Paper 248

A Generalized Rounding Alternative

Jeff Palmer, Statistics Collaborative, Inc., Washington, DC

ABSTRACT

It is important to understand the consequences of using the ROUND function in the SAS® System. By default, numbers with a trailing 5 are rounded away from zero (i.e., 0.15 to 0.2 and -0.25 to -0.3), neglecting any numerical precision errors that a computer's hardware limitations may introduce. In some instances, this type of rounding can lead to an overestimation of the true mean of a set of numbers, as illustrated by the simulation described in this paper. The macro code presented in this paper eliminates this bias by randomly rounding up or down, depending on the outcome of a normally distributed random variable.

BACKGROUND

PRECISION ERROR

Numerical precision errors can sometimes produce rounding results that do not match up with, what some people term, your "pencil-and-paper" results. The first problem we encounter is the computer's limitation in its ability to represent the infinite domain of the real number system. The computational answer to this problem is found in *floating point arithmetic*, where an arbitrary number, *x*, is represented by the expression:

$$x = \pm\, 0.d_1 d_2 \ldots d_k \times 10^n.$$

Here, *k* is the number of *significant digits*, and *n* is the *exponent* of the number *x*. Suppose we try to represent an irrational number such as π with this expression. Our resultant number will undoubtedly be truncated after *k* significant digits, and will not be the *true* value we are after, no matter how "precise" we get.

Please note that the level of precision, or number of significant digits, used in floating point arithmetic, depends on the hardware specifications of your computer and not on the computational abilities of the SAS System.

With this background in mind, we will now focus on how SAS's ROUND function handles numerical precision. The SAS System uses the exact number straight from memory, while some other software applications round the number stored in memory behind the scenes before performing any calculations. This internal rounding procedure, commonly referred to as adding a "fuzz factor", is used to offset potential calculation errors resulting from numerical imprecision.

In the following hypothetical table, suppose we want to use the ROUND function to round each value to the nearest 10^{-3} (thousandth). As a default rule in SAS, we would expect the numbers to be rounded up when there is a trailing 5:

Obs	Value on screen	Value in memory
1	0.09040000	0.09040…001
2	0.09050000	0.09049…999
3	0.09940000	0.09940…001
4	0.09950000	0.09950…001

When we use the ROUND function on these values, we would expect observations 2 and 4 to both be rounded up to 0.091 and 0.100, respectively. Unfortunately, because of the numerical precision errors inherent in this example, we would end up with 0.090 and 0.100 instead.

The macro %NEWROUND provided as an attachment here solves the numerical precision problem of rounding in SAS by using a fuzz factor. This fuzz factor is implemented by adjusting the input value by one-tenth of the desired rounding level to ensure that the value is rounded in the expected direction.

BIASED ROUNDING

What is this expected direction of rounding? In addition to potential precision errors, always rounding away from zero in the trailing 5 situation leads to bias.

For example, if it is your job to report any trends of abnormally low serum sodium levels for subjects in a randomized clinical trial, then you do not want to overestimate and potentially overlook possibly significant results by consistently rounding your values up. If the levels are recorded to the nearest tenth, and you are reporting them to the nearest integer, then any summary statistics you may present on the distribution of the rounded numbers may be biased.

The simulation presented here will illustrate the difference in effect between rounding up and randomly rounding on a normally distributed random sample.

SOLUTION

MACRO CODE

The attached macro %NEWROUND (Attachment 2) can be used as a regular procedural statement in the body of a data step. The call to the macro should be in the form:

%NEWROUND(<**variable**>, <**roundoffunit**>);

where the input parameter **variable** is the name of the variable to be rounded, and **roundoffunit** is the unit. The unit level is in terms of the number of places to the right of the decimal point you are rounding to (*i.e.*, a precision level of 2 will round your variable to the nearest 10^{-2}, and a precision level of -3 will round to the nearest 10^{3}).

Note that the call to this macro differs from the functional format of the call to the conventional ROUND function:

<**new variable**>=ROUND(<**variable**>,<**roundoffunit**>);.

Also note that, by using this macro, we eliminate the aforementioned precision error with the introduction of a fuzz factor. The algorithm adds (subtracts), to the input variable, the following adjustment level which is equal to 1 tenth of the desired outcome precision:

```
+(sign(rannor(0)))*10**(-(&PREC + 1));
```

The macro variable **&PREC** is the value given by the user for the **roundoffunit**.

For example, if we are trying to round the value 0.1555 to the nearest thousandth, the macro will first adjust this value by adding (subtracting) 0.0001, and then use the

ROUND function to arrive at the desired result.

As a follow up to the previous discussion about biased rounding, this fuzz factor randomly chooses the direction in which to round a number with a trailing 5. The symmetry of the normal distribution (with mean $\mu = 0$) about the y-axis implies that the call to the RANNOR function (with a seed of 0) is expected to return a negative value half of the time. The sign of this result determines whether the value will be rounded up or rounded down. This provides us with our “generalized” alternative to SAS’s default of always rounding away from zero.

The user of this macro must be sure that the variables _SUBSTR_ and _TEMP_ do not already exist in the data set from which the macro is called.

EXPECTATION

Under certain assumptions, it is possible to compute the exact expectation of the difference in the behavior between the ROUND function and the %NEWROUND macro. The most extreme case occurs when you measure a variable to a certain digit of precision then round to one digit of precision less. (*i.e.*, measure your variable to two decimal points, then round to one.) In this case, assuming that each of the ten digits is equally likely to appear in the second decimal position, the expected probability of a 5 appearing in this position is 0.1.

Now we are faced with the choice of rounding this variable away from zero, or rounding in an arbitrary direction. If we also restrict ourselves to positive numbers, then we would expect the two methods to produce dissimilar results exactly 5% of the time.

SIMULATION

The simulation described here is provided mainly as a visual aid. It shows that it is possible to alter a distribution inadvertently by applying the ROUND function to a variable.

First, we produce 20 normally distributed random values with mean 100 and variance 1. These values are generated using SAS’s random number generator in the form of the RANNOR function and are stored to the nearest hundredth. We expect that the digit 5 will show up 10 percent of the time in the second decimal position.

The next step is to create another variable by rounding this original sample to the nearest tenth using the ROUND function. We now run a t-test comparing the two distributions and record the resultant t-statistic. We iterate this procedure 5,000 times. Next, we repeat this entire process again, except this time, using the %NEWROUND macro instead of the ROUND function.

The attached graphic (Attachment 1) shows what happens to the distribution of these t-statistics when we run this simulation on samples of size N = 20, 40, 100, and 1000. The worst case occurs when we run the simulation on a sample of size N = 1000. The graph illustrates how the deviation between the two distributions (for ROUND and %NEWROUND) increases as we increase the sample size.

CONCLUSION

Programmers must understand the computational limitations of their computer and the process by which their software handles such limitations. SAS users should recognize that SAS does not automatically adjust values in a way that will always produce the correct result when numbers are rounded. SAS does, however, automatically choose the direction in which it rounds when it encounters a trailing 5. This paper offers a simple macro as a solution to the computational limitation problem and an alternative to the default rounding rule.

REFERENCES

Grossman, Stanley I., *Elementary Linear Algebra, Fifth Edition*, Fort Worth, TX. Sanders College Publishing, 1994, p.A-19.

