity greater than 5 ms^{-1}) and motion are recorded. Software is available to plot these tracks as a function of time in absolute coordinates.

It is important to note that a number of adjustable parameters are used in the cell-tracking algorithm. These parameters include the maximum number of cells to be tracked, the threshold vertical velocity defining a "cell," the minimum horizontal area of a cell, *etc*. Please examine subroutine CELTRK for more details.

7.6.3. Optimal pattern translation

The translational motion of a scalar pattern determined objectively from real or model data can also used to define a moving reference frame. The technique described herein is based on a least squares technique proposed by Gal-Chen (1982). We consider the frozen-turbulence approximation (Taylor's hypothesis) as applied to the motion of a scalar quantity ϕ,

$$\frac{\partial \phi}{\partial t} + U\frac{\partial \phi}{\partial x} + V\frac{\partial \phi}{\partial y} = 0, \qquad (7.6.1)$$

where U and V are constants. It can be verified that the solution of (7.6.1) is

$$\phi = f(x - Ut, y - Vt),$$

where f is any arbitrary function of two variables. Introducing the Galilean transformation $x' = x - Ut$ and $y' = y - Vt$, the solution can be written as $\phi = f(x', y')$, in which there is no time dependence. The Galilean transformation has a simple physical interpretation: the new coordinate system is moving at the speed U in the x-direction and speed V in the y-direction with respect to the original coordinate system. Thus, a pattern satisfying (7.6.1) would appear

stationary in a frame of reference moving with velocity components U and V. Alternatively, we can say that (7.6.1) describes the translation of a scalar pattern at speeds U and V in a fixed reference frame.

Although there is no *a priori* reason why a scalar (*e.g.*, vertical velocity or rainwater) should satisfy the frozen-turbulence approximation (7.6.1), experience has shown that meteorological phenomena over a wide range of scales often appear as organized structures or patterns that "advect" or "propagate" at some characteristic speed, at least for certain periods of time. The U and V components describing this pattern-translation are not necessarily associated with the mean fluid velocity, although there often is a close correspondence between the two. Further discussions of this approximation and its limitations can be found in Hinze (1975), Townsend (1976), Panofsky and Dutton (1984) and in references therein.

With ϕ and its time derivative known (*e.g.*, from real or model data), it is possible to evaluate the terms in our pattern-translation equation locally and thus obtain local estimates of U and V. However, since we wish to obtain global (constant) U and V values, we seek the least squares solution of (7.6.1) over the whole domain for a given time-window. Toward that end, define the local error E of the frozen-turbulence approximation as: $E = \partial\phi/\partial t + U\partial\phi/\partial x + V\partial\phi/\partial y$, and define the total (squared) error as $J = \iiiint E^2 dx\, dy\, dz\, dt$, or:

$$J = \int\int\int\int \left(\frac{\partial \phi}{\partial t} + U \frac{\partial \phi}{\partial x} + V \frac{\partial \phi}{\partial y} \right)^2 dx\, dy\, dz\, dt\,.$$

To minimize the total error, we set $\partial J/\partial U = 0$ and $\partial J/\partial V = 0$ simultaneously, obtaining two linear equations:

$$AU + BV = D,$$

$$BU + CV = E,$$

where A-E are given in terms of known data by:

$$A = \int\int\int\int \left(\frac{\partial \phi}{\partial x} \right)^2 dx\, dy\, dz\, dt, \qquad B = \int\int\int\int \frac{\partial \phi}{\partial x}\frac{\partial \phi}{\partial y} dx\, dy\, dz\, dt,$$

$$C = \int\int\int\int \left(\frac{\partial \phi}{\partial y} \right)^2 dx\, dy\, dz\, dt, \qquad D = -\int\int\int\int \frac{\partial \phi}{\partial t}\frac{\partial \phi}{\partial x} dx\, dy\, dz\, dt,$$

$$E = -\int\int\int\int \frac{\partial\phi}{\partial t}\frac{\partial\phi}{\partial y} dx\, dy\, dz\, dt.$$

The solution for U and V is thus:

$$U = \frac{EB - DC}{B^2 - AC}, \qquad V = \frac{BD - EA}{B^2 - AC}. \tag{7.6.2}$$

Numerous grid-translation experiments with simulated multicell and supercell storms suggest that the vertical velocity field, w, is a suitable candidate for the diagnosis of the U and V components. On the other hand, because of their discontinuous distribution in space and time, moisture variables (*e.g.*, rainwater) were less reliable. When applied to an isolated supercell storm, the w-based grid-translation algorithm kept the storm nearly in the center of the domain. However, in the case of a splitting storm, the clear air between the two sibling storms was kept nearly in the center of the domain, while both storms eventually left the domain. Since we were mostly interested in tracking the stronger storm (the right-mover), we decided to modify this w-based technique to give more weight to the areas of the domain that had stronger vertical velocities. This was accomplished by tracking higher powers of w rather than w itself, a legitimate modification since the frozen-turbulence approximation for any arbitrary function of w is valid if the frozen turbulence approximation is valid for w itself. A technique based on w^4 gave the desired improvement and enabled us to "follow the strongest pattern." If a user wishes to use a different scalar for the optimal pattern translation, the necessary code changes are straightforward.

