Term Report 1(revision) & 2

On the Development and Parallelization of an Air Temperature Spatial Interpolation and Prediction Program

By Erik LaBerge

December 14, 2010

Contents

Phase One 3
Air Temperature Spatial Interpolation Algorithms 3
Inverse Distance Weighting 3
Polynomial Regression 7
Air Temperature Prediction Algorithms 9
Polynomial Prediction 9
Unscheduled Programs 11
Display program 11
Phase Two 16
Air Temperature Prediction Programs 16
Artificial Neural Network 16
Unscheduled Programs 19
Prediction Graph 19

Phase One

Air Temperature Spatial Interpolation Algorithms

Inverse Distance Weighting

Pseudo code

This program takes in the data from the sensors, makes a grid large enough to cover all of the sensors, breaks up the grid into as much detail as the user wants, and estimates the temperature for each grid point using the Inverse Distance Weighting method.

The way it does this is:

- Read the sensor data from the file name input by the user
- Initialize grid
    - Get the largest and smallest latitude and longitude from all the sensor readings and make the grid x and y distances
    - To get the actual size of the grid multiply the x and y distance by the granularity multiplier input by the user and this is the final size of the grid
- Set grid
    - Loop through every point in the grid and call the Inverse Distance method
        - First get the sum weight of all the sensors for that point in the grid. This is based off the distance from the point. The farther away a sensor is, the less weight it has with estimating the temperature of the grid point.
        - Loop through all of the sensors
            - Get the weight of the current sensor
            - Adjust the temperature of the sensor to account for the elevation change from the sensor to the grid point
            - Multiply the weight of the sensor (sensor weight/sum weight) with the temperature of the sensor and add it to the grid temperature
- Print output
    - After all the grid points have been set my program outputs a file of temperatures to be input into the Display program I have worked on as well as a separate output for the temperature prediction program which needs information such as the latitude, longitude, and elevation of a grid point in addition to its temperature.

How to use

To find out what the argument options are run the program .jar file with no arguments. This will trigger the output directions for argument types, defaults and formatting. The output should look like:

```
-if    inputFileName          required
-df    displayFileName        optional
-pf    predictionFileName     optional
-g     granularity            optional
```

```
-p     pValue                  optional
-r     radius                  optional

help   for descriptions of each
exit   to exit program

-->
```

There are default settings for every argument except for the input file. The inputFileName should be the name of the file that holds the data from the sensors; it needs to be in the format of:

sensorName<Tab>latitude<Tab>elevation<Tab>longitude<Tab>airTemperature<\r\n>

The displayFileName and predictionFileName will rename the two output files to what you want. These files are formatted to work correctly in the Display and Prediction programs.

The higher the granularity the more detail there will be in the data output into the output files. This is the amount of estimations for every square unit of latitude and longitude. Note that depending on the curvature of the earth the latitude and longitude lines are closer or farther away from each other so this number may need to be larger or smaller than the default. See figure 5 and 6 for examples.

pValue is part of the inverse distance equation that you can change between 1 and 2 to smooth out the results. See figures below. 1 is the default value and only lets sensors have an effect on the area immediately around them where as 2 gives a much smoother look. It may seem better with the smoothing but when elevation is included it is more accurate without smoothing.