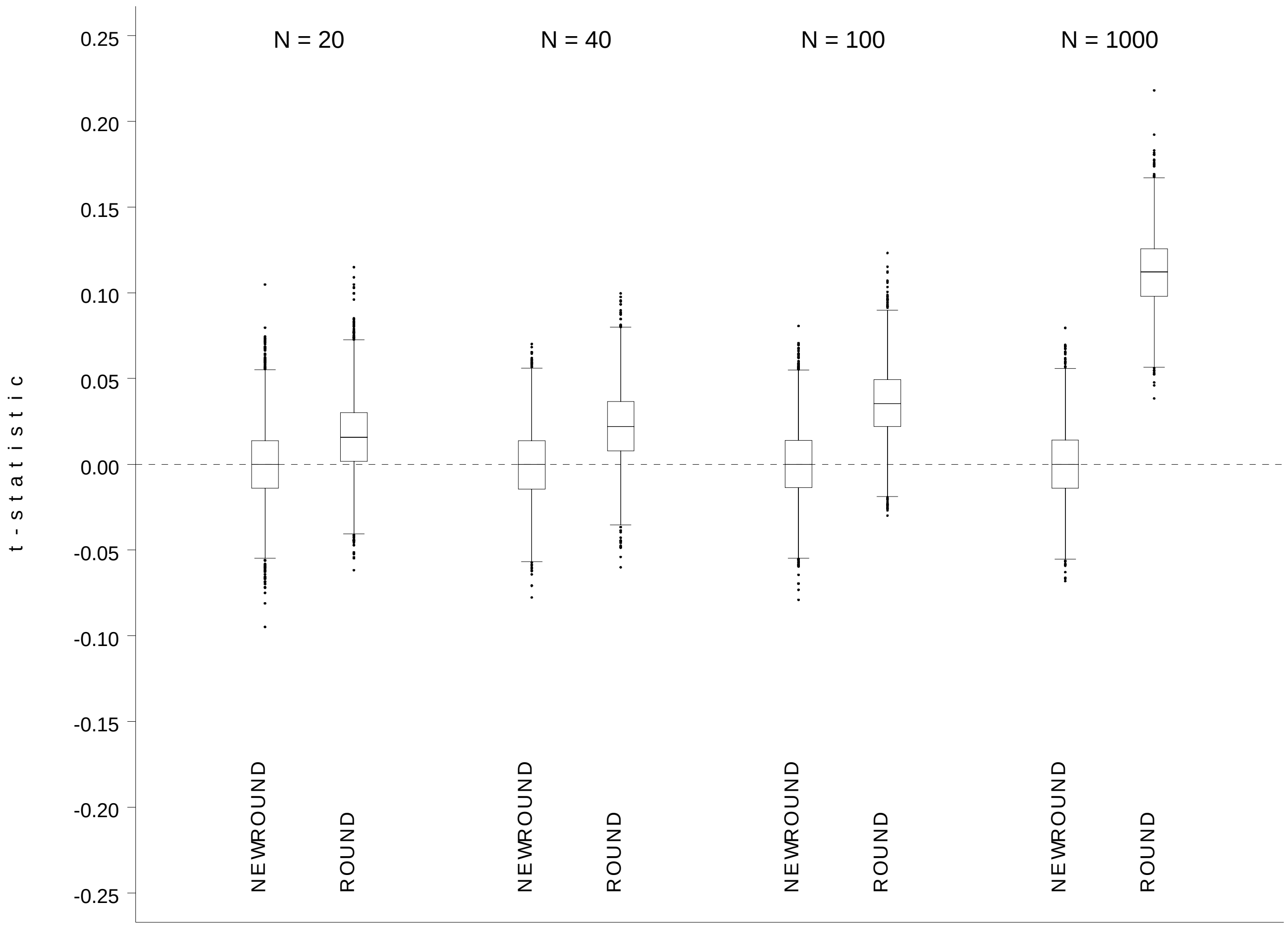
SAS Institute Inc., *Combining and Modifying SAS Data Sets: Examples, Version 6, First Edition,* Cary, NC: SAS Institute Inc., 1995. p.137.

CONTACT INFORMATION

Any comments, suggestions, or requests for documentation of the simulation, in response to this poster should be forwarded to:

Jeff Palmer
Statistics Collaborative, Inc.
1710 Rhode Island Ave. NW, Suite 200
Washington, DC
(202) 429 - 9267
jeff@statcollab.com

ATTACHMENT 1: DISTRIBUTIONS OF t-STATISTICS FROM SIMULATION



ATTACHMENT 2: MACRO CODE FOR %NEWROUND

```
%macro newround (VAR, PREC);

  if . < &PREC < 0 then _SUBSTR_ = abs(&PREC);
  else _SUBSTR_ = 1;

  if  ((substr(reverse(round(&VAR, 10**(-%eval(&PREC + 1)))), _SUBSTR_, 1) = "5") AND
       (reverse(round(&VAR, 10**(-%eval(&PREC + 1)))) = reverse(&VAR))) then do;

        _TEMP_ = &VAR + (sign(rannor(0)))*10**(-(&PREC + 1));
        _TEMP_ = round(_TEMP_, 10**(-%eval(&PREC)));
  end;

  else _TEMP_ = round(&VAR, 10**(-%eval(&PREC)));
  &VAR = _TEMP_;

  drop _TEMP_ _SUBSTR_;

%mend newround;
```