The 2016 Gwx Report
Clopper Almon

Inforum regulars will recall that for about five years now I have been working on rewriting the G7 regression and model-building program with open-source, cross-platfrom tools, specifically with wxWidgets for creating the graphical user interface. I have not only been writing the program but recording the process of writing in the form of what have come to be called “tutorials”. The work is now sufficiently advanced that I have made these tutorials available in a print-on-demand book, *The Gwx Story*, which is available from amazon.com and its affiliates worldwide.

My work has been entirely under Ubuntu Linux, but the C++ code and the wxWidgets should work also on Windows and OSX (Mac) operating systems. I have been using the Code::Blocks inter-active development environment (IDE), which is also widely used on Windows and OSX. I recently had to use Windows 8.2 to get a feature of G7 not yet available in Gwx; the contrast between Windows and Linux was striking. Windows not only claimed a lot of time for its own housekeeping but put itself ahead of me in priority. The whole interface was complicated, obscure and time-consuming. If you are not absolutely prohibited from doing so, at least give your computer a dual boot with Linux and begin to learn the operating system of the future. It is already well ahead of Windows on new systems because all the Android devices use Linux.

Last year in Bangkok, I reported on work up as far as creating a VAM file for handling vectors and matrices. Later, Leonardo Ghezzi remarked that he would like to use Gwx for building his macro model of Italy. This comment caused me to change the direction of my work to round out the Gwx facilities for building macro models, especially quarterly ones. Already available were the abilities to read from and create databanks compatible with G7, to read in data from a text file, to create new series by algebraic formulas and by the use of logarithm and exponential functions, to perform OLS regression, and to draw and save (as .png files) graphs including the graphs of regression fits. To this list there have now been added:

the *save* command to save the results of regression in a form for use by Build or IdBuild, the programs that put together equations and identities to form macro models and Interdyme multisectoral models.

the *catch* command to catch the results of what goes onto the screen in files to be used in documents.

the *@atoq()* function to convert an annual series to a quarterly one.

the *@atoqi()* function to perform this conversion with an indicator or guide series.

the *@mtoq()* function to convert a monthly series to a quarterly one.

the @cum() function to create a stock from a flow, e.g. to create a capital stock from investment.

the *con* command to impose “softly” linear constraints on the regression coefficients.

the *sma* command to impose “softly” polynomial constraints on a distributed lag.

the *gname* command to control the name of the .png file into which the next graph will be written.

With these techniques available, estimating the equations of a macro model should now be within the capabilities of Gwx. I have not yet converted the Build program to run under Linux, but since it is pure C++ with no GUI to worry with, I do not expect major problems.

