Progress on Gwx: the G Regression and Model-building Program
Rewritten with Open-Source, Cross-platform Tools.

Clopper Almon

At the 20th Inforum World Conference, I announced (in absentia) that I was undertaking to rewrite the G7 regression and model-building system with open-source, cross-platform tools, namely with wxWidgets and the GNU C++ compiler. A year later, I am beginning to get results, but the program is still several months from reaching a useful stage. In this report, I want to explain again my motivation for undertaking this work and to give a report on progress to date.

Motivation

Initially, my motivation was simply fear. Gradually, however, other, nobler motives have emerged.

The fear motive arises from the fact that G7 is built with Borland Builder 6 and the Borland C++ compiler. Borland went out of business in 2003, and Builder was taken over first by CodeGear and then by Embarcadero Software, which offers a current version for about $2000. Before it went out of business, Borland produced a version of Builder that worked on a small number of Linux distributions but does not work on current distributions. Builder 6 did not work on Windows Vista but worked again on Windows 7. (Corel PhotoPaint also stopped working under Vista.) I do not know whether or not it works on Windows 8, with which I have had only enough experience to know that I don't like it. Recently, Microsoft has announced that it is terminating support for Windows XP; machines running XP in the University of Maryland are being converted to Windows 7. I mention these facts to emphasize (1) that there is a very close connection between Builder and Builder-like programs and the operating system and (2) that Microsoft has not the least interest in maintaining support for Builder, a program which for two years or so was far ahead of anything Microsoft had to offer.

On the positive side, the reasons for wanting to use cross-platform tools are increasing. The current versions of the Apple computers are doing very well commercially, in part because they are running on a solid, Unix-based operating system. Linux, while a minor player in the desk-top and lap-top market, is the major player in the hand-held devices, where it accounts for about two-thirds of the new sales, with Apple's similar Unix-based systems being most of the rest of the market. Consequently, a young programmer will definitely want to learn to deal with Unix and its look-alike Linux. I personally use Ubuntu Linux but with the Gnome desktop, which is more traditional than the Unity desktop currently the Ubuntu default. I took the time to become fairly proficient with Unity – which aims to be nearly the same on laptops and hand-held devices, hence the name – but in the end, I found it less efficient than the traditional Gnome desktop. (One of the nice things about Linux is that you have a choice of a number of different desktops; Apple and Windows remind one of Henry Ford's famous dictum: “You can have your Ford in any color you want so long as it is black.”)

I find Ubuntu Linux a very stable operating system with a wide variety of high-quality, free, open-source software available with a few mouse clicks. The LibreOffice suite, which installs with Ubuntu, includes Writer (a word-processor superior to MS Word), Calc, Impress, and Base closely paralleling Excel, PowerPoint, and Approach. I also use Gimp for image processing (a free alternative to PhotoShop), MuseScore for writing musical notation (a free alternative to Finale), the OpenShot video editor (a superior alternative to MS MovieMaker), the Gramps genealogical program, the Audacity audio editor, Google Earth and various utilities for Internet access, scanning, OCR conversion, and the full set of Unix command-line
software. Code::Blocks with wxSmith and the GNU C++ compiler is fully the equivalent of Borland Builder 6 for writing programs – and it is cross-platform. All of that is free and most of it open-source. In fact, everything I ever want is there – except a good time-series regression and model-building program like G7. I would like for G7 to join this distinguished company.

As I dug into the project, I found that there were other reasons for doing it. The code for G7 has become very complicated and difficult to read. Any programmer new to the code who was expected to step into it and make some correction or add some feature would have a very difficult time. The original code was much simpler and clearer. The documentation of how it was supposed to work was, from the outset, limited to comments in the code. But many changes have been made in the code, usually with the original code left as a comment and the new code marked with the changer's initials and date but with no explanation of why the change was made. In the work that I have done so far, I have been able to simply copy G7 code in only three instances: (1) writing data to a standard G data bank, (2) the inversion part of the regression program, and (3) the GDate class. Elsewhere, the G7 code has become hopelessly – and I often feel – needlessly complex. I want to produce code which is eminently readable by humans and is documented not only with comments in the code but with a tutorial description of what it is doing. To date, this description is a document of over 150 pages. While I intend to make the whole code open-source, I also intend to keep my own copy and to keep it intelligible. I want it – in conjunction with the tutorials – to be easy and clear reading for anyone who understands the basics of C++ and the algorithms being used. The tutorials should serve as an introduction for anyone who later needs to maintain and update the code.