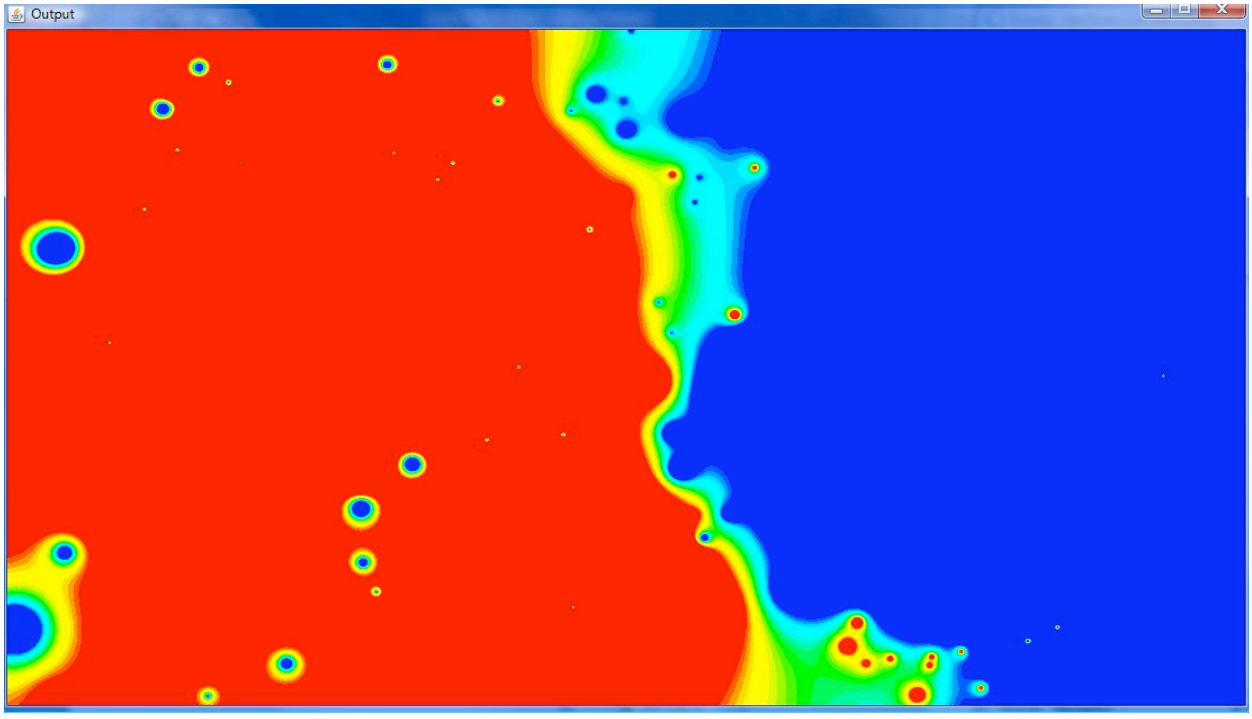

Figure 1: P = 1

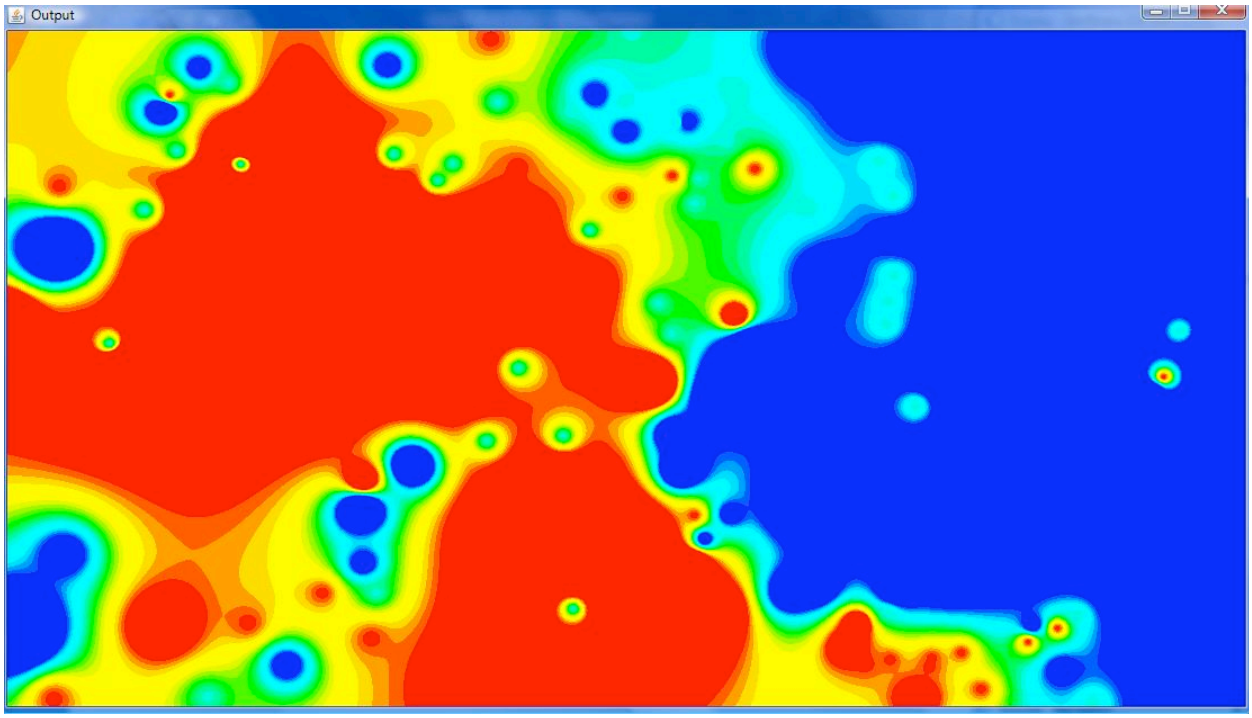

Figure 2: P = 2

The radius argument affects the radius around each sensor that it affects. The smaller the radius the smaller the affect any given sensor has on the area around it. In figure 1 and 2 the radius is maxed out so that there are no gaps where as Figure 3 shows an estimation with a radius of 0.2 and 4 has a radius of 1.5. You can see how figure 4 has no estimation data on the top right corner of the picture because there are no sensors nearby. The radius is a unit of latitude and longitude. So 1 would be about the distance between the latitude lines or longitude lines.

