AA Technical Note 2011-02

The Error in the Double Precision Representation of Julian Dates

Approved for Public Release

George Kaplan, Jennifer Bartlett, & William Harris

April 6, 2011

Abstract

An evaluation of the error associated with representing Julian dates in IEEE 754 double precision floating-point numbers demonstrates that Julian dates near the current epoch can be represented to a precision not worse than 20.1 microseconds.

1 Introduction

The representation of time variables as Julian dates is ubiquitous in software for positional astronomy. In many cases, the Julian date is stored as a double precision floating-point number. Standard 754 of the Institute of Electrical and Electronic Engineers (IEEE) [1] specifies the format of double precision numbers now widely used in computer systems. (See also [2] and [3].) This representation has about 16 decimal digits of precision. For all Julian dates of practical interest, seven decimal digits are taken up by the day count (to the left of the decimal point), leaving nine decimal digits for the fraction of a day. Thus, we expect that Julian dates can be represented to an accuracy of about 10^{-9} day $\approx 10^{-4}$ s = 100 microseconds. But what is the actual accuracy? What is the true error in Julian dates stored in IEEE 754 double precision format?

2 Method

Two Fortran programs were written to test the precision of Julian dates stored as double precision floating-point numbers. One of us (WTH) converted the second program to C.

Program 1 generates random time epochs as Julian dates between 1900 and 2100, computed and stored as extended precision (REAL*16) floating-point numbers. For each such random

Julian date, the program then sets a double precision variable (REAL*8) to equal the extended precision Julian date. The program subtracts the two values using extended precision arithmetic. The result should be the error in the double precision representation. The program loops through as many of these sequences as requested by the user, gathering statistics on the errors found.

Unfortunately, not all Fortran compilers support extended precision arithmetic, which restricts the systems on which Program 1 can be run.

Program 2 is identical to the first in overall structure and operation but does not rely on extended precision variables or arithmetic. It can, therefore, be used with virtually any Fortran compiler and system. It assembles each random Julian date in two separate pieces, the integral day count and the fraction of a day, each stored in its own double precision (REAL*8) variable. Next, the program adds the two pieces and stores the result in one double precision variable. Then, the fraction of a day is extracted from this variable. At this point, there are two representations of the fraction of a day, original and extracted; the original will be much more precise. The program subtracts the two and, just as in the first program, the result should be the error in the double precision representation of the complete Julian date.

3 Results

A number of tests was run on different systems and compilers:

Program	Language	Compiler	Operating System	Chip
1	Fortran	Absoft	Mac OS X	PowerPC
2	Fortran	Absoft	Mac OS X	PowerPC
1	Fortran	gfortran	Mac OS X	Intel
1	Fortran	Lahey-Fujitsu	Windows 7	Intel
2	Fortran	Lahey-Fujitsu	Windows 7	Intel
2	Fortran	gfortran	Linux 2.6.9	Intel
2	C	gcc	Mac OS X	Intel

All except the last test were run in 32-bit mode. The C language test was run in 64-bit mode. Regardless of which program was run or which language/compiler/system combination was used, the statistical results were always the same, and the output from a typical set of runs is shown in the table below.

The table shows the statistics of a large number of sample Julian dates chosen at random. Each line represents a run with a different random number "seed." The number of Julian dates analyzed in the run is the first number in each line, labeled "N". The other numbers in the line are all Julian date error statistics expressed in units of days. The average error is the average of the absolute values of the errors, and is somewhat less than the root-mean-squared (RMS) error. The average error is about half the maximum error. Once N is sufficiently large, there is apparently a "hard" upper limit on the Julian date error of $\pm 2.33 \times 10^{-10}$ days, which equals ± 20.1 microseconds.