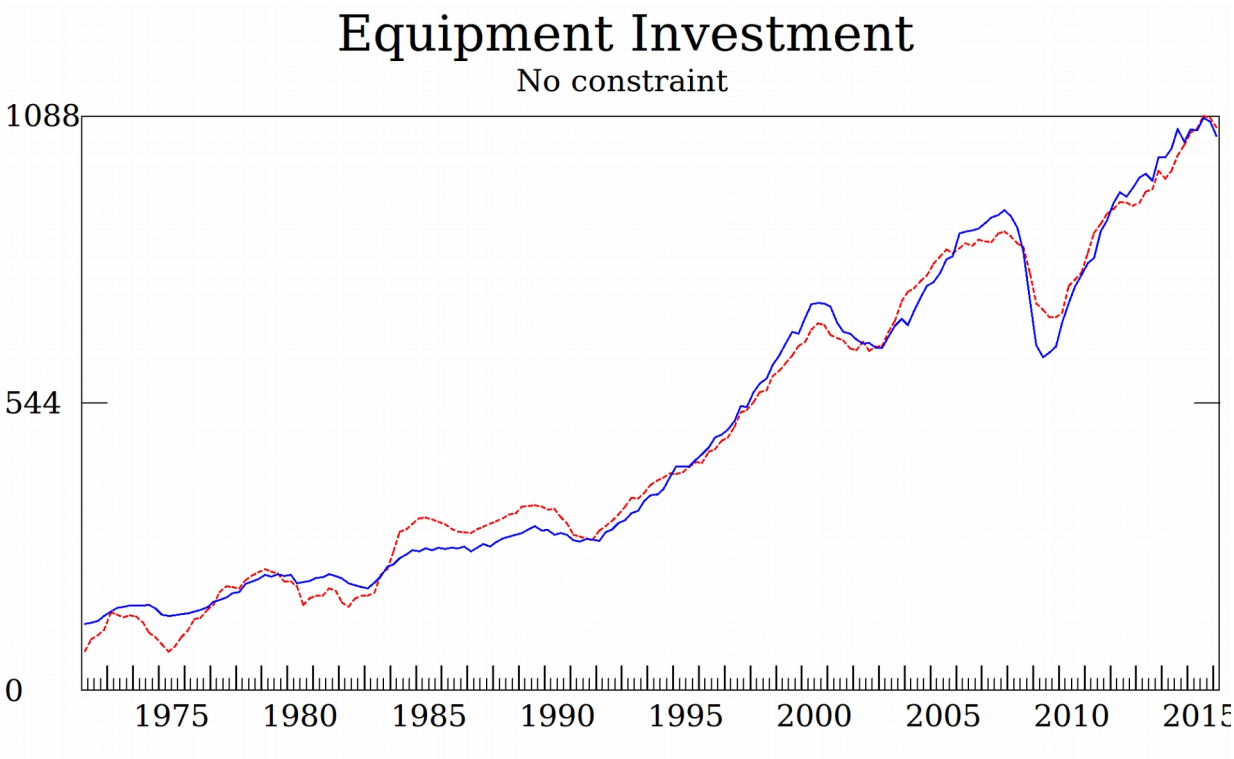
An Example

To illustrate some of the new techniques, we may take the example of developing an equation for equipment investment, vfnreR in the Quip bank naming scheme (read the name as: inVestment, Non-Residential, Equipment, Real). Current U.S. national account databanks begin in 1969. The commands have been put into a file called vfnreR.reg. To start with, it reads as follows:

```
catch vfnreR.cat
save vfnreR.sav
ti Equipment Investment and Wearout
bank quip161 # Quarterly Income and Product 1969q1 to 2016q1
fdates 1969q1 2016q1 # formula calculation dates
tdates 1969q1 2016q1 # dates for displaying data by the “type” command
rdates 1972q1 2016q1 # regression dates
gdates 1972q1 2016q1 # graph dates
f d = gdpR – gdpR[1] # first difference of real GPP
f one = 1.0;
f ub05 = @cum(ub05,one,0.05) # unit bucket with 5% spill
f capstock = @cum(vstock,vfnreR,.05)/ub05 #capital stock
f replace = .05*capstock # capital wearout
gname InvAndWear # Name for next graph
vr 0 # vertical range: put bottom of graph at 0
gr replace, vfnreR # Graph these series; note comma!
ti Equipment Investment
subti No constraint
r vfnreR = d,d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],replace
gname Nocon
gr *
#=============
subti
ti
catch off
save off
```

When run through Gwx it gives these results:

```
:                                Equipment Investment
                                     No constraint
SEE =       32.3 RSQ =    0.988 RHO =    0.908 DW =    0.184
  Variable name              RegCoef  Mexval   Elas  NorRes     Means
  1 intercept              -27.81968     6.5  -0.06   80.99      1.00
  2 d                        0.12509     3.2   0.02   79.53     65.66
  3 d[1]                     0.15843     4.7   0.02   78.16     65.56
  4 d[2]                     0.14510     3.9   0.02   76.98     65.55
  5 d[3]                     0.16747     5.1   0.02   75.81     65.26
  6 d[4]                     0.15367     4.4   0.02   74.86     65.38
  7 d[5]                     0.13860     3.7   0.02   73.76     64.62
  8 d[6]                     0.10378     2.0   0.01   73.10     64.34
  9 d[7]                     0.11118     2.4   0.01   72.55     63.29
 10 d[8]                     0.14505     4.2   0.02   71.50     62.37
 11 replace                  1.09866   745.6   0.89    1.00    394.94
```



The fit could hardly be better, but the coefficient on “replace” indicates that nearly 110% of what wears out is being replaced. I would like for it to be somewhat below 1.00. So let's use the con command to pull it down. We just add before the r command

```
subti Pull a11 (a eleven) towards .80
con 100000 .8 = a11
gname Pull
gr *
```

