David Cavaceci

Meteo 496

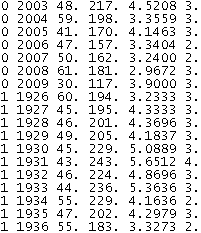
November 9, 2009

**Report: Fifth Milestone**

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Task** | **Code** |
| *V* | *Extend the program code to do graphical output of the results.* | *VBA in Excel* |

Graphing the results of the statistical analysis performed on the collected temperature observations involved Visual Basic for Applications (VBA) in Microsoft Excel. This language was used to read-in the results data from a text file created by Fortran after analysis was complete. The final Fortran output provided the sign, year, number of runs, sum of the runs, mean value, and the median value. The figure below provides a snapshot of the output file from the Fortran analyses.

**Final Fortran Output**



**Figure 1.** *From left to right, the sign indicator, year, number of runs, sum of the run lengths, the mean, and the median of the State College temperature data.*

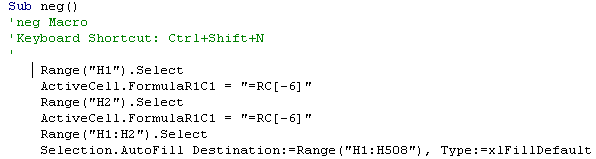
VBA in Excel is an excellent tool for developing or improving the coder’s programming ability for the same reason that Adobe Dreamweaver is great for all levels of website authors: The *Record Marco* ability allows the Excel user to record tasks to be repeated and apply the repetition to other data, thus saving time. In addition, however, the VBA code which corresponds to the tasks recorded by the macro is automatically logged into VBA as the recording is made. Thus, all you need to know in order to program in VBA is what you want and how to achieve it normally in Excel. This made the learning curve for VBA drop to practically nothing as compared to higher-power languages such as Perl or Fortran. Much more sophisticated task macros and programs, however, require a more thorough understanding of the VBA coding style, as many possibilities available in VBA are not available through the Excel interface alone. This was illustrated in exporting Excel charts to a new location as PNG images, which is not a possibility through Excel’s normal interface (save copying and pasting).

The first task for this milestone was getting the results into Excel. One of the main goals of the project is to attain automation, and thus start from scratch and end with State College historical temperature trends analyses without any manual tasks in between. In VBA for Excel, this means calling my *modules*, or *subroutines*, under the *This* Workbook block in VBA. By calling my subroutines here and selecting *Open* in the drop-down dialogue, the tasks I code will commence and automatically complete upon opening the workbook. Since I knew Excel could read-in data from external sources using the *Get External Data* button under the *Data* tab, I started by recording a macro that imported the Fortran results from its location.

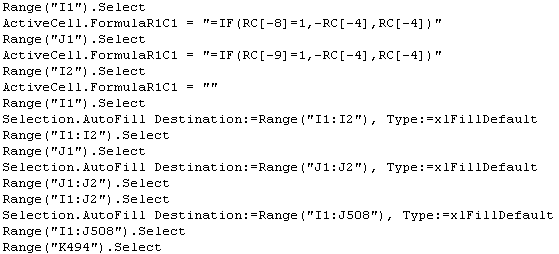


This block automatically located my data text file, read it, determined how the columns were delimited, and pasted the data accordingly into the first sheet starting with A1. The code is somewhat extensive, because many parameters can be tweaked upon getting external data, depending on what type of data is stored, how it is delimited, whether or not header lines are present, etc. The lines above were created automatically after I initiated the recording of a new macro, chose *From Text* in the *Get External Data* drop-down box under the *Data* tab, and followed the instructions given by the import wizard. However, now that I can see the VBA code that corresponds to the recorded macro, I can alter the import process from within VBA and without re-recording steps that are how I need them. For example, if instead of the output file displayed above I wanted to import data that was separated by commas, I could change the value of *.TextFileTabDelimiter* from True to False, and *.TextFileCommaDelimiter* from False to True. This would successfully import the data if the file located contained comma-delimited data, and an error message if some other delimiting method was used. Note also that the *filename (*right after *“TEXT;”*), *destination*, and even *column fixed widths* (of the columns to store the imported data) can be modified from the VBA editor.

