SESUG Paper 122-2018

When SAS® Doesn't Behave As Expected

Megan McNeill, Rho®; Gregory Weller, Rho

ABSTRACT

SAS does not always behave in a way a programmer would expect. Sometimes this is due to errors in programming, while other times this is due to how SAS processes data and performs computations.

This paper will provide two examples of instances where SAS does not provide the result a programmer was expecting, and give tips on how to program defensively in order to avoid such issues. One example will highlight a case where the result computed from SAS's internal process differs from the mathematically correct result. The other example will touch on a broader issue of SAS processing date variables differently between a data step and a macro. This paper will cover different processing methods and the importance of knowing how SAS behaves in these situations.

INTRODUCTION

Regardless of the amount of experience a programmer has, it is difficult to know how SAS will perform every calculation in every situation. In the overwhelming majority of situations, the results will be as expected. The focus of this paper is on those situations where the output does not match the intended result, and the programmer is left wondering what went wrong.

EXAMPLE 1

This first example shows how the result of a calculation performed by SAS can differ from the mathematically correct value. Consider the data step below:

```
data check;
   var1=2.8;
   var2=3.5;
   var3=(0.8)*var2;
   if var3=var1 then flag=1;
   else flag=0;
run;
```

When you multiply the numbers 3.5 and 0.8, you come up with a value of 2.8. Thus, in the data step above the resulting value of FLAG should be 1. However, this is the dataset that is output by SAS:

	VAR1	VAR2	VAR3	FLAG
1	2.8	3.5	2.8	0

Table 1. Dataset CHECK Output

You can see the value of VAR3 is showing as 2.8, but the value of flag is 0. What is going on here? The following data step provides the answer:

```
data check1(keep=var3 diff);
   set check;
   diff=var3-2.8;
run;
```

If VAR3 = 2.8, then you would expect the value of DIFF to be 0. However, as the output of dataset CHECK1 below indicates, the difference between the actual value of VAR3 and 2.8 is not zero.

	VAR3	DIFF
1	2.8	4.440892E-16

Table 2. Dataset CHECK1 Output

The reason for the difference is a floating point error, something that can occur often when computing. There are many papers and articles which discuss the issue of floating point error. The easiest way to avoid this is to use the ROUND function in SAS when it is practical. In doing so, a programmer can limit the number of decimal places of a numeric result. For example, if you run the following code you will see the value of DIFF comes out as 0:

```
data check2(keep=var3 diff);
   set check;
   diff=round(var3,0.1)-2.8;
run;
```

	VAR3	DIFF
1	2.8	0

Table 3. Dataset CHECK2 Output

EXAMPLE 2

The second example relates to a more general issue of SAS processing variables differently between a data step and a macro. Even for programmers who are more experienced with writing macros, it can be easy to get tripped up trying to use dates or other numeric variables correctly within the macro language. Say for example you would like to write code that is date dependent, as in it will execute one of two ways based on the current date. In the example below, which is based on an actual experience in a clinical trial, a sponsor wants to keep subject treatments blinded until a specific date. Up to and including that date, dummy treatment values based on the RANUNI function are to be used. After that date, the actual treatment assignments are to be used.

To achieve this, a macro was written with the intention of comparing the system date (&sysdate) with the delivery date (15FEB18 in this example). If the system date falls before or on February 15, 2018 the dummy treatment code should execute. If the system date falls after February 15, 2018 the real treatment code should execute. For simplicity, the dummy treatment code utilizes the RANUNI function to randomly generate a number between 0 and 1, using the uniform distribution, for each subject in the demographics dataset. There were two treatment options, and if the number generated was less than 0.5 the first treatment was assigned, while a number greater than or equal to .5 assigned the second treatment. An example of a macro written to accomplish this is shown below:

```
%macro ranuni;
 %if "&sysdate"d <= "15FEB18"d %then %do;
    %put running for dummy treatments;
    x=ranuni(12345);
    if x < .5 then do;
       armcd='A';
       arm='Treatment A';
    end;
    else if x >= 0.5 then do;
       armcd='B';
       arm='Treatment B';
    end;
 %end;
```

```
%else %do;
   armcd=strip(ravetrt_std);
   arm=strip(ravetrt);
%end;
%mend;
```

However, running this code after February 15, 2018 does not always result in the actual treatments being assigned. For example, if you run the code on Mar 5, 2018, the top portion of the code (the code used to generate and assign dummy treatments) still executes. As a way to investigate why this might be happening, you might create a dataset outside of the macro to check what each date variable "&sysdate"d and "15FEB18"d) resolves to. For example, executing the following data step code on March 5, 2018:

```
data test;
   date1=”&sysdate”d;
   date2=”15FEB18”d;
run;
```

shows that each date resolves to a numeric date variable (date1=21248, date2=21230). Since the current date falls after February 15, 2018, and the variables are formatted correctly, why is the dummy treatment code still executing?

As it turns out, what should be tested instead is what each macro variable resolves to, instead of using a data step to check the values. If SAS processes the variables differently between a data step and the macro, this could be where you are going wrong. In fact, if you use %put with each macro variable as shown in the following code:

```
%put “&sysdate”d “15FEB18”d;
```

you will obtain the resulting values of character strings “05MAR18”d and “15FEB18”d. Since it is early into the next month (before the 15th), SAS executes the dummy treatment code because the beginning characters “05 are ‘less’ than the “15 of the expiration date. Once the character by character comparison hits 0 and 1, SAS treats the early March character date as before the mid-February date.

Now that SAS’s ‘unexpected’ behavior is understood, the next step is to get the date variables to resolve to numeric within the macro. To accomplish this, you can use the %SYSEVALF function and apply it to each date variable. Changing the clause from:

```
%if “&sysdate”d <= “15FEB18”d %then %do;
```

to:

```
%if %sysevalf(“&sysdate”d) <= %sysevalf(“15FEB18”d) %then %do;
```

results in the desired comparison of numeric date values 21248 and 21230. Finally, the dates are being compared in the correct format. The dummy treatment code is skipped over since the date falls after February 15, 2018, and the real treatment code executes instead.

It is prudent to be wary of cases when you intend to use numeric variables within a macro. In our case of intending to use date variables for a comparison within a macro, SAS processing these variables differently in a data step was initially not accounted for. A check of the resulting values within a data step would only further confuse you until it is realized you should check the behavior of these variables when processed in a macro.

CONCLUSION

It is important for programmers to be aware that SAS will not always produce the results they expect. In the example of encountering floating point error, a programmer must recognize this type of error can occur quite frequently. Making use of the ROUND function will help avoid this issue. In the example of macro variable resolution, programmers must be aware that SAS processes variables differently depending on the context. Programmers should test the resulting values consistently within the same

context to ensure they are achieving the desired result. For example, as a check programmers should test the resolution of a variable used within a macro by using a %put statement, and not compare this to the resolution of the variable within a data step. When needing to use a variable as numeric within a macro, programmers must use a function such as %SYSEVALF if the variable is not already in numeric format, such as in our example of needing a numeric date.

CONTACT INFORMATION

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

Megan McNeill
Rho
Megan_McNeill@rhoworld.com

Gregory Weller
Rho
Greg_Weller@rhoworld.com