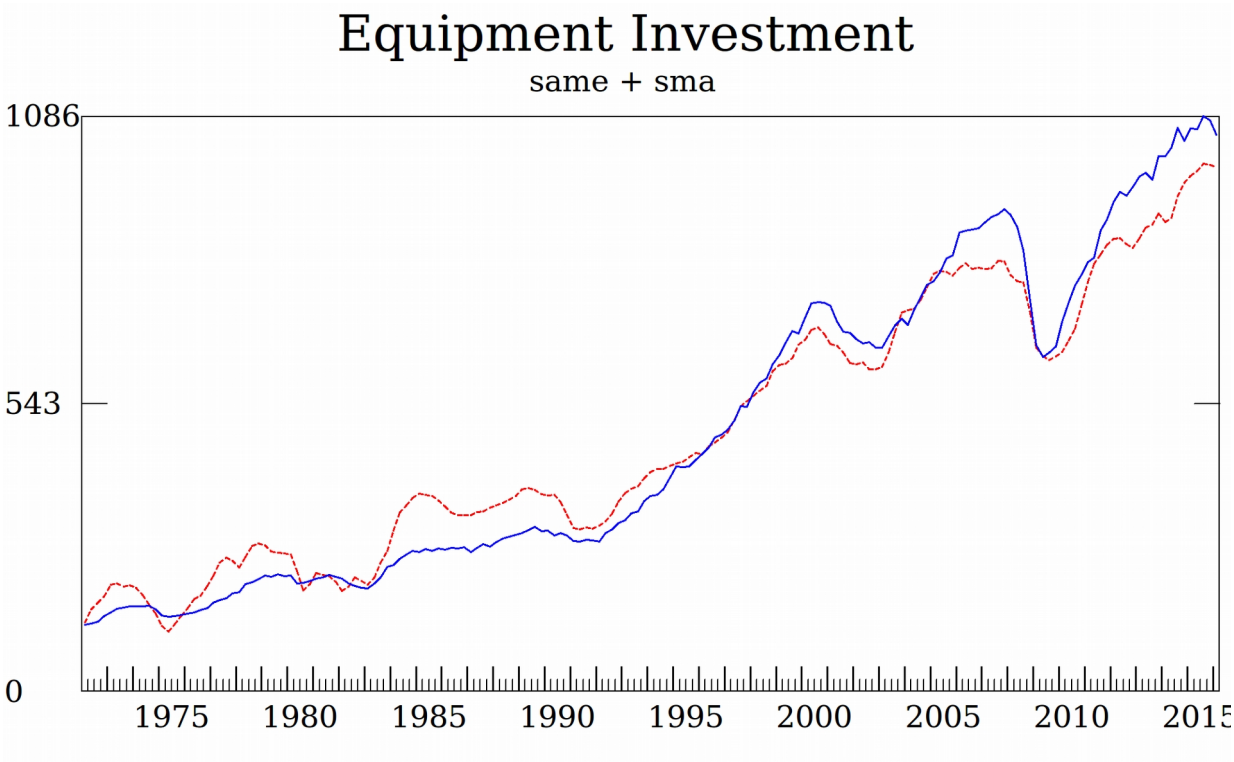
And we get these results:

```
:                              Equipment Investment
                          Pull a11 (a eleven) towards .80
SEE =      116.0 RSQ =    0.841 RHO =    0.958 DW =    0.084
  Variable name              RegCoef  Mexval   Elas  NorRes      Means
  1 intercept               31.04642     0.8   0.06    6.27       1.00
  2 d                        0.15307     0.4   0.02    6.16      65.66
  3 d[1]                     0.17944     0.5   0.02    6.06      65.56
  4 d[2]                     0.16429     0.4   0.02    5.96      65.55
  5 d[3]                     0.19079     0.5   0.03    5.87      65.26
  6 d[4]                     0.16730     0.4   0.02    5.80      65.38
  7 d[5]                     0.16883     0.4   0.02    5.71      64.62
  8 d[6]                     0.11718     0.2   0.02    5.66      64.34
  9 d[7]                     0.12603     0.2   0.02    5.62      63.29
 10 d[8]                     0.20574     0.7   0.03    5.54      62.37
 11 replace                  0.91308   231.1   0.74    1.00     394.94
```

The problem with a11 has been fixed, but the distributed lag weights jump about a bit irregularly. In particular, the last one jumps up to be the largest of all. Let's softly constraint the last eight of them to lie on a cubic polynomial by the command

```
sma 100000000 a3 a10 3
```

The results are:




```
:                              Equipment Investment
                                    same + sma
SEE =      116.5 RSQ =    0.839 RHO =    0.967 DW =    0.066
  Variable name              RegCoef  Mexval   Elas  NorRes     Means
  1 intercept               32.91736     0.9   0.07    6.22      1.00
  2 d                        0.14025     0.3   0.02    6.11     65.66
  3 d[1]                     0.20417     0.9   0.03    6.11     65.56
  4 d[2]                     0.16414     2.5   0.02    6.11     65.55
  5 d[3]                     0.15456     1.8   0.02    6.11     65.26
  6 d[4]                     0.16226     2.3   0.02    6.10     65.38
  7 d[5]                     0.17412     4.5   0.02    6.07     64.62
  8 d[6]                     0.17697     4.0   0.02    5.82     64.34
  9 d[7]                     0.15767     2.0   0.02    5.63     63.29
 10 d[8]                     0.10306     1.2   0.01    5.53     62.37
 11 replace                  0.91375   230.3   0.74    1.00    394.94
```

The problem with the tail end of the distributed lag has been fixed. The regression, of course, is not perfect, but is intended only to illustrate the use of the *con* and *sma* commands as well as the *@cum()* function in Gwx. In particular, the ρ value is very close to 1.0, and indeed the graph indicates the very high autocorrelation of the errors. Perhaps interest rates have a little something to do with investment.

What goes into @atoq() and @atoqi()?