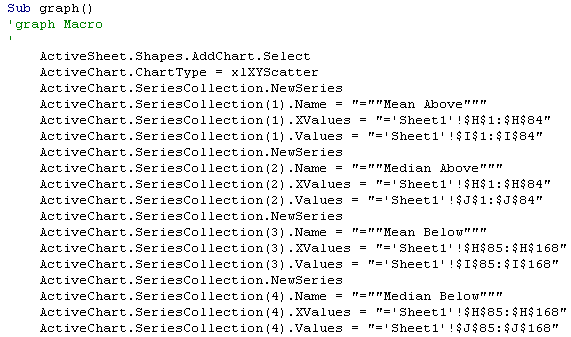
At this point, the analysis data produced by Fortran has been successfully loaded into Excel and is ready to be illustrated graphically. Expanding on Figure 1 above, the list of output data is organized by the type of run (above or below normal), the year, and the temperature category (max min and avg). In order to increase the value of each graph, I decided to hold the temperature category constant and graph the relationship between year and run length for both the means and medians. That is, the average above-normal run length, the average below-normal run length, the median above-normal run length, and the median below-normal run length were each plotted over all years for maximum temperature, minimum temperature, and average temperature. This required a conditional statement to be *Autofilled* through two new columns which would display the year’s mean and median either positively or negatively, depending on whether the run was above or below normal. This was achieved by recording a macro which wrote the condition for the mean of the first year, the condition for the median of the first year, and filled the test down through all years, temperature categories and indicator values. The resulting VBA code contained many more lines of code than I expected or understood, but was successful in manipulating the data as needed.



This first block creates a column that simply copies the year values from their original column to make transformations easier later in the project. Next, the mean column values were copied either directly or negatively into the column adjacent to the new year column, and the same was done for the median values. The test performed, starting with the first case, is *If the trend in question is* ***below normal****, then this cell equals the* ***negative*** *value of the mean (which is three columns left), and if not, this cell equals the* ***positive*** *value of the mean.* By *Autofilling* this conditional one cell to the right, I’m saying *Do the same test and this time, manipulate the* ***median*** *as needed.* By *Autofilling* these two cells of tests down through all rows of data, I’m saying *Manipulate the* ***rest*** *of the* ***means and medians*** *in my period of record, based on whether the trend is above or below the normal, as before.* In this manner of thinking, the value of *Autofill*, *Record Macro*, and *VBA* become crystal clear.

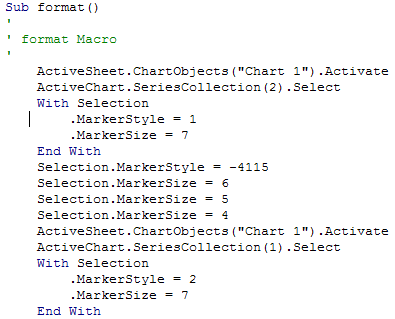


After this code block has run, the three working columns of data, the year, the mean, and the median, are ready to be plotted. In the same way as reading-in the data and manipulating it for sign association, graphing the results was a matter of pressing the record button and creating an individual scatter plot for each temperature category. Normally in Excel, a scatter plot is created by selecting the type of plot to insert and then choosing the data you want represented. An example portion of the graphing module is shown below, with a following explanation of key lines.

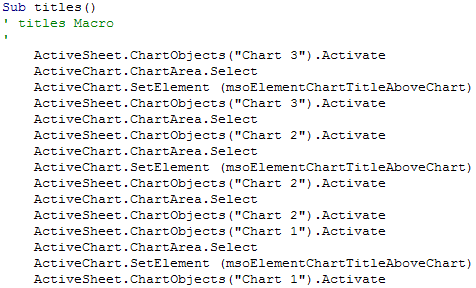


If we go line-by-line through the first segment in the *graph* subroutine, it’s easy to see how the plotting process is commanded in VBA. Starting with the first indented line, *In the active sheet, create a new chart of type scatter plot, and choose Select Data to add the first series. The series name is ‘Mean Above’ (corresponding to the mean value of run length above the normal), the x-coordinate values are found in the range ‘H1:H84’ (the years), and the y-coordinate values are found in the range ‘I1:I84’ (the means). After you’ve added those data for series one, add three more series containing the mean below-normal lengths and the median lengths above and below normal.* This produces a graph which displays the mean and median run lengths for maximum temperature above **and** below normal for each year. Graphing the same data for minimum and average temperatures was simply a matter of copying and pasting the above block, and modifying the *Series Name* and *Range* values accordingly.