It can also be noted that if a storm is initialized with a hot bubble in a horizontally homogeneous environment, data from the first few time steps of the simulation can yield meaningless system translation velocities. An upper bound formulated on the CFL criterion is applied to the U and V components to limit these early inaccurate values. If the time window is larger than a few model time steps (as is almost always be the case) these first few inaccurate values will have a negligible impact on the running time-mean of U and V.

The optimal grid-translation algorithm is designed to "follow the dominant pattern" but does not necessarily keep that pattern in the center of the domain. For instance, an isolated storm may be kept nearly in the center of the domain but if it generates stronger cells on the cold outflow some distance away, the newer convection would be tracked without being repositioned to the center of the domain. The optimal grid-translation algorithm works only with the speed of scalar patterns, not with the locations of the patterns. The advantage of this technique is that pattern translation velocities are finite regardless of the time window, whereas translation

velocities based on the locations of patterns can become quite large (theoretically infinite in the case of discontinuous propagation, *i.e.*, new cell formation) and can be quite sensitive to the duration of the time window.

The optimal grid translation subroutines can be found in file *grdtrns3d.f*. Subroutine ADJUVMV (which calls GALILEI) adjusts the model variables at the past and present model time levels (see next section) whenever the domain translation speed changes, regardless of the algorithm that caused the change in the domain translation speed (user-specified change, automatic grid translation, cell-tracking update, *etc.*). Subroutine AUTOTRANS computes the optimum (least squares) pattern translation speed.

7.6.4. Redefining quantities in a moving reference frame

Once the grid translation velocity is determined (*e.g.*, by the cell tracking procedure or optimal pattern translation procedure), all model variables are redefined in the moving reference frame. This redefinition also occurs if the model is initialized from a restart file in which the *umove* and *vmove* components differ from those in the current input file. The appropriate redefinition for the current values of the horizontal wind field u, and v, and for all scalars ϕ is:

$$u = u - U$$
$$v = v - V$$
$$\phi = \phi$$

where U and V are the new system translation components (or the change in the system translation components if the old system was already translated). Because ARPS currently uses the leapfrog scheme to integrate the governing equations, model variables at both present and past time levels must be redefined in the moving reference frame. However, care must be taken that the time derivative of the transformed variables (based on present and past values) satisfies the Galilean transformation: $\partial/\partial t = \partial/\partial t' - U\,\partial/\partial x' - V\,\partial/\partial y'$. The necessity for this condition can be seen by considering the example of a flow that is stationary in a moving reference frame. Such a flow would not be stationary in a fixed reference frame (it would appear as a translating pattern), hence the local derivatives in the fixed and moving reference frames should not be the same.

An appropriate redefinition for the 2 time levels is,

At the "present" time:

$$u(i,j,k,tpresent) = u(i,j,k,tpresent) - U$$
$$v(i,j,k,tpresent) = v(i,j,k,tpresent) - V$$
$$\phi(i,j,k,tpresent) = \phi(i,j,k,tpresent)$$

At the "past" time:

$$u(i,j,k,tpast) = u(i,j,k,tpast) - U - U\,\Delta t\,\partial u/\partial x - V\,\Delta t\,\partial u/\partial y$$
$$v(i,j,k,tpast) = v(i,j,k,tpast) - V - U\,\Delta t\,\partial v/\partial x - V\,\Delta t\,\partial v/\partial y$$
$$\phi(i,j,k,tpast) = \phi(i,j,k,tpast) - U\,\Delta t\,\partial \phi/\partial x - V\,\Delta t\,\partial \phi/\partial y$$

where Δt is the model big time step.

Currently, centered space derivatives are used to approximate the spatial derivatives. The time level of the spatial derivatives is half-way between the past and present time levels (*i.e.*, an average is taken).

7.6.5. Some practical considerations

In the current version of the code, the user is asked to specify several grid translation input parameters in ARPS input file. The parameter *grdtrns* determines which algorithm is used to compute the system translation components. If *grdtrns* = 1 then *U* and *V* are just the user-specified values *umove* and *vmove*. If *grdtrns* = 2 then *U* and *V* are determined from the cell-tracking algorithm (see Section 7.5.2). If *grdtrns* = 3 then *U* and *V* are determined from the optimal pattern translation technique described in Section 7.5.3.

If *grdtrns* = 3 (optimal pattern translation) the user must also specify two additional parameters: *chkdpth* and *twindow*. The *chkdpth* parameter sets the depth above the *k=3* physical grid point over which *U* and *V* are calculated (the reason the minimum *k* level is set to 3 is because the optimal pattern translation is based on the vertical velocity field, which first becomes non-zero at level 3). The user is cautioned that *chkdpth* should be $\geq dz$. The *twindow* parameter sets the time interval (in seconds) between successive updates of the grid translation speed. *twindow* also defines the interval over which the running average of the system translation speed is computed.