Rolling Regressions with PROC FCMP and PROC REG

Mark Keintz, Wharton Research Data Services, University of Pennsylvania

ABSTRACT

Although the technique of applying regressions to rolling time windows is commonly used in financial research for a variety of uses, SAS offers no routines for directly performing this analysis. The recent introduction of PROC FCMP, along with the long-standing capability of PROC REG to accept sum-of-squares and cross-products matrices, provides an easily-programmed way to have the flexibility of PROC REG (or other PROC's) for rolling windows. This presentation shows how to use the two PROCs, without the usual performance-sapping burden of generating large intermediate data sets of rolling window data.

INTRODUCTION

Rolling windows regression, (more generally all rolling window analysis) is a statistical technique that lets the analyst explore patterns and relationships that change over time. However, instead of estimating a single model of a complete time series in which specific time periods might be modeled as predictor variables, the rolling window technique generates a model estimate for every window of a given size in the series. For instance a 5 year monthly data series would contain 49 windows of 12-months length.

The resulting primary SAS® programming issue is how to efficiently form, or how to avoid forming, intermediate data sets that have each time point in the series included for analysis multiple times– once for each window containing that time point (we'll use DATE for the rest of this paper). This is an issue whose significance becomes relevant (if not dominant) only as the size of the data series or size of the windows becomes large. In our case, the is a data set with 29,435 publically traded company, with an average of 2,810 daily stock prices and returns. Analyzing rolling windows of 120 days would generate almost 10 billion rows from this data set.

This paper will demonstrate the simplest (brute force) technique, which literally creates a data set of all the rolling windows to be passed on to the analysis procedure (PROC REG), followed by a more efficient way of constructing rolling window series. It will also show two other techniques (one using PROC EXPAND and one using PROC FCMP) that replace the creation of a rolling data series with a rolling series of sum-of-squares and cross products (SSCP), ready for submission to PROC REG. The latter has the advantage of passing a much smaller data set to PROC REG, with resulting performance benefits.

THE BRUTE FORCE APPROACH

The simplest way (conceptually) to do a rolling analysis of any sort in SAS is to create a new data set in which every window is populated with data from each date covered by the window. It's essentially a task of reading each input record and writing that record out once for each window containing it. Writing these records in the proper order allows the use of a BY statement in the analysis procedure, yielding one regression for each window. Consider the data set MYSERIES with variables DATE, X, and Y. Example 1 performs rolling regression of Y on X for windows of size 90.

Example 1: Simple Program to Make Rolling Windows

```
DATA rwin / view=rwin;
  ws = 90;
  nwin = nrecs - ws +1;
  do w=1 to nwin;
    do p=w to w + ws -1;
      set myseries point=p nobs=nrecs;
      output;
    end;
  end;
  stop;
run;
```

```
proc reg data=rwin noprint outest=myests;
   by w;
   model x=y;
quit;
```

After calculating the number of 90-day windows (NWIN) to be generated, this program reads in 90 records for each window and writes them out. Every record (except the first 89) is read 90 times! The need to read the MYSERIES data multiple times (150 times for windows of 150 days) is ample demonstration that the only merit of this program is its simplicity. However the program does result in a data set (myests) with one observation per window containing default regression statistics: intercept, beta, and root-mean-squared-error (other regression stats can be requested).

HOLDING AND WRITING

A second approach is to write out the complete rolling window data series, as in example 1, but to read the incoming data set just once. Instead of constantly re-reading MYSERIES, the program below simply maintains arrays of the most recent WS (e.g. 90) values of X and Y, so they are available in memory for output.

Example 2: Maintaining Arrays for Writing Rolling Window Time Series

```
DATA rwin / view=rwin;
  array _X {90} _temporary_ ;
  array _Y {90} _temporary_ ;
  set myseries;
  I=mod(_N_-1,90)+1;
  _X{I}=x;
  _Y{I}=y;
  if _N_>=90 then do I= 1 to 90;
    x=_X{I};
    y=_Y{I};
    output;
  end;
run;
proc reg data=rwin noprint outest=myests;
   by date;
   model x=y;
quit;
```

Notable in this example are:

1. The _X and _Y arrays are temporary, which means (a) values in the arrays are automatically retained, and (b) the arrays do not become variables in the output data set.
2. The automatic variable _N_ is (in this example) the observation number of the record in hand (i.e. for 100,000 observations, _N_ ranges from 1 to 100,000).
3. The mod (for “modulo”) function returns the remainder of _N_-1 divided by 90. After adding back the 1, X from the first observation is stored in _X{1}, the 90th X goes to _X{90}, and the 91st goes back to _X{1}. So at any point in the data step, the _X and _Y arrays have the most recent 90 values.
4. Using the “DO I=1 to 90” will generate all the data for a given window, but it will typically not be in chronological order. In the case of PROC REG, order is not important. If a time-series analysis were being performed (e.g. PROC TSCSREG), within-window chronological order would be necessary.

Also the PROC REG has a "BY DATE" statement. The DATE variable is read from the MYSERIES data set, and is NOT stored in an array. The result is that the DATE variable is a constant for each window, taking its value from the record read just before the window data is written – i.e. the last date in the window.

This program is much faster than example one, due entirely to reading the data set only once no matter what window size is implemented. And this is probably about the most efficient technique for writing actual rolling window data. Further improvements, shown below, will depend upon constructing rolling SSCP matrices prior to running the PROC REG. that look like the above.

SERIES BY ID

It's worth a brief detour from progressively more efficient rolling regression techniques to address a common situation, namely one where there is a time series (stock returns in our case) for each of a large number of entities (~29,000 companies). In other words, how should the program above be modified to address data set MYSERIES containing four variables (ID, DATE, X, and Y), sorted by ID and DATE? The program now has to construct windows such that the first window from one ID is not contaminated by the last records of the prior id. This is done by creating a "within-ID" counter (N) to replace the automatic _N_ variable used in example 2.

Example 3: Rolling Series by ID

```
DATA rwin / view=rwin;
  array _X {90} _temporary_ ;
  array _Y {90} _temporary_ ;
  set myseries;
  by id;
  retain N 0;
  N = ifn(first.id,1,N+1);
  I=mod(N-1,90)+1;
  _X{I}=x;
  _Y{I}=y;
  if N>=90 then do I= 1 to 90;
    x=_X{I};
    y=_Y{I};
    output;
  end;
run;
proc reg data=rwin noprint outest=myests;
   by id date;
   model x=y;
quit;
```

The key here is to use the variable N as a record counter, reset to 1 when a new id is encountered, otherwise it's incremented by 1. The program then uses N to control when window writing starts ("IF N>= 90 …"), and the range of records to write ("… DO I= N-89 to N"). Because this technique only starts to write windows upon encountering the 90th record for a given ID, it will have replaced all the _X and _Y array elements taken from the prior ID. In other words, there is no spillover from one ID to the next.

PASS ON ROLLING SSCP, NOT ROLLING DATA

The techniques above generate actual data series for each window, which means, however efficient they are, the size of the data set they pass to PROC REG will be directly proportional to the size of the rolling window. The bigger the window the more data must be transferred to the PROC REG and the longer the task will take. However PROC REG can also accept SSCP (Sum of Squares and Cross Products) data, which will require only NV+2 rows

per window (NV is the number of variables in the regression model), regardless of the window size. For a simple regression of Y on X, only 4 rows per window need to be transferred to PROC REG. If a way could be found to efficiently generate rolling SSCP data, the time required to run rolling windows regressions might shrink considerably, simply because of the decrease in data volume.

THE TYPE=SSCP DATA SET

For a few statistical procedures including PROC REG, SAS has established a number of additional special data set types (CORR, COV, SSCP, EST and FACTOR) that are acceptable input. Running a PROC CONTENTS on one of these data sets will report this attribute with text on the listing file such as:

"Data Set Type: SSCP"

This data set attribute is used by PROC REG to recognize the type of data being used. Table 1 shows what the actual SSCP data looks like for a single window, usable for regression of Y on X (or X on Y). It has the BY variables (ID and DATE) used to identify the window and a _TYPE_ variable describing the nature of the row data. _NAME_ identifies the row variable of the SSCP matrix, and the actual SSCP values are in the last 3 columns. For example in this case, the sum of Y-squared is 0.1754, the sum of the XY cross-product is 0.0030, and the simple sum of Y is 0.5353 (in the "Intercept" row), for a 90-day window (i.e. row "N" is all 90's).

