PharmaSUG 2026 - Paper AP-108

The Problems Surrounding Rounding

Jennifer McGrogan, Priovant Therapeutics;
Mario Widel, Independent Contractor

ABSTRACT

Rounding is not a novel concept—examples can be found in ancient civilizations, such as approximations made by the Mesopotamians. Since then, the need for rounding has not diminished, it has rather increased. It was necessary long before the use of computers for numerical calculations and introducing their use has made the rounding process more complex and indispensable. Based on the referenced literature and our own experience, we will show typical problems including:

1. Rounding is unavoidable, for
    a. table readability,
    b. validation of results,
    c. keeping results within reasonable precision, and
    d. ensuring accurate assignment of CTCAE toxicity grades.
2. Computer representation of numeric results may
    a. affect number precision,
    b. be inadvertently rounded,
    c. introduce calculation errors due to rounding, or
    d. cause compare differences on numbers that appear identical.

Throughout this paper we will cover some impacts as well as mitigations and solutions with respect to clinical analysis.

INTRODUCTION

Rounding is an inherent necessity when performing calculations using a computer. The publications included in the references cover a wide variety of problems and solutions related to rounding, in particular – The Table Maker's Dilemma, 1988. The dilemma in question is a simple concept; how to always produce correctly rounded results when computing elementary functions. The solution is not as simple as it may seem, and this problem still exists today, particularly so for any programmer who is performing clinical analyses and presenting their results.

Rounding and its associated problems have existed almost as long as numbers themselves. The introduction of computers to perform numerical calculations did not resolve these rounding problems, if anything the issues became more apparent. This is due to the computer's internal representation of numbers. From the original introduction of computers, each manufacturer used their own conventions to represent fractions, resorting to some flavor of fixed and/or floating point. It was not until 1985 when the Institute of Electrical and Electronics Engineers introduced the IEEE-754 document, where a standard format for floating point number representation is described. After this point manufacturers had a common standard to rely on for this representation, and for the most part adhere to this standard today. In particular, SAS uses the IEEE-754 representation of double-precision 64-bits for all numbers.

ROUNDING IS UNAVOIDABLE

TABLE READABILITY vs ACCURACY

As mentioned in the publication, "The Table Maker's Dilemma is the problem of always getting correctly rounded results when computing the elementary functions." Statistical programmers face this dilemma

every day at their jobs – when producing clinical results, a programmer is performing mathematical calculations using a computer, and presenting those results on a consolidated table, where they must be reasonable and accurate, displayed with appropriate precision, but then still fit on the space provided, typically an electronic document that mimics a piece of paper.

Independently from performing any calculations that may be part of the analysis, the computer has often already approximated the numbers so they can be represented within the software. Any rounding the programmer sets up will be applied in addition to the inherent approximation.

VALIDATION OF RESULTS

Due to the computer floating point representation (for which we will provide details), the comparison of results may fail for numbers that look the same and are actually just very similar in value. Examples of this can be found in Shaoji Xu's publication, where the author shows and explains an example where adding 0.1 + 0.2 does not equal 0.3.

An interesting phenomenon occurs when performing double-programming validation, and the production and validator use different methods to produce results; there may be cases where those results, although very similar, are not identical and do not cleanly compare.

Below is a modified example from the SAS documentation to illustrate a simple case of performing calculations in different orders. Imagine that the variable SUM1 is produced by the production programmer and SUM2 is produced by the validation programmer. Both of these calculations are reasonably following the same specifications but will not produce identical numbers.

Figure 1. Order of Operations Example (SAS)

```
data test;

      /* Assign values */
       x1 = -1./3.;
      x2 = 22./7.;
      x3 = -1234567891.;
      x4 = 1234567890.;

      /* Add the numbers in two different orders. */
      sum1 = x1 + x2 + x3 + x4;
      sum2 = x4 + x3 + x2 + x1;

      put sum1=;
      put sum2=;

      format x: sum: hex16.;

run;
```

Running this sample code will produce the following messages in the log, verifying that although SUM1 and SUM2 are each created as the sum of the same 4 numbers, when added in different orders, they produce different results that will not cleanly compare. Note the syntax – decimals have been included on the assignment of values so that SAS is forced not to process them as integers. This is not explicitly necessary here as SAS will process integers and floating-point in the same manner, but this may be relevant as you test this in other languages.

