Monte Carlo Arithmetic:
a framework for the statistical analysis of roundoff error

D. Stott Parker
stott@cs.ucla.edu
Computer Science Department
University of California
Los Angeles, CA 90095-1596

Paul R. Eggert
eggert@twinsun.com
Twin Sun, Inc.
360 N. Sepulveda Blvd., Suite 2055
El Segundo, CA 90245-4462

Brad Pierce
pierce@cs.ucla.edu
Computer Science Department
University of California
Los Angeles, CA 90095-1596

March 30, 1997

Abstract

Monte Carlo Arithmetic (MCA) is an extension of standard floating-point arithmetic that exploits randomness in basic floating-point operations. MCA uses randomization to implement random rounding — which forces roundoff errors to be randomly distributed — and random unrounding — which randomly extends the inputs of arithmetic operations to arbitrary precision. Random rounding can be used to produce roundoff errors that are truly random and uncorrelated, and that have zero expected bias. Random unrounding detects catastrophic cancellation, which is the primary way that significant digits are lost in numerical computation. Randomization also can be used to vary precision dynamically, and to implement inexact values (values known to only a few significant digits).

Randomization has both theoretical and practical benefits. It has the effect of transforming any floating-point computation into a Monte Carlo computation, and roundoff analysis into statistical analysis. By running a program multiple times, one directly measures the sensitivity of particular outputs to random perturbations of particular inputs. Also, MCA gives a way to avoid some anomalies of floating-point arithmetic. For example, while floating-point summation is not associative, Monte Carlo summation turns out to be 'statistically associative' up to the standard error of the sum. Generally, it gives a different perspective on the study of error.

Keywords: floating-point arithmetic, floating-point rounding, roundoff error, random rounding, ANSI/IEEE floating-point standards, significance arithmetic, Monte Carlo methods

AMS(MOS) subject classifications: 65C05, 65C20, 65G05, 65G10, 65J05, 68M07, 62P99

1 Introduction

In this paper we sketch a practical method for *a posteriori* roundoff error analysis. We show how to answer the following question:

> How much sensitivity to roundoff error is there in the results that were generated by a particular code running on a particular machine applied to a particular input?

Formal roundoff error analysis is indispensable to the design of high-quality numerical algorithms. After design, however, algorithms must still be implemented as routines in a high-level computer language, then combined with other routines into a complete program. Moreover, before a program can be used, it must be translated by a compiler into the code of the particular machine on which it will be run. Results are generated by applying this code to a specific input problem. How can one determine the degree to which roundoff error has clouded these *particular* results?

Monte Carlo Arithmetic (MCA) is a model of floating-point arithmetic in which arithmetic operators and their operands are *randomized* (perturbed with random values) in certain ways. For example, rather than insist that floating-point addition obey

$$x \quad \oplus \quad y \quad = \quad fl[\, x \, + \, y \,] \quad = \quad (x \, + \, y)\,(1 \, + \, \delta)$$

where δ is some deterministically-defined value (such as the relative error incurred by rounding to the nearest floating-point number, as in IEEE floating-point arithmetic [1, 15, 43]), we allow δ to be a random variable. The result of every arithmetic operation is randomized in a predefined way. As a result, the addition $x \oplus y$ can yield different values if evaluated multiple times.

The foundation of MCA is a proposed enhancement of floating-point arithmetic that would do away with certain of its arbitrary conventions and instead use randomization. The conventions we call into question concern the handling of rounding, and of rounded-off operands. We describe how randomization can be efficiently[1] implemented, making it possible to use Monte Carlo methods for estimating the accumulation of roundoff error.

There are a number of other useful approaches for analyzing roundoff error in numerical computations, e.g., running in higher precision, computing condition numbers, using interval arithmetic, and performing a formal roundoff analysis. However, each of these has drawbacks. Monte Carlo error analysis is not intended to replace these methods, but rather to complement them.

> Competent engineers rightly distrust all numerical computations and seek corroboration from alternative numerical methods, from scale models, from prototypes, from experience, ...
> — W. Kahan [44, p.34]

We simply hope that Monte Carlo analysis of roundoff error will prove to be a useful addition to the toolbox of competent engineers.

With randomized floating-point arithmetic, if a program is applied to the same input several times, then each recalculation yields a slightly different answer. These answers constitute a sample distribution to which the whole array of standard statistical methods can be applied. If $\widehat{\mu}$ is the sample mean and $\widehat{\sigma}$ is an estimate of the standard deviation of the sample mean, then $\widehat{\sigma}$ is a rough estimate of the error in $\widehat{\mu}$.

Each recalculation is an experiment in a Monte Carlo simulation — a simulation of the sensitivity to rounding of this particular combination of input and program. (Monte Carlo methods are described in Section 4.2.) The practitioner must decide whether the level of sensitivity suggested by this simulation is acceptable for the problem at hand.

It should be emphasized that Monte Carlo error analysis can give no iron-clad guarantees that serious roundoff error is absent in a computation. As Kahan [44, p.24] points out, computations can be "virulently unstable but in a way that almost always diverges to the same wrong destination".

[1] Fast random number generators can be implemented efficiently with feedback shift registers [19, §7.15–16].

2 Some Examples

2.1 A simple example

Kahan (e.g., [24, 41, 42]) has stressed that even computations as simple as solving $ax^2 - bx + c = 0$ present interesting problems for floating-point arithmetic. Consider solving the equation

$$7169\,x^2 \quad - \quad 8686\,x \quad + \quad 2631 \quad = \quad 0.$$

With $a = 7169$, $b = -8686$ and $c = 2631$, the C statements

```
r1 = (-b + sqrt(b*b-4*a*c))/(2*a);
r2 = (-b - sqrt(b*b-4*a*c))/(2*a);
```

yielded Table 1, using IEEE floating-point with the default rounding (round to nearest).

precision	r1	r2
exact solution (rounded)	.60624386632168620	.60536165746126819
IEEE single precision	.606197	.605408

Table 1: Roots of $7169\,x^2 - 8686\,x + 2631 = 0$, computed with IEEE floating-point.

When we instead randomize the inputs and outputs of floating-point operations, the results vary each time we execute the program. (The coefficients a, b, c and the constants 2 and 4 were not randomized as they are exact, i.e., representable precisely in floating-point format.) Using gcc version 2.7.2 with no options (except '-lm') on a Sun SPARCstation 20 model 501-2324 running SunOS 5.5, running the program 5 times with single precision MCA yielded Table 2.

run	r1	r2
1	.606168	.605333
2	.606205	.605343
3	.606191	.605391
4	.606249	.605323
5	.606252	.605301
computed average:	.606213	.605338
standard deviation:	.000037	.000033
standard error:	.000016	.000015

Table 2: Roots of $7169\,x^2 - 8686\,x + 2631 = 0$, computed with single precision MCA.