Overview of Progress to Date

At last year's Inforum conference, I was able to report getting the basic G7 mechanism set up with a command drop-down box, a results window, and two buttons, one for picking a starting directory and one to draw and save the Giotto figure – a red circle and an octagon around it. Since then have been added the following commands:

- **data** for adding time series one at a time to the workspace bank
- **matdata** for adding a number of time series arranged in a matrix to the workspace bank
- **type** for displaying a time series in the workspace numerically on the results screen
- **f** for forming a new variable in the workspace bank by arithmetic operations with existing series. The log() and exp() functions are also recognized.
- **r** for performing linear regression of one variable on a number of others
- **gr** for graphing one or more variables.
- **title** to provide a title for a regression or graph
- **subtitle** to provide a subtitle for a regression or graph
- **dates** tdates, fdates, rdates, and gdates, to provide date ranges for the type, f, r, and gr commands
- **add** to execute any of the above command from a file specified following the add command.
- **quit** to exit the program.

While that is a fairly short list, users of G7 will recognize that these commands cover much of the basic
operation of the program. The main missing pieces are:

- enrichment of the graph command with vertical range control, saving and various options.
- other bank commands – bank, cbk, hbk, and vam
- extension of the r command with con, sma, showt, and the saving commands sav and cat.
- tabular display of items in a vam file
- matrix algebra on matrices and vectors in a vam file
- more functions for use in f commands.
- specialized regression commands recur and sur

I intend to work on these more or less in the order given.

In building models, G7 works with Build for macro models and IdBuild for multisectoral models. Both Build and IdBuild are pure command-line C++ programs and should convert to the cross-platform GNU compiler rather easily. I do not anticipate any major problems there.

Much recent work on G7 has concerned interaction with a spreadsheet program. Unfortunately, Excel was chosen instead of Calc so that none of that work can be carried over easily to the open-source environment, and I do not have on my personal agenda any plans for extension of Gwx in that direction.

**Illustrations of Gwx at Work**

I put some made-up data in a file called integers.dat and gave Gwx the command add integers.dat. Here is what I got in the Results pane, with the input set in bold italics

```plaintext
# Example with date on first line
data integers 1970
  0 1 2 3 4 5 6 7 8 9
  10 11 12 13 14 15 16 17 18 19
  20 21 22;
tdates 1970 1990
tdateA input 1970
tdateB input 1990
Year 1970 Frequency 1 Period 1 Day 0.
Year 1990 Frequency 1 Period 1 Day 0.
type integers
Here is integers:
  1970   0.0000   1.0000   2.0000   3.0000   4.0000
  1975   5.0000   6.0000   7.0000   8.0000   9.0000
  1980  10.0000  11.0000  12.0000  13.0000  14.0000
  1985  15.0000  16.0000  17.0000  18.0000  19.0000
  1990  20.0000
```
# Example without date on the first line:
data threes
1970 3 6 9 12 15
1975 18 21 24 27 30
1980 33 36 39 42 45;
type threes
Here is threes:
1970 3.0000 6.0000 9.0000 12.0000 15.0000
1975 18.0000 21.0000 24.0000 27.0000 30.0000
1980 33.0000 36.0000 39.0000 42.0000 45.0000
1985 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
1990 -0.0000

# Example with date and all data on the first line
type fives
Here is fives:
1970 0.0000 -5.0000 10.0000 -15.0000 20.0000
1975 -25.0000 30.0000 -35.0000 40.0000 -45.0000
1980 50.0000 -0.0000 -0.0000 -0.0000 -0.0000
1985 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
1990 -0.0000

Handling of dates has been improved. Initially, G used floating point numbers to store dates. Annual
The monthly dates looked rather strange, and the system could not be extended to daily dates, which become
important in financial data. So a new system called GDate was devised and is available in G7 and Gwx,
where it is implemented as a C++ structure, a struct. Most of the coding here is new and simpler than in G7
for Windows.

A GDate can be specified by the user in any of the just-mentioned ways and also as shown by the
following examples:

2010q1 the first quarter of 2010
2010m01 January of 2010
2010m07d04 2010 July 4

Note that the month and day must always be two characters.

When Gwx starts, a daily time series called timed is placed in the workspace bank. We can use it to illustrate
daily dates and the type command with daily data.

tdates 1970m01d01 1971m12d31
tdateA input 1970m01d01
tdateB input 1971m12d31
Year 1970 Frequency 13 Period 1 Day 1.
Year 1971 Frequency 13 Period 12 Day 31.
ty timed
Here is timed:

<table>
<thead>
<tr>
<th>Year</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>1.000</td>
<td>2.000</td>
<td>3.000</td>
<td>4.000</td>
<td>5.000</td>
<td>6.000</td>
<td>7.000</td>
<td>8.000</td>
<td>9.000</td>
<td>10.000</td>
<td>11.000</td>
<td>12.000</td>
<td>13.000</td>
<td>14.000</td>
<td>15.000</td>
<td>16.000</td>
<td>17.000</td>
<td>18.000</td>
<td>19.000</td>
<td>20.000</td>
<td>21.000</td>
<td>22.000</td>
<td>23.000</td>
<td>24.000</td>
<td>25.000</td>
<td>26.000</td>
<td>27.000</td>
<td>28.000</td>
<td>29.000</td>
<td>30.000</td>
<td>31.000</td>
</tr>
<tr>
<td>1970</td>
<td>2.000</td>
<td>3.000</td>
<td>4.000</td>
<td>5.000</td>
<td>6.000</td>
<td>7.000</td>
<td>8.000</td>
<td>9.000</td>
<td>10.000</td>
<td>11.000</td>
<td>12.000</td>
<td>13.000</td>
<td>14.000</td>
<td>15.000</td>
<td>16.000</td>
<td>17.000</td>
<td>18.000</td>
<td>19.000</td>
<td>20.000</td>
<td>21.000</td>
<td>22.000</td>
<td>23.000</td>
<td>24.000</td>
<td>25.000</td>
<td>26.000</td>
<td>27.000</td>
<td>28.000</td>
<td>29.000</td>
<td>30.000</td>
<td>31.000</td>
<td></td>
</tr>
</tbody>
</table>

Note that the output looks a bit like a calendar, except that every month begins on a Sunday. Note also that the program knows that January has 31 days and that February of 1970 had only 28 days. It also knows the full rule for detecting a leap year. Daily data is given a frequency of 13.

Now here is real, quarterly data on coal usage introduced with the `matdata` command:

```plaintext
matdata 2009q1 21
CoalElec CoalCoke CoalOthInd CoalTotal
# The 21 in the first line means to skip the first 21 spaces on each data line. A line beginning with a # is just a comment.
# U.S. Coal Consumption
# (Thousand Short Tons)
# Year and Electric Coke Other Total
# Quarter Power Plants Industry
#
# #2009
# January - March 236842 4398 12075 254383
# April - June 216502 3402 10542 231110
# July - September 244445 3450 11107 259621
# October - December 235838 4076 11590 252363
#
# #2010
# January - March 246445 4857 12600 264939
# April - June 229469 5353 11914 247344
# July - September 267943 5491 12284 286361
# October - December 231195 5391 12490 249870
#
# #2011
# January - March 234847 5188 12489 253541
# April - June 223540 5392 11036 248611
# July - September 261534 5407 11168 278638
# October - December 208637 5447 11543 226233
```
We can illustrate the \texttt{f} command with these lines:
\begin{verbatim}
f CoalSum = CoalElec + CoalCoke + CoalOthInd
f CoalResid = CoalTotal - CoalSum
f ShareElec = 100.*CoalElec/(CoalElec + CoalCoke + CoalOthInd + CoalResid)
type ShareElec
\end{verbatim}

with this result

Here is \texttt{ShareElec}:
\begin{verbatim}
2009q1  93.104  93.679  94.155  93.452
2010q1  93.020  92.773  93.568  92.526
2011q1  92.627  92.905  93.862  92.222
\end{verbatim}

And we can now illustrate regression with a somewhat nonsensical example:
\begin{verbatim}
title Coal Used for Coke
r CoalCoke = CoalResid,CoalOthInd
\end{verbatim}

The result looks like this:
\begin{verbatim}
:                               Coal Used for Coke
SEE =      567.6 RSQ =    0.870 RHO =    0.735 DW =    0.531
Variable name              RegCoef  Mexval   Elas  NorRes     Means
1 intercept             -162.19922     0.3  -0.03    7.69      1.00
2 CoalResid               -1.86559    12.2  -0.29    4.31    757.25
3 CoalOthInd               0.54496   107.5   1.33    1.00  11736.50
\end{verbatim}

The graph of the fit is then obtained by
\begin{verbatim}
gr predic, depvar
\end{verbatim}

One tiny point should be noted: the comma between the two variable names. It is there because Gwx allows expressions in the list of items to be graphed, not just names of variables already in the data banks. Since spaces are legal in expressions, the comma is necessary to delimit the items to be graphed.

Here is the graph.
It is basically working, but it still lacks labeling of the vertical axis and the labeling of the horizontal presently works only with fairly short series of quarterly data. Moreover, the graph was not properly saved as a .png file but caught with a screen shot program. With a few days work, however, it should be a pretty competent graphing program.

Chinese and Russian colleagues may be pleased by these two graphs which were obtained simply by putting the Chinese and Russian on the title command.

This is somewhat less than I had hoped to have ready to present, but it at least represents progress on basics and shows that the project is still alive. Moreover, there is the 150+ page document Tutorials on the Construction of Gwx, a Cross-Platform Version of G7 with 12 tutorials explaining the program thus far.