Tips about Using Data Step Option Point access

Jingren Shi, Syntel Inc., Troy, Michigan
Shiling Zhang, Wayne State University, Detroit, Michigan

Abstract

This paper discusses how to use SAS data step option **POINT =** in data manipulation among observations. For example, how to access observations directly from within a data set to get the useful information or how to generate a random sample from a data set. It is not uncommon for an inexperienced SAS programmer taking several steps to manipulation a data set into a desired format when it can be done in a single step. All examples used here are based on questions posted on comp.soft-sys.sas or sas-l.

Terminology and Definition

SAS data step option **POINT =** variable-name reads SAS data sets using random (direct) access by observation number. With the **POINT=** option, one names a temporary variable whose value is the number of the observation one wants the SET statement to read. Because the **POINT =** option reads only those observations specified in the option, the SAS system cannot read an end-of-file indicator as it would if the file were being read sequentially. Therefore one must include a **STOP** statement to stop processing the data step [1]

Program data vector(PDV) [2] **----** Area of memory where the SAS system builds your data set, one observation at a time. When the program executes, data values are read from the input buffer or created by SAS language statements and assigned to the appropriate variables in the program data vector. From here the variables are written to the SAS data set as a single observation.

SAS processes observations sequentially. Point access option is very useful when one wants to re-shape or manipulate a data set out of sequence. It reads observations directly according to the observation number or point. Hence any unnecessary information is not read into the PDV and the performance may be improved.

Here is a typical example of using SAS data step option **POINT =.**

```
Data out;
   Do I = 1, 3, 5
      Set  dsn point=I;
      Output;
   End;
   Stop;
Run;
```

Note the data step only reads observation 1, 3, 5 from the input data set and writes them to the output data set.

Case 1

Using SAS data step option **POINT =** to subset a data set is very efficient when one knows which observations he wants.

Here is an example.

```
51   data t1;
52     do x = 1 to 1e6;
53      output;
54     end;
55   run;

NOTE: The data set WORK.T1 has 1000000 observations and 1 variables.
NOTE: The DATA statement used 4.0 seconds.

56
57   data t2;                          ❶
58     do i = 1,3,5;
59       set t1 point=i;
60       output;
61     end;
62     stop;
63   run;

NOTE: The data set WORK.T2 has 3 observations and 1 variables.
NOTE: The DATA statement used 0.22 seconds.

64
65   data t2;                          ❷
66     set t1;
67     if _n_ in (1 3 5) then output;
68   run;

NOTE: The data set WORK.T2 has 3 observations and 1 variables.
NOTE: The DATA statement used 4.5 seconds.
```

The log file shows that ❶ is 20 times faster than ❷ when it subsets a data set having one variable with 1 million observations.

Case 2

Question: The last observation in a data set has the summary data. How do I output just the LAST observation?

❶ using the END option on the SET statement.

```
data summary;
  set rawtests end = eof;
    if eof then output;
run;
```

❷ using point access option.

```
data summary;
  set rawtests point = nobs nobs = nobs;
    output;
    stop;
run;
```

❶ reads all observations into PDV and writes the last one into summary**.**
❷ only reads and writes the last one from an input data set and an output data set.
❷ is more efficient than ❶.

Another example is how to access the minimum or maximum value of a variable if a data set is sorted by the variable.

If values of a variable are sorted, then the first and the last values are minimum and maximum respectively. Hence it is very efficient to retrieve data with the point access option

```
data out;              ❶
  do i=1, n;           ❷
    set dsn point=i nobs=n;
    output;
  end;
  stop;
run;
```

❶ Output data step has two observations.
❷ it reads and writes only two observations.

Case 3

Question: One sorts a data set in a wrong direction. How can one re-sort the data without using PROC SORT?

In this case, one can use the point access option to access the data from bottom to top. The approach should be better than using PROC SORT.

```
data c_ord;
 do _i=nobs to 1 by -1;
   set w_ord nobs=nobs point=_i;
   output;
 end;
 stop;
run;
```

Case 4

Question: Can anyone give me some suggestions of a systematic sample selection.

A systematic sample selection involves a random selection of a starting point and a given range to determine the location of the next sample. A good approach is to use the point access option.

```
*start point range and value of a step;
%let srange=20;
%let step=20;

data sample(drop= start);
 start=ceil(&srange *ranuni(-1));          ❶
 do _j_=start to nobs by &step;
   set pop nobs=nobs point=_j_;
   output;
 end;
 stop;
run;
```

❶ randomly selects a starting point.