Figure 2. SAS logs from Order of Operations Example

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
72
73         data test;
74
75               /* Assign values */
76               x1 = -1./3.;
77               x2 = 22./7.;
78               x3 = -1234567891.;
79               x4 = 1234567890.;
80
81               /* Add the numbers in two different orders. */
82               sum1 = x1 + x2 + x3 + x4;
83               sum2 = x4 + x3 + x2 + x1;
84
85               put sum1=;
86               put sum2=;
87
88               format x: sum: hex16.;
89
90         run;

sum1=3FFCF3CF40000000
sum2=3FFCF3CF3CF3CF3D
```

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
72
73         data test;
74
75               /* Assign values */
76               x1 = -1./3.;
77               x2 = 22./7.;
78               x3 = -1234567891.;
79               x4 = 1234567890.;
80
81               /* Add the numbers in two different orders. */
82               sum1 = x1 + x2 + x3 + x4;
83               sum2 = x4 + x3 + x2 + x1;
84
85               put sum1=;
86               put sum2=;
87
88               format x: sum: 24.22;
89
90         run;

sum1=1.8095238208770700000000
sum2=1.8095238095238000000000
```

In the left panel of Figure 2, the values of SUM1 and SUM2 are displayed in HEX16 format, and in the right panel the values are displayed in standard numeric format of one digit per byte. In both logs it is clear that although the numeric value begins with digits that match, after a certain number of digits the numeric approximation of the values in SUM1 and SUM2 begin to diverge. One can imagine the issues this would cause while attempting to achieve a clean compare in clinical programming, despite the fact that the values in SUM1 and SUM2 are reasonably similar.

This impact on the final results is not caused specifically by the SAS software, and instances of this can be found in other programming languages as well. R is another software that is often used by statistical programmers, particularly for the ability to create data visualizations. Completing this exercise in R will yield the same results that we saw in the SAS program.

Figure 3. Order of Operations Example (R)

```
> x1 <- -1./3.
> x2 <- 22./7.
> x3 <- -1234567891.
> x4 <- 1234567890.
> s1 <- x1 + x2 + x3 + x4
> s2 <- x4 + x3 + x2 + x1
>
> s1 == s2
[1] FALSE
```

Shown above is a representation of the R console after submitting the same calculation performed in SAS. Like SUM1 and SUM2, S1 and S2 in this example do not cleanly compare due to having been calculated in slightly different manners.

To provide additional examples, this behavior is not limited to any specific programming softwares, rather applies to any software using floating point representation. The same results are yielded even in Excel, and Python, as illustrated below.

Figure 4. Order of Operations Example (Microsoft Excel)

	A	B	C
1	Label	Value	Excel Text
2	X1:	-0.333333333	=-1/3
3	X2:	3.142857143	=22/7
4	X3:	-1234567891	=-1234567891
5	X4:	1234567890	=1234567890
6	S1:	1.809523821	=B2+B3+B4+B5
7	S2:	1.80952381	=B5+B4+B3+B2
8	S1 = S2:	FALSE	=IF(B7=B6,TRUE,FALSE)

Figure 5. Order of Operations Example (Python)

```
>>> x1 = -1./3.
>>> x2 = 22./7.
>>> x3 = -1234567891.
>>> x4 =  1234567890.
>>> a = x1+x2+x3+x4
>>> a
1.8095238208770752
>>> b = x4+x3+x2+x1
>>> b
1.8095238095238095
>>> a == b
False
>>>
```

Interestingly, a similar inequality occurs when using the same numbers but applying multiplication instead of addition. Furthermore, the values will still not match if rounded to 9 decimal places, which is a commonly used strategy to avoid these issues. In this scenario, it will match when rounded to 6, so a clean compare is achievable, but we recommend not using a general solution of always rounding to 9.