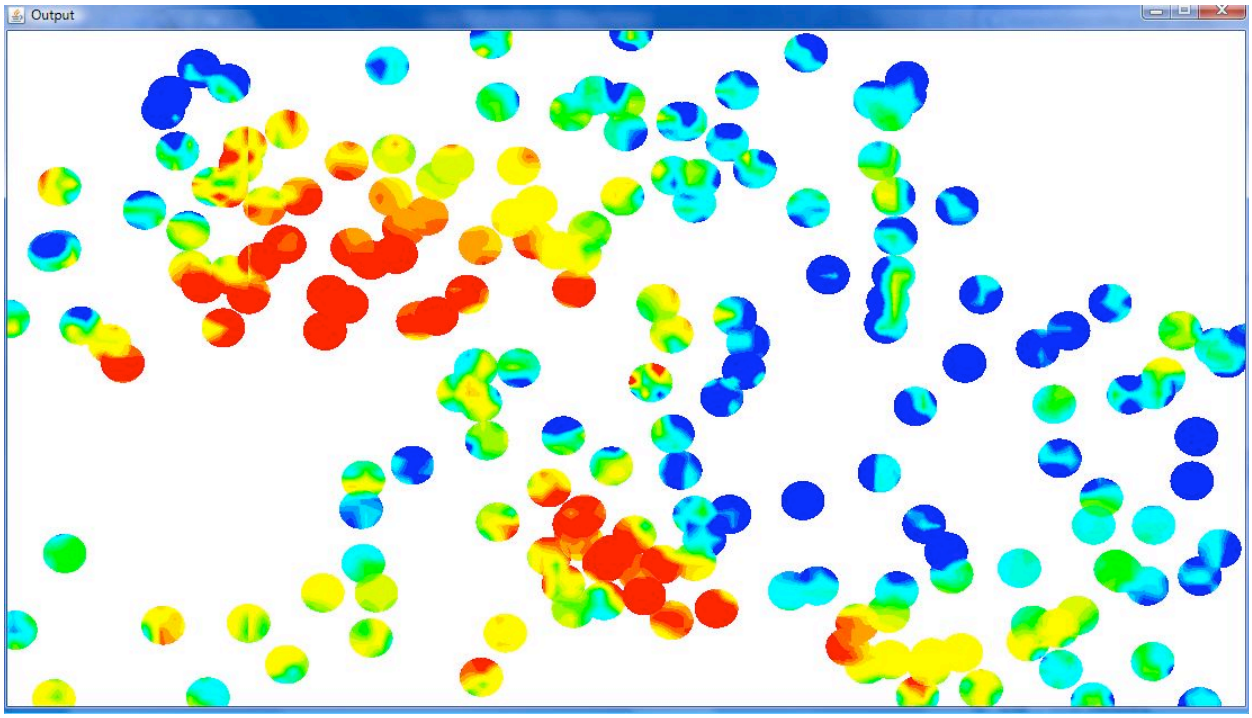

Figure 3: Radius = 0.2

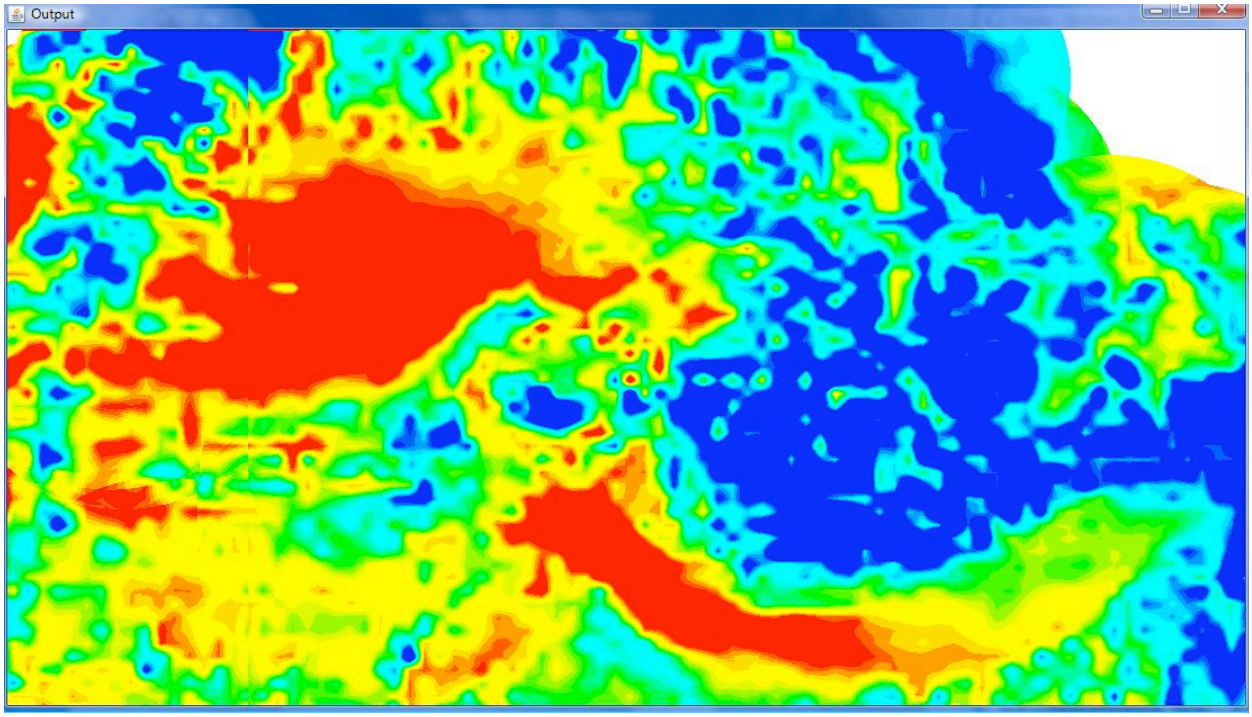

Figure 4: Radius = 1.5

Polynomial Regression

Pseudo code

This program takes in the data from the sensors, makes a grid large enough to cover all of the sensors, breaks up the grid into as much detail as the user wants, and estimates the temperature for each grid point using the Polynomial Regression method.

The way it does this is:

- Read the sensor data from the file name input by the user
- Initialize grid
    - Get the largest and smallest latitude and longitude from all the sensor readings and make the grid x and y distances
    - To get the actual size of the grid multiply the x and y distance by the granularity multiplier input by the user and this is the final size of the grid
- Set grid
    - Create the polynomial based on the sensor data
        - Essentially you make a polynomial for each sensor data point based on its latitude, longitude, elevation and temperature
        - You then solve those polynomials into one using Gaussian elimination
    - Loop through every point in the grid and call the Polynomial Regression method
        - All the polynomial method does is input the latitude, longitude, and elevation into the polynomial and it gives you the temperature for the point
- Print output
    - After all the grid points have been set my program outputs a file of temperatures to be input into the Display program I have worked on as well as a separate output for the temperature prediction program which needs information such as the latitude, longitude, and elevation of a grid point in addition to its temperature.

How to use

To find out what the argument options are run the program .jar file with no arguments. This will trigger the output directions for argument types, defaults and formatting. The output should look like:

```
-if    inputFileName          required
-of    outputFileName         optional
-pf    predictionFileName     optional
-g     granularity            optional

help   for descriptions of each
exit   to exit program

-->
```

There are default settings for every argument except for the input file. The inputFileName should be the name of the file that holds the data from the sensors; it needs to be in the format of:

sensorName<Tab>latitude<Tab>elevation<Tab>longitude<Tab>airTemperature<\r\n>

The displayFileName and predictionFileName will rename the two output files to what you want. These files are formatted to work correctly in the Display and Prediction programs.

The higher the granularity the more detail there will be in the data output into the output files. This is the amount of estimations for every square unit of latitude and longitude. Note that depending on the curvature of the earth the latitude and longitude lines are closer or farther away from each other so this number may need to be larger or smaller than the default. You can see the difference in figure 5 and 6.

