INITIALIZING GRIDDED FIELDS FROM NUMERICAL MODELS

T. LeFebvre[1], M. Romberg, T. Hansen

NOAA Research−Forecast Systems Laboratory

Boulder, CO

1. INTRODUCTION

For the past decade, the National Oceanic and Atmospheric Administration's (NOAA) National Weather Service (NWS) has worked toward utilizing the latest accomplishments in computer technology through its massive modernization effort. A component of the modernization, the Advanced Weather Interactive Processing System (AWIPS), allows NWS forecasters to display and combine a variety of meteorological datasets on a single display. Now that AWIPS is in use operationally, the NWS personnel are turning their attention to modernizing the set of products and services delivered to the public. The Interactive Forecast Preparation System (IFPS), being developed jointly by the NWS's Meteorological Development Laboratory and NOAA's Forecast Systems Laboratory (FSL), is a software component of AWIPS that allows NWS forecasters to define weather forecast elements in a gridded digital format. Products based on these digital grids can take many forms including the typical text−based products the NWS issues today, graphical representations that can describe the weather more precisely, and the gridded data per se for customers that require the maximum detail (LeFebvre et al. 1996).

2. Graphical Forecast Editor

For nearly a decade, FSL, guided by NWS field forecasters, has been developing the Graphical Forecast Editor (GFE), which is the forecast grid−editing component of IFPS. Using the GFE, forecasters manipulate gridded values of surface−based sensible weather elements such as temperature, dew point, and wind that define the future state of the atmosphere. Hansen et al. (2000) discusses the methodology forecasters employ to modify these grids. The GFE consists of a set of editors, each with its own set of tools that modify the gridded data. The Spatial Editor furnishes tools to edit the grids in a plan view or map format. The Temporal Editor contains displays in time−series format along with tools to modify the grids temporally. The Grid Manager provides tools to manage the grid inventory, perform time interpolation, and initialize the forecast based on numerical model output.

About two years ago, with the essential components of GFE complete, we turned our attention to developing a set of sophisticated tools that enforced a particular meteorological law or concept. These new "Smart Tools" were required to work effectively during all seasons in disparate climate regimes. Many tools would need to be customized to each local forecast area, further adding to the challenge.

The solution was not to develop a set of tools, but a framework within which forecasters, researchers, and software developers could employ their own ingenuity to invent their own tools. LeFebvre et al. (2000) describes this Smart Tool framework. This type of extensible architecture offers several advantages:

- An enormous amount of communication would need to take place between field forecasters and developers in order to develop an efficient set of editing tools, if developers were responsible for writing them. This communication has been shown to be very time−consuming and greatly slows the process of developing effective software. The need for this communication is greatly reduced once forecasters are allowed to write their own tools.

- No matter how tools are developed, local forecasters will need to modify them because of local climatological and topographical effects. The Smart Tools framework gives forecasters complete control over every tool.

- The editing system can be easily extended without a major redesign and thus is able to adjust to new requirements quickly and with minimum effort.

- Most important, these Smart Tools encapsulate some meteorological concept or rule of thumb from the human knowledge−base at the local forecast office. The tools may be easily shared among colleagues in other forecast areas. With each of dozens of forecasters contributing a piece of weather knowledge, we can quickly build a large library of tools that when combined, allows forecasters to express complex meteorological concepts to the editing system (GFE), which then performs the complex task of calculating the actual weather values.

Recently, the Smart Tool framework concept has been extended to another area of GFE development. In order to quickly populate a gridded forecast, the GFE requires algorithms that transform numerical weather prediction models, such as Eta, into grids of surface sensible weather elements. We call this set of algorithms *Smart Initialization*. It should be noted that the word "initialization" used here does not mean defining and balancing the initial state of the

1Corresponding author's address: Tom LeFebvre, NOAA Forecast Systems Laboratory, 325 Broadway R/FS6, Boulder, CO, 80305;
e−mail: lefebvre@fsl.noaa.gov

atmosphere within a numerical model. Rather, we use it to describe the process of populating an undefined gridded forecast database with weather values consistent with a particular numerical model. These Smart Initialization algorithms are described in detail in the sections to follow.

3. SMART INITIALIZATION

The gridded forecast database comprises hundreds of grids that define the state of the atmosphere for particular weather elements over a specified time period. As time marches forward, these grids eventually expire when their forecast valid time passes. Relative to the current time, grids can be thought of as moving, since their valid times get perpetually closer to the current time. We call this type of database a rolling database.