KEEPING RESULTS WITHIN REASONABLE PRECISION

When presenting results, the values must represent the true value, applying a reasonable level of precision, that is, an appropriate number of significant figures for the measurement being displayed. However, when programming intermediate steps, such as computations, derivations, or results that are put into datasets for storage, rounding must be applied very carefully. Rounding too early, or too much, may lead to catastrophic differences.

Some guidance for rounding can be found in the PhUSE working group forum titled *SDTM/ADaM IG Nuances*. See below an excerpt from that document:

> "Storing a rounded value in AVAL is not good practice as it typically results in a loss of precision for calculations in the tables. Storing rounded values in AVALC goes against the ADaM rule that there has to be a 1-1 mapping of AVAL to AVALC."

Overall, be careful when converting units. While there is strength in keeping trailing digits per PhUSE, if no values are rounded when you have performed a transformation to the data (like a change of units), you may be sending an FDA reviewer values that seem to have magically acquired an enormous amount of precision. It is ultimately most important that final values are reasonable and accurate.

ENSURING ACCURATE ASSIGNMENT OF CTCAE TOXICITY GRADES

When converting lab units, rounding can lead to an incorrect assignment of normality status – this is particularly illustrated when calculating CTCAE grades. The National Cancer Institute publishes Common Terminology Criteria for Adverse Events to dictate a standard for assignment of severity grades. Items are graded along a scale ranging from one to five, or "mildly abnormal" to "death". The grading criteria is dictated by various ranges in various units, for example, Grade 1 for CTCAE Term Anemia is described as Hemoglobin (Hgb) < LLN – 10.0 g/dL; < LLN – 6.2 mmol/L; < LLN – 100 g/L. Depending on the units used to collect hemoglobin and where the value falls in the range, a lab can be assigned as meeting the criteria for Grade 1 or not.

As an example of how rounding can introduce some complexity to this conversation, let us review the grading criteria for hypercalcemia based on corrected serum calcium from version 6.0 of *CTCAE*:

Figure 6. Excerpt of Hypercalcemia grading criteria from CTCAE version 6.0

Hypercalcemia

Grade 1	Grade 2	Grade 3	Grade 4	Grade 5
Corrected serum calcium of >ULN - 11.5 mg/dL; >ULN - 2.9 mmol/L; Ionized calcium >ULN - 1.5 mmol/L	Corrected serum calcium of >11.5 - 12.5 mg/dL; >2.9 - 3.1 mmol/L; Ionized calcium >1.5 - 1.6 mmol/L; symptomatic	Corrected serum calcium of >12.5 - 13.5 mg/dL; >3.1 - 3.4 mmol/L; Ionized calcium >1.6 - 1.8 mmol/L; hospitalization indicated	Corrected serum calcium of >13.5 mg/dL; >3.4 mmol/L; Ionized calcium >1.8 mmol/L; life-threatening consequences	Death

Definition: A disorder characterized by laboratory test results that indicate an elevation in the concentration of calcium (corrected for albumin) in blood.

Note Grade 3 for this term – 3.4 mmol/L, by definition, is a value corresponding to Grade 3 Hypercalcemia. The conversion factor from mmol/L to mg/dL is 4.007. Taking the 3.4 mmol/L value and applying this conversion factor:

$$3.4 * 4.007 = 13.6238\ mg/dL$$

$$round(3.4 * 4.007, 1) = 13.6\ mg/dL$$

Note that Grade 4 for this term is defined as values greater than 13.5 mg/dL. Thus, simply by converting this item, and not advertently or purposefully modifying the inherent value, the same adverse event will have changed grades during this conversion process.

Similarly, but in the opposite direction, let us review the grading criteria for hypocalcemia using the same lab assessment.