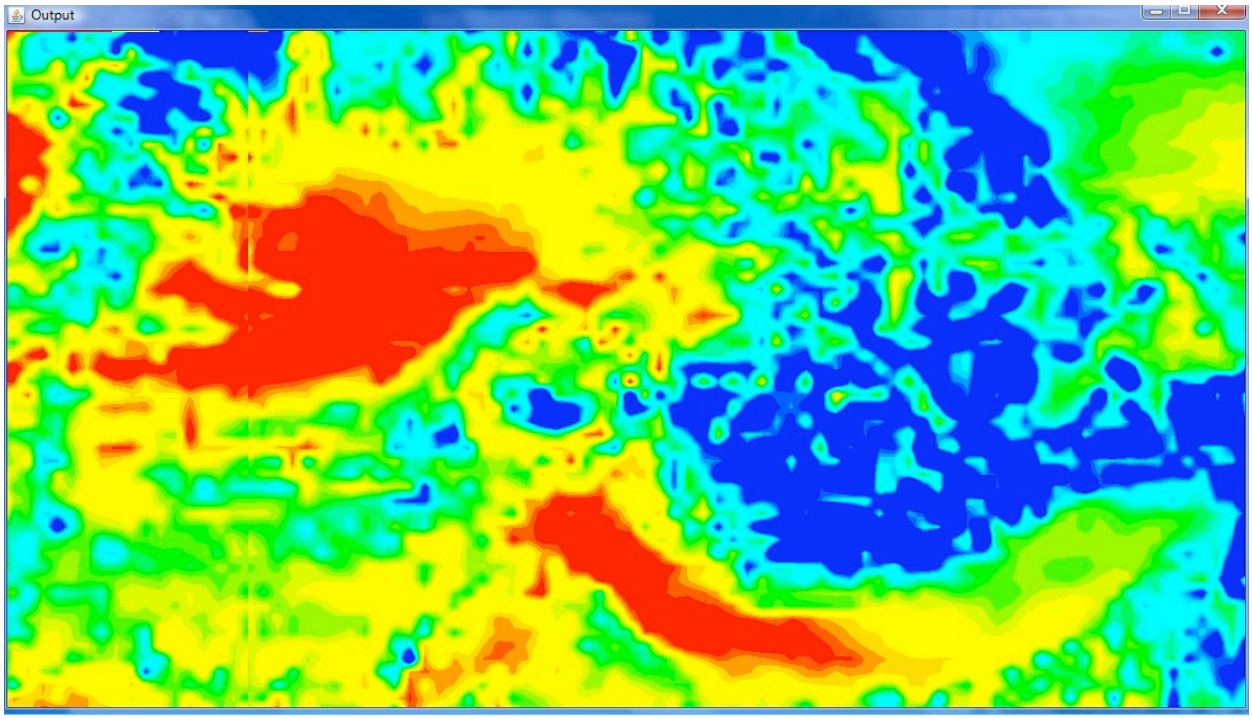

Figure 5: Granularity 100

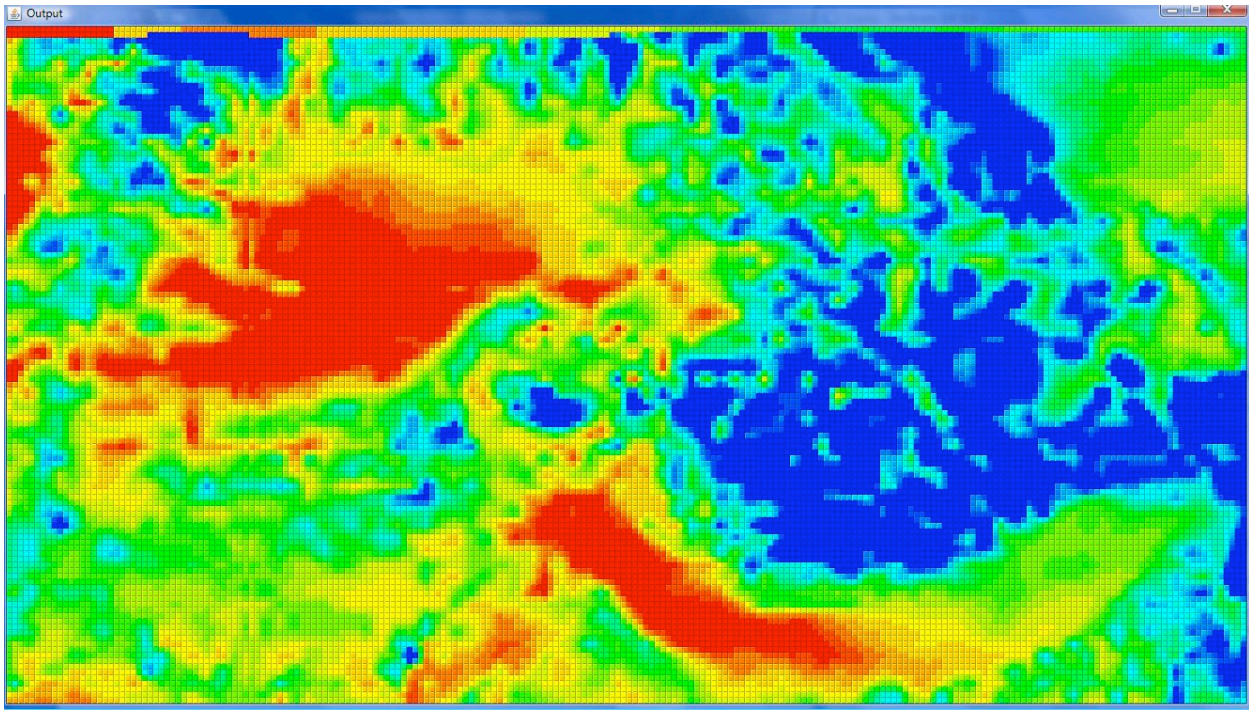

Figure 6: Granularity 20

Air Temperature Prediction Algorithms

Polynomial Prediction

Pseudo code

This program takes the output from the air temperature spatial interpolation programs and predicts the temperature from two hours after sunset until sunrise.

The way it does this is:

- Read the temperature estimation file that has a temperature, latitude, longitude, and elevation for every point on the grid
- Loop through every point on the grid
    - Predict the temperature
        - Get hours to predict based on the difference between two hours after sunset and sunrise
        - For every hour to be predicted use the equation T2 + (b * sqrt(i-2)) to estimate
            - T2 is the temperature 2 hours after sunset
            - i is the hour being estimated starting (i = 0 == two hours after sunset)
            - b is the equation ((Tp - T2) / (sqrt(n - 2)))
                - n is the number of hours between sunset and sunrise
                - Tp is the minimum temperature for that point

        - It is the multiplier times the temperature two hours after sunset plus the offset
            - Multiplier
                - to get the multiplier do the SLOPE function from Microsoft Excell: SLOPE(Y's, X's)
                - where the Y's are all the observed minimum tempuratures
                - and the X's are all the observed tempuratures taken 2 hours after sunset
                - Microsoft Excell SLOPE = ((sum((x-xMean) * (y-yMean)))/(sum((x-xMean)^2))
            - Offset
                - to get the offset do the Intercept function from Microsoft Excell: INTERCEPT(Y's, X's)
                - where the Y's are all the observed minimum tempuratures
                - and the X's are all the observed tempuratures taken 2 hours after sunset
                - Microsoft Excell INTERCEPT = AVERAGE(Y's) - SLOPE(Y's, X's) * AVERAGE(X's)
                - Microsoft Excell AVERAGE = you should know how to take an average
                - Microsoft Excell SLOPE = see Multiplier
- Print output
    - I print the output so that for every point there are all the temperatures that were estimated

How to use

To find out what the argument options are run the program .jar file with no arguments. This will trigger the output directions for argument types, defaults and formatting. The output should look like:

```
THIS PROGRAM IS MEANT TO RUN ON TEMPERATURES THAT WERE TAKEN 2 HOURS AFTER SUNSET
-if    inputFileName           required
-of    outputFileName          optional

help   for descriptions of each
exit   to exit program

-->
```

There are default settings for every argument except for the input file. The inputFileName should be the name of the file that holds the data from the Inverse Distance or Polynomial Regression programs; it needs to be in the format of:

airTemperature<Tab>latitude<Tab> longitude<Tab>elevation<\r\n>

The first line of the file has to have the height and width of the grid that was output into the file in the following format:

height<space>width<\r\n>

The outputFileName will rename the output file to what you want. This file is formatted to work correctly in the Display program.

Unscheduled Programs

Display program

Pseudo code

This program takes a file that has temperatures output from either the estimation or prediction programs listed above and displays them in a colorful visualization of a grid.

The way it does this is:

- Read the file with temperatures
- Make a window that shows all the data points while staying within a max height and width
- Take all the temperatures for each grid point (could be one to any amount of temperatures per grid point) and find the average from all of them.
- Then find the standard deviation based on the mean and divide it by a number input by the user for more or less granularity (this gives a range from almost 0 to 2 standard deviations from the mean for the color range of the output)
- Loop through every set of temperatures
    - Loop through every point on the display grid and color it based on the temperature of that grid point in respect to standard deviation and color range

How to use

To find out what the argument options are run the program .jar file with no arguments. This will trigger the output directions for argument types, defaults and formatting. The output should look like:

```
Arguments: fileName standDevDivisor
   fileName        (required, ex: C:/Users/gridOutput.txt || gridOutput.txt)
   standDevDivisor (optional, int, >0, def = 1)
Just type a 0 for optional arguments you don't want to use
```

There are default settings for every argument except for the input file. The fileName should be the name of the file that holds the data from the Inverse Distance, Polynomial Regression, or Prediction programs; it needs to be in the format of:

(optional as many times as you want)airTemperature<Tab>(end optional section)airTemperature<\r\n>

The first line of the file has to have the height and width of the grid that was output into the file in the following format:

height<space>width<\r\n>

The standardDevDevisor is a number that the standard deviation will be devided by. This makes the color range of the output larger or smaller from almost 0 to 2 times the standard deviation. You can see in figure 7 and 8 the difference it makes. In figure 7 the colors span the full 2 times the standard deviation range so since most of the temperatures are in the middle of the range the colors are not as vivid. While the same data in figure 8 has vivid colors because the color range only covers a smaller range of temperatures.

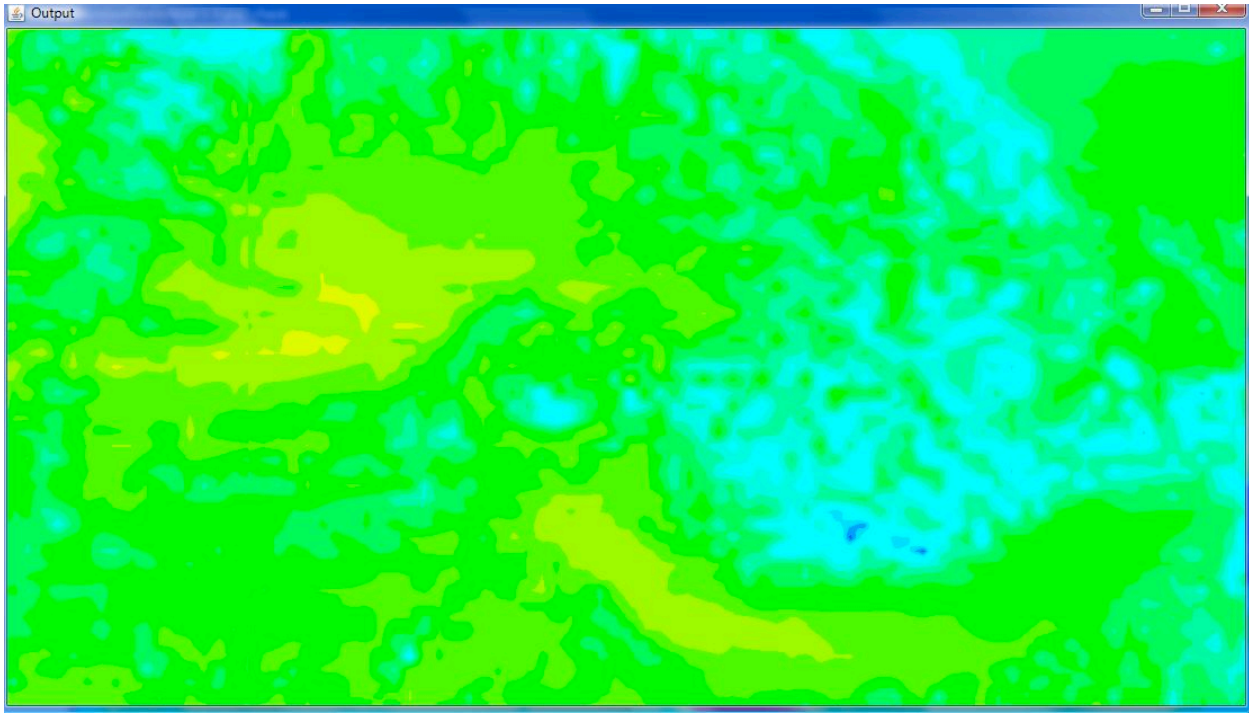

Figure 7: standardDevDevisor = 1

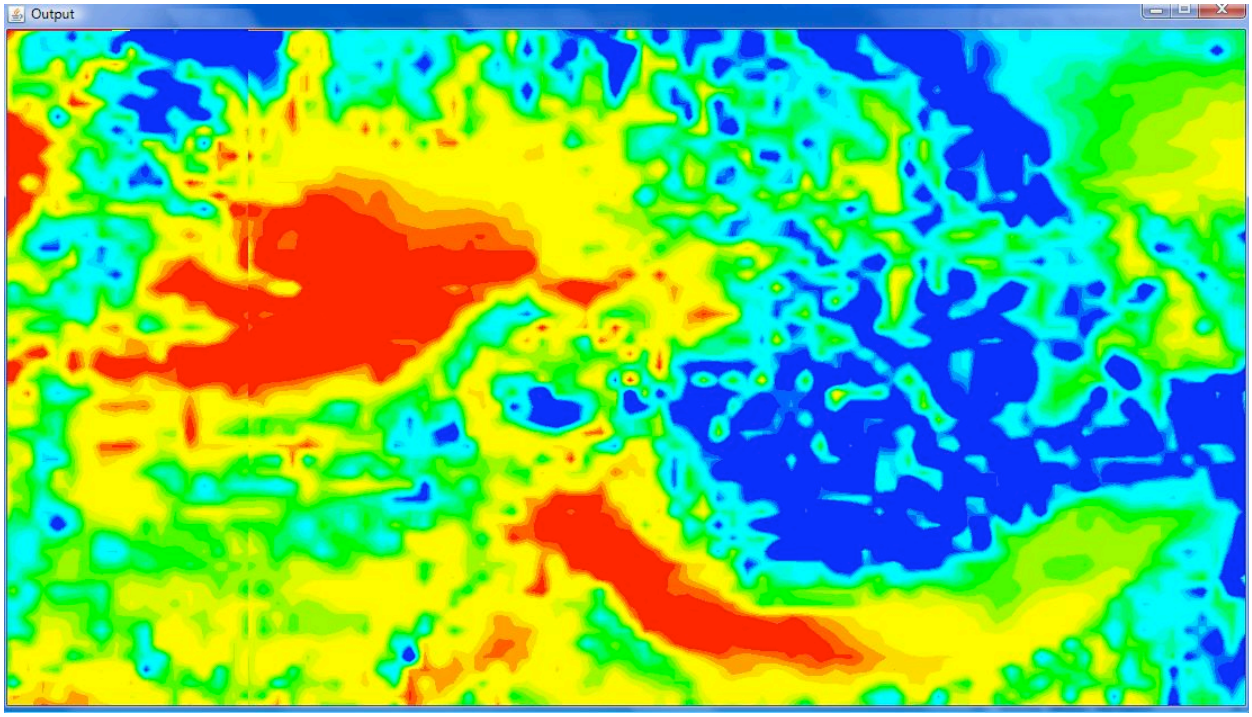

Figure 8: standardDevDevisor = 5

One of the features of the Display program is that it can display a single frame from one of the estimating programs or it can display a series of frames from the Prediction program which has a separate temperature for every grid point for every hour of nighttime. Figure 9, 10, and 11 show the progression you might see from the Prediction program output with three hours of night time. Notice how the color range is spread out over all the temperatures from all the frames.

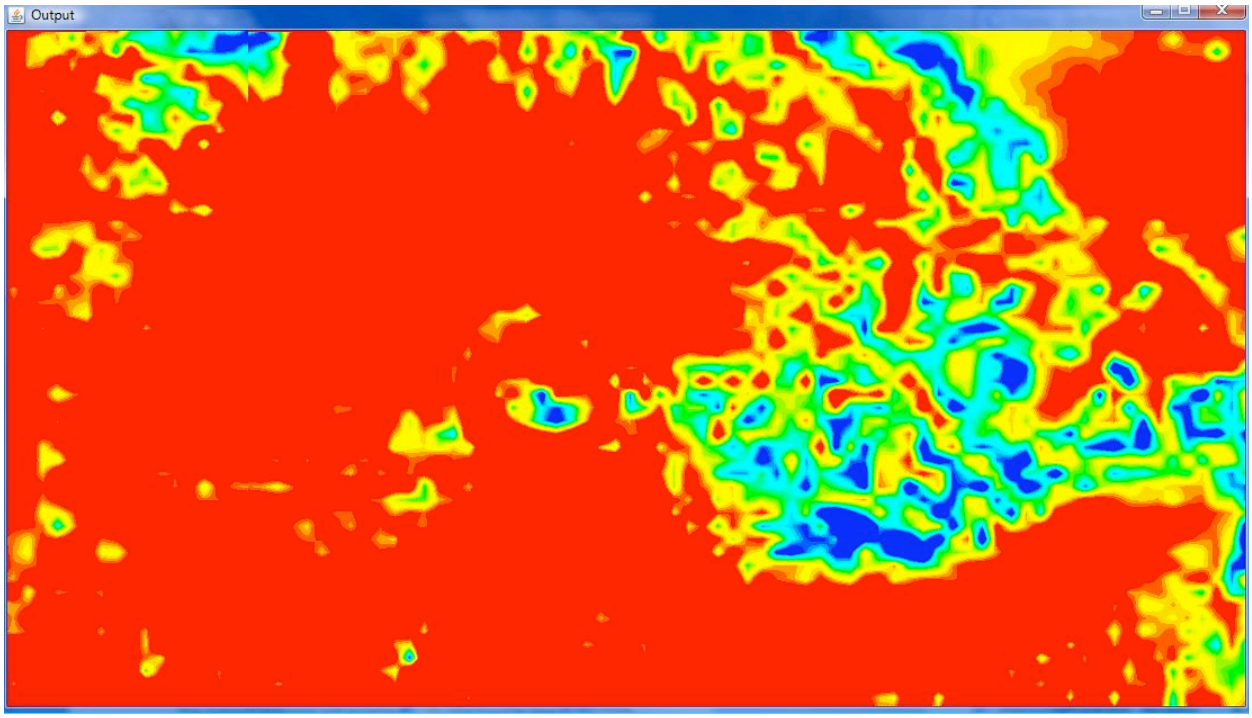

Figure 9: Frame 1/3

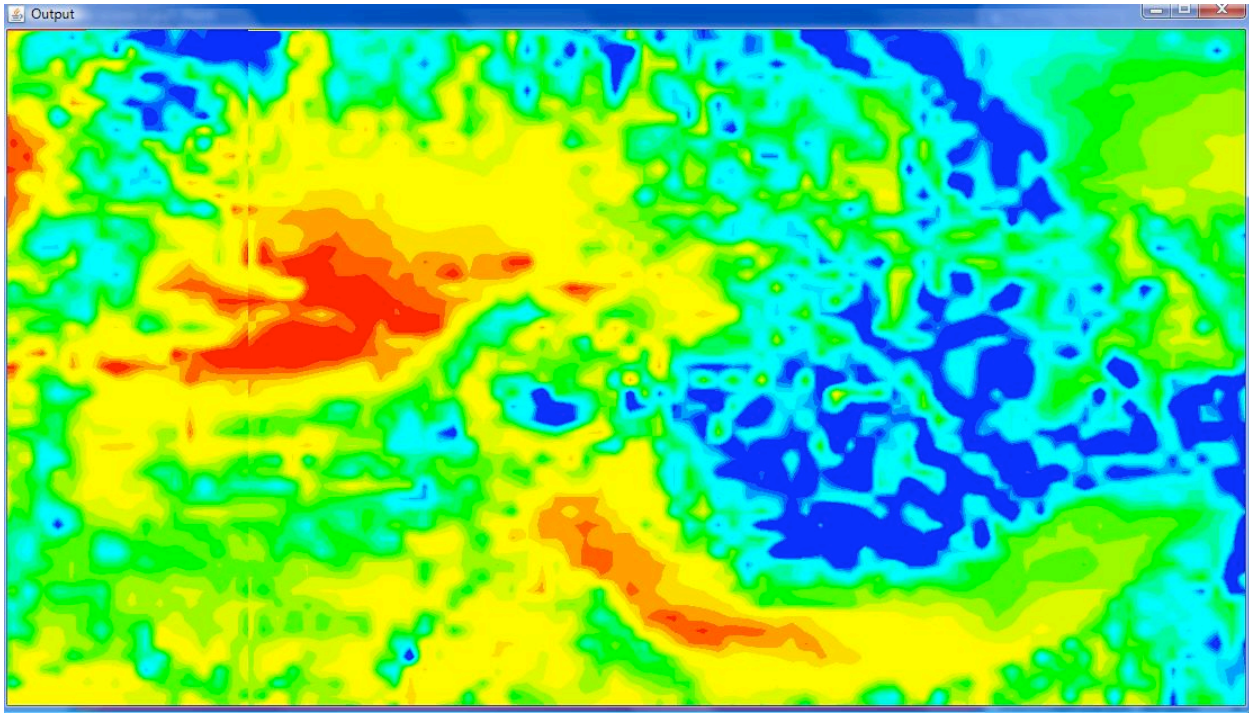

Figure 10: Frame 2/3

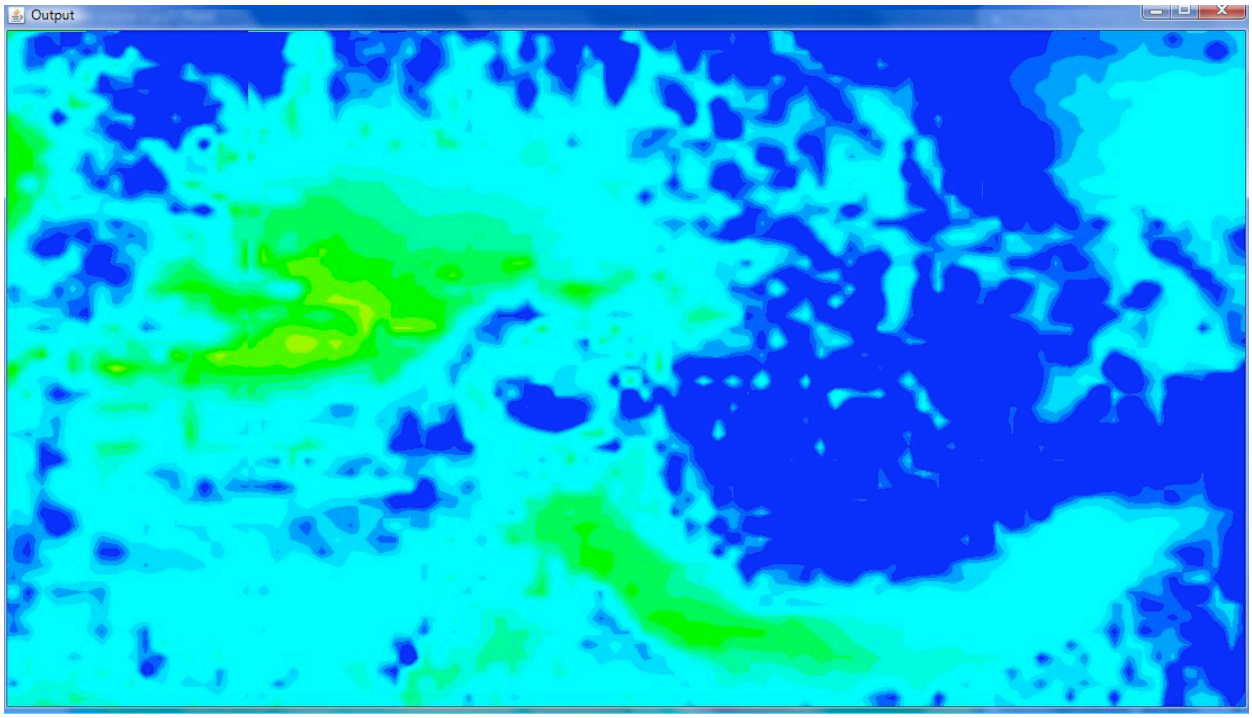

Figure 11: Frame 3/3

Phase Two

Air Temperature Prediction Programs

Artificial Neural Network

Pseudo Code

There are lots of code paths that this program could possibly go through but the main ones are as following.

Run program with no arguments:

- Looks in the current directory for a saved instance of the program
    - if found,
        - it loads it
    - otherwise
        - it initializes a new instance with no historical data to do predictions on
- Output input options as shown below and then shows a command line arrow for the next argument

```
Every time arguments:
false  if you have already initialized and do not want to reset        required
-if    inputFileName           required
-bf    backupFileName          optional
-of    outputFileName          optional

First time through arguments:
true   if it is the first time running for the location or want to reset the prediction variables      required
-if    inputFileName           required
-bf    backupFileName          optional
-of    outputFileName          optional
-fA    functionA               optional
-fB    functionB               optional
-fC    functionC               optional
-sh    startHour               optional
-eh    endHour                 optional
-pph   minuteIncrament         optional
-w     minWeight, maxWeight    optional

Help   for descriptions of each
exit   to exit program


-->
```

Initialization:

Run the program with at least the following arguments: true –if fileNameHere

- Load data on hard drive if available
    - Initialize program if not found
    - Load program if found
- Set all first-time-run properties if input as arguments
- Determine what type of file to read (SQL or regular .txt format)

- Put all data into data array
- Normalize all temperatures (between 0 and 1)
- Allocate array with enough prediction engines for every possible prediction need (from every 10 minutes to every other ten minutes within a range. Default range 5pm to 8am)
- Initialize weights of all engines by randomizing them
- Call the train() function on every engine with the correct starting and ending time temperatures and learning rate to change the weights so they are more accurate
- Undo normalization of the data
- Put the prediction programs instance for that sensor into an array to be used later

Notice that it does not do a prediction at this point and you can only initialize one sensor at a time.

Prediction:

Run the program with at least the following arguments: false –if fileNameHere

- Figure out what sensor the data is for
- Find the sensor in the array
- Call NNPRun() with the command line arguments and line of data from the file
- Check the prediction instance for this sensor is initialized with the minimum information needed
- Set the new data
- Normalize the data
- Predict temperature
    - Find the start time from the data
    - Find all the times to be predicted
    - For each start and end time prediction bring up the correct historical data and run the feedForward() with the inputs from the line in the data file
- Undo normalization
- Print predictions to file in the following format with time, prediction and then real data if available

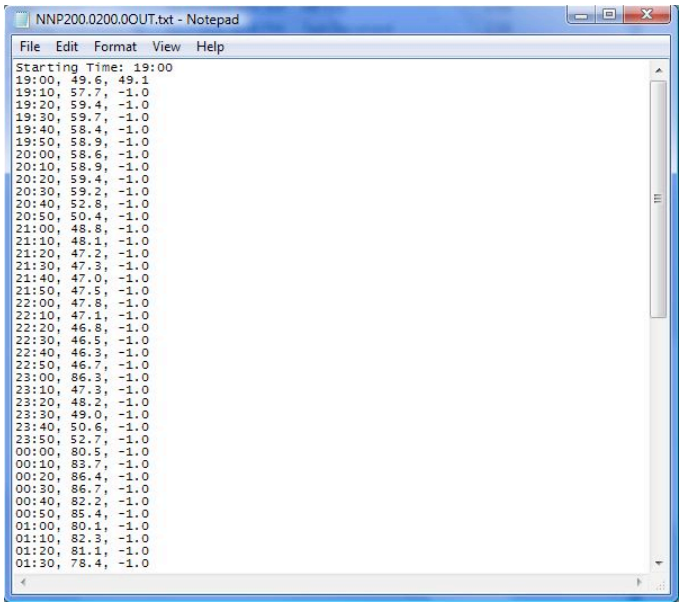

Figure 12: Prediction output

How to use

To add a sensor with historical data the program reads from either a SQL file or a file with the data from SQL formatted into a .txt file. The formats are listed below in figure 13 and 14.

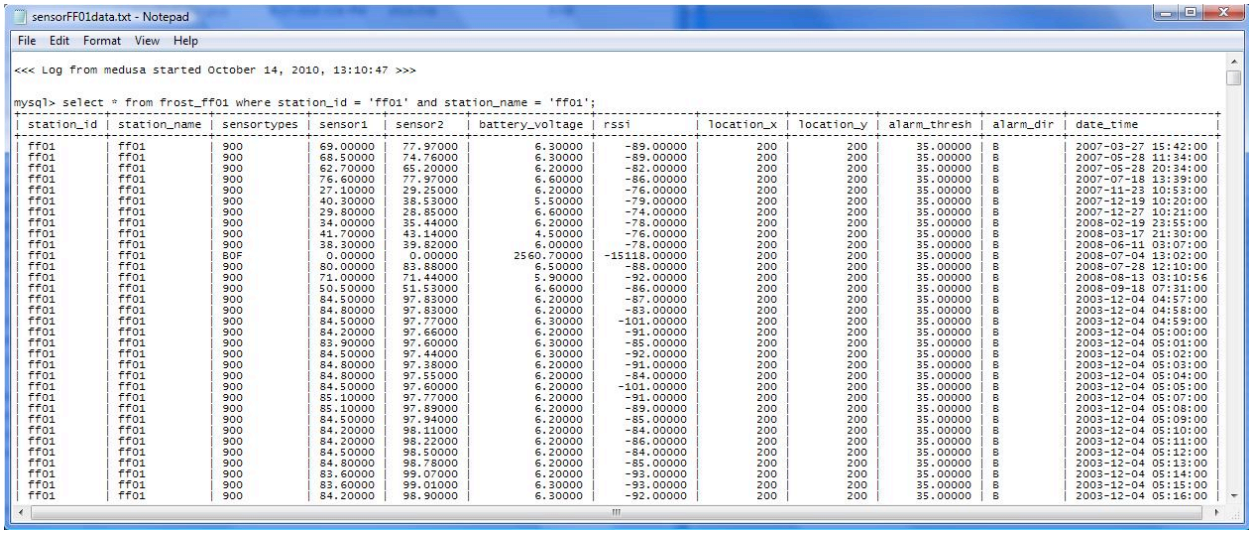

Figure 13: SQL formatted data

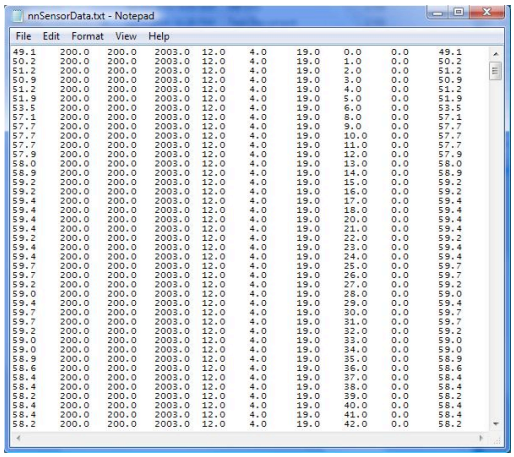

Figure 14: Data formatted for .txt file

The basic layout for the data in figure 14 is

Temperature<tab>Latitude<tab>Longitude<tab>Year<tab>Month<tab>Day<tab>Hour<tab>Minute<tab>Second<tab>Temperature

The temperature is repeated at the beginning and the end of the line.

To do predictions just enter a file with a line of data for every sensor that needs to have a prediction. It will output the predictions to different files for each prediction. The lines of data should be the same format as in figure 14.

From here the program is pretty straightforward. When you input a file that holds data to a new sensor that is not in the program yet it automatically detects the new data based on the latitude and longitude. It will load the data into an instance of the Neural Network code and store it in an array for later use.

Unscheduled Programs

Prediction Graph

Pseudo Code

This program takes the file output from the Artificial Neural Network program and displays it in a graph.

- Input the data from the file into an array
- Find the range of the data (minTemperature, maxTemperature, numberOfPredictions)
- Set up the size of the graph grid
- Use the data values to index into the correct block of the display screen and color it with either the prediction or real temperature color
- If there are multiple graphs in one file you can press enter to repeat this process but as of more recent versioning of the Artificial Neural Network program this is not going to be output

How to use

The file format has to be like the format in figure 12 where the start time is at the top of the file followed by lines of data for every prediction including prediction time, prediction, and actual temperature.

“Starting Time: “startPredictionTime<\r\n>
predictionTime”,”<space>prediction”,”<space>realTemperature<\r\n>

Run the program with no arguments to get directions on what you can put in as arguments.