The rest of this report is a recently written section of the book *The Gwx Story.* It deals with exactly how the two annual-to-quarterly conversion functions are written. The user of G7 or Gwx may well wonder how this manufacturing of data is done; and, indeed, these functions are some of the most mathematically interesting in the whole program. The text is reproduced here exactly as it stands in the book, so there are a few references – mainly about where this or that is to be found in the program – which won't be meaningful out of context. But mostly the text should be clear.

Making a series of one frequency from one of a lower frequency obviously involves data creation, and there is no perfect way to do it. All we can hope for is plausibility. We will assume that the series are flows – such as GDP or investment – not values of a specific date, such as stock exchange closing prices. We therefore want the sum of the four quarterly values of the series being created from an annual series to equal the given annual total.[1] Similarly, in conversion of a quarterly series to a monthly one, the sum of the three monthly values should equally the quarterly total.

The simplest way to achieve this correct summing when creating a quarterly series is to make all the quarterly values in a given year equal to one fourth of the annual total. That method, however, creates a highly implausible-looking series with big jumps between the fourth quarter of one year and the first of the next, but no change within the year.

The atoq command uses a slightly more complicated method based on polynomial interpolation which largely avoids this problem. Let a1, a2, and a3 be the annual values in three successive years. Now plot the four (t,y) points:

(0, 0)

(1, a1)

(2, a1+ a2)

(3, a1+ a2 + a3),

and find the third degree polynomial, y = P(t), which passes through these four points. (There is one and only one such polynomial.) Let

q1 = P(1.25) – P(1)

q2 = P(1.5) – P(1.25)

q3 = P(1.75) – P(1.5)

q4 = P(2.0) – P(1.75)

be the values of the quarterly series in year 2. The sum of these four quarterly values is

P(2) – P(1) = (a1 + a2) – a1 = a2

So the four quarterly values in year 2 sum to the annual total, as required. If the annual series is growing, with a1 < a2 < a3, P(t) will have a positive second derivative and the q's will be increasing, providing a more plausible quarterly series than the one with all quarters equal. In the first year of the annual data, the cubic fitted through the first three years is used; and in the last year of annual data, the cubic fitted through the last three years is used. In other years, the cubic fitted through the given year and those on either side is used.

Experience with this sort of interpolation has shown that it gives generally plausible results, but of course misses unusual spikes or troughs.

The next question is how to calculate the cubic polynomial passing through four points. As is usual in numerical analysis, we employ Lagrangian polynomials. These neat polynomials have the property that the first one is 1 at the first point and 0 at the other three points, the second is 1 at the second point and 0 at the other three points, and so on. The polynomial that passes through the four

1 Actually, because Gwx is likely to be used with national accounts data where the quarterly series are at annual rates, the atoq command will multiply the calculated quarterly values which sum to the annual total by 4 to yield a series at annual rates, but we will leave that adjustment to the end and consider that we want the quarterly values to sum to the annual value.

points is simply the sum of the four Larangian polynomials each weighted by the desired value at the point where it is 1.

Let t_0, t_1, t_2, and t_3 be the points where the polynomial has the values $v_{0,}$ $v_{1,}$ v_2 and v_3 . Then define the Lagrangian polynomials

$$L_0(t) = \frac{(t-t_1)(t-t_2)(t-t_3)}{(t_0-t_1)(t_0-t_2)(t_0-t_3)}$$

$$L_1(t) = \frac{(t-t_0)(t-t_2)(t-t_3)}{(t_1-t_0)(t_1-t_2)(t_1-t_3)}$$

$$L_2(t) = \frac{(t-t_0)(t-t_1)(t-t_3)}{(t_2-t_0)(t_2-t_1)(t_2-t_3)}$$

$$L_3(t) = \frac{(t-t_0)(t-t_1)(t-t_2)}{(t_3-t_0)(t_3-t_1)(t_3-t_4)}$$

Then

$$P(t) = v_0 L_0(t) + v_1 L_1(t) \; v_2 L_2(t) + v_3 L_3(t)$$

is the desired polynomial, as is easily seen.

In G7, the Lagrangian polynomials are evaluated in the code every time @atoq() is called. For Gwx, it seemed to me more instructive to calculate them once and for all with a spreadsheet program, show the results here, and use the resulting numerical values in the code – which is then far simpler than the G7 code. The table below shows the results of these calculations. The first panel shows the coefficients to be used in an internal year, that is one that is neither the first nor the last for which we have data. We will call it the target year. *a1* is the annual total for the preceding year, *a2* is the annual total for the target year, and *a3* is the annual total for the following year. We read the formula for the value of the series in the first quarter of the target year, Q1, from the first column of the table:

Q1 = .05469a1 + .23438a2 - .03906a3

and similarly for the other three quarters. Notice that the sum of the three coefficients in each quarter is (except for rounding) 0.25. Thus, if we increase a1, a2, and a3 each by 1.0, each quarterly amount will be increased by .25 and the sum the four quarterly amounts will be increased by 1.0, just as it should be. An increase of 1 in a2 will increase the sum of the four quarterly values by 1.0, just as it should, but an increase of 1 in a1 or a3 will have no effect on the sum of quarterly values in the middle year. All that is just as it should be and is more important than the precise values in the individual quarters.