Figure 7. Excerpt of Hypocalcemia grading criteria from CTCAE version 6.0

Hypocalcemia

Grade 1	Grade 2	Grade 3	Grade 4	Grade 5
Corrected serum calcium of <LLN - 8.0 mg/dL; <LLN - 2.0 mmol/L; Ionized calcium <LLN - 1.0 mmol/L	Corrected serum calcium of <8.0 - 7.0 mg/dL; <2.0 - 1.75 mmol/L; Ionized calcium <1.0 - 0.9 mmol/L; symptomatic	Corrected serum calcium of <7.0 - 6.0 mg/dL; <1.75 - 1.5 mmol/L; Ionized calcium <0.9 - 0.8 mmol/L; hospitalization indicated	Corrected serum calcium of <6.0 mg/dL; <1.5 mmol/L; Ionized calcium <0.8 mmol/L; life-threatening consequences	Death

Definition: A disorder characterized by laboratory test results that indicate a low concentration of calcium (corrected for albumin) in the blood.

Hypocalcemia defines a range of less than 8.0 to exactly 7.0 mg/dL corrected serum calcium corresponding to Grade 2. Performing a similar calculation to above yields the following. Keep in mind, based on everything we have discussed so far, that dividing by a number may not produce the exact same results as multiplying by the inverse, which is what we have included in our calculations below.

$$8.0 * 1/4.007 = 1.9965061143\ mmol/L$$

$$round(8.0 * 1/4.007, 1) = 2.0\ mmol/L$$

Note the conversion factor used above, 4.007. Based on the information provided in the CTCAE criteria, it is apparent that the conversion factor used was rounded to 4. This is another consequence of rounding to keep in mind – in addition to any rounding you apply, and inherent approximations made by the computer, values can sometimes be rounded by other users. This also needs to be kept in mind so that grades can be carefully and accurately assigned.

This problem is described in detail by Carol Matthews' paper *Assigning NCI CTC Grades to Laboratory Results*. Please refer to this for a greater discussion on potential issues and resolutions. Note as well that a similar problem may arise regarding assigning normality.

Like other points that this paper has made thus far, this behavior is not introduced by SAS software and can be found in any software that uses floating point arithmetic – thus this will be encountered when performing calculations in R, Python, Excel, etc.

COMPUTER REPRESENTATION OF NUMERIC RESULTS

NUMBER PRECISION

Sections 7.4 and 7.5 of the IEEE-754 cover Overflow and Underflow, respectively, when a computer is processing numeric results. Essentially, overflow occurs when the absolute value of a number is calculated that is too large to be properly represented by a computer, whereas underflow is the converse situation, where a number is trending so close to zero that the computer cannot accurately represent how small it is. While the approximations produced are not truly accurate due to this limitation, every software that calculates numeric results will have the same, or similar problems. Thus, the approximations are considered true enough for practical purposes.

While it can be helpful to understand this behavior, effectively there is no need to address it. Typically for work on clinical trials, the numbers most often exist between these bounds and overflow and underflow errors are not applicable.

In Figures 1 through 5 (Order of Operations, above) an example is illustrated using a set of 4 numbers of varying precision, that when summed in different orders produce different results that will not compare cleanly. It is displayed that this phenomenon will occur regardless of the software being utilized, producing these same results in all tested softwares. In general, any software using floating point representation is susceptible to this rounding error.

Similar behavior is explained in Xu's publication *Is .1 + .2 Equal to .3?* In the described example, he finds that when different calculations are used to reach 0.3 (as shown by the default format), the decimal value may appear similar but there may be differences which can be more easily seen in the hexadecimal format.

In the example below, the SAS variable VAR1 is calculated as 0.1 + 0.1 + 0.1. VAR2, however, is just assigned to the value of 0.3. While the default SAS format makes it appear as if these values may be identical, we can tell from the hexadecimal versions of these values (H_VAR1, H_VAR2) that these values are not exactly the same.

