Paper 093-2008

Easy Rolling Statistics with PROC EXPAND

Premal P. Vora, Penn State Harrisburg, Middletown, PA.

ABSTRACT

When analyzing a time series of data, a researcher frequently desires to output and analyze moving or rolling statistics such as moving averages, moving standard deviations, and rolling correlations. While it is possible to implement this in the DATA step, SAS® provides some powerful features in PROC EXPAND (which is a component of SAS/ETS) that allow the researcher to output a variety of moving statistics with minimal effort. In this presentation, which is intended for basic to intermediate level SAS users, I will provide two examples of DATA step code that outputs the desired moving statistics and the equivalent code in PROC EXPAND and compare both to demonstrate the power of PROC EXPAND.

INTRODUCTION

Time series analysis is important in several fields of research such as finance and economics. For example, finance researchers are interested in the behavior of stock prices and interest rates while economists are interested in studying the rate of inflation, the gross domestic product, the rate of unemployment and so on, over time. While specialized packages geared toward time series analysis are readily available no one package has the wide scope and many features that appear in the SAS system. Combined with the many other features available in SAS/ETS, SAS makes for a complete system for time series analysis, other statistical analysis and data management.

One of the peculiar features of time series analysis is it frequently requires researchers to combine the current observation with its lagged values like in a moving average. Other more complicated examples are calculating a moving standard deviation, or – for bivariate analysis – calculating a rolling correlation coefficient. It is possible to implement all of these via the DATA step in SAS; however, as I will demonstrate below, it is much easier and more elegant to accomplish the same result using the powerful features that are available in PROC EXPAND of SAS/ETS compared to the cumbersome code in the DATA step.

One task that PROC EXPAND excels at and is useful for time series analysis is transforming a set of series into another by operating on it in numerous ways. For instance, it has a built-in ability to calculate the sum, the mean, median, sum of squares, etc. Further one can select the length of the window and how missing observations should be treated. It can also convert a time series into a higher frequency or aggregate it into a lower frequency. Additionally, if some desired feature is not available off the shelf, the user can program it by combining the strong built-in features of SAS. Some of these ideas are demonstrated in the two examples below to illustrate the power and elegance of PROC EXPAND.

COMPARISON OF DATA STEP AND PROC EXPAND

Two examples are provided below to bring out the power and elegance of PROC EXPAND relative to the data step. In the first example a simple moving average over a rolling 5-day window is calculated. In the second example a rolling correlation coefficient over a window of 55 days is calculated. Both examples are illustrated with the relevant DATA step code followed by the equivalent PROC EXPAND code.

EXAMPLE 1: CALCULATING A MOVING AVERAGE

Suppose I want to calculate a moving average of the variable xi over a rolling centered 5-day window. I need to define a variable that retains the sum of xi (sumxi in the code below) but at the same time because I am not interested in any observation that is more than 5 days old I need the 5-day lagged value of xi (xi5 in the code below). Also, suppose when the value of xi is missing during any 5-day window it is acceptable to calculate the average for the remaining days. Thus, I need to let the code know not to fill the window surrounding any missing values with a missing average. I accomplish this by setting any missing values of xi and xi5 to zero and by creating two new variables, obs and obs5, which contain a flag when the current and lagged xi are missing. Here is the DATA step code (all code is tested on actual data to ensure that it works):

```
/*-------------------------------------------------------------------*/
/*  Pass through the data to flag missing observations and to create  */
/*  lagged values of the variables of interest                        */
/*-------------------------------------------------------------------*/
data cma; set cma;

 if missing(xi) then
   do;
     OBS = 0;
     xi = 0.0;
   end;
 else OBS = 1;

 XI5 = lag5(xi);
 OBS5 = lag5(obs);

 if missing(xi5) then xi5 = 0.0;
 if missing(obs5) then obs5 = 0;

 LDATE = lag2(date);
 format ldate date9. ;

/*-------------------------------------------------------------------*/
/*  Now calculate the moving average for just the 5-day window...     */
/*-------------------------------------------------------------------*/
  if _N_ = 1 then
      do;
       SUMXI = 0.0;
       N = 0;
      end;
   else;

  sumxi = sumxi + xi - xi5;
  n = n + obs - obs5;
  MEANXI = sumxi / n ;
  retain sumxi n;
run;
```

The PROC EXPAND code to accomplish the same result is:

```
proc expand DATA = cma   OUT = cmaout;
 convert xi = MEANXI / METHOD = none   TRANSFORMOUT = (cmovave 5);
run;
```

In the above example the convert statement is used to convert xi into a centered moving average (meanxi) over a 5-day window. This is captured in the “cmovave 5” option. The “method = none” option above tells SAS not to extrapolate any missing values which SAS will do unless it is asked not to. If I wanted to have SAS extrapolate missing values I have a number of choices on how the extrapolation is done. SAS takes care of all the issues related to missing values, removing lagged values of variables, alignment of dates, etc., behind the scenes letting the researcher focus on the results of the analysis. Note that the DATA step contains 26 lines of code compared to 3 lines for PROC EXPAND. Apart from the difference in size of the programs and the effort required to create the proper algorithm, the DATA step code is more susceptible to errors and therefore may require more debugging. The equivalent PROC EXPAND code will require the programmer to learn the basics of the procedure, which may involve a steep learning curve, but once the basics of the procedure are understood the code requires little or no debugging to get it to run and to do what the programmer wants it to do.