For simplicity, we used IEEE single precision floating-point representation in our implementation of MCA. Similar results can be produced in any precision. The C source code for MCA and all examples here is available from the authors.

In this table, notice the standard deviation estimates the absolute error in the computed roots. That is, the roots in each run are within a few standard deviations $\widehat{\sigma}$ of the exact solution. Furthermore the computed average $\widehat{\mu}$ also lies within these bounds. So the standard deviation $\widehat{\sigma}$ gives a rough estimate of the error in $\widehat{\mu}$.

Running a program n times with MCA ultimately gives, for each value x being computed, n samples x_i that disagree on the random digits of their errors. These samples have an underlying distribution with **mean** μ and **standard deviation** σ, which are respectively estimated by the computed **average** $\widehat{\mu}$ and **(unbiased) standard deviation** $\widehat{\sigma}$ of the n samples $x_1, \ldots, x_n$:

$$\widehat{\mu} \quad = \quad \frac{1}{n} \sum_{i=1}^{n} x_i \qquad\qquad \widehat{\sigma} \quad = \quad \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\, x_i \,-\, \widehat{\mu}\,)^2}.$$

The expected error in $\widehat{\mu}$ is measured by the standard deviation of $\widehat{\mu}$, called the standard error:

$$\textbf{standard error} \quad = \quad \sigma/\sqrt{n}, \qquad \text{which is estimated by } \widehat{\sigma}/\sqrt{n}.$$

Use of standard error is common in scientific work (e.g., [61, 71]), and it is traditional to write

$$\text{“}\mu \quad = \quad \textbf{average} \;\pm\; \textbf{estimated standard error}\text{”} \qquad\qquad (\text{“}\,\mu \quad = \quad \widehat{\mu} \;\pm\; \widehat{\sigma}/\sqrt{n}\,\text{”}).$$

This notation is misleading since it does not give hard bounds on the error, but instead gives ‘one standard deviation’ bounds. Thus $\widehat{\sigma}$ actually overestimates the absolute error in $\widehat{\mu}$, while the standard error estimates it well, and this estimate improves with the number of samples n.

Now, if we modify the problem by reducing a and c, asking to solve

$$7\,x^2 \;-\; 8686\,x \;+\; 2 \;=\; 0,$$

then the computed second root is quite inaccurate, even in double precision, as shown by Table 3.

precision	r1	r2
exact solution (rounded)	1240.8569126015164	.00023025562642454231
IEEE double precision	1240.85691260152	.000230255626385250
IEEE single precision	1240.86	.000279018

Table 3: Roots of $7\,x^2 \;-\; 8686\,x \;+\; 2 \;=\; 0$, computed with IEEE floating-point.

Running the program 5 times with MCA gives Table 4; the standard error reflects the inaccuracy.

run	r1	r2
1	1240.86	.000198747
2	1240.86	.000248582
3	1240.86	.000251806
4	1240.86	.000177380
5	1240.86	.000203571
computed average:	1240.86	.000216017
standard deviation:	.000	.000032739
standard error:	.000	.000014641

Table 4: Roots of $7\,x^2 \;-\; 8686\,x \;+\; 2 \;=\; 0$, computed with single precision MCA.

This example shows that by ‘sampling’ the randomized floating-point computation we obtain useful statistical accuracy estimates, even with a small number of samples. We get good average values (in this case, slightly better than those for IEEE single precision floating-point), but more important we also get standard deviation values that measure disagreement among the samples, and standard error values estimate the absolute error in the computed average.

2.2 Two startling examples

Consider the following two marvelous examples adapted from [5]. First, define the sequence (x_k)

$$\begin{aligned} x_0 &= 1.5100050721319 \\ x_{k+1} &= (3x_k^4 - 20x_k^3 + 35x_k^2 - 24) \ / \ (4x_k^3 - 30x_k^2 + 70x_k - 50). \end{aligned}$$

As demonstrated in Table 5, *depending on the precision of one's machine, the sequence converges to either 1, 2, 3, or 4.* Actually IEEE double precision converges to 3, and IEEE single precision converges to 2.

precision	x_{30} (computed with decimal arithmetic)
30 digits	`3.00000000000000000000000000000`
24 digits	`3.00000000000000000000000`
20 digits	`3.0000000000000000000`
16 digits	`3.000000000000000`
12 digits	`1.99999999990`
10 digits	`4.000000000`
8 digits	`1.9999980`
6 digits	`1.99990`
4 digits	`2.097`
2 digits	`1.0`

Table 5: Values of x_{30} computed with rounded decimal arithmetic of different precisions.

With MCA, the extremely unstable nature of the iteration is discernible from enormous standard deviation of x_{30}. With 10 samples, again using uniform input and output randomization, we obtained the values in Table 6. Single-precision computation for this iteration converges to 2, but the large standard deviations in this table show that, with MCA, results other than 2 are obtained with high probability. Many standard deviations in this table are larger than the average values, suggesting a complete loss of significant digits.

k	x_k average	x_k std. dev.
1	1.51001	.00000
2	2.37745	.00002
3	1.50995	.00015
4	2.37776	.00084
5	1.50778	.00593
6	2.39149	.03451
7	1.25269	.52123
8	5.17069	10.2108
9	5.65173	8.29322
10	4.76598	6.29415
⋮	⋮	⋮
30	2.40002	1.26492

Table 6: Average and standard deviation of x_k computed with single precision MCA (10 samples).

A second example illustrates that in some cases the exact solution to a problem cannot be computed in finite precision. Define the sequence (u_k) by

$$\begin{aligned} u_0 &= 2 \\ u_1 &= -4 \\ u_{k+1} &= 111 - 1130/u_k + 3000/(u_k\, u_{k-1}). \end{aligned}$$

Then for example u_{30} is

$$\frac{9901760258702227179708 67}{1648741172159345399092 07} = 6.00564868877\ldots$$

and $\lim_k u_k = 6$. However, with IEEE floating-point arithmetic and the default rounding we get the very different results shown in Table 7.

precision	*value of* u_{30}
exact value (rounded)	6.005648688771420
IEEE double precision	100.000000000000
IEEE single precision	100.0000

Table 7: The value of u_{30} computed with IEEE floating-point.