Samples of Errors in Random Julian Dates 1900–2100

N	Bias	Avg Err	RMS	Min Err	Max Err
15	4.85D-11	1.43D-10	1.52D-10	–2.27D-10	2.16D-10
20	–3.31D-11	9.94D-11	1.15D-10	–1.77D-10	2.08D-10
200	–8.38D-13	1.12D-10	1.29D-10	–2.32D-10	2.27D-10
1000	–5.82D-13	1.18D-10	1.36D-10	–2.32D-10	2.33D-10
1500	1.43D-12	1.16D-10	1.34D-10	–2.32D-10	2.33D-10
2000	2.74D-12	1.13D-10	1.31D-10	–2.33D-10	2.33D-10
4000	–5.61D-13	1.14D-10	1.32D-10	–2.33D-10	2.33D-10
9900	–8.46D-13	1.17D-10	1.35D-10	–2.33D-10	2.33D-10
12000	–4.39D-13	1.17D-10	1.35D-10	–2.33D-10	2.33D-10

Further tests revealed that, as we might expect, the distribution of errors is uniform between -2.33×10^{-10} and $+2.33\times10^{-10}$ days; that is, between the two limits, all errors are equally likely. For such a distribution, centered on zero, the average error is just half the absolute value of the maximum error, as the numerical tests showed. Also, for a uniform distribution, the RMS error does not have the same meaning in terms of probability that it has for a Gaussian distribution; here, only $\sqrt{\frac{1}{3}} = 58\%$ of the errors have magnitudes less than the RMS.

These results are for Julian dates with truly random fractions. Julian dates with fractions that are multiples of powers of two (.0, .5, and multiples of .25, .125, etc.) can be exactly represented. Julian dates with other common fractions (e.g., multiples of .1) have errors that are restricted to a finite set of values, but all are within the distribution envelope found for the randomized fractions.

4 Interpretation

A plausible explanation follows for how the maximum error comes about.

Any Julian date from 2,097,152 (in year 1029) through 4,194,303 (in year 6771) can be written as $1.z \times 2^{21}$, a form that is analogous to the floating-point representation in a computer. In a computer, the fraction z, here encompassing at most a span of 2,097,151 days, is represented as a series of binary digits (bits). In IEEE 754 format, 52 bits are allocated to the fraction. (It is sometimes said that the IEEE format has 53 bits of precision, but that includes the 1 to the left of the decimal point, which is not stored; it is called the "hidden" or "implicit" bit.) So the lowest-order bit in the fraction represents $\{0 \text{ or } 1\}\times2^{-52} \times 2,097,151$ days = $\{0 \text{ or } 1\}\times4.66\times10^{-10}$ days. The number 4.66×10^{-10} thus represents the minimum difference in days between two double precision Julian dates—that is, the Julian date "resolution" or "granularity," the numerical consequence of flipping the lowest-order bit from 0 to 1 or vice versa. But the IEEE specification also provides that the results of double precision operations are rounded to the precision specified. Therefore, the maximum error in representation would be half the resolution, just as the maximum error in an integer representation (resolution = 1) of a floating-point number is 0.5. By this reasoning, we get the 2.33×10^{-10} number we found experimentally for the magnitude of the largest error, expressed in days.

This maximum error of 20.1 microseconds for double precision Julian dates is significantly

better than the estimate of 0.1 milliseconds (100 microseconds) given in the User's Guide to NOVAS 3.0 [4], near the bottom of pages F-18 and C-20. That is, there, the maximum error is conservative by a factor of five. Based on the results described here, in the User's Guides to NOVAS 3.1, the Julian date error should be revised in those sentences to read 20.1 microseconds.

References

[1] IEEE Computer Society (2008), *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2008 (revision of IEEE Std 754-1985), The Institute of Electrical and Electronic Engineers, Inc. (New York: IEEE)

[2] Pittsburgh Supercomputing Center (2010), "The IEEE Standard for Floating Point Arithmetic," http://www.psc.edu/general/software/packages/ieee/ieee.php (Pittsburgh: Carnegie Mellon University & University of Pittsburgh)

[3] Stallings, W. (1989), *Computer Organization and Architecture* (New York: Macmillan), pp. 222–233

[4] Kaplan, G. H., Bangert, J. A., Bartlett, J. L., Puatua, W. K., & Monet, A. K. B. (2009), *Users Guide to NOVAS Version 3.0*, USNO Circular 180 (Washington, DC: USNO)