Research | October 02, 2017

A Multiprecision World

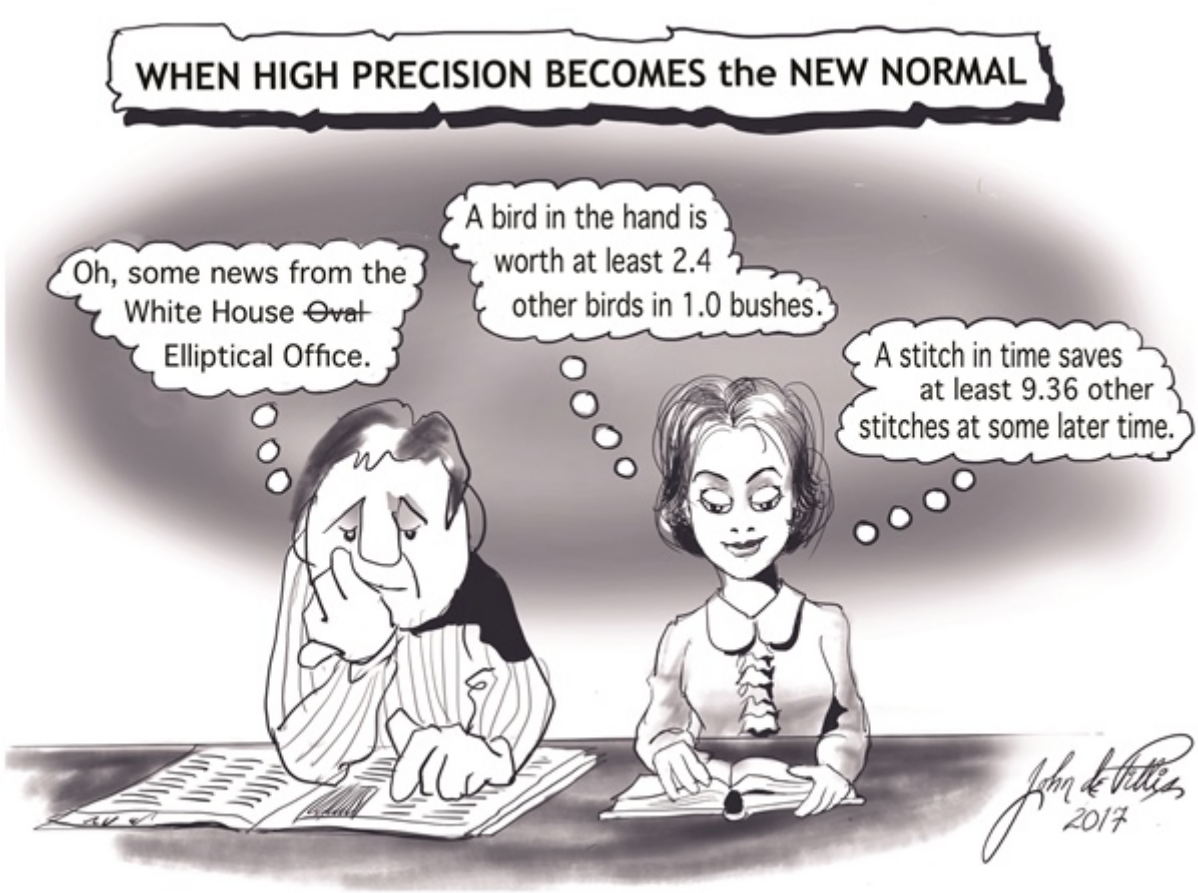
By Nicholas Higham (https://sinews.siam.org/About-the-Author/nicholas-higham)

Traditionally, floating-point arithmetic has come in two precisions: single and double. But with the introduction of support for other precisions, thanks in part to the influence of applications, the floating-point landscape has become much richer in recent years.

To see how today's multiprecision world came about, we need to start with two important events from the 1980s. The IEEE standard for binary floating-point arithmetic was published in 1985. It defined single precision (32-bit) and double precision (64-bit) floating-point formats, which carry the equivalent of about eight and 16 significant decimal digits, respectively. This led to the relatively homogeneous world of floating-point arithmetic that we enjoy today, which contrasts starkly with the 1970s, when different computer manufacturers used different floating-point formats and even different bases (hexadecimal in the case of some IBM machines). The second important event in the 1980s was the introduction of the Intel 8087 coprocessor, which carried out floating-point computations in hardware (in conjunction with an 8086 processor) and enabled much faster scientific computations on desktop machines. Intel went on to incorporate the coprocessor into the main processor in the Pentium and subsequent series of processors.

Throughout the 1990s, we had the choice of working in single or double precision arithmetic in most computing environments. Single precision did not intrinsically run faster than double precision on Intel chips, but its lower storage requirement could lead to speed benefits due to better use of cache memory.

The picture started to change in 1999 when Intel introduced streaming single instruction, multiple data (SIMD) extensions (SSE), which allowed single precision arithmetic to execute up to twice as fast as double. A few years later, the Cell processor, designed by Sony, Toshiba, and IBM for use in the Sony PlayStation 3 gaming system, offered single precision arithmetic running up to 14 times faster than double precision, thus presenting interesting opportunities for scientific computing. These developments directed efforts towards algorithms with the ability to exploit two precisions to solve a problem faster or more accurately than just one precision. The concept of such algorithms is not new. Up until the 1970s, many computers could accumulate inner products at twice the working precision and no extra cost, and the method of iterative refinement for linear systems—first programmed by James Hardy Wilkinson on the Pilot ACE in 1948—exploited this capability to improve the accuracy of an initial solution computed with LU factorization. A new form of iterative refinement that employs single precision to accelerate the double precision solution process was developed in [3].



Cartoon created by mathematician John de Pillis.