Generally forecasters using the GFE start with the current forecast and then make nominal adjustments using simple tools or the Smart Tools briefly described above. This way they can take full advantage of the work done by their predecessors who worked earlier forecast shifts. As the rolling database moves forward, an undefined time segment will be created during the latest time periods. For example, grids valid seven days from now will be eventually six days away, leaving an unspecified 24−hour time segment in the six to seven day forecast period. Forecasters could use the tools available to draw these grids by hand, but doing so is very tedious and time consuming, particularly in forecast areas with complex terrain.

Occasionally, for a variety of reasons, a forecaster's opinion may sharply deviate from the current forecast. Clearly, forecasters will not have the time to create all of the grids by hand in time to meet product deadlines.

In order to provide forecasters with a good starting point when populating an empty forecast period, we built a program that creates surface−based sensible weather grids based on numerical model output. Using such a program, forecasters can "initialize" their forecast based on the numerical model of their choice. This allows them to quickly populate their "empty" forecast with reasonable values, such that the weather elements are internally consistent, e.g., clouds in areas of rain. Unfortunately, numerical models do not generally produce fields of sensible weather at the surface. In addition, these models run at coarser spatial resolution (40−80 km) when compared to a typical GFE resolution (less than 5 km).

The GFE has employed this technique for the past several years (Wier 1998) but the method by which the original algorithms were implemented had some significant drawbacks. Previous versions of Smart Initialization were written in C++. This demanded that we develop and compile the algorithms in a development environment. This technique offered no method for forecasters to modify or adjust these tools once they arrived at the local office. Even the simplest suggestion from forecasters required many months before the modified code could be delivered to them because of the elaborate code management system imposed on all AWIPS software.

The solution to this problem was to implement the algorithms in a way that forecasters could modify them based on their meteorological knowledge of the local climate regime. That way forecasters themselves had the opportunity to immediately implement any new idea, thus eliminating the long turn−around time required by the old precompiled method. We chose to implement the Smart Initialization algorithms in a programming language called Python. Python is an "interpreted" language, which means it is executed line−by−line and does not require compiling. In addition, Python's syntax is very easy to read and learn; forecasters with any programming experience can learn it quickly.

Interpretive languages such as Python also have their drawbacks, however. The fact that interpretive languages are compiled at execution time generally means that they execute slowly, and Python is no exception. The extensive numerical calculations required by many of the algorithms meant that forecasters had to wait several minutes per operation in some cases, before the results were ready.

Recently, the performance issue has been greatly alleviated by employing an extension to Python called Numerical Python.[2] This package operates on entire grids in a single step rather than iterating over every grid point. The result is that the same algorithm runs up to 20 times faster when implemented in Numerical Python.

With many of these problems solved, we created Smart Initialization algorithms to convert model output into surface−based weather elements. In the following sections, we document each of the 17 different weather element algorithms in varying detail. In many cases the algorithm is somewhat dependent on the particular model used, since some models do not offer the same output parameters, especially near the earth's surface. Since space is limited, we will document a few algorithms in detail, and briefly describe the rest.

4. INITIALIZING SURFACE WEATHER ELEMENTS

4.1 Wind

Wind is one is one of the simpler tools since the algorithm extracts the 10 m wind from the model and

2 Numerical Python was designed and implemented by a group at Lawrence Livermore National Laboratory. For more information, see the Numerical Python web site at: http://www.pfdubois.com/numpy

then just performs a unit conversion. The Numerical Python code for this is shown below.

```
def calcWind(self, wind_FHAG10):
    mag = wind_FHAG10[0]  # extract the magnitude
    dir = wind_FHAG10[1]  # extract the direction
    mag = mag * 1.94      # convert from ms−1 to knots
    dir = clip(dir, 0, 359.5)  # confine direction values
    return (mag, dir)
```

Note: FHAG refers to fixed height above the ground, in this case, 10 meters.

4.2 Temperature

Most models produce temperature near the surface (2 meters) but the coarse resolution of most models produces only smooth representation of the surface temperature. Using a high−resolution topography dataset, we can add detail to the model's temperature forecast by adjusting the values based on elevation.

Figure 1 illustrates that the model topography can greatly deviate from the "true" topography as extracted from the GFE's 1−km resolution dataset. Inside the temperature algorithm, we calculate this difference as well as the model temperature lapse rate to determine a more precise temperature value. The result is a detailed temperature grid that more closely follows the terrain.