Table 7 is disturbing since the results of IEEE single and double precision agree, yet the two values are completely incorrect. For this innocuous-looking problem, the standard technique of checking the accuracy of a program's results by running it in higher precision fails. In fact, *in any bounded-precision floating-point computation, the computed limit of the sequence will be 100.*

With MCA, this instability of the computation does not escape unnoticed if we trace its evolution. Table 8 gives the initial iterates u_k, using single precision MCA, averaged over 10 samples. The standard deviations increase rapidly at first, but ultimately decrease as k grows; this is a consequence of the inevitable convergence to the attractive fixed point at 100.

k	u_k average	u_k standard deviation
0	2.00000	.0000000
1	-4.00000	.0000000
2	18.50000	.0000294
3	9.37834	.0001614
4	7.80073	.0017465
5	7.14897	.0225447
6	6.72940	3.316322
7	5.24188	4.68842
8	-3519.71	11337.3
9	105.407	26.2545
10	100.023	1.42100
⋮	⋮	⋮
30	100.000	.0000000

Table 8: Average (and standard deviation) of u_k computed with MCA (10 samples).

Although MCA will not detect error in every iterative process, in these two examples the huge variances among intermediate iterates reflects tremendous instability in arriving at the fixed point, and give strong warnings about the results of the iterations.

3 Monte Carlo Arithmetic

We assume the reader is familiar with basic issues in floating-point arithmetic. Goldberg's tutorial [27] and Higham's encyclopedia [36, chs.1–5] are great references. A more complete presentation of most of this material is in [57].

3.1 Exact and inexact values

Exact values are real numbers that can be exactly represented in a given floating-point format. **Inexact values**, by contrast, are either real values that must be rounded (to an approximation) in order to fit this format, or real values that are not completely known. Rounding discards information, hence increases inexactness.[2]

This is an important distinction. In scientific computation most quantities are inexact; constants like Avogadro's number 6.0225×10^{23} are known only to a few digits [45, §4.2.2.B]. The inexactness can be due to ignorance, uncertainty, estimation, measurement or computational error, but the upshot is that we have only a few significant digits. Sterbenz [59] mentions a variety of proposals for implementing this distinction in computer arithmetic. For example, it was implemented in the NORC calculator in the early 1950s [59, p.196].

Because an inexact value could represent a whole range of possibilities, one can only guess which of these real numbers it 'actually' represents. Conventional floating-point arithmetic, which adopts the assumptions of forward error analysis, makes the simplest possible guess — it treats all operands as if they were exact. For example, it extends single-precision numbers to double-precision format by padding the significands with zeros. In the IEEE floating-point standard [1], the 'inexact result' (or 'precision') flag bit in the status word indicates whether the result would be exact if the operands were exact[3].

3.2 Randomization

An alternative approach is to guess a *random* real value for an inexact floating-point number. This alternative gives a consistent way of dealing with two common situations:

1. The inexact value is the result of rounding of a higher-precision computation.
2. The inexact value is known to only a few significant digits (like Avogadro's number).

In both situations, the random real value can be viewed as a possible value for the inexact value, and the difference between the two can be modeled as random error.

We create an random value $\widetilde{x}$ that agrees with the value x to s digits with the **randomization**

$$\widetilde{x} \quad = \quad \mathsf{inexact}\,(x,\, s,\, \xi) \quad = \quad x \;+\; 2^{e-s}\,\xi$$

where e is the order of magnitude of x, and we are using floating-point arithmetic to base 2. Here s is a real value (typically a positive integer), and ξ is a random variable (scaled random error) that can be discrete or continuous, and can depend on x, generating values from the interval $(-\frac{1}{2}, \frac{1}{2})$.

Floating-point arithmetic differs from real arithmetic only in that additional random errors are needed to model the loss of significance caused by the restriction of values to limited precision. Given a value x and a desired precision t, the randomization of x to t digits is implemented as

$$\mathsf{randomize}\,(x) \quad = \quad \begin{cases} x & \text{if } x \text{ can be expressed exactly with } t \text{ digits} \\ \mathsf{inexact}\,(x,\, t,\, \xi) & \text{otherwise.} \end{cases}$$

[2] This inexactness is often compounded during the course of a computation as inexact floating-point numbers take part in arithmetic operations that yield results that are even more inexact.

[3] Or, more loosely, whether the operation introduced *additional* roundoff error.

This yields x if x is exact within t digits, and otherwise superimposes a random perturbation so that its significance is bounded by t digits.

3.3 Random Rounding

Random rounding is simply rounding of a randomized value to the machine precision:

$$\mathsf{random\_round}\,(x) \quad = \quad \mathsf{round}\,(\ \mathsf{randomize}\,(x)\ ).$$

Here in the definition of $\mathsf{randomize}\,(x)$ we almost always take the random variable ξ to be uniformly distributed over $(-\frac{1}{2}, \frac{1}{2})$, with mean 0.

The advantage of this choice for the distribution is the absence of bias: the resulting rounding method has zero expected error. For a bounded arithmetic expression (which can also involve inexact values), the expected value of its computed Monte Carlo average will be its expected real value. Formally, if z is a random variable, and '$\mathsf{E}[\ z\ ]$' denotes the expected value (average value) of z, then with the uniform distribution just given, $\mathsf{E}[\ \mathsf{randomize}\,(z)\ ] \ = \ \mathsf{E}[\ z\ ]$, and moreover $\mathsf{E}[\ \mathsf{round}\,(\mathsf{randomize}\,(z))\ ] \ = \ \mathsf{E}[\ z\ ]$. When t equals the machine precision, in fact, $\mathsf{round}\,(\mathsf{randomize}\,(z))$ implements the same random rounding method proposed by other analysts, including Forsythe, Hull & Swensen, and Callahan (see p.13).

3.4 Random Unrounding

A natural randomized arithmetic comes from using input randomization: if '$\odot$' is the floating-point approximation to the '$\bullet$' operation with input randomization, then

$$x \ \odot \ y \quad = \quad \mathsf{round}\,(\ \ \mathsf{randomize}\,(x) \ \bullet \ \mathsf{randomize}\,(y)\ \ ).$$

Input randomization can be viewed naturally as **random unrounding** [58], which is the random conversion of a (rounded-off, inexact) floating-point value to a real value. In [58], Pierce develops a detailed implementation that maintains 'real' values in registers with higher precision than in memory, and uses rounding and random unrounding — storing to and loading from memory — only when necessary. Effectively, Pierce proposes implementing $p \gg t$, where p is the machine floating-point precision in registers and t is the precision in memory. This aspect of his scheme is similar to the IEEE 754 standard's Double-Extended scheme, and he in fact proposed an implementation using IEEE Double-Extended precision as a basis.

3.5 Monte Carlo Arithmetic

Full **Monte Carlo Arithmetic** (MCA) results if we combine random rounding with random unrounding. If '$\odot$' is the floating-point approximation to the '$\bullet$' operation, then in full Monte Carlo Arithmetic

$$x \ \odot \ y \quad = \quad \mathsf{round}\,(\ \mathsf{randomize}\,(\ \ \mathsf{randomize}\,(x) \ \bullet \ \mathsf{randomize}\,(y)\ \ )\ ).$$

Although this definition requires real arithmetic in theory, it can be implemented efficiently with finite precision, of course. Random digits can be generated incrementally, as needed.