Case 5

Question: Can anyone give me some suggestions as to the proper code for a random sample selection. For instance, I randomly select 10 observations out of a data set that contains 36 observations with no replacement.

This type of question is frequently posted at comp.soft-sys.sas or sas-l.

A usual solution will be,
1) creating a data set that contains a variable, say, ran_x with 36 observations of random values,
2) merging the data set with a sampling data set;
3) sorting it by the ran_x and output the first 10.

This approach involves several steps. It is quite inefficient. One could do it in a single data step with the data step point access. The code is simple and the logic is easy to understand if one has a little knowledge about conditional probability concept. Here is an example to illustrate the idea.

There are three boys and three identical balls except in color. One ball is red and the rest are green. Every boy prefers red one. In order to be fair to everyone they decide to do a random draw from an opaque bag. The rule is that everyone can draw only once

with one ball.

❶ Three boys draw simultaneously.
❷ Three boys draw sequentially with no replacement.

❶ and ❷ are equivalent, i.e., everyone has exactly $\frac{1}{3}$ in probability to have the red ball in both approaches.

Here is a naïve proof. There is a millionth second difference in picking up a ball when three boys draw 'simultaneously'. Hence the 'simultaneously' becomes the 'sequentially'.

The mathematical proof is simple too. Under the sequential approach, the first boy has the red ball with $\frac{1}{3}$ chance and no red ball with $\frac{2}{3}$ chance. The second boy has the red one depending upon the result of what the first boy has. If the first boy has the red ball, the second boy has the red ball in 0 probability. Otherwise he has red one with $\frac{1}{2}$ probability. The probability for the second boy having the red ball is $\frac{1}{3} \times 0 + \frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$. The probability for the third boy having the red ball is $1 - \frac{1}{3} - \frac{1}{3} = \frac{1}{3}$.

Because SAS processes observations sequentially, the above outlined proof can be deemed an implementation of approach ❷.

The following codes is based on approach ❷.

```
data out(drop = size n);
   size = 10;
   n=nobs;
   do i = 1 to nobs;
     if ranuni(-1) le ( size/n ) then do;
       set dsn point = i nobs=nobs;
       output;
       size = size-1;
       if size=0 then stop;
     end ;
     n=n-1 ;
   end;
run;
```

Case 6

Question: One wants to roll over samples to do some rolling regressions. For example, a data set has 60 observations. One wants to do regressions using 1-30 obs, 2-31 obs, 3-32 obs,..., 31-60 obs.

One can use point= access reshape the data easily and clearly first, then one can use a 'by statement' in regression to get all 31 regressions.

```
*create a testing data set;
data t1;
  do x=1 to 60;
     y=2+3*x+rannor(0);
     output;
  end;
run;

*reshape the data set such that;
*each group has 30 consecutive observations;
data t2;
  do flag=1 to 31;
    do i = flag to flag+29;
      set t1 point=i;
      output;
    end;
  end;
  stop;
run;

proc reg data=t2;
  model y=x;
  by flag;
quit;
run;
```

Case 7

Question: How do you do LEAD (as opposed to a lag) in SAS. For example;

```
    data temp;
      set test01;
      xxx = lead1(yyy);
    run;
```

SAS does not have a lead function. Can one emulate the lead function in this example? The answer is positive. Here is a solution of using point access option. The solution is provided by Paul Dorfman on sas-l.

```
data t1;
  retain x 'x' y 'y' n m 0;
  length x y z $8;
  do l = 1 to 1e1;
     z=compress('z'||l);
     output;
  end;
run;

%let leadn=2;
%let leadvar=l;
%let dsn=t1;

data out;
   set &dsn nobs=nobs;
   _i_=_n_+&leadn;
   if (nobs<_i_) then lead_&leadvar =.;
```

```
  else set t1(keep=&leadvar
              rename= (&leadvar=lead_&leadvar ))
              point = _i_;  ❶
run;
```

❶ It accesses the next value of a given variable as its current value.

Case 8

Question: How do you take a product summation without the square terms? i.e.

$$\frac{1}{2}\sum_{i,j=1,i\neq j}^{n} x_i * x_j$$

Using data step point access point is very efficient to handle the problem. Note the looping times will reduce to almost a half if one uses the fact that $x_i * x_j$ is symmetric.

```
*create a testing data set;
data t1;
  do x=1,3,5,7;
     output;
   end;
run;

data t2(keep=y);
  do i = 1 to n;                    ❶
   do point=i to n;
    set t1 point=point nobs=n;
    if point=i then z=x;
    else y + z*x;                   ❷
   end;
  end;
  output;
  stop;
run;
```

❶ Note the looping index i is from 1 to the number of observations and index point is from i to the number of observations because $x_i * x_j$ is symmetric.