Figure 8. SAS Display Example 1

Total rows: 1 Total columns: 4 Rows 1-1

	var1	h_var1	var2	h_var2
1	0.3	3FD3333333333334	0.3	3FD3333333333333

It is notable that if the programmer is to increase the number of decimal places on the original result, it will not make this issue visible. The values will still appear to be identical, so it is easiest to catch this issue by converting the values to hexadecimal format, as shown below.

Figure 9. SAS Display Example 2

Total rows: 2 Total columns: 4 Rows 1-2

	category	var	h_var	d_var
1	0.1 + 0.1 + 0.1	0.3	3FD3333333333334	0.3000000000000000000000
2	0.3	0.3	3FD3333333333333	0.3000000000000000000000

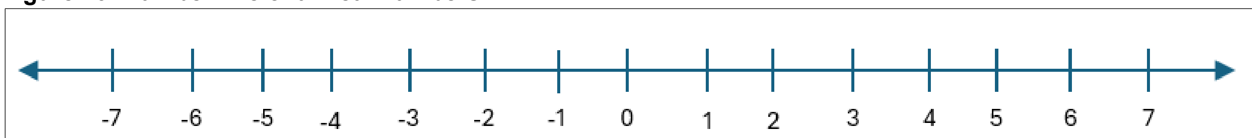
While this difference in values will occur the same in R as it does in SAS, in R it may be visible in the decimal version of the numbers whereas in SAS it was not. When performing the same calculations in R, the difference is visible in the decimal result if it is displayed to the 60th digit. This is helpful, as R does not allow you to apply a hexadecimal format like SAS does.

INADVERTENT ROUNDING

We have discussed various scenarios that can arise when performing numeric calculations using floating point representation but have not yet addressed why this behavior occurs.

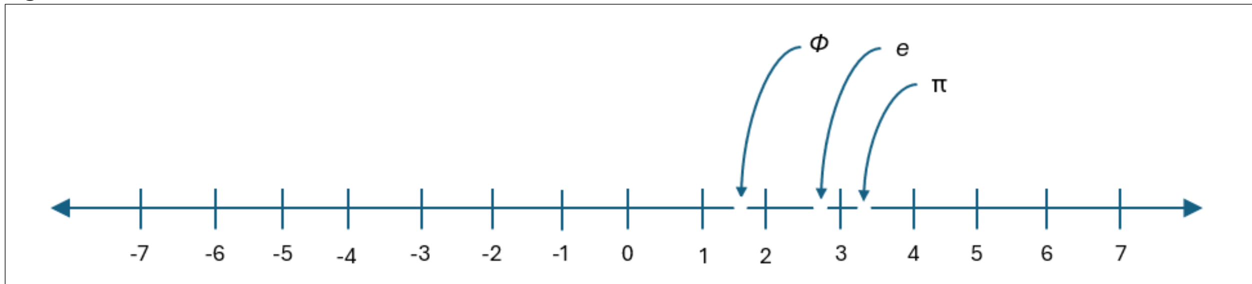
Visualize the number line, continuously representing the set of all real numbers. The line is horizontal and continuous, containing all rational and irrational numbers as values on the line.

Figure 10. Number line of all real numbers



Consider the concept of the number line and its relationship to different sets of numbers, such as the rational set (Q). Although from a high-level the rational number line may look similar to the real number line, upon zooming in there are gaps in the values and the line is not continuous.

Figure 11. Number line of rational numbers

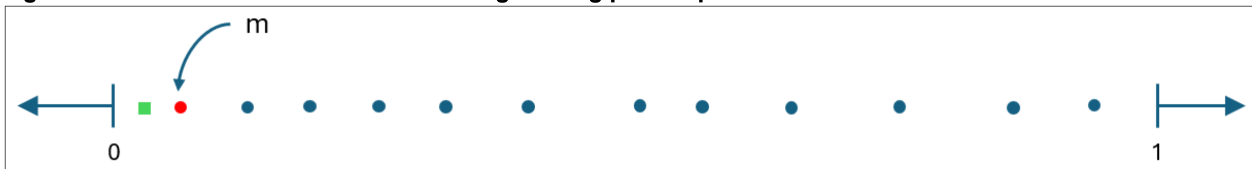


While this does not represent all of the gaps, some samples of the gaps are included here. Understand that these gaps span the number line as far as the represented values.

Now apply the same logic to computer floating point representation, but note that the representable values are fewer, with larger gaps in between. This is due to the restrictions we have explained regarding display of results of numeric calculations; a computer can only represent the full set of values which can be represented as a linear combination of the power of order two. If needed to represent a value for which that is not true, the value will be approximated due to the finite number of bits allowed to represent it.

Consider below the visualization of a number line of values that can be represented without approximation by a computer. Note that gaps exist between every pair of integers but consider 0 and 1 for the sake of example.

Figure 12. Number line of exact values using floating point representation



While this image is notably not to scale, imagine the red dot, *m*, is the smallest, exactly representable, non-zero value by a computer, while the unlabeled blue dots are other exactly representable, greater-than-zero values. Any actual value between 0 and *m*, such as the green square, will be approximated to either 0 or *m*, depending on which is closer, when attempted to be stored in the computer. This is not unique to this segment of the number line – this behavior will span the entire number line of values.

Note that most of the number of possible values from real life, when stored in a computer using floating point representation, will be mapped into a finite number of sequences of bits. While some of them will have an exact representation, most real-life values will be approximated. However, for most practical purposes, these approximations are perfectly adequate.

There do exist real-life values which cannot be represented or approximated by the computer at all. Not a Number is a value that cannot be represented or approximated. Values that would be considered Not a Number are those which are too large or too small to be represented with the available bits. For example, consider assigning a value of 10^{2000} in different computing softwares.

When this calculation is attempted in a SAS data step, it does not produce a result, and the user will receive a message in the log that states *NOTE: Invalid argument(s) to the exponential operator "**"*. This is similar to Excel behavior, which produces a #NUM! error when this calculation is attempted. When it is instead attempted in R, it will actually produce a non-numeric result of 'Inf', but there will be no message or warning associated with this. This value can be used in further calculations, so it will not prevent R from producing results like in SAS, however, if infinite Not a Number values are encountered in clinical work, the program likely needs to be investigated. Notably, when calculating 10^{2000} in Python, it does produce a numeric result due to a difference in software architecture. For the interested reader, more detailed

information about floating point representation and its associated values in general can be found in the IEEE-754 standard.

CALCULATION ERRORS DUE TO ROUNDING

Calculation errors due to rounding are going to occur very often with categorizing values existing at range thresholds. As displayed in a previous example, section *Ensuring Accurate Assignment of CTCAE Toxicity Grades*, depending on the precision used in calculating CTCAE grades, different grades can be produced for the same results. This, however, does not necessarily imply that either grade could be true. In fact, it acts as a counterargument to the best practice of not storing rounded values in AVAL – it would not be appropriate to shift a laboratory grade based on an improperly precise value having a numeric range applied. Additionally, by rounding too much the grade may also be shifted. Work with caution as you are programming complex numeric derivations and verify with each step that whatever precision being utilized is reasonable for the given assessment to ensure the proper grades are being assigned.

COMPARE DIFFERENCES

With all of these various rounding and approximation behaviors discussed, a clinical programmer is likely asking a singular question – how will this realistically impact my work? Most probably the answer lies in comparison of production and validation data.

Knowing that these behaviors occur does not make a comparison of values moot – rather it provides context for situations where there is otherwise no explanation for why visually identical numbers are not producing a clean compare. We recommend in particular to use a strategy that we illustrated above (see text surrounding Figure 9) and modify the format of the value to identify where the discrepancy is occurring. Once identified, determine through whatever means are available if this is reasonably a difference caused by how the computer processes the numbers and thus may be resolved by appropriate rounding, or if this is a true difference that needs to be addressed in the calculations.