In the last few years, the advent of half precision arithmetic (16 bits) has enriched the floating-point landscape. Although the 2008 revision of the IEEE standard originally defined it only as a storage format, manufacturers have started to offer half precision floating-point arithmetic in accelerators such as graphics processing units (GPUs). Half precision offers both speed benefits (it operates up to twice as fast as single precision, though only the top-end GPUs attain the factor 2) and lower energy consumption. The main application driver for half precision is machine learning (and in particular, deep learning), where algorithms have been found empirically to perform satisfactorily in low precision.

I am not aware of any rigorous analysis that explains the success of machine learning algorithms run in half—or even lower—precision. One possible explanation is that we are solving the wrong optimization problem (as the correct one is too difficult to solve) and thus do not need to solve it accurately. Another is that low precision has a beneficial regularizing effect. Yet from the traditional numerical analysis point of view, half precision is dubious. The usual rounding error bound for the inner product of two n-vectors contains the constant nu, where u is the unit roundoff, so in half precision (which has $n \approx 5 \times 10^{-4}$), we cannot guarantee even one correct significant digit in the computed inner product once n exceeds 2,048. Indeed, the set of half precision numbers is small: there are only 61,441 normalized numbers, and the spacing between 32,768 and the largest number, 65,504, is 32.

People will be tempted to use half precision as it becomes more accessible in hardware, potentially with serious consequences if relative errors of order 1 are obtained in critical applications. The limitation that half precision has a range of only $10^{\pm 5}$ means that in many problems, one is just as likely to obtain NaNs as output

(resulting from overflow) as completely incorrect numbers. This presents work for our community to better understand the behavior of algorithms in low precision, perhaps through a statistical approach to rounding error analysis instead of the usual approach of proving worst-case bounds.

The precision landscape has been getting more interesting at the higher end as well. The 2008 IEEE standard revision introduced a quadruple precision floating-point format, which is available almost exclusively in software (the IBM z13 processor being a rare exception), perhaps as a compiler option. *Arbitrary* precision arithmetic is available in several environments, including Maple, Mathematica, Sage, Julia through its BigFloat data type, and MATLAB with the Symbolic Math Toolbox or the Multiprecision Computing Toolbox (Advanpix). Several of these systems utilize the GNU MPFR Library, an open source C library for multiple precision floating-point computations. Having arbitrary precision floating-point arithmetic at our fingertips is not something many of us are accustomed to. I first became intrigued with the possibility during a visit to the University of Toronto (U of T) in the 1980s, when Tom Hull introduced me to Numerical Turing. Turing was a Pascal-like language developed in U of T's Department of Computer Science for teaching, and Hull's Numerical Turing augmented it with variable precision decimal floating-point arithmetic.

Field-programmable gate arrays, which have always been configurable for different precisions of fixed-point arithmetic but now can additionally support floating-point arithmetic, also have a role to play. These low-power devices offer the possibility of customizing the floating-point format in hardware to meet the precision requirements of an application.

Once arithmetic of several precisions is available (half, single, double, quadruple), we want to harness it to compute results of the desired accuracy as efficiently as possible, bearing in mind the relative costs of the precisions (https://nickhigham.wordpress.com/2017/08/31/how-fast-is-quadruple-precision-arithmetic/). A natural scenario is iterative methods such as Newton's method, where there may be no point in computing iterates accurately in the early stages of an iteration when far from the solution; increasing the precision during the iteration may reduce execution time. We can also ask whether using just a little extra precision in certain key parts of an algorithm can bring benefits to the speed or accuracy, and whether it can stabilize a potentially unstable algorithm. See [2, 5] for some recent work along these lines.

If we aim to achieve a given fairly low level of accuracy or residual with an iterative method, say t bits, we can ask what the best choice of precision ($p > t$ bits) is in which to run the computations. It turns out that for Krylov methods (for example), the number of iterations can depend strongly on the precision [4], meaning that the fastest computation might not result from the lowest precision that achieves the desired accuracy.

SIAM News readers may remember "A Hundred-dollar, Hundred-digit Challenge" announced by Nick Trefethen in January 2002. That challenge asked for 10 problems to be solved to 10-digit accuracy. Although high precision arithmetic could be used in the solutions as part of a brute force attack, it turned out to be generally not necessary [1]. This example serves as a reminder that mathematical ingenuity in the choice of algorithm can enable a great deal to be done in double precision arithmetic, so one should always think carefully before resorting to higher precision arithmetic, with its attendant increase in cost. Nevertheless, today's multiprecision computational landscape offers great scope for clever exploitation, presenting exciting opportunities to researchers in our community.

References

[1] Bornemann, F., Laurie, D., Wagon, S., & Waldvogel, J. (2004). *The SIAM 100-Digit Challenge: A Study in High-Accuracy Numerical Computing*. Philadelphia, PA: Society for Industrial and Applied Mathematics.

[2] Carson, E., & Higham, N.J. (2017). Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions. *MIMS EPrint 2017.24*, The University of Manchester.

[3] Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A., & Dongarra, J. (2016). Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. Tampa, FL.

[4] Roldao-Lopes, A., Shahzad, A., Constantinides, G.A., & Kerrigan, E.C. (2009). More Flops or More Precision? Accuracy Parameterizable Linear Equation Solvers for Model Predictive Control. In *17th IEEE Symposium on Field Programmable Custom Computing Machines* (pp. 209-216).

[5] Yamazaki, I., Tomov, S., & Dongarra, J. (2015). Mixed-Precision Cholesky QR Factorization and its Case Studies on Multicore CPU with Multiple GPUs. *SIAM J. Sci. Comp., 37*, C307-C330.

Nicholas Higham is the Richardson Professor of Applied Mathematics at the University of Manchester. He is the current president of SIAM.