	For the quarters of an internal year				
	Q1	Q2	Q3	Q4	Sum
a1	0.05469	0.00781	-0.02344	-0.03906	0.00000
a2	0.23438	0.26563	0.26563	0.23438	1.00000
a3	-0.03906	-0.02344	0.00781	0.05469	0.00000
Sum	0.25000	0.25000	0.25000	0.25000	1.00000
	For the quarters of the last year				
a1	-0.03906	-0.02344	0.00781	0.05469	0.00000
a2	0.17188	0.07813	-0.04688	-0.20313	0.00000
a3	0.11719	0.19531	0.28906	0.39844	1.00000
	0.25000	0.25000	0.25000	0.25000	1.00000
	For the quarters of the first year				
a1	0.39844	0.28906	0.19531	0.11719	1.00000
a2	-0.20313	-0.04688	0.07813	0.17188	0.00000
a3	0.05469	0.00781	-0.02344	-0.03906	0.00000
	0.25000	0.25000	0.25000	0.25000	1.00000

Within the individual quarters, we can detect an almost anthropomorphic behavior on the part of the formulas. In the first quarter of an internal year, the formula is still quite attached to the previous year, doesn't quite believe the current year and totally mistrusts the next year. By the second quarter, it has almost forgotten the previous year, is fully in the swing of the present year, but maintains a reduced mistrust of the promised next year. The second half of the year mirrors the first half but with the previous year and the next year changing roles.

For the first year and the last year, we have to use different coefficients, as shown in the lower panels of the table. These panels also have the appropriate row and column sums.

Since these sums and the general pattern of the coefficients are more important than the precise values in the tables, I have – to avoid cluttering the program with long numbers – rounded the numbers to three decimal places in the code but carefully preserved the sums.

Annual to Quarterly Conversion – @atoq()

The usage of the function is similar to that of @log() and @exp(). The user can write, for example,

```
f qann = @atoq(ann)
```

where *ann* is an annual series and *qann* is the quarterly series created from it. The *fdate*s must be quarterly for the command to function correctly. The argument of the function – *ann* in the example – must be the name of an annual series *already* in the workspace bank or the assigned bank. It cannot be a function; evaluating a function of annual data requires annual fdates, and – as just noted – the fdates must be quarterly for @atoq() to work properly.

We need to add *atoq* to the list of functions recognized by an *f* command. In the *functions()* routine in *functions.cpp* we add the line shown in bold here:

```
  if (strcmp(s,"log") == 0) funcrtn = logarithm(f);
else if (strcmp(s,"exp") == 0) funcrtn = exponential(f);
else if (strcmp(s,"cum") == 0) funcrtn = cumulate(f);
else if (strcmp(s,"atoq") == 0) funcrtn = atoq(f);
```

The main addition to the the code is the *atoq()* function which is found in *functions.cpp* right after the *cumulate()* routine. The fundamental difference of atoq() from log() and exp() is that we cannot call rhs() to get the argument to the function because the fdates are quarterly – and must be quarterly to store the result correctly – while the argument to the function is of course an annual series. So instead of calling rhs() to get the annual series from which we start, we pull it directly from the workspace or the first assigned bank with a call to getseries(), shown in bold in the code. Then – since the fdates don't tell us where to begin and end working on the annual series – we simply search it to find where it starts and stops and create a quarterly series covering those same years. The command which will eventually store the series we create will use the frequency from the fdates – namely quarterly – but will store the whole series we create, which may start or stop before or after the fdates. Here is the rather simple code.

```
short atoq(float *f){
    char name[MAXNAMELENGTH];
    short i,j,n,err,astart,astop,qstart,qstop,q1;
    float aseries[NOMAX]; // The annual series.

   if (((err = chop(name)) != '(')) {
        printg("Expected ( after @atoq.\n");
        goto error;
        }
   if ((err = chop(name) ) != 'a'){
        printg("Expected variable name in @atoq()\n");
        goto error;
        }
    //Get the series to be converted.
    // The -1 for series length causes nobs to be used.

    nopy = getseries(name,aseries,-1);

    // Find where aseries begins
    i = 0;
    while (aseries[i]== MISSING) i++;
    astart = i;

    // Find where aseries ends
    while(aseries[i] != MISSING && i <= NOMAX) i++;
    astop = i-1;
    // printg("astart = %d astop = %d \n", astart,astop);
    qstart = astart*4;
    qstop = astop*4 +3;
    if(qstop >= NOMAX - 1){
        printg("Series too long for me.\n");
        return ERR;
        }

    // interpolate the first year
    f[qstart]   = .399*aseries[astart] - .203*aseries[astart+1] + .054*aseries[astart+2];
    f[qstart+1] = .289*aseries[astart] - .047*aseries[astart+1] + .008*aseries[astart+2];
    f[qstart+2] = .195*aseries[astart] + .078*aseries[astart+1] - .023*aseries[astart+2];
    f[qstart+3] = .117*aseries[astart] + .172*aseries[astart+1] - .039*aseries[astart+2];

    // interpolate the last year
    f[qstop]   = .399*aseries[astop] - .203*aseries[astop-1] + .054*aseries[astop-2];
    f[qstop-1] = .289*aseries[astop] - .047*aseries[astop-1] + .008*aseries[astop-2];
    f[qstop-2] = .195*aseries[astop] + .078*aseries[astop-1] - .023*aseries[astop-2];
    f[qstop-3] = .117*aseries[astop] + .172*aseries[astop-1] - .039*aseries[astop-2];

    // Interpolate internal years
    for (i = astart+1;i<= astop-1;i++){
        q1 = i*4;
        f[q1]   =  .055*aseries[i-1] + .234*aseries[i] - .039*aseries[i+1];
        f[q1+1] =  .008*aseries[i-1] + .266*aseries[i] - .024*aseries[i+1];
        f[q1+2] = -.024*aseries[i-1] + .266*aseries[i] + .008*aseries[i+1];
        f[q1+3]=  -.039*aseries[i-1] + .234*aseries[i] + .055*aseries[i+1];
        }
```

```
// Convert the  quarterly data to annual rates
for(i = qstart;i<= qstop; i++){
    f[i] = 4.0*f[i];
    }

return OK;

error:
return err;
}
```