EXAMPLE 2: CALCULATING A ROLLING CORRELATION COEFFICIENT

If you are not already convinced of the power and elegance of PROC EXPAND, here is another example that brings out both those aspects of PROC EXPAND. Suppose you are asked to write a program that calculates a rolling or centered moving correlation coefficient between two variables xi and yi in a 55-day window. Here is the relevant DATA step code:

```
/*--------------------------------------------------------------------*/
/*  Pass through the data to flag missing observations and to create  */
/*  lagged values of the variables of interest                         */
/*--------------------------------------------------------------------*/
data cma; set cma;
 if missing(xi) or missing(yi) then
   do;
     OBS = 0;
     xi = 0.0;
     yi = 0.0;
   end;
 else OBS = 1;

 XI55 = lag55(xi);
 YI55 = lag55(yi);
 OBS55 = lag55(obs);

 if missing(xi55) then xi55 = 0.0;
 if missing(yi55) then yi55 = 0.0;
 if missing(obs55) then obs55 = 0.0;

 LDATE = lag27(date);
 format ldate date9. ;

/*--------------------------------------------------------------------*/
/*  Now calculate the correlation coefficient between xi, yi          */
/*--------------------------------------------------------------------*/
 if _N_ = 1 then
     do;
      SUMXI = 0.0;
      SUMYI = 0.0;
      SUMXISQ = 0.0;
      SUMYISQ = 0.0;
      N = 0;
      SUMPRODXIYI = 0.0;
     end;
  else;

 sumxi = sumxi + xi - xi55;
 sumyi = sumyi + yi - yi55;
 sumxisq = sumxisq + xi*xi - xi55*xi55;
 sumyisq = sumyisq + yi*yi - yi55*yi55;
 n = n + obs  - obs55;
 sumprodxiyi = sumprodxiyi + xi*yi - xi55*yi55;

 R = (sumprodxiyi - sumxi*sumyi/n) /
     sqrt(( sumxisq - (sumxi**2)/n) * (sumyisq - (sumyi**2)/n)) ;

 retain sumxi sumyi sumxisq sumyisq sumprodxiyi n;
run;
```

The PROC EXPAND code to accomplish the same result is:

```
data cma ; set cma;
```

```
      OBS = 1;

      if missing(xi) OR missing(yi) then
        do;
          xi = . ;
          yi = . ;
          obs = . ;
        end;

      PRODXIYI = yi * xi;
    run;


    proc expand DATA = cma   OUT = cmaout;
     convert yi = YISUM / METHOD = none   TRANSFORMOUT = (cmovsum 55);
     convert xi = XISUM / METHOD = none   TRANSFORMOUT = (cmovsum 55);
     convert prodxiyi = PRODXIYISUM / METHOD = none   TRANSFORMOUT = (cmovsum 55);
     convert obs = N  / METHOD = none   TRANSFORMOUT = (cmovsum 55);
     convert xi = XICSS / METHOD = none   TRANSFORMOUT = (cmovcss 55);
     convert yi = YICSS / METHOD = none   TRANSFORMOUT = (cmovcss 55);
    run;


    data results; set cmaout;
     R = (prodxiyisum - (yisum*xisum)/n) / ( sqrt(xicss)*sqrt(yicss)) ;
    run;
```

PROC EXPAND does not contain a facility to operate on two or more different variables at the same time to create bivariate or multivariate statistics. Thus, the data requires some additional preparation before it is sent to PROC EXPAND and further processing after PROC EXPAND performs its magic. The PROC EXPAND code above transforms six different time series which are then used as inputs to the correlation coefficient in the DATA step that follows. The code above calculates the 55-day centered moving sum for four series by using the “cmovsum 55” option and the centered moving corrected sum of squares for two series by using the “cmovcss 55” option.

In this example the difference in the number of lines between the DATA step code and PROC EXPAND code is not as large (36 lines compared to 21 lines) as it was in the previous example. However, the effort required to conceive of the algorithm for the DATA step code is much larger than for the PROC EXPAND code. Further, the potential for errors is larger and therefore the DATA step code would require more debugging.

CONCLUSION

This paper provides two examples that demonstrate the power and elegance of PROC EXPAND. This procedure excels at dealing with time series data and provides facilities for transforming or converting one time series into another. While it is possible to create code that would accomplish the same thing in a DATA step, the resulting DATA step code is long, cumbersome, requires careful attention to algorithm and usually requires substantial debugging. Compared to that the PROC EXPAND code is short, easy to read, and requires little debugging. Of course, PROC EXPAND requires a steep learning curve. However, if you frequently deal with time series data and statistical analysis of that data, the above examples are presented to demonstrate to you that it is well worth your time to go up that learning curve.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Premal P. Vora
Associate Professor of FInance
Penn State Harrisburg
777 W. Harrisburg Pike
Middletown, PA 17057.
Work Phone: (610) 873-1943

E-mail: fpv@psu.edu
Web: http://www.personal.psu.edu/faculty/f/p/fpv/