❷ It adds up the cross product $x_i * x_j$.

Case 9

Question: One has a very large uncompressed SAS data set. The file is created in production weekly and contains almost 35 million observations and 20 variables. The file is sorted by a 16-digit numeric key variable. The marketing department periodically receives a small list of non-duplicate accounts, somewhere from several tens to several hundred thousand. One has to select only those records from the large file whose accounts coincide with the accounts contained in the small one.

Now we use,

```
PROC SORT DATA=small OUT=osmall;
BY keyvar;
RUN;

DATA VYTJAG;
  MERGE large(IN=K) osmall(IN=L);
  BY keyvar;
  IF K AND L;
RUN;
```

It works fine, as the large file has already been sorted in production, but takes a very long time. I wonder if anyone in the group could indicate a method which could do it much faster. Please note: The large file is not indexed, and one can only read it. All suggestions would be greatly appreciated.

In this case the transaction data set is very SMALL and the base data set is very BIG, and more important, it is sorted.

If one knows the range of the key variable in the transaction data set, then one can limit observations from base data set by applying FIRSTOBS and OBS options, something like,

```
DATA VYTJAG;
   MERGE large(firstobs=&start obs=&end IN=K)
   osmall(IN=L);
   BY keyvar;
   IF K AND L;
RUN;
```

So the problem boils down to 'how to find the range of the key variable in the transaction data set'. A solution is to partition or mark the base data set into hundreds of blocks. By using point access, one may easily find a small range, but big enough to cover the values of the key variable in the transaction data set.

```
%let bnum=1e8;
%let snum=1e4;
%let parts=500;

data t1;
  retain x1-x20 0;
  do id = 1 to &bnum;
     output;
  end;
run;                                ❶

data t2;
  retain y1-y20 0;
```

```
  start = max(int(&snum*ranuni(0)),1);
  do until (cnt > &snum  ) ;
     cnt+1;
     start + ceil(10*ranuni(0));
     id = start;
     output;
  end;
drop cnt start;
run;                                    ❷

data _null_;
  do point=1, nobs;
     set t2 point=point nobs=nobs; cnt+1;
     call symput (compress('pt'||cnt),id);
  end;
  stop;
run;                                    ❸

data _null_;
  i=1;
  do until (i>&bnum);
    set t1 point=i;
      if id < &pt1 then call symput
('start',max(int(i-&bnum/&parts),1));
      if id > &pt2 then do;
         call symput ('end',i);
         stop;
      end;
      i+int(&bnum/&parts);
  end;
  stop;
run;                                    ❹

%put >>>>>>>>>>&pt1 &pt2 &start &end;

data t3;
  merge t1(firstobs=&start obs=&end in=a) t2(in=b);
  by id;
  if a and b;
run;
```

❶ creates the base data set
❷ creates the transaction data set
❸ calculates the range of transaction data set
❹ finds the range in large data set.

If the value range of the key variable in the transaction data set is hundredth of that of the base data set, the above approach should run very fast. However, if the values of the key variables are uniformly distributed on the range of key variable in base data set, the binary search would be preferred. One can find more details in "Professional SAS Programming Secrets"[3].

Conclusion

POINT = access is very useful option. It is more important to know when one should consider using point access option.

1) When one wants to subset a data set and knows which observations are needed in data manipulation. For example, case 1, case 2, and case 4.

2) If program logic is clearer with point access option, one should consider using it.
For example, case 3, case 5, case 6, and case 8.

3) Program performance can be improved. For example, case 3, case 7 and case 9.

However, the point access option has its limitations. The point access is slower that the sequential access in a reading process. It cannot work with
a) SAS data views
b) Compressed SAS data sets
c) The index of a SAS data set
d) A by statement
e) A where= data set option or where statement

When one use it properly, not only will program performance be improved, but program logic is clear as well.

[1] SAS® Language: Reference Version 6, First Edition, Cary, NC: SAS® Institute Inc., 1990, p485.

[2] SAS® Language: Reference Version 6, First Edition, Cary, NC: SAS® Institute Inc., 1990, p16.

[3] Professional SAS Programming Secrets,1991 by Rick Aster & Rhena Seidman, Published by the McGraw-Hill Companies, Inc. p415-423.

Authors:
Jingren Shi
Syntel Inc.
jshi2@ford.com

Shiling Zhang
The Department of Mathematics
Wayne State University
shiling@math.wayne.edu