I took advantage of the symmetry between the first and last year to just copy the first year code and then change the subscripts on the variables to get the last year code.

I tested the code with this example:

```
data ann
1980  2  4  6  8 11 14 18 25 30 36
1990 42 47 51 55 58 57 53 48 45 48 ;
fdates 1980q1 2000q4
f aqtest = @atoq(ann)
tdates 1980q1 1999q4
type aqtest
```

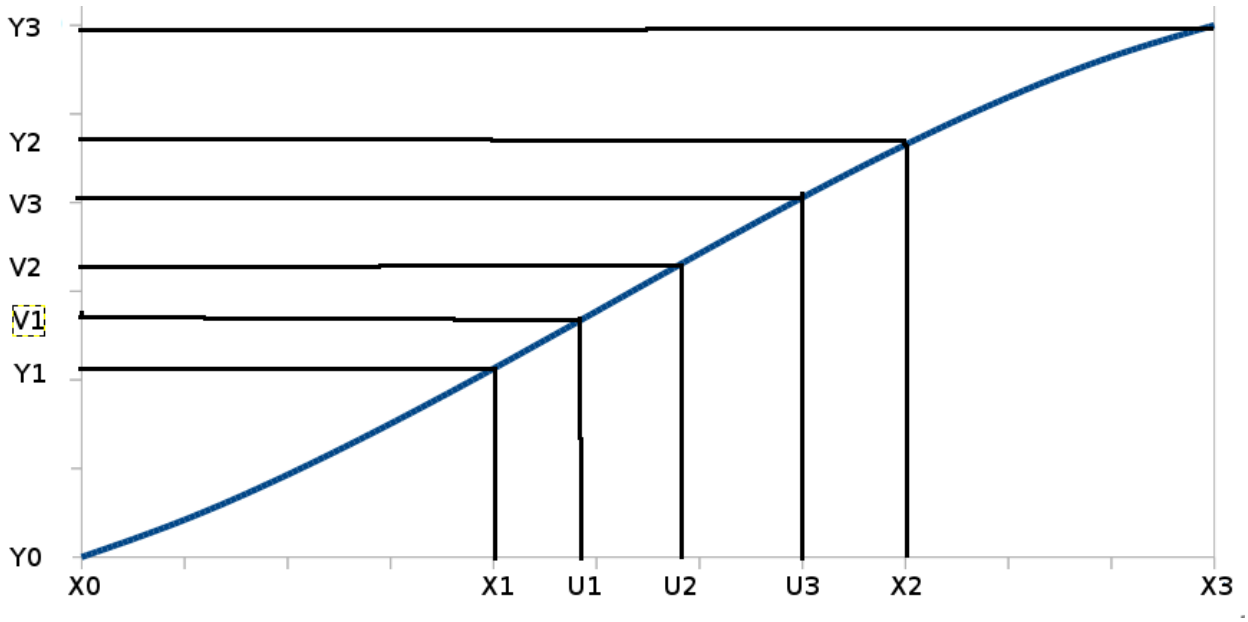
with this rather satisfactory result where one can see the effect of the value on either side of a year on interpolation within the year:

```
1980q1         1.240        1.752        2.256        2.752
1981q1         3.248        3.744        4.256        4.752
1982q1         5.248        5.744        6.256        6.752
1983q1         7.092        7.648        8.288        8.972
1984q1         9.872       10.616       11.384       12.128
1985q1        12.716       13.520       14.416       15.348
1986q1        16.028       17.200       18.608       20.164
1987q1        22.680       24.296       25.832       27.192
1988q1        27.964       29.264       30.672       32.100
1989q1        33.744       35.232       36.768       38.256
1990q1        39.900       41.328       42.736       44.036
1991q1        45.276       46.456       47.608       48.660
1992q1        49.496       50.488       51.512       52.504
1993q1        53.652       54.584       55.480       56.284
1994q1        57.496       58.000       58.256       58.248
1995q1        57.844       57.416       56.776       55.964
1996q1        54.660       53.608       52.456       51.276
1997q1        49.568       48.448       47.424       46.560
1998q1        45.192       44.808       44.808       45.192
1999q1        45.936       47.064       48.564       50.436
```

Note particularly the behavior in turning point years such as 1994 and 1998.

Annual to Quarterly Conversion with an Indicator Series – atoqi().

In the @atoq() function, we worked without any information about how the annual flow might have been distributed among the quarters of the year. We just produced a smooth curve with the right annual sum. Sometimes, however, we have a different series that can be used as a guide or indicator of the movement of the series we need to convert to a quarterly frequency. If we had annual data on fuel oil consumption and quarterly data on heating degree days, we might use the latter to guide us in making a quarterly series of the former. If the annual series is in constant proportion to the quarterly series, the problem is easily solved, but if the proportion is smoothly changing a more flexible tool is needed.



Annual to Quarterly Interpolation with Indicator, atoqi()

We again turn to polynomial interpolation, but with a new wrinkle. Instead of using equally spaced points on the x axis, we use points spaced in proportion to the values of the indicator series. The method is illustrated in the diagram below. If α is the annual series we wish to interpolate and $\alpha 1$, $\alpha 2$, and $\alpha 3$ three successive values, then we define four points on the vertical axis by

$y0 = 0$

$y1 = y0 + \alpha 1$

$y2 = y1 + \alpha 2$

$y3 = y2 + \alpha 3.$

Likewise, if the annual sums of the indicator series are $\beta 1$, $\beta 2$, and $\beta 3$, we define

$x0 = 0$

$x1 = \beta 1$

$x2 = x1 + \beta 2$

$x3 = x2 + \beta 3.$

The interpolating polynomial, L(x), is the cubic passing through the four points (x0,y0) = (0,0), (x1,y1), (x2,y2), (x3,y3), as shown in the graph below. (The curve is, in fact, a cubic drawn with LibreOffice Calc, not my freehand creation.)

Now let $\gamma 1$, $\gamma 2$, $\gamma 3$ and $\gamma 4$ be the quarterly values of the indicator series, so that

$\beta 2 = \gamma 1 + \gamma 2 + \gamma 3 + \gamma 4$

and define points on the x axis

$u1 = x1 + \gamma 1$

$u2 = x1 + \gamma 1 + \gamma 2$

u3 = x1 + γ1 + γ2 + γ3.

x2, already defined, is x1 + γ1 + γ2 + γ3 + γ4.

The corresponding points on the y axis are v1 = L(u1), v2 = L(u2), and v3 = L(u3), and the quarterly series we seek is

q1 = v1 – y1, q2 = v2 – v1, q3 = v3 – v2, and q4 = y2 – v3.

These quarterly numbers sum to the yearly total. Since quarterly national accounts usually presented as quarterly flows at annual rates, we multiply these quarterly numbers by 4 before reporting the result.

As in the case of @atoq() without the indicator series, the first and last years require special handling. Because the programming is tedious and I am somewhat pressed for time at the moment, I have taken the shortcut of simply throwing in the @atoq() code for these two years.

Here is the code, which is in functions.cpp. There are numerous comments, and the notation matches the text and the graph, so I hope it is clear without further explanation.

```
// Annual to quarterly interpolation with an indicator series.
short atoqi(float *f){
    char namea[MAXNAMELENGTH],namei[MAXNAMELENGTH];
    short i,j,j0,n,err,astart,astop,istart,istop;
    float aseries[NOMAX],iseries[NOMAX];
    float x1,x2,x3,y1,y2,y3,u1,u2,u3,v1,v2,v3;
    float d1,d2,d3;

   if (((err = chop(namea)) != '(')) {
        printg("Expected ( after @atoq.\n");
        goto error;
        }
   if ((err = chop(namea) ) != 'a'){
        printg("Expected variable name in @atoq()\n");
        goto error;
        }
   if (((err = chop(namei))) != ',') {
        printg("Expected , between arguments.\n");
        goto error;
        }
   if ((err = chop(namei) ) != 'a'){
        printg("Expected name of indicator series.\n");
        goto error;
        }

    //Get the series to be converted.
    // The -1 for series length causes nobs to be used.
    nopy = getseries(namea,aseries,-1);
    if(nopy != 1) {
        printg("%s is not annual.\n",namea);
        goto error;
        }
    // Get the indicator series
    nopy = getseries(namei,iseries,-1);
    if (nopy != 4){
        printg("%s is not quarterly.\n",namei);
        goto error;
        }

    // Find where aseries begins.
    i = 0;
    while (aseries[i]== MISSING && i <= NOMAX) i++;
    if(i >= NOMAX){
        printg("The annual series is empty.\n");
        goto error;
        }
    astart = i;

    // Find where aseries ends.
```

```
    while(aseries[i] != MISSING && i <= NOMAX) i++;
    astop = i-1;
    printg("astart = %d astop = %d \n", astart,astop);

    // Find where iseries starts

    istart = astart*4;
    while ((iseries[istart] == MISSING) && (istart < NOMAX)) istart += 4;
    astart = istart/4;
    istop = astop*4 +3;

    if(istart >= istop){
        printg("No overlap of series.\n");
        goto error;
        }

    if(istop >= NOMAX - 1){
        printg("Series too long for me.\n");
        goto error;
        }

    // Do the internal years; i is the year index.
    for(i = astart+1; i < astop; i++){
       // Compute the known cumulative points of the annual series on the vertical axis.
        y1 = aseries[i-1];
        y2 = y1 + aseries[i];
        y3 = y2 + aseries[i+1];

       // Compute the corresponding annual cumulative points of the indicator series
       //    on the horizontal axis.
        j0 = 4*(i-1);
        x1 = iseries[j0]+iseries[j0+1]+iseries[j0+2]+iseries[j0+3];
        x2 = x1+iseries[j0+4]+iseries[j0+5]+iseries[j0+6]+iseries[j0+7];
        x3 = x2+iseries[j0+8]+iseries[j0+9]+iseries[j0+10]+iseries[j0+11];

       // Compute the quarterly cumulative points of the indicator series-by-series
       //    on the horizontal axis.
        u1 = x1+iseries[j0+4];
        u2 = u1+iseries[j0+5];
        u3 = u2+iseries[j0+6];

        // Compute the denominators of the Lagrangian polynomials,
        // which are constant within a year.
        d1 = x1*(x1-x2)*(x1-x3);
        d2 = x2*(x2-x1)*(x2-x3);
        d3 = x3*(x3-x1)*(x3-x2);

// Use the polynomial to compute the quarterly cumulative points on the vertical axis.
        v1 = y1*((u1)*(u1-x2)*(u1-x3)/d1) +
             y2*((u1)*(u1-x1)*(u1-x3)/d2) +
             y3*((u1)*(u1-x1)*(u1-x2)/d3);

        v2 = y1*((u2)*(u2-x2)*(u2-x3)/d1) +
             y2*((u2)*(u2-x1)*(u2-x3)/d2) +
             y3*((u2)*(u2-x1)*(u2-x2)/d3);

        v3 = y1*((u3)*(u3-x2)*(u3-x3)/d1) +
             y2*((u3)*(u3-x1)*(u3-x3)/d2) +
             y3*((u3)*(u3-x1)*(u3-x2)/d3);

               // Take differences of the cumulative points to get the quarterly flows.
        f[i*4] = v1 - y1;
        f[i*4 + 1] = v2 - v1;
        f[i*4 + 2] = v3 - v2;
        f[i*4 + 3] = y2 - v3;
        }

    // Interpolate the first year (without benefit of the indicator).
    f[istart]   = .399*aseries[astart] - .203*aseries[astart+1] + .054*aseries[astart+2];
    f[istart+1] = .289*aseries[astart] - .047*aseries[astart+1] + .008*aseries[astart+2];
    f[istart+2] = .195*aseries[astart] + .078*aseries[astart+1] - .023*aseries[astart+2];
    f[istart+3] = .117*aseries[astart] + .172*aseries[astart+1] - .039*aseries[astart+2];

    // Interpolate the last year (also without benefit of the indicator).
```

```
f[istop]   = .399*aseries[astop] - .203*aseries[astop-1] + .054*aseries[astop-2];
f[istop-1] = .289*aseries[astop] - .047*aseries[astop-1] + .008*aseries[astop-2];
f[istop-2] = .195*aseries[astop] + .078*aseries[astop-1] - .023*aseries[astop-2];
f[istop-3] = .117*aseries[astop] + .172*aseries[astop-1] - .039*aseries[astop-2];

// Convert the quarterly data to annual rates
for(i = istart;i<= istop; i++){
    f[i] = 4.0*f[i];
    }

return OK;

error:
return ERR;
}
```

How much difference does the use of the indicator make? It depends on the indicator. If is smooth, like GDP, – not much. When the *ann* series of the test was interpolated using GDP, no point differed by more than half a percent from the interpolation without the indicator. But when it was interpolated using residential construction, vfrR in the Quip bank, there were differences of up to five percent, as shown in the figure below.