SOLUTIONS, MITIGATIONS, AND MUSINGS

Based on the introduction of IEEE-754 and the general adoption of this standard by computer softwares, most calculations performed by computers will be using floating point representation. However, note that the reason for this behavior is actually caused by a finite number of bits to represent any numeric value. Therefore, this may also occur using fixed point representation as well. There are systems that can overcome this problem, but they are not practical for clinical purposes.

Let us reconsider the compare issues displayed in Figures 1 through 5. Recall that despite the same calculations being performed with the same raw values, the resulting numbers did not match due to the difference in order of operations. While enlightening to understand why this happens in computer approximation of numeric results, it is not very helpful in a clinical programming environment where dual programming is often the standard and clean compares are an important step to producing validated results. So, what can practically be done?

As a defensive strategy, it is always good practice, although the feasibility is questionable, for programmers to add numbers in order of increasing magnitude to ensure calculating the result with the highest level of accuracy. The reason for the loss of precision when adding numbers in order of *decreasing* magnitude is due to where the approximation is forced to occur. By adding the values in order of increasing magnitude, you will also produce the value that is closest to the true result.

When ordering operations similarly does not resolve the compare differences, programmers have other strategies available in their toolbelt. See Richann Watson and Louise Hadden's paper, *Quick, Call the Fuzz!* for an in-depth discussion on using COMPFUZZ logic to achieve a clean compare.

The most important takeaway is to be aware that this can happen and while solutions exist for each particular case, we have not been able to find a general solution. With that said, a general solution may obscure cases that should be more heavily scrutinized.

CONCLUSION

Computers can be considered close to a magical artifact, and it is difficult to imagine life without them. Luckily for most people, computers can be used for their purposes without having to intimately understand their inner workings. However, for us programmers, familiarizing ourselves with at least the basics of their internal processes may often benefit us in our line of work. We have discussed cases where the floating-point algorithm may trigger inconsistencies when performing validation, as well as various pitfalls regarding conversions and rounding. Ultimately, understanding that these scenarios may arise will be essential to finding appropriate solutions.

REFERENCES

IEEE Computer Society
Developed by the Microprocessor Standard Committee
IEEE Standard for Floating Point Arithmetic
IEEE Std 754-2019

The Table Maker's Dilemma [Research Report]
LIP RR-1998-12
Laboratoire de l'informatique du parallélisme.
Vincent Lefevre, Jean-Michel Muller, Arnaud Tisserand
https://hal-lara.archives-ouvertes.fr/hal-02101765v1/file/RR1998-12.pdf

Shaoji Xu
Is .1+.2 equal to .3?
https://www.lexjansen.com/nesug/nesug08/ff/ff07.pdf

PhUSE Working Groups
Good Programming Practice Guidance
https://advance.hub.phuse.global/wiki/spaces/WEL/pages/26804848/Good+Programming+Practice+Guidance

Common Terminology Criteria for Adverse Events (CTCAE) v6.0 (MedDRA 28.0)
July 22, 2025
https://dctd.cancer.gov/research/ctep-trials/for-sites/adverse-events/ctcae-v6.pdf

Carol Matthews
Assigning NCI CTC Grades To Laboratory Results
https://www.lexjansen.com/pharmasug/2010/PO/PO03.pdf

Richann Watson, Louise Hadden
Quick, Call the "FUZZ": Using Fuzzy Logic
https://www.lexjansen.com/sesug/2019/SESUG2019_Paper-149_Final_PDF.pdf

ACKNOWLEDGMENTS

The views and opinions expressed in this paper are those of the authors and do not necessarily reflect the official policy or position of Priovant Therapeutics or any affiliated organizations.

We would like to acknowledge Veronica (Vee) Gonzalez for the inception of the paper topic – while pursuing a practical solution she inspired an exciting, theoretical conversation about some of the more granular nuances of rounding. We would also like to extend additional appreciation to Vee, as well as Martha O'Brien and Keith Shusterman for reviewing our work. Finally, of course, we thank the authors of our references and recommended reading for providing key information to completing this publication.

