Table of Contents

Precision and Scientific Notation in Liberty BASIC

Reproduced with minor editing from Newsletter 139

This is a brief outline of the most a Liberty BASIC programmer is likely to need to know about precision and scientific notation. This article will probably be of most interest to science students and includes functions they can incorporate into their programs for outputting numbers in scientific notation.

Precision and Rounding

ALL numbers in Liberty BASIC are stored and manipulated in double precision. This means that if there is an error from the computer's limit on how many decimal places it can keep, it will appear in the 15th digit after the decimal point. Missing a few points at the fifteenth decimal is the equivalent of a few people going missing out of a population about 500,000 times the earth's population. Accurate enough for government work and even most scientific and engineering work.

Such tiny errors can occur with no calculation as soon as the data is input because of the way numbers are stored internally, particularly the limited amount of storage devoted to the part of the number after the decimal point. Such errors have been demonstrated in the conforums posts in threads on "Rounding".

Rounding, as most readers will know, is a controlled lowering of precision by "correcting" to the nearest whole number of a specified position in the decimal representation. To see this in action, run the final demo below.

You can most simply output numbers to the desired precision, using the USING() function of LB. If you never use more than 14 # characters after the decimal point you will never see these tiny errors - and why would you want to?

Once again, see the final demo in this article for an example of rounding.

Precision and Accuracy

There is a technical distinction between precision and accuracy. Accuracy refers to the degree of error potentially inherent in a measurement. Precision is about the amount of fine information in the numerical expression of the measurement. You can have a high precision expression of a low accuracy measurement - very misleading at best. And you can have a low precision expression of a highly accurate measurement - possibly a bit wasteful of good information. The two terms are sometimes used interchangeably.

Scientific Notation

For very large and very small numbers, scientists long ago agreed on a standard way of representing them so that they would all be easily able to compare results. This became the standard for all numbers in scientific and engineering journals.

The syntax is:

D.d{d}eN

D - is a single digit between 1 and 9 before the decimal point
d - represents digits after the decimal point which may be zero
e - is the letter "e"
N - is a positive or negative integer called the exponent representing the power of ten to multiply by

Examples:
1.234e3 = 1.234 x 10 x 10 x 10 = 1.234 x 1000 = 1234
9.67e-7 = 9.67 x (1/10000000) = 0.000000967

So a positive exponent means "shift the decimal point this many places to the right"

and a negative exponent means "shift the decimal point this many places to the left".

We can represent any number in this form except zero. The scientific notation for zero is 0 - though some

sticklers may put 0e0.

The part of the number before the "e" is usually called the mantissa but may also be called the significand (sic).

General Function for Scientific Notation

The following LB function will represent any number (positive or negative or zero) in scientific notation.

Try entering any numbers you want. Note especially how the system represents small numbers like 0.0000000000067 in an exponential form but not the standard scientific notation. The function given will "fix" these and will accept any non-standard exponential form such as 234.76e-7 thanks to the latest expansion of LB.

```
' test program for scientific notation function
' released to public domain, August 2006 by Grahame King
a=1
while a<>0
input "test number ="; a
a$ = str$(a)
print "test number as string = "+a$
print "test number in scientific notation = "+ScNotation$(a)
wend
end
function ScNotation$(numin)
' function to express a number as a string in 'scientific notation"
if numin = 0 then ScNotation$ = "0" : exit function
' separate number into sign, significant numeric part (mantissa), and
exponent
' first extract the sign
absNumIn = abs(numin)
sgnNumIn = absNumIn/numin
if sgnNumIn>0 then
sgnNumIn$ = ""
else
sgnNumIn$ = "-"
end if
' represent unsigned number as a string using str$
absNumIn$ = str$(absNumIn)
' find the "e" in case LB has already put it in a form with an exponen
t
inde = instr(absNumIn$,"e")
```

```
if inde>0 then
expon = val(mid$(absNumIn$,inde+1))
mantissa$ = mid$(absNumIn$,1,inde-1)
else
expon = 0
mantissa$ = absNumIn$
end if
' move the decimal point in the old mantissa so that there is just one
 nonzero digit in front of it
' and adjust the exponent accordingly
' start by finding the decimal point and breaking mantissa into its wh
ole part and fractional part
indDot = instr(mantissa$,".")
if indDot = 0 then ' for integers
indDot = len(mantissa$)
whole$ = mantissa$
fract$ = ""
else
whole$ = mid$(mantissa$,1,indDot-1)
fract$ = mid$(mantissa$,indDot+1)
end if
while val(whole$) > 9 ' to move decimal point to the left if necessary
indDot = indDot-1
fract$ = right$(whole$,1)+fract$
' add the last digit from the whole number part to the beginning of fr
act$
whole$ = left$(whole$,len(whole$)-1)
' and then remove that digit from the whole$
expon = expon+1   ' adjust exponent to keep the net result correct
wend
while val(whole$) <= 0
' to move decimal point to the right if necessary
indDot = indDot+1
whole$ = whole$+left$(fract$,1)
' add the first digit of fractional part to the whole number part
fract$ = mid$(fract$,2)
' and then remove that digit from the beginning of the fractional part
expon = expon-1   ' adjust exponent to keep the net result correct
wend
' build Scientific Notation string from the three parts - sign, new ma
ntissa and new exponent
ScNotation$ = sgnNumIn$+str$(val(whole$))+"."+fract$+"e"+str$(expon)
end function
```

General Rounding Function

The following LB function will round any number (positive or negative) to any number of places. For run-of-the-mill accounting applications, simple rounding to two decimals with the USING function should do.

```
' demo of rounding with over-
USING, reasonable USING and the round() function below:
num = 77725.214 +0.0005
print num, "from print num, with no formatting in print statement"
print
print "The next three lines with the using function show what over-
precision does."
print "77725.214 = ", using("#####.##################",77725.214)
print " 0.0005 = ", using("#####.##################",0.0005)
print "77725.214 +0.0005 = ", using("#####.##################",num)
print
print "Precision within double-
precision bounds removes the wierdness:"
print "77725.214 = ", using("#####.########",77725.214)
print " 0.0005 = ", using("#####.########",0.0005)
print "77725.214 +0.0005 = ", using("#####.########",num)
print
print "Using the mathematical rounding function:-"
print
"Rounding to 4 decimals, 3 decimals ...., and finally to the nearest t
housand: "
for i = -4 to 3
print using("##",i); " ... "; round(num,i)
next
end
function round(x,places)
'positive "places" rounds to whole number places,
'negative "places" rounds to fractional places
round = sgn(x)*int((abs(x)*10^(-1*places))+0.5)*(10^places)
end function 'round
function sgn(x)
if x<0 then
sgn = -1
else
sgn = 1
end if
end function 'sgn
```

What's Wrong with Using USING() for Rounding?

Nothing most of the time. The main advantage of the mathematical form is that it easily handles variable

precision. Suppose you want to vary the number of decimal places in your output depending on the results of a set of calculations without knowing before hand what format will be required. This is messy to do with the using() function particularly if you can't predict where the decimal point might be.

Beyond Double Precision

Finally, if you're thinking of doing calculations which will produce results smaller than say about 1.0e-10, you will need to be very mathematically savvy in how you go about it. I imagine only scientists and pure mathematicians might be interested in doing this and they would need to know where to learn about techniques for ensuring accurate results. Remember that 1.0e-8 squared is 0 to double precision, so if you are evaluating expressions containing squares and cubes etc, your results can quickly become hopelessly inaccurate as the input numbers become smaller.

Table of Contents