Formatting the graph required individual manipulation as well. The original chart created by Excel with the subroutine above produces a cluttered display with data points that are too large, axes with too wide a range, and no title. Furthermore, changing the median run length data point marker to a horizontal line made interpretation easier, since the median trend is a stepwise function. These and other minor preferential format alterations are completed programmatically in the same way as before. By pressing the *Record Macro* button and making all of the changes I desired to **each** graph, the alterations could be completed automatically upon initial insertion of the chart by calling the *function* subroutine after the *graph* subroutine. The block below displays the VBA code that completes the formatting tasks.



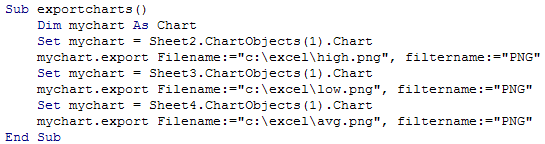
In English, the code above reads, *select the first chart in the active sheet (since there’s only one) and then select the second series of data. For that selection, change the marker style to style 1 and the size to 7.* The lines after *End With* are the result of me changing the style and size around to make sure I liked my first choice. It’s clear to see how we continue on formatting the rest of the data series by simply repeating the same block and modifying the sheet, chart object, and series as needed. Titles were added to the graphs in the same way, but with a different module. This was because I discovered that for some reason, the macro would not record both **inserting** a new chart title and the **content** I chose to input as the title. Only the “Chart Title” template would be made from the first macro. This problem was solved by simply creating a new macro just for the titles, though a solution could have also been reached by manually updating the *format* module to specify the titles. The *titles* subroutine code block is illustrated below.



Since the content of my chart titles was not recorded by my first macro, I decided to structure my *titles* macro so that each title would first be created, and then each would be customized. This is seen in the block above, where *msoElementChartTitleAboveChart* is seen after selecting each of the three graphs. This corresponds to a centered title above each plot.

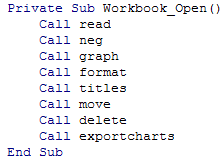
Two additional minor modules were created for very specific purposes that need not have their code block explained. The first issue was that the first chart created had three extra series each time the workbook was opened and run. One module was developed in response to this, and it simply records the deletion of these elements. Additionally, to allow for further modification to the graphs and easier visualization, each chart was moved to separate worksheets. This process was also quite simple and involved merely pressing record, cutting and copying each graph to their new location, and pressing stop. These were the final steps in creating sufficient graphs for the State College temperature trends since 1926.

Since this project is all about automation, each of the three graphs needed to be saved as images somewhere on the machine, to allow for access by other programs. This created a larger problem than anticipated, because Excel (in the standard interface) doesn’t support such exporting. Luckily, VBA does facilitate such a task, and saving each of the charts automatically was a breeze with the *Export* command.

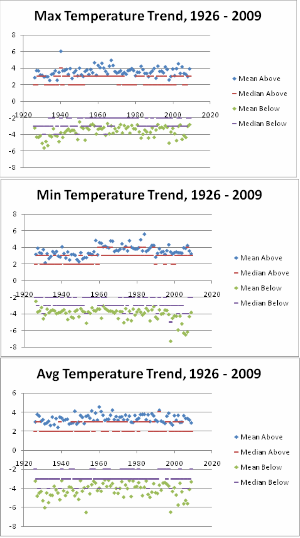


Ironically, the closing subroutine in this milestone was the only one that involved strictly programmatic control, as is indicated with a lack of commented lines and indication of the shortcut command in the block above. We start my declaring the variable *mychart* to be of type *Chart*. Next, *mychart* is passed the value of the chart of concern, the max temperature chart in this case. Finally, the *mychart.export* command saves the graph as an image at the location defined after *Filename:=* and in the PNG format, as is specified after *filtername:=*. For the remaining two charts, we simply reassign *mychart* and repeat the export command, altering only the file name. Upon exportation, the graphing of statistical results is complete. Below, the master program flow control and exported PNG images are displayed.

**Master Flow Control**



**Figure 2.** *This is the block which is found under ‘This Workbook’ in VBA. It calls each of the detailed subroutines in order, resulting in the PNG images below.*



Full program code is available at www.personal.psu.edu/dac5039/496/MS\_V