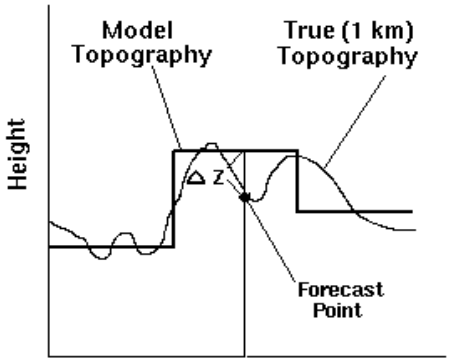

Figure 1: This figure conceptually illustrates the difference between the model topography and the 1 km true topography dataset.

The Python code for this algorithm is shown below.

```
def calcT(self, t_FHAG2, t_BL3060, stopo, topo):
 # Determine the lapse rate between 2 meters and
 # the 30−60 mb above ground layer in deg / millibar
 # using a form of the hydrostatic equation
 dz/dp = 287.04 * t_FHAG2 / (p_SFC / 100 * 9.8)
# 30 − 60 mb layer averages 45 mb above the sfc
dz = dz / dp * 45

 lapse = (t_FHAG2 − t_BL3060)  / dz

# Calculate the true temp. by multiplying the
# lapse rate by the difference in topography
t = t_FHAG2 + lapse * (stopo − topo)

# Convert from Kelvin to Fahrenheit and return
return self.KtoF(t)
```

4.3 Mixing Height

The mixing height algorithm calculates the potential temperature as a function of height. Then it calculates a running average of potential temperature from the ground upward. When the environmental potential temperature exceeds the running average by some threshold, a precise mixing height is interpolated. Figure 2 illustrates this method.

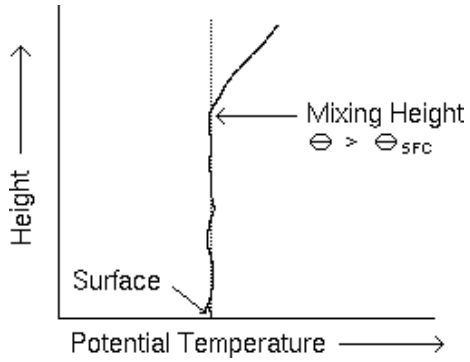

Figure 2: The diagram illustrates that the Mixing Height algorithm is calculated by generating a potential temperature profile as a function of height and finding the level at which it deviates from the surface potential temperature.

```
def calcMixHgt(self, T, topo, t_c, gh_c):

    # Make a mask for above ground
    mask = greater_equal(gh_c, topo)

    # Make a potential temperature sounding cube
    pt = []
    for i in xrange(len(self.pres)):
        p = self._empty + self.pres[i]
        tmp = t_c[i] * pow((1000 / p), 0.286)
        pt = pt + [tmp]
    pt = array(pt)

    # Zero out below ground
    pt = where(mask, pt, 0)

    # Get the running average of the pot. temp.
    avg = add.accumulate(pt, 0)
```

```
count = add.accumulate(mask, 0)
mh = self._minus

# Go through sounding cube and get a running avg.
for i in xrange(1, avg.shape[0]):
   runavg = avg[i] / (count[i] + .0001)
   diffpt = pt[i] − runavg

# Perform a linear interpolation to get a mixing ht.
tmh = self.linear(pt[i], pt[i−1],
        gh_c[i], gh_c[i−1],runavg)

# Assign mixing height at first point where
# there is a difference of more than 3 between the
# running average and the potential temperature
mh = where(logical_and(logical_and(mask[i],
     equal(mh, −1)),greater(diffpt, 3)), tmh, mh)

# Convert to feet AGL
return (mh − topo) * 3.28
```

4.4 Maximum Temperature

The maximum temperature algorithm calculates the maximum value found in all temperature grids that overlap the time period of interest.

4.5 Minimum Temperature

The minimum temperature algorithm calculates the minimum value found in all temperature grids that overlap the time period of interest.

4.6 DewPoint

This algorithm calculates the mixing ratio from the surface temperature and relative humidity, then determines the dew point from the calculated surface temperature and mixing ratio.

4.7 Quantitative Precipitation Forecast

The QPF algorithm uses the total precipitation grid from the model and converts units to inches.

4.8 Sky Cover

The Sky Cover algorithm calculates a sky layer coverage at each model level based on the relative humidity. Then it integrates these levels to produce total sky coverage by calculating the amount of clear sky obscured by each level. For example, if one level has 50% coverage and other has 50% coverage, then the total sky cover is 75%.

4.9 Probability of Precipitation

This initialization tool uses the model's total precipitation to calculate the PoP using a linear scale. Where the total precipitation is less than 0.02 inches and there is a high mean relative humidity PoP values are increased slightly. Figure 3 graphically depicts this function.

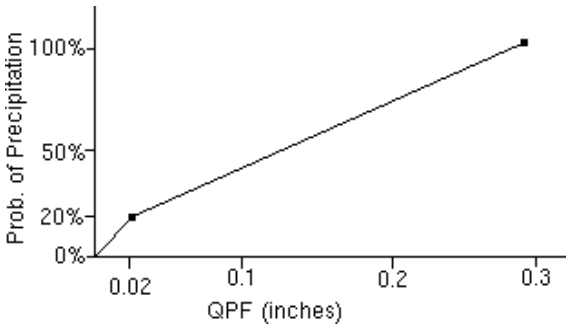

Figure 3: The diagram graphically depicts the function used to calculate Probability of Precipitation based on the Quantitative Precipitation Forecast (QPF).

4.10 Freezing Level

To calculate freezing level, this algorithm searches vertically for the first occurrence of the model temperature that is below freezing, and interpolates downward to determine the "actual" level of the freezing temperature.

4.11 Snow Amount

The Snow Amount tool calculates a snow ratio based on the surface temperature. In areas that are 1000 feet or greater above the freezing level, the snow amount is the QPF multiplied by the snow ratio. In areas below 1000 feet of the freezing level, the snow amount is assigned a zero value.

4.12 Haines Index

The Haines Index algorithm uses the temperature and relative humidity fields to calculate dewpoint at 500 mb and 700 mb, and, from this, calculates the Haines Index.

4.13 Free Air Wind

The Free Air Wind tool interpolates between wind values (calculated from the u and v components) at model levels to obtain a value at 3000 feet above the surface.

4.14 Transport Wind

The Transport Wind method calculates the average wind in the layer bounded by the surface and the mixing height. If there is no mixing height, then the wind grid is zero.

4.15 Weather

The Weather algorithm is responsible for defining the weather type, coverage, and intensity. To accomplish this, it uses the total precipitation and convective precipitation grids to calculate the convective precipitation ratio, which in turn is used to determine whether shower or stratiform precipitation is expected. The model's lifted index is used to determine the likelihood of thunderstorms. The surface temperature determines whether the precipitation type is rain or snow.

4.16 Chance of Wetting Rain

This is similar to the PoP algorithm (see above) in that it uses the total precipitation but with different thresholds.

4.17 Lightning Activity Level

Starting at zero, the LAL is increased by one for each of the following conditions that are true:

- ratio of convective precipitation to stratiform is greater than 50%
- midlevel (~500−600 mb) moisture is high and accompanied by dry air at the surface
- lifted index is less than −3
- lifted index is less than −5

5. CONCLUSION

Smart Initialization provides forecasters complete control over the algorithms that calculate surface−based sensible weather elements from a numerical model. The framework also gives forecasters an opportunity to contribute their meteorological knowledge to the gridded forecast system (GFE). The use of Python permits the kind of flexibility that is needed in NWS field offices. The Numerical Python extension increases the performance of these algorithms to an acceptable level.

In the near future, we plan to implement a verification facility which will allow forecasters to evaluate the accuracy of these algorithms, providing valuable data critical to improving these algorithms. We invite anyone with suggestions for new algorithms to contact the main author at the postal address or e−mail address listed on the first page of this paper.

6. REFERENCES

Hansen, T. L., M. Mathewson, M. Romberg, 2000: Forecast Methodology using the GFESuite. Preprints, 17th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Albuquerque, NM, Amer. Meteor. Soc.

LeFebvre, T.J., C. Bacco, and M. Romberg, 1996: Gridded Forecast Products and Services Under Development at FSL. Preprints, 15th Conference on Weather Analysis and Forecasting, Norfolk, VA, Amer. Meteor. Soc.

LeFebvre, T. J., M. Mathewson, T. Hansen, M. Romberg, 2000: Injecting Meteorology into the GFESuite. Preprints, 17th International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Albuquerque, NM, Amer. Meteor. Soc.

Wier, S.K., 1998: Surface Forecast Guidance Made from Numerical Models. 16th Conference on Weather Analysis and Forecasting, Atlanta, GA, Amer. Meteor. Soc.