ID	DATE	_TYPE_	_NAME_	Intercept	X	Y
1001	19860515	SSCP	Intercept	90.000	0.0916	0.5343
1001	19860515	SSCP	X	0.0916	0.0048	0.0030
1001	19860515	SSCP	Y	0.5343	0.0030	0.1754
1001	19860515	N		90	90	90

Table 1: Representative SSCP Data Contents

So the task now becomes one of replacing the creation of a rolling windows data series with a rolling windows SSCP series – i.e. generating SUMs of daily data that look like the above.

ROLLING SSCP VIA PROC EXPAND

SAS users with a license for SAS/ETS® have available a ready-made tool for creating sums over rolling windows – PROC EXPAND. While the most commonly emphasized features of PROC EXPAND is its ability to convert data frequencies (e.g. monthly to quarterly, etc.) or smooth a data series, generating simple sums for fixed window length is a 4 or 5 lines statement. The example below creates rolling window SSCP data for X and Y:

Example 4: CREATING ROLLING WINDOW SUMS VIA PROC EXPAND

```
DATA vtemp / view=vtemp;
  set myseries;
  xx=x*x; xy=x*y yy=y*y;
run;

proc expand data=vtemp out=sscpdata (where=(_n=90)) method= none;
  by id;
  id date;
  convert y=_n  / transformin=(*0)  transformout=(+1 MOVSUM 90);
  convert x y xy xx yy /            transformout=(+1 MOVSUM 90);
run;
```

Because the task is to create sums of SQUARES and sums of CROSS_PRODUCTS, the data step first generates squares (XX and YY) and cross-product (XY) for each original record. PROC EXPAND then produces SSCPDATA which has all the SSCP needed information in one record per window. The most relevant elements of the procedure are:

1. The first convert statement tells proc expand to multiply each Y by 0 upon input, and at output to add 1 to each y, and generate the 90-day moving sum of the result. The resulting variable is named _N and is will be used to indicate the window size.
2. The second convert statement tells expand to generate the 90-day moving sum of the variables (X Y), their squares (XX YY) and cross-product (XY). The resulting variables will keep the original name.
3. In the "PROC EXPAND" statement "method=none" tells expand to do no smoothing of the input series.

The only problem with the output of expand is that it needs to be re-shaped into a TYPE=SSCP data set, which is demonstrated in Appendix 1.

A brief review of the performance of this technique showed no change in time-to-completion when varying the window size from 90 to over 200 days. In our case the performance of the PROC EXPAND became superior to example 3 at about 150 days. Of course your mileage is likely to vary.

CONSTRUCTING THE SSCP IN A DATA STEP WITH PROC FCMP

For those lacking PROC EXPAND, or simply choosing to avoid it, the rolling SSCP data could be also generated directly in a DATA step. While this task could be done entirely with standard data step arrays, we took the opportunity to apply PROC FCMP, which allows the construction of a matrix-oriented subroutine that can be called at any time within a data step. The complete code is in Appendix 2 (the PROC FCMP subroutine code) and Appendix 3 (the data step that calls the subroutine. The basic outline of the data step is as follows:

Example 5a: DATA STEP TO CALL RSSPC SUBROUTINE

```
DATA rwin (type=SSCP keep=id date _TYPE_ _NAME_ Intercept X1 Y)
     / view=rwin ;
  length _NAME_ $32 _TYPE_ $8;
  array var  {2} X Y;
  array vnames {2} $32 _temporary_ ('x','y');

  ** Allow for up tO 25,200 daily values of X and Y **;
  array data{25200,2}   _temporary_;  ** Send to FCMP      **;
  array rssc{25200,2,2} _temporary_;  ** Receive from FCMP **;
  array rsum{25200,2}   _temporary_;  ** Receive from FCMP **;

  array datevar {25200} _temporary_;

  ** Copy complete data series for an ID to _DATA matrix **;
  do nr=1 by 1 until (last.id);
    set MYSERIES (keep=id date x y);
    by id;
    do v=1 to 2; data{nr,v}=var{v}; end;
    datevar{nr}=date;  ** Save the DATE var **;
  end;
```

```
** Ask FCMP subroutine for rolling SSCP and SUM **;
if nr>=90 then call rsscp(nr,90,2,data,rssc,rsum);

  ** Put the results into a "TYPE=SSCP" data set **;

  ... (Remaining code in Appendix 3) ...

proc reg data=rwin noprint outest=results;
  by id date;
  model y=x
quit;
```

This process will work on series with up to 25,200 dates (100 years of trading dates). The data step has 3 stages: (1) copy the model variables into a the matrix named _DATA (needed to transfer the data set to the FCMP subroutine, (2) call the RSSCP subroutine, with parameters to be described below, and (3) take the results of RSSCP (in matrixes RSSC and RSUM) and write an SSCP data set for proc reg. Note this code deals with only variables X and Y, but could easily be modified to have four variables (say Y, X1, X2, X3), mostly by changing the "2" that appears in array bounds and do-loop limits to "4" (and changing the 3rd parameter of rsscp as well).

So the real work takes place in the RSSCP subroutine, which calculates individual squares and cross products, sums them up for each window and stores the results in the RSSC and RSUM matrixes. Parts of the subroutine code follows: :

Example 5b: SUBROUTINE RSSCP IN PROC FCMP

```
proc fcmp outlib=sasuser.temp.subr;
  subroutine rsscp(nobs,ws,nv,_data{*,*},_rssc{*,*,*},_rsum{*,*}) varargs;
  outargs _rssc,_rsum;
  /*  Arguments:                                                  */
  /*  NOBS:        Number of populated rows in _DATA matrix      */
  /*  WS:          Window Size to develop                        */
  /*  NV:          N of variables (columns) in _DATA matrix      */
  /*  _DATA{*,*}   Data Items passed to this subroutine          */
  /*  _RSSC{*,*,*} Rolling SSCP to return,                       */
  /*               with dimensions NOBS,NV,NV                    */
  /*  _RSUM{*,*}   Rolling simple sums to return (NOBS,NV)       */

  /* Generate Squares, Cross-Prods for observation 1 only */
     ... Code in Appendix 2 ...

  /* Starting at obs 2 add the current SQ & CP to previous sums */
  do obs=2 to ws;
    do r=1 to nv;
      _rsum{obs,r} = _rsum(obs-1,r) + _data{obs,r} ;
      _rssc{obs,r,r} = _rssc{obs-1,r,r} + _data{obs,r}**2 ;
      if r<nv then do c=r+1 to nv;
        _rssc{obs,r,c} = _rssc{obs-1,r,c} +_data{obs,r}*_data{obs,c} ;
```

```
      _rssc{obs,c,r} = _rssc{obs,r,c};
    end;
  end;
end;

/* At obs ws+1 start subtracting observations leaving the window*/
if nobs>ws then do obs=ws+1 to nobs;
  do r=1 to nv;
    _rsum{obs,r} = _rsum(obs-1,r) + _data{obs,r} - _data{obs-ws,r};
    _rssc{obs,r,r}=_rssc{obs-1,r,r}+_data{obs,r}**2-_data{obs-ws,r}**2;
    if r<nv then do c=r+1 to nv;
      _rssc{obs,r,c} = _rssc{obs-1,r,c} + _data{obs,r}*_data{obs,c}
                     - _data{obs-ws,r}*_data{obs-ws,c};
      _rssc{obs,c,r} = _rssc{obs,r,c};
    end;
  end;
end;

endsub;
quit;
```

Instead of calculating the sum of 90 values for each (90-day) window, this routine keeps a running total and modifies it for each new observation. For instance the code in italics above calculates the new sum of squares for variable **r** at observation **obs**:

- The prior sum of squares: (_rssc{obs-1,r,r})
- Plus the square of the new observation: (_data{obs,r}**2)
- Minus the square of the observation leaving the window: (_data{obs-ws,r}**2)

Initially it was tempting to use the matrix multiplication capabilities of PROC FCMP in generating SSCP values (i.e. [XY]'[XY] for each window. The programming would be a bit more intuitive, but much less efficient, since each row of the original data would be involved in 90 matrix multiplications (more for larger windows). In this program, each row is subjected to calculation of squares and cross-products only twice: when it enters a window and when it exits a window.

A few tests were run on this program, and like the PROC EXPAND technique the time-to-completion was not effected by increasing window size.

CONCLUSIONS

There are essentially two SAS programming approaches to performing rolling windows regressions: generating a series of rolling windows data to submit to a regression procedure, or generating a series of rolling SSCP data (also to be submitted to a regression procedure. The earlier technique seems to perform well when window sizes are relatively small (less then120 on our platform), but time-to-completion grows proportionately to the window size. The rolling SSCP technique, however, has a virtually constant time-to-completion regardless of the size of the window. The rolling SSCP can be generated in a single data step, using FCMP subroutines.

ACKNOWLEDGMENTS

CONTACT INFORMATION

This is a work in progress. Your comments and questions are valued and encouraged. Please contact the author at:

Author Name:	Mark Keintz
Company:	Wharton Research Data Services
Address:	305 St. Leonard's Court 3819 Chestnut St Philadelphia, PA 19104
Work Phone:	215.898-2160
Fax:	215.573.6073
Email:	mkeintz@wharton.upen.edu

APPENDIX 1: SSCP VIA PROC EXPAND

```
** Step 1: Generate Squares and Cross-Products **;
data vtemp / view=vtemp;
  set myseries;
  xy=x*y;   xx=x*x;   yy=y*y;
run;

** Step 2: Sum the squares and cross-products **;
proc expand data=vtemp out=sscpdata (where=(_n=90)) method=none;
  by id ;
  id date;
  convert Y= _n        / transformin=(*0) transformout=(+1 MOVSUM 90);
  convert x y xy xx yy /                  transformout=(   MOVSUM 90);
run;

** Step 3: Reshape the data into a TYPE=SSCP data set **;
data rsscp (type=SSCP keep = id date _TYPE_ _NAME_ intercept x y)
     / view=rsscp;
  retain id date _TYPE_ _NAME_ intercept x y;
  set sscpdata;
  length _TYPE_ $8  _NAME_ $32 ;

  ** Store for later use **;
  _sumy=y;  _sumx=x;

  _TYPE_="SSCP";  /* For the record type, not the data set type*/
  ** First output record is just N, and sums already in each original variable **;
  _NAME_='Intercept';
    Intercept=_n;
    y=_sumy;  x=_sumx;
    output;
  _name_="X";
    intercept=_sumx;  x=xx;   y=xy;
    output;
  _name_="Y";
    intercept=_sumy;  x=xy;   y=yy;
    output;
  _TYPE_='N';
  _NAME_=' ';
    Intercept = _n;   Y=_N;   X=_N;y
    output;
run;

proc reg data=rsscp noprint outest=results;
  by id date;
  model y=x1;
quit;
```

APPENDIX 2: FCMP ROLLING SSCP SUBROUTINE

```
proc fcmp  outlib=sasuser.temp.subr;
  deletesubr rsscp;
run;

proc fcmp outlib=sasuser.temp.subr;
  subroutine rsscp(nobs,ws,nv,_data{*,*},_rssc{*,*,*},_rsum{*,*}) varargs;
  outargs _rssc,_rsum;
  /*  Arguments:                                                  */
  /*  NOBS:        Number of populated rows in _DATA matrix      */
  /*  WS:          Window Size to develop                        */
  /*  NV:          N of variables (columns) in _DATA matrix      */
  /*  _DATA{*,*}   Data Items passed to this subroutine          */
  /*  _RSSC{*,*,*} Rolling SSCP to return,                       */
  /*               with dimensions NOBS,NV,NV                    */
  /*  _RSUM{*,*}   Rolling simple sums to return (NOBS,NV)       */

  /* Generate Squares, Cross-Prods for row 1 only */
  do obs=1 to 1;
    do  r=1 to nv;
      _rsum{obs,r} = _data{obs,r};
      _rssc{obs,r,r} = _data{obs,r}**2;
      if r<nv then do c=r+1 to nv;
        _rssc{obs,r,c} = _data{obs,r}*_data{obs,c};
        _rssc{obs,c,r} = _rssc{obs,r,c};
      end;
    end;
  end;

  /* Starting at obs 2, add current SQ & CP to previous total  */
  do obs=2 to ws;
    do r=1 to nv;
      _rsum{obs,r} = _rsum(obs-1,r) + _data{obs,r} ;
      _rssc{obs,r,r} = _rssc{obs-1,r,r} + _data{obs,r}**2 ;
      if r<nv then do c=r+1 to nv;
        _rssc{obs,r,c} = _rssc{obs-1,r,c} +_data{obs,r}*_data{obs,c} ;
        _rssc{obs,c,r} = _rssc{obs,r,c};
      end;
    end;
  end;

  /* At obs ws+1 start subtracting observations leaving the window*/
  if nobs>ws then do obs=ws+1 to nobs;
    do r=1 to nv;
      _rsum{obs,r} = _rsum(obs-1,r) + _data{obs,r} - _data{obs-ws,r};
      _rssc{obs,r,r} = _rssc{obs-1,r,r} + _data{obs,r}**2 - _data{obs-ws,r}**2;

      if r<nv then do c=r+1 to nv;
```

```
      _rssc{obs,r,c} = _rssc{obs-1,r,c} + _data{obs,r}*_data{obs,c}
                       - _data{obs-ws,r}*_data{obs-ws,c};
      _rssc{obs,c,r} = _rssc{obs,r,c};
    end;
  end;
end;

endsub;
```

APPENDIX 3: DATA STEP TO CALL FCMP ROLLING SSCP SUBROUTINE

```
data rwin (type=SSCP keep=id date _TYPE_ _NAME_ Intercept X1 Y) / view=rwin ;
  length _NAME_ $32 _TYPE_ $8;
  array var  {2} X Y;
  array vnames {2} $32 _temporary_ ('x','y');

  ** Allow up to 25200 daily values for an ID **;
  array data{25200,2}   _temporary_;  ** Send to FCMP      **;
  array rssc{25200,2,2} _temporary_;  ** Receive from FCMP **;
  array rsum{25200,2}   _temporary_;  ** Receive from FCMP **;
  array datevar {25200} _temporary_;  ** Needed, but not part of Subroutine**;

  ** Copy vars to _DATA matrix **;
  do nr=1 by 1 until (last.id);
    set MYSERIES (keep=id date x y);
    by id;
    do v=1 to 2; data{nr,v}=var{v}; end;
    datevar{nr}=date;  ** Save the DATE var **;
  end;

  ** Ask FCMP subroutine for rolling SSCP and SUM **;
  if nr>=90 then call rsscp(nr,90,2,data,rssc,rsum);

    ** Put the results into a "TYPE=SSCP" data set **;
    if nr>=90 then do I=90 to nr;
      date=datevar{I};
      _type_= 'SSCP';
      ** Write the "Intercept" Line **;
      _name_= "Intercept";
      intercept=90;
      do C=1 to &nv; var{C}=_rsum{I,C}; end;
      output;

      ** Write an sscp line for each variable **;
      do R=1 to 2;
        _NAME_ = VNAMES{R};
        Intercept=_rsum{I,R};
        do C=1 to 2; var{C}=_rssc{I,R,C}; end;
        output;
      end;

      ** Write the "N" line **;
      _type_='N';
      _name_=' ';intercept=90;
      do C=1 to 2; var{C}=90; end;
      output;
    end;
  run;
```

```
proc reg data=rwin noprint outest=results;
  by id date;
  model y=x;
quit;
```