RECOMMENDED READING

Amol Waykar, et al.
Generating TFLs in R - Challenges and Successes compared to SAS
https://www.lexjansen.com/phuse-us/2020/ct/CT05.pdf

Erik W. Tilanus
Randomized Rounding
https://support.sas.com/resources/papers/proceedings/proceedings/sugi28/105-28.pdf

Christopher F. Ake
Rounding After Multiple Imputation With Non-binary Categorical Covariates
https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/112-30.pdf

Jeff Palmer
A Generalized Rounding Alternative
https://support.sas.com/resources/papers/proceedings/proceedings/sugi24/Posters/p248-24.pdf

Robert D. Sands
A SAS® MACRO FOR THE CONTROLLED ROUNDING OF ONE- AND TWO-DIMENSIONAL TABLES OF REAL NUMBERS
https://www.lexjansen.com/nesug/nesug03/st/st001.pdf

Vikash Jain,
Standardization of Reporting Digits: Significant Vs. Rounding
https://www.lexjansen.com/nesug/nesug10/po/po39.pdf
Ying Zhou, Lali Sandalic, Holger Bohnemeier, David Haryanto, Jens Klement,
Overlooked Tiny Numeric Representation Errors in SAS may Lead to Substantially Misleading p-Values
https://www.lexjansen.com/phuse/2018/as/AS01.pdf

Imelda C. Go
Rounding in SAS®: Preventing Numeric Representation Problems
https://analytics.ncsu.edu/sesug/2008/PO-082.pdf

Osmel Brito Bigott, Yenireth Gil, and María Victoria Daboín,
Working with large decimals: how precise is my data with SAS?
https://www.lexjansen.com/sesug/2022/SESUG2022_Paper_106_Final_PDF.pdf

Adam Miller
Rounding Up the Stragglers: Dynamically Detecting and Processing All Existing Data Sets
https://www.lexjansen.com/nesug/nesug12/bb/bb18.pdf

Adeline J. Wilcox
WHEN ROUNDING THE RESULT OF A RANDOM NUMBER GENERATOR IS WRONG
https://www.lexjansen.com/nesug/nesug00/cc/cc4026.pdf

Paul Stutzman
What do you mean 0.3 doesn't equal 0.3? Numeric Representation and Precision in SAS and Why it Matters
https://www.lexjansen.com/pharmasug/2014/CC/PharmaSUG-2014-CC50.pdf

Paul Gorrell
Numeric Data In SAS® : Guidelines for Storage and Display
https://www.lexjansen.com/nesug/nesug02/at/at002.pdf

Matthias Lehrkamp
Do Not Trust Your Own Computer-generated Numbers
https://www.lexjansen.com/phuse/2019/dh/DH08.pdf

Nicola Tambascia
Introduction to numeric precision and representation issues. Why 4.8 minus 4.6 is not always equal to 0.2
https://www.lexjansen.com/ phuse/2011/pp/PP12.pdf

Monika Ludwinska
Mr. (R)ight – Why R Should Be Used for Validating TLFs in Clinical Trials?
https://www.lexjansen.com/phuse/2023/ct/PAP_CT10.pdf

Deb Cassidy
Comparing Datasets: Using PROC COMPARE and Other Helpful Tools
https://www.lexjansen.com/pharmasug/2012/TF/PharmaSUG-2012-TF14.pdf

Michael Walshe
R Evolution: A Guide for Training SAS® Programmers
https://www.lexjansen.com/phuse/2024/tt/PAP_TT09.pdf

Joachim Blume
Laboratory Result conversion and the Rounding Problem
17th German CDISC User Group Meeting 24-Sep-2013

CONTACT INFORMATION

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

Jennifer McGrogan
Priovant Therapeutics
(609) 923-2896
jenn.mcgrogan@priovant.com
https://priovant.com/

Mario Widel
Independent Contractor
(317) 459-5520
mhwidel@gmail.com

Any brand and product names are trademarks of their respective companies.