By randomizing both the inputs and the output, full MCA achieves two important properties. First, input randomization detects catastrophic cancellation, which is the primary way that significant digits are lost in numerical computation. Second, output randomization gives random rounding, producing roundoff errors that are random and uncorrelated, with zero expected bias. All errors are modeled with random variables, and the virtual precision t can be changed as needed. The examples in Section 2 were executed with full MCA.

An additional feature of MCA is the ability to vary the effective precision of computation (even dynamically, if that is desired). The **virtual precision** t is the precision to which arithmetic values are represented (in memory). If implemented in floating-point with register precision p, we require $t \leq p$. By varying t in the definition of $\mathsf{randomize}(x)$ we implement arithmetic of any desired precision $t \leq p$. The ability to vary the virtual precision can be very useful, such as in evaluating the hardware precision requirements of a particular computation, so we allow t to differ from p.

3.6 Why Monte Carlo Arithmetic works

Catastrophic cancellation is a major source of loss of precision in floating-point computation. This cancellation is the loss of leading significant digits caused by subtracting two approximately equal operands (i.e., two operands whose difference has a smaller exponent than either operand). This is shown in Figure 1; boxed values denote floating-point values.

operand 1	$\boxed{\texttt{+3.495683} \times 10^{0}}$
operand 2	$\boxed{\texttt{+3.495681} \times 10^{0}}$
difference	$\texttt{+0.000002} \times 10^{0}$
normalized difference	$\boxed{\texttt{+2.000000} \times 10^{-6}}$

Figure 1: Catastrophic cancellation

The difference computed in Figure 1 has at most one significant digit, yet there is no way to detect this in modern computers. Floating-point arithmetic lacks a mechanism for recording that the trailing zero digits introduced by normalization are not significant.

Catastrophic cancellation occurs in all of the examples discussed above, and also is often the primary problem in horrific examples of numeric inaccuracy found in the numerical analysis literature. For example, in the first quadratic equation example earlier, catastrophic cancellation arises in computing the difference $(-b) - (\sqrt{b^2 - 4ac})$, since the two operands differ only in the final decimal digit.

Randomizing the trailing zero digits detects catastrophic cancellation. If subtraction loses ℓ leading digits, then ℓ trailing random digits will be in the result. This is illustrated in Figure 2. When computed multiple times, these randomized results will disagree on the trailing ℓ digits.

operand x	$\boxed{\texttt{+3.495683} \times 10^{0}}$	
$x' = \mathsf{randomize}(x)$		`+3.49568320391695941600884...`
operand y	$\boxed{\texttt{+3.495681} \times 10^{0}}$	
$y' = \mathsf{randomize}(y)$		`+3.49568191870795420835463...`
difference $x' - y'$		`+0.00000228520900520765421...`
rounded difference $\mathsf{round}(x' - y')$	$\boxed{\texttt{+2.2852090} \times 10^{-6}}$	

Figure 2: Example of input randomization (in a difference with catastrophic cancellation) with eight-digit decimal arithmetic ($t = p = 8$, $\beta = 10$). Boxed values are floating-point values.

In full MCA, randomization is used in all arithmetic operations. Input randomization detects catastrophic cancellation. Output randomization gives random rounding, producing roundoff errors that are random and uncorrelated, with zero expected bias. All errors are modeled with random variables, and the virtual precision t can be changed as needed.

4 Related Work

4.1 The statistical theory of error

The statistical nature of error has been important to scientists and numerical analysts for centuries, with a particularly strong early emphasis in the field of astronomy [60]. An important first step was taken by Thomas Simpson (developer of Simpson's rule for quadrature), who in a paper read to the Royal Society of London on 10 April 1755 proved that it was more accurate to compute a mean of six measurements than to take a single observation, under specific assumptions about the distribution of the observations.

> The conceptual development in Simpson's paper was his decision to focus, not on the observations themselves or on the astronomical body being observed, but on the errors made in the observations, on the differences between the recorded observations and the actual position of the body being observed. To those of us who are now used to dealing with such matters, this may seem like a trivial, even semantic, difference. To those in the mid-eighteenth century, ... it was the critical step that was to open the door to an applicable quantification of uncertainty. Simpson began by assuming a specific hypothesis for the distribution of the errors. He was able to focus his attention on the mean *error* rather than on the mean *observation*. Even though the position of the body observed might be considered unknown, the distribution of errors was, for Simpson, known. By basing his analysis upon this known distribution, he was able to come to grips with a stochastic structure for the unknown position. —— Stigler [60, pp.91-94].

Simpson built directly upon the work of De Moivre, who in 1738 had succeeded in showing that Bernoulli's binomial distribution tended asymptotically to the normal distribution [60, p.82]. The **normal distribution** of a variable x, having mean μ and standard deviation σ, is

$$\mathsf{Pr}[\, x \;\leq\; t \,] \quad = \quad \Phi(t) \quad = \quad \frac{1}{\sqrt{2\pi}\;\sigma} \int_{-\infty}^{t} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\, dx.$$

Many basic foundations of mathematical statistics were then developed by Laplace. Remarkably, although he published works on the normal distribution, error distributions, and Laplace transforms (developed from De Moivre and Simpson's generating functions), Laplace at first missed the idea of adopting the normal distribution as an error distribution, and e^{-x^2} as an error curve [60, p.143]. He spent much of the 1770s working on the idea of an error curve, and focused particularly on the probability density $\phi(x) = (m/2)\, e^{-m|x|}$ [60, p.111].

The relationship between the normal distribution and error was established in 1795 when, at the age of eighteen, Gauss conceived the method of least squares.[4] Inspired by the work of Laplace, Gauss' initial theory of errors [71, Chs.6–7] took errors to be normally distributed, which was supported by his hypothesis of elementary errors, i.e., that the total error in these observations is a sum of individual, independent errors of small variance. The probability that x has a value that lies within λ standard deviations of its mean is given by the **error function**

$$\mathsf{Pr}[\, |x-\mu| \;\leq\; \lambda\,\sigma \,] \quad = \quad \mathrm{erf}(\,\lambda/\sqrt{2}\,) \quad = \quad \frac{1}{\sqrt{2\pi}} \int_{-\lambda}^{\lambda} e^{-\frac{1}{2}z^2}\, dz.$$

For example, in [25, §9], Gauss noted that the probabilities of an error lying within $\lambda = 1$, $\lambda = 2.57$, and $\lambda = 3.89$ standard deviations are, respectively, 0.6827, 0.99, and 0.9999. Gauss used this connection to complete the principle of least squares, under which the solution to a system

[4] Or so Gauss later claimed, since his work was not published until 1809. Gauss became the target of scornful allegations by Legendre, who had published his treatise on least squares in 1805 (cf. [29, p.210], [60, pp.145-146]).

of linear equations with least squares from the observed values is also the solution maximizing the probability of the observed values, assuming that the observations are of the same degree of accuracy and are normally distributed [60, pp.140-141].

The impact of the method of least squares was immense; it was widely adopted in astronomy and geodesy [60, pp.39-40]. The normal distribution is still commonly referred to as the 'Gaussian' distribution, although it was developed much earlier by both DeMoivre and Laplace.

Seizing on Gauss' work, Laplace produced the Central Limit Theorem (see Appendix C.3) in 1810 and his classic *Théorie analytique des probabilités* in 1812. The latter introduced the normal law of experimental errors, the theory that governs the sum of a number of experimental errors and justifies a normal distribution with the Central Limit Theorem.

Gauss' justification of the normally distributed errors and the principle of least squares was somewhat circular, and Gauss himself found it unsatisfying. In the 1820s Gauss revised the theory of errors and the method of least squares in *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*, of which an excellent recent edition by Stewart is now available [25].

This work is remarkable in many ways, and mainstream scientific and statistical discipline have so thoroughly adopted its ideas that much of it today seems like common sense. Several noteworthy advances of this work were that Gauss removed his earlier assumption that the error distributions be normal (proposing instead to measure the central value and precision of a distribution by its mean μ and standard deviation σ, which he formulated in terms of moments [25, §5–8]), considered the problem of estimating the precision of the samples from their standard deviation, and criticized the use of the 'biased' standard deviation estimate

$$\widehat{\sigma}_{\text{naive}} \quad = \quad \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \widehat{\mu})^2} \ ,$$

where $\widehat{\mu}$ is the average of the samples. Gauss argued that it tends to overstate the precision, and that it should be replaced [25, §38] by the unbiased estimate

$$\widehat{\sigma} \quad = \quad \sqrt{\frac{1}{n-\rho} \sum_{i=1}^{n} (x_i - \widehat{\mu})^2}$$

where ρ (typically 1) is the number of parameters involved. This estimate is called **unbiased** since the expected value of $(\widehat{\sigma}^2 - \sigma^2)$ is zero.

Another significant contribution of *Theoria Combinationis*, on its first page, was the careful distinction between **random error** and **regular error** (systemic error resulting from the experimental mechanism, such as consistent error resulting from some measurement device). This distinction is still one of the first topics in texts on experimental error (cf. [61]), and is fundamental to experimental design. Random errors are much easier to analyze, both formally and experimentally.

The central idea behind Gauss' approach is the identification of random errors with random variables, which have underlying distributions and independence properties. An experimental quantity $\widetilde{x}$ is then viewed as having a true value x and an observational error (inaccuracy, inexactness, uncertainty) ξ, which is modeled as a random variable.

4.2 Monte Carlo methods

Monte Carlo methods [19, 32, 33] solve problems by treating them as *experiments involving random variables*. The random variables can represent truly random physical processes, or just abstractions of deterministic processes. The experiments are performed multiple times ('Monte Carlo simulation'), and sampling methods or techniques of statistical inference are used to make conclusions about the results of these experiments.

If we think of an experiment obtaining a vector-valued result $\mathbf{R}$, which is a function of random variables $\xi_1, \xi_2, \ldots, \xi_m$, then for some number $n > 0$ the Monte Carlo simulation computes

$$\mathbf{R}(\, \xi_1, \xi_2, \ldots, \xi_m \,)$$

for n sequences of m random values $(\, \xi_1, \xi_2, \ldots, \xi_m \,)$. Their average is an unbiased estimate of

$$\int\int \cdots \int \mathbf{R}(\, x_1, x_2, \ldots, x_m \,)\; dx_1\; dx_2\; \cdots\; dx_m.$$

The true average μ and variance σ^2 of the random variables (assuming a common distribution) determine the accuracy of this integral. The computed standard deviation $\widehat{\sigma}$ gives an estimate of σ that improves as the number of samples n increases. Computer implementations rely on high-quality pseudo-random number generators, intelligent sampling, and a number of techniques (such as variance reduction) for eliminating the uncertainty arising from this experimental approach.

Monte Carlo methods were developed around 1947, with much of the early impetus for their formal development coming in 1945 from the successful nuclear tests at Alamogordo and the completion of ENIAC, the first general-purpose electronic computer. A review of ENIAC in 1946 brought together Enrico Fermi, Stanislaw Ulam, and John von Neumann. In order to simulate random neutron diffusion in fissile material (extending the atomic bomb development of World War II), Ulam and von Neumann developed the Monte Carlo approach throughout 1947 [33, 53]. By 1949 the method worked impressively on the ENIAC and von Neumann, Ulam, Fermi, Metropolis and others had produced elegant theoretical results [49].

Interest in Monte Carlo methods in the 1950s spread to many areas of numerical analysis, and this is reflected in the paper by Forsythe and Leibler [20] (elaborating an idea of von Neumann and Ulam), and in Householder's book [38]. Although practical frustrations, notably meager computing resources, put a damper on this naïve enthusiasm, Monte Carlo methods have found heavy use in physical simulation, optimization, and evaluation of multi-dimensional and awkward integrals. Today Monte Carlo analysis is enjoying a resurgence of interest [19, 54].

4.3 Statistical roundoff analysis

The use of statistical methods in numerical analysis arguably began with the pioneering works of von Neumann and Goldstine in 1947 [55] and 1951 [28], who decided to show that the 1943 gloomy exponential forward error bounds for Gaussian elimination of Hotelling [37, p.7] were not reflective of practice. An excellent summary of these papers is given by Wilkinson [72], and also by Higham [36, §9.6], who reproduces the following commentary by Goldstine about this work:[5]

> We did not feel it reasonable that so skilled a computer as Gauss would have fallen into the trap that Hotelling thought he had noted ... Von Neumann remarked one day that even though errors may build up during one part of the computation, it was only relevant to ask how effective is the numerically obtained solution, not how close were some of the auxiliary numbers, calculated on the way to their correct counterparts. We sensed that at least for positive definite matrices the Gaussian procedure could be shown to be quite stable. — Goldstine [30, p.290]

In [55, p.1036], von Neumann and Goldstine argued that modeling roundoff errors with independent probabilistic estimates is natural since we know their average and worst-case values, and are ignorant of their exact distribution. In the sequel paper [28], following the development of Monte Carlo, Goldstine and von Neumann gave statistical bounds on the results of Gaussian elimination using the norms of a random matrix. Turing, in his 1948 analysis of Gaussian elimination that originated the idea of *LDU* matrix decomposition, had made an analysis for random matrices also [62, p.299].

[5]This perspective of Von Neumann addresses only Gaussian elimination, and not, for example, iterative computations like those in Section 2.2.

Early statistical analyses of rounding methods, leading digit frequencies, etc., are surveyed by Knuth [46] and Sterbenz [59, §3.1.2]

Probably the best recent work in statistical analysis of error in numerical computations is by Chaitin-Chatelin and her coworkers, who analyze the effects of specific perturbations on the robustness of numerical algorithms (e.g., [12, 13], and very recently [14]). Her work is exceptionally clear and rigorous, and careful about its assumptions. Since 1988 Chaitin-Chatelin has developed a MATLAB toolbox called PRECISE that allows users to perform statistical backward error analysis and sensitivity analysis experiments, with emphasis on linear system solution, eigencomputations, polynomial root finding, and general nonlinear (matrix or polynomial) equation solving under componentwise or normwise perturbations [14].

4.4 Random rounding

Let x be a real number that is to be rounded to a floating-point format F. Let $x_\downarrow$ be the greatest (normalized) floating-point number in the format F such that $x_\downarrow \le x$ and let $x^\uparrow$ be the least (normalized) floating-point number in the format F such that $x \le x^\uparrow$. The *random rounding* of x is implemented by drawing a random variable r from the uniform distribution $[\,0, x^\uparrow - x_\downarrow\,]$, then chopping $x + r$. The probability that this will result in $x_\downarrow$ is proportional to $(x^\uparrow - x)$, and that it will result in $x^\uparrow$ is proportional to $(x - x_\downarrow)$. This definition is equivalent to that in Section 3.3.

Random rounding was considered in the early 1950s by George Forsythe [21, 22]. He noted [23] that, although Huskey's recent work [40] on the ENIAC showed roundoff errors in some problems are not distributed like independent random variables[6], better error estimates could be obtained if they *were*. He suggested that they can be forced to be random:

> It seems clear that in the integration of smooth functions the ordinary rounding-off errors will frequently not be distributed like independent random variables.
> To circumvent [the problems of correlated errors ϵ noted by Huskey], the present writer [21] has proposed a *random rounding-off* procedure which make ϵ a true random variable. Suppose, for example, that a real number u is to be rounded off to an integer. Let $[u]$ be the greatest integer not exceeding u, and let $u - [u] = v$. In the proposed procedure u is "rounded up" to $[u] + 1$ with probability v, and "rounded down" to $[u]$ with probability $1 - v$, the choice being made by some independent chance mechanism. The rounding-off error is thus a random variable with $E(\epsilon) = 0$ and $E(\epsilon^2) = v(1 - v)$. Since $v(1 - v) \le \frac{1}{4}$ one can give probabilistic bounds for the accumulated error which are independent of the distribution of v. The method can be reasonably simulated in machine computation: On a decimal machine, instead of adding a 5 in the most significant position of the digits to be dropped (ordinary rounding off), one adds a random decimal digit to each of the digital positions to be dropped. —— G. Forsythe [23]

In concluding [21], Forsythe announces: "Tests with I.B.M. equipment indicate that random round-off probably eliminates *a priori* the peculiarities of round-off found by Huskey on the ENIAC."

Henrici [34, ch.5] gives a good review of early statistical analyses of roundoff error in numerical integration, and argues that it can be analyzed statistically. Random rounding ('probabilistic rounding') was used by Hull and Swenson [39] in order to test his hypothesis that ordinary rounding can be modeled statistically as a random process. Their experiments confirmed that the cumulative roundoff errors obtained with random rounding were similar to those obtained with ordinary rounding, and that probabilistic models of roundoff propagation are generally valid. Henrici was intrigued by this experimental approach, and showed that their results could be predicted analytically to within an error of 10% [35].

Callahan [10] demonstrates the use of random rounding in signal processing applications, stressing the usefulness of truly random roundoff errors. He discusses an often-used simple recursive integrator that can have "dramatic correlated quantization error effects unless random rounding is employed" [10, p.501].

[6]See also [44] and [36, §1.17,§2.6].

4.5 Automatic monitoring of significance

Significance arithmetic was popularized by Ashenhurst and Metropolis in a series of papers arguing the need for numerical systems to track the accuracy (significant digits) of their results [3, 4, 50]. A basic problem is defining what is meant by "the number of significant digits" [59, §3.1]. Significance arithmetic is difficult to implement when this measure is a real number, but is less compelling when it is always an integer.

Their work was one of the inspirations for the Project Stretch design [9]. Its floating-point system included a distant relative of random normalization, called 'noisy mode', that could be used to monitor significance loss:

> To aid in significance studies, a *noisy mode* is provided in which the low-order bits of results are modified. Running the same problem twice, first in the normal mode and then in the noisy mode, gives an estimate of the significance of the results. —— [9, p.25]

> After an extensive search, the most effective technique [for monitoring significance loss] turned out to be both elegant and remarkably simple. ... Any operand or result that is shifted left to be normalized requires a corresponding number of zeros to be shifted in at the right. ... In the *noisy mode* these numbers are simply extended with 1s instead of zeros (1s in a binary machine, 9s in a decimal machine). Now all numbers tend to be too large in absolute value. The true value, if there had been no significance loss, should lie between these two extremes. Hence, two runs, one made without and one made with the noisy mode, should show differences in result that indicate which digits may have been affected by significance loss.
> The principal weakness of the noisy-mode procedure is that it requires two runs for the same problem. A much less important weakness is that the loss of significance cannot be guaranteed to show up — it merely has a very high probability of showing up — whereas built-in significance checks can be made slightly pessimistic, so that actual significance loss will not be greater than indicated. On the other hand, little extra hardware and no extra storage are required for the noisy-mode approach. Furthermore, significance loss is relatively rare, so that running a problem twice when significance loss is suspected does not pose a serious problem. What is serious is the possibility of *unsuspected* significance loss. —— [9, p.102]

Interval computation [51, 52], which generalizes floating-point numbers from single values to statistical objects, has failed to gain widespread popularity, because it is very pessimistic [72, pp.566–567]. Interval arithmetic and significance arithmetic were considered by Turing as early as 1946 [72, p.566].

4.6 Stochastic computer arithmetic

Randomized numerical methods have been studied since the early 1970s by Vignes, who presented the idea at the IFIP conference in 1974 [63]. The paper [48] apparently spawned the 'permutation-perturbation' method, performing Gaussian elimination on a matrix both with random initial permutation of the matrix columns, and with random perturbation of some of the matrix entries (setting their least significant bits to 0 or 1, i.e., perturbing by 0 or $\pm 2^{-p}$). In [63] 'permutation' has evolved to mean changing the order in which additions are performed, and 'perturbation' is the addition of a random 0 or 1 value to the least significant bit of a floating-point fraction. These were both implemented in a single FORTRAN function `P`, later called `PEPER` [64]. Vignes and Ung patented the idea in Europe in 1979 [65], and in the USA in 1983 [66].[7] The retrospective survey [67] reviews the results of a decade of research on the permutation-perturbation method.

In dozens of subsequent papers, Vignes and coworkers refer to the 'permutation-perturbation' method as the CESTAC (Contrôle et Estimation STochastique des Arrondis de Calcul) method. As described in [8], CESTAC works on numerical programs in which each floating-point expression '$X \bullet Y$' has been preprocessed to $\mathtt{PER}(X \bullet Y)$, so for example $X+Y+Z$ becomes $\mathtt{PER}(\mathtt{PER}(X+Y)+Z)$,

[7] Kahan remarks that the CESTAC patents can be circumvented with IEEE standard arithmetic by randomly toggling the IEEE directed rounding control bits [44, p.19].

where PER is a function that implements random perturbation. With CESTAC, the program is executed 2 or 3 times, and its results (output variables) are stored. Finally the mean value and number of significant digits of these results are computed and printed by a module that performs a Student t computation on their 2 or 3 values. Vignes' survey [69] accompanies an entire journal issue with papers describing applications of CESTAC.

Both PER and the perturbation mode of PEPER work identically [8, 69]: when the underlying machine arithmetic works with ordinary rounding, PER adds a least significant bit 1 with probability $\frac{1}{4}$, adds 0 with probability $\frac{1}{2}$, and subtracts 1 with probability $\frac{1}{4}$. When the underlying arithmetic is chopping, it adds 0 with probability $\frac{1}{2}$, and adds 1 with probability $\frac{1}{2}$. This method can be implemented efficiently, and without extended precision.

More recently, CESTAC has been reformulated as **stochastic arithmetic** and incorporated in the CADNA (Control of Accuracy and Debugging for Numerical Applications) library, which implements stochastic arithmetic for FORTRAN and Ada programs [70]. Stochastic arithmetic includes not only the basic arithmetic operators with perturbation, but also operators for comparing stochastic values (one can statistically test the hypotheses $x < y$ or $x = y$, for stochastic values x and y). This is a significant change from CESTAC. It requires a different implementation, in which each arithmetic *expression* is treated like a CESTAC *program*: it is evaluated 2 or 3 times, then the results averaged (and tested statistically if desired). A value that has no significant digits is treated as an 'informational zero' [68], and when tested by a comparison operator these zeroes produce exceptions.

Kahan [44] has raised strong objections to the CESTAC approach, taking special issue with its assumption that roundoff errors are normally distributed. Kahan demonstrates examples for which a CESTAC-based software package makes "extravagantly optimistic" claims of accuracy. He takes a strong stand, saying at one point that "probabilistic estimates of error are probably useless or worse" [44, p.7]. Notice that MCA makes no such assumptions or claims; a detailed comparison of MCA with CESTAC and CADNA is made in [57].

Another significant problem presented by CADNA is that comparisons, and hence boolean values, can become inexact. In CADNA, comparisons are based on statistics [2, p.133]:

$$x \otimes y \quad \equiv \quad \widehat{\mu}_x - \widehat{\mu}_y \quad < \quad \lambda_\alpha \sqrt{\widehat{\sigma}_x^2 + \widehat{\sigma}_y^2}$$

where $\widehat{\mu}_x$ is the computed average of x, $\widehat{\sigma}_x$ is its computed standard deviation, and λ_α is half the Student t confidence interval width depending on the desired confidence level $1 - \alpha$. Thus statistical inference is embedded in the comparison operator, and stochastic arithmetic is similar to the classical theory of propagation of error (or uncertainty) [61].

MCA presents this problem as well. In MCA, we recommend defining comparisons in terms of the underlying arithmetic operations, as programmers often assume this [27]:

$$x \otimes y \quad \equiv \quad (x \ominus y) \quad < \quad 0.$$

Actually, *every* fixed-precision arithmetic system will present this problem. Only in the idealized situation when the operands are exact is there a clear conceptual difference between the '$\otimes$' operation (float × float → *boolean*) and the '$\ominus$' operation (float × float → *float*), since only in this situation can the former operation be implemented exactly. When the operands are inexact, on the other hand, both operations can give inexact results.

Chatelin and Brunet warn [12] that perturbation of the algorithm itself (affecting its branching structure, and not just perturbing its input) probably makes the function computed by the program nonanalytic, and requires qualitatively different analysis. We agree. Inexact real arithmetic has both an intuitive appeal and a practical usefulness that inexact boolean algebra decidedly lacks. Although we suspect they are inherently inferior, programs whose execution behavior differs over different samples clearly present serious problems for both CADNA and MCA, and for now we leave it open how best to deal with them — either precluding them wholesale, or allowing certain classes of such programs that permit formal analysis.

Other statistical studies of computer arithmetic include [6, 7, 17, 47, 56]. In [56], Parker develops a statistical theory of relative errors in floating-point computation that generalizes floating-point numbers to real-valued distributions (represented to arbitrary precision by their mean and higher moment values), and floating-point operations to operations on distributions. However, while elegant and easily implemented, it usually gives poor results in computations that produce catastrophic cancellation, and it cannot represent distributions whose support interval includes zero (as relative error is undefined at zero).

5 Some Benefits of Monte Carlo Arithmetic

MCA makes computer arithmetic more like real arithmetic. Randomization transforms roundoff from systemic error to random error, and random errors are much easier to deal with, both formally and informally. By transforming floating-point arithmetic into a Monte Carlo discipline we obtain many useful statistical properties.

5.1 Some floating-point anomalies are avoided

It is well-known that floating-point arithmetic is not associative. Knuth gives the following example for eight-digit decimal arithmetic [45, p.196]:

$$\begin{array}{rclcl} (\ 11111113. \oplus -11111111.\ ) \oplus 7.5111111 & = & 2.0000000 \oplus 7.5111111 & = & 9.5111111; \\ 11111113. \oplus (\ -11111111. \oplus 7.5111111\ ) & = & 11111113. \oplus -11111103. & = & 10.000000. \end{array}$$

This anomaly is a direct manifestation of catastrophic cancellation. Associativity fails after the cancellation removes all but the least significant digit, which is affected by roundoff errors.

	(11111113. ⊕ −11111111.) ⊕ 7.5111111			11111113. ⊕ (−11111111. ⊕ 7.5111111)		
n	$\widehat{\mu}$	$\pm$	$\widehat{\sigma}/\sqrt{n}$	$\widehat{\mu}$	$\pm$	$\widehat{\sigma}/\sqrt{n}$
10	9.62506	±	0.11484	9.40092	±	0.27888
100	9.49476	±	0.04241	9.42260	±	0.06533
1000	9.51095	±	0.01295	9.49816	±	0.02042
10000	9.50977	±	0.00411	9.51206	±	0.00645
100000	9.51014	±	0.00129	9.51396	±	0.00204
1000000	9.51093	±	0.00041	9.51159	±	0.00065
10000000	9.51112	±	0.00013	9.51111	±	0.00020

Table 9: Result of performing the indicated sums in single precision MCA with uniform input and output randomization. Notice the convergence to the exact sum value 9.5111111.

With MCA, these anomalies are avoided because of the zero expected bias of random rounding. In particular, addition becomes associative. Table 9 gives a computational demonstration, showing that standard errors decrease with increasing numbers of samples. Different computed sums agree up to the standard error, and the average converges to the exact sum in the limit where the number of samples goes to infinity. Even for small values of n, the error in the average is bounded by a constant times the standard error, so we can say 'addition is associative up to the standard error of the sum'. With the right statistical caveats, we can thus formally prove that Monte Carlo arithmetic avoids certain floating-point anomalies. See [57] for more details.

5.2 Roundoff errors actually do become random

Kahan [44] and others (e.g., Higham [36, §1.17,§2.6]) argue that statistical analyses of roundoff error are improperly founded because they assume roundoff errors are random. For functions like

$$rp(x) \quad = \quad \frac{622 \ - \ x \cdot (751 - x \cdot (324 - x \cdot (59 - 4 \cdot x)))}{112 \ - \ x \cdot (151 - x \cdot (72 - x \cdot (14 - x)))}$$

which are sensitive to perturbation, they show its roundoff errors are not randomly distributed at all with a plot like that in Figure 3.

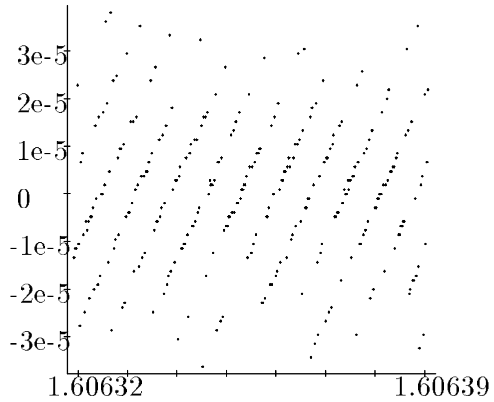

Figure 3: $rp(x) - rp(u)$, for $u = 1.60631924$ and $x = u, (u+\epsilon), \ldots, (u+300\epsilon)$ where $\epsilon = 2^{-24}$ —— computed with single precision IEEE arithmetic.

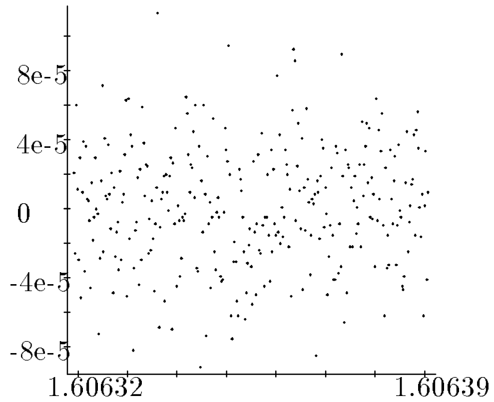

Figure 4: $rp(x) - rp(u)$, for $u = 1.60631924$ and $x = u, (u+\epsilon), \ldots, (u+300\epsilon)$ where $\epsilon = 2^{-24}$ —— computed with single precision MCA (uniform input randomization, round to nearest).

With MCA, this argument does not work. This is demonstrated with the equivalent plot, Figure 4, produced with MCA using uniform input randomization and deterministic IEEE default rounding. Randomization forces the roundoff errors to be random. Forsythe predicted randomization would have this effect [21], but because this problem has considerable cancellation, we have used input randomization, and not just the random rounding proposed by Forsythe. Figure 5 also shows the distribution of these values. The computation of rp involves 16 arithmetic operations, and the resulting distribution is quite close to normal.

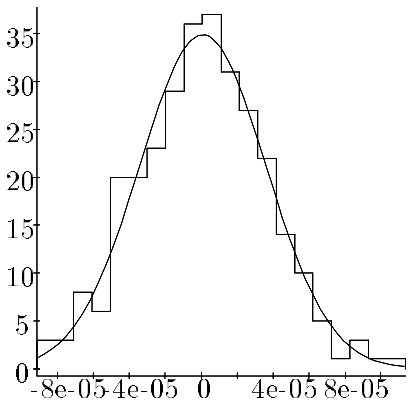

Figure 5: Histogram of values of $rp(x) - rp(u)$ computed with MCA in the previous figure. The normal density (for the average and standard deviation of these values) is superimposed.

5.3 Variable precision is supported

Figure 6 shows the result of varying the virtual precision while computing the Tchebycheff polynomial $T_{20}(0.75)$, using the factored representation

$$T_{20}(z) \quad = \quad 1 \; + \; 8\,z^2\;(z-1)\;(z+1)\;\left(4\,z^2+2\,z-1\right)^2\;\left(4\,z^2-2\,z-1\right)^2\;\left(16\,z^4-20\,z^2+5\right)^2.$$

This computation involves only very mild cancellation and small constants. As the figure shows, increasing the virtual precision t to the single precision maximum of 24 has the desired effect of gradually increasing the accuracy of the result, modulo peculiarities in the binary representation of the intermediate results. Thus it is possible to get a qualitative sense of the accuracy of 15-bit or 10-bit computation, for example.

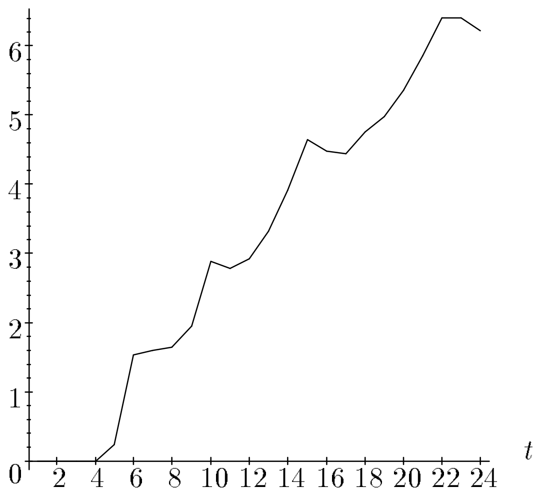

Figure 6: Number of significant decimal digits (negative base-10 log of the relative error) in the value $T_{20}(0.75)$ computed with full MCA (100 samples), at the indicated binary virtual precision t.

6 Conclusion

We have argued that Monte Carlo Arithmetic (MCA) has interesting practical uses in numerical computations. We sketched how MCA can give useful results without imposing a great deal of overhead. Other examples and case studies are presented in [57]. Although it is certainly no panacea, MCA does give perspective and does give reasonable estimates on the accuracy of computed results.

MCA thus appears to give an alternