Newly Interpolated Meso-scale Boundaries Using Surface Stations

Final Project

Advanced Geographic Information Systems (GIS II)

CE 4940

Brian Hirsch

Dr. Matisziw

May 9, 2016

**Introduction**

Weather conditions change every second of everyday. In order to create the forecasts that inform you what clothes you should wear, the numeric weather prediction model needs to know what the current weather conditions are within the model domain. One of the many factors that can improve forecasting is the injection of large amounts of weather observations (both surface and upper air). Another important factor in providing a good forecast is providing the forecaster with a map of current observations. Without a map the forecaster has to rely on remembering where the individual stations are and what the station is observing to draw the map in their head.

**Objective**

Newly Interpolated Meso-scale Boundaries Using Surface Stations (NIMBUSS) was designed to map the [real-time Missouri Mesonet Stations](http://agebb.missouri.edu/weather/stations/index.htm) and look for any atmospheric boundaries within the data. As stated in the introduction, a visualization of meteorological data is vital to situating the forecaster in the current state of the atmosphere. There is no publically available visualization of the Missouri Mesonet data. To aid forecasters, NIMBUSS was developed to visualize the Missouri Mesonet data. NIMBUSS also processes the Mesonet data and looks for an atmospheric boundary. This process is still under development and currently outputs meteorologically inaccurate representations of atmospheric boundaries. NIMBUSS also creates a place file that that allows the user to display the Mesonet Stations in Gibson Ridge (GR) software (e.g. GR2Analyst, GRLevel3, etc.). Once NIMBUSS has its initializing input form the user (map production intervals), it will run until the user closes it.

**System Requirements and Setup**

NIMBUSS has several easily obtained system requirements. The first of which is Microsoft Windows 7/8/10 with ArcGIS 10.2 and Python 2.7 installed on it. Python needs six modules installed that are not default to Python 2.7. These modules include: schedule, arcpy, request, beautifulsoup4, geodesic, and arcpy\_metadata. These modules can be installed into the “Python” directory using pip; with the exception of arcpy, which will be installed alongside ArcGIS 10.2. Once the modules are installed, the user needs to be check that the secondary scripts that NIMBUSS (main script) depends on are in the same directory. These scripts include: GrabData, PlacefileMaker, StationShape, FrontFinder, PolylineMaker, and MapMaker. The function of these scripts will be explained in the secondary script section below. Three sub-directories are also needed within the NIMBUSS directory. The “JPGs” directory is where NIMBUSS will output the maps as a .jpeg. The “placefile” directory is where NIMBUSS will store the place file. The “JPGs” and “placefile” directories can be empty. The final directory “templates” is where NIMBUSS looks for the map template and map layers when creating the outputted map. This directory needs six different files: “FrontTemplate.lyr”, “MOCounties.lyr”, “MOStations.lyr”, “MOStationsLabel.lyr”, “WindDirection.cal”, and “MapTemplate.mxd”. The final requirement in the NIMBUSS directory is the “NIMBUSS.gdb” geodatabase. The only feature class that is required in the geodatabase is the “MOCounties” feature class. If the users does not alter the NIMBUSS directory, all of these required directories and files should be present and correctly placed.

**Running NIMBUSS**

To run NIMBUSS either, double click NIMBUSS.py or run NIMBUSS.py in the command line. When NIMBUSS asks how often you would like a map of the surface stations, enter the numeric value (in hours) and press enter. NIMBUSS is now running.

**Methodology**

As alluded to in the System Requirements and Setup section above, NIMBUSS runs off of one main script which calls several secondary scripts. The reason for using several scripts rather than one script was to see the overall workflow more easily and allow for an easy implementation of scheduled workflow. The scheduled workflow allows for a place file to be created every time the Mesonet stations data updates and for a map to be created at a user defined interval. By using this schedule, NIMBUSS can run in the background with no input from the user for an indefinite amount of time. The purposes of each script will be briefly described below. For more information see the code itself and the comments within the code.

The main script is called NIMBUSS.py. This script runs the rest of the scripts at defined intervals. First this script displays a friendly introduction to the user and asks the user how often they would like a map to be produced. It then sets up a queue for the tasks. These tasks being creating a place file and creating a map. The “create a place file” task runs every five minutes because that is how often the real-time Mesonet URLs update. The “create a map” task runs at the user defined interval.

The first task “create a place file” runs in a function called placefile. The placefile function first calls a function on a secondary script called GrabData then creates a place file using a function called makePlaceFile from a different secondary script.

The second task “create a map” runs in a function called mapper. The mapper function, first uses GrabData to download the Mesonet data. Mapper then using a makeShapefile function found in a secondary script. Next mapper calls a function called findFront in another secondary script. If findFront finds a front mapper will then create a polyline representing the front using a function called PolylineMaker then map the front and Mesonet stations using a Mapmaker function. If a front is not found mapper will only map the Mesonet stations.

The GrabData function can be found in a script called GrabData. The function downloads the Missouri Mesonet Station ([for example](http://agebb.missouri.edu/weather/realtime/albany.php#more-conditions-page)) HTML using the request module and uses the BeautifulSoup4 HTML parser to obtain the data from the HTML. GrabData currently obtains the station’s temperature, dewpoint, wind speed, wind direction, and pressure from the HTML. GrabData creates a list for each station containing: the stations name, latitude, longitude, temperature, dewpoint, wind speed, wind direction, and pressure (if available). GrabData then adds this list to another list that will contain the data for each station. This process is repeated for each real-time Mesonet station. GrabData returns two lists: global\_stations which contains wind directions in degrees, and text\_stations which contains the wind directions as cardinal directions (e.g. N, E, S, W, NNE, etc.). The global\_stations list is used in the mapper function (second task) while text\_stations is used in the placefile function (first task).

The makePlaceFile function can be found in the PlacefileMaker script. A place file is simply a text file. The text file must be formatted in accordance to GRLevelX’s [Place File Specification Documentation](http://www.grlevelx.com/manuals/gis/files_places.htm). To do this the makePlaceFile function, first writes a header containing copyright and data source information. The next line is the placefile title; followed by the refresh rate, fonts, view threshold, and text color. In the body of the place file is where the dynamic data is stored. Each body line is a different station that will be plotted in GR software. The body lines start out with a “Text:” flag indicating the data will be displayed as text. Next in the line is the latitude and longitude, followed by the font ID and displayed temperature. Each of these items are separated by commas. The next item is the hover text. The hover text is the text that displays when the user hovers their cursor over the station. I chose to display: temperature, dewpoint, wind speed, wind direction, pressure, and observation time in the hover text. The makePlaceFile function then saves the newly created place file in the “placefile” directory. See Sample.txt in the placefile directory for an example of a place file.

The makeShapefile function can be found in the StationShape script. The name of this function and script is misleading, the function does not create a shapefile but rather a feature class. The function writes an attributed table using data in the global\_stations list. It writes six fields with the raw output from the GrabData function and four fields which contain data that was converted from the raw data to a standard meteorological surface station plot format. This format will be discussed in the output section below. The arcpy\_metadata module is used to write metadata to the new feature class. The feature class is saved in the NIMBUSS.gdb geodatabase with using the name format of “MOStations” followed by the date and time (e.g. MOStations201605091416 where the date and time is May 9, 2016 at 14:16 UTC). The makeShapefile function returns the file path of the outputted feature class, as well as, the feature class name.

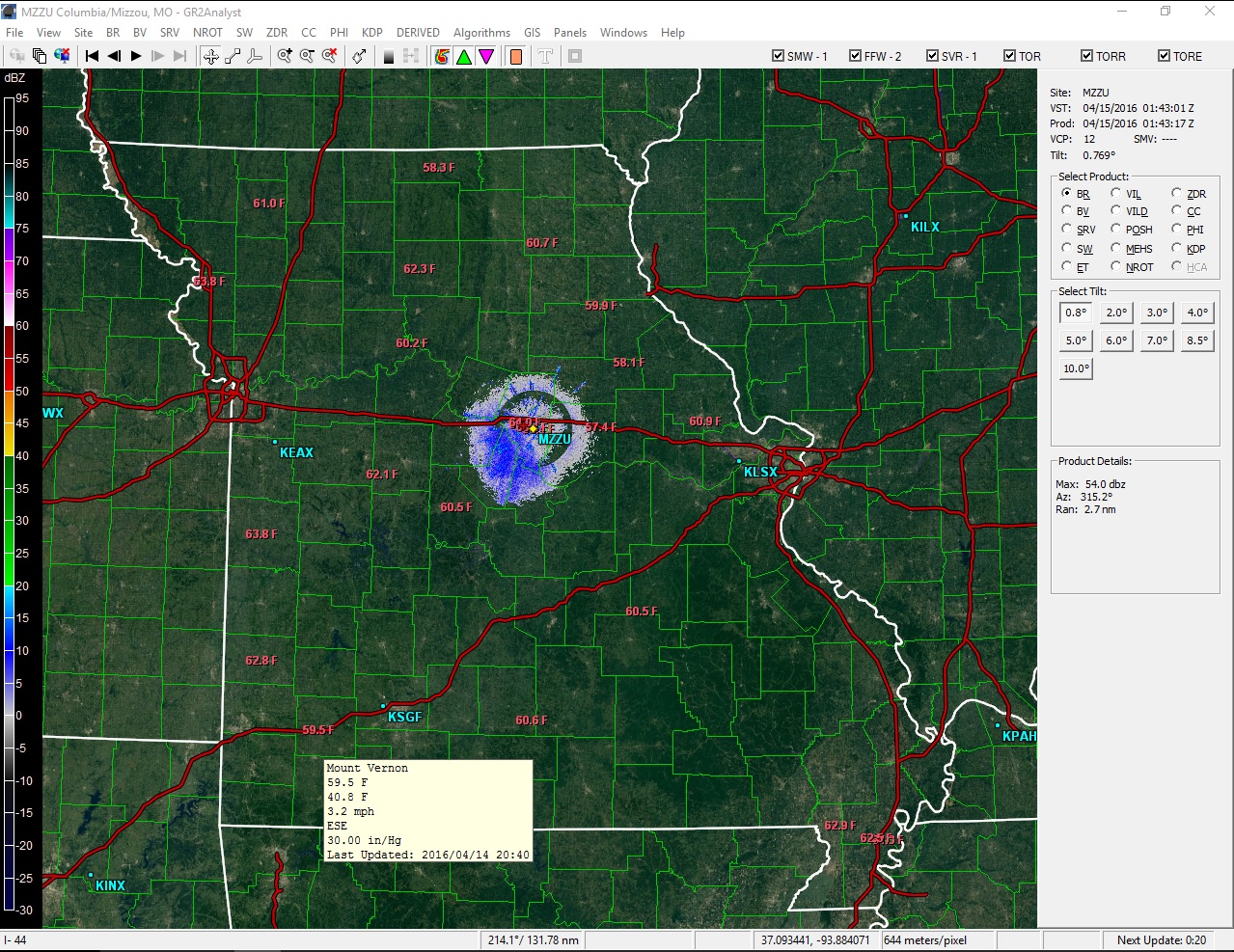
The findFront function can be found in the FrontFinder script. This script looks through each possible pair of stations and checks if the stations are less than 1.2 or greater than 0.1 decimal degrees apart. If the station pair meets the distance criteria, the function then checks if the difference in temperature and dewpoint between the stations is greater than or equal to 10 degrees or if the wind direction between the stations changes by 150 degrees. If a station pair meets the temperature and dewpoint, or wind direction criteria, the midpoint between the stations is calculated using the geodesic module and stored in a list with other midpoints that were found. These midpoints represent the front location. If the function finds at least five midpoints to orders the list of points to the shortest distance between the points. This is done so that the PolylineMaker function will make a line across that will not turn back on itself or look similar to a bowl of spaghetti when mapped. The findFront function returns a list of ordered coordinates and a Boolean variable indicating if a front was found or not.

The PolylineMaker function can be found in the PolylineMaker script. This function makes a polyline representing the front using the ordered midpoints found using the findFront function. The PolylineMaker function outputs a feature class called PolyFront in the NIMBUSS.gdb geodatabase. The PolylineMaker function returns the file path of the outputted polyline front feature class.

The MapMaker function can be found in the MapMaker script. As the name suggests, this script makes a map using the newly created data. It updates the data in the label classes of the station feature classes, adds the wind speed and directions layer and symbology, adds the front polyline layer and symbology, the base map of Missouri, and updates the title to represent the time of the observations. The MapMaker function outputs a .jpeg of the map in the “JPGs” directory. The outputted map is saved using the date and time scheme used in the makeShapefile function.

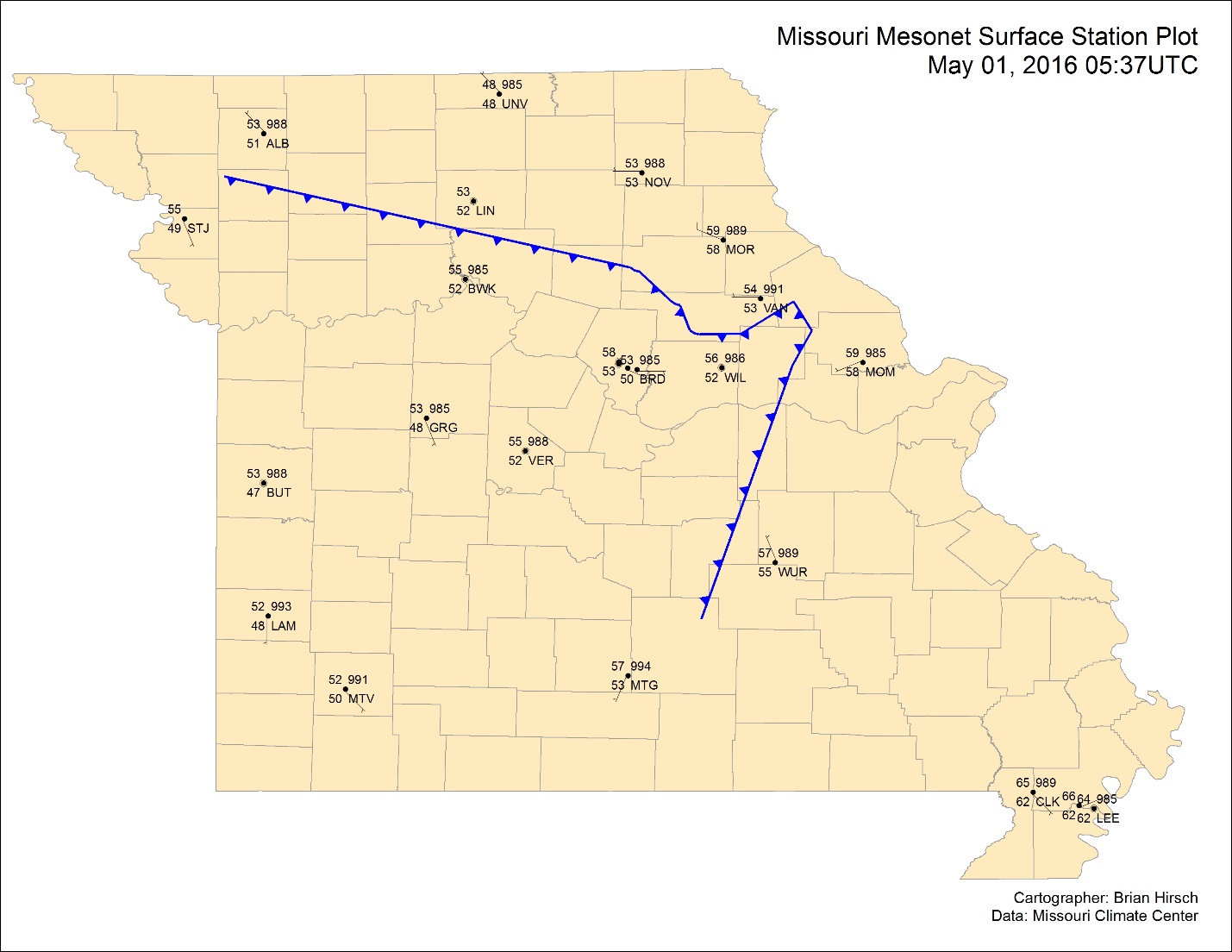
**Output**

The outputted place file displayed in GR2Analyst can be seen below in figure 1. One can see the temperatures for each station displayed in red and the hover text for Mount Vernon being displayed in southern Missouri.

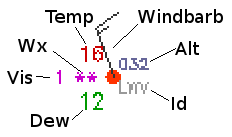


**Figure 1:** The outputted place file displayed in GR2Analyst

An example of a map outputted by the mapper function can be seen below in figure 2. The standard meteorological surface station plot format can be seen in figure 3. Temperature is the number in the upper left. Dewpoint is the number in the low left. Station ID is in the lower right. And pressure is in the upper right. Pressure is displayed in in/Hg and the first digit and decimal is removed. If the pressure starts with a 6, 7, 8, or 9 add a 2 to the beginning of the displayed temperature (i.e. 974 = 29.47 in/Hg). If the pressure starts with a 0, 1, or 2 add a 3 to the beginning of the displayed temperature (i.e. 012 = 30.12 in/Hg). The wind barbs are pointed to where the wind is coming from, therefore the winds in southwest Missouri in figure 2 are out of the south. The flags at the end of the wind barb represent the wind speed. If there is no wind barb, the wind is calm at that station. If there is a half flag on the wind barb the wind speed is 5 mph (see ALB and STJ in figure 2). There is a full flag on the wind barb with wind speed is 10 mph (see MOR in figure 2).



**Figure 2:** Example of a map outputted by the mapper function



**Figure 3:** The standard meteorological surface station plot format

**Troubleshooting**

The only known runtime problem occurs if there is a lock on one of the files in the geodatabase. This occurs sometimes if the script is ran, closed, then ran again. The exact cause of this lock has yet to be determined. However the way to fix this error, is to close all Python scripts and shells then end all python processes in the Windows Task Manager.

**Known Problems and Future Work**

One issue with NIMBUSS is the front it maps. Fronts and atmospheric boundaries, like the one seen in figure 2, do not have sharp, jagged curves. A smoothing algorithm will be written at a later date to fix this problem. Sometimes NIMBUSS draws a front with a loop in in. These loops also need to be removed.

Another problem associated with the front is the direction the triangles are pointing in the symbology. According to convention, the triangles need to point in the direction of movement. Therefore, if the front is moving eastward the triangles point to the east.

Another feature that will be built in at a later date is identifying and labeling different types of atmospheric boundaries (e.g. cold front, warm front, occluded front, dry line, and trough). The labeling of low and high pressure centers would also be useful. To accomplish this, NIMBUSS needs to ingest more data, which it will obtain from National Weather Service’s Automated Surface Observing Stations (ASOS) among other sources. Perhaps, ingesting data from outside Missouri and processing the data into a raster will also aid in identifying atmospheric boundaries and pressure centers.

Yet another feature that could be built in is expanding the extent to continental United States or even the entire world and displaying the data on a website. This website would let people easily explore the weather systems of the world in a navigable and zoomable map.

**Conclusion**

In conclusion, NIMBUSS is a useful script that ingest and displays a map of Missouri Mesonet data that previously could only be seen in a text format. This allows for easier interpretation of weather conditions in Missouri. However, several features could be added to NIMBUSS to allow for better interpretation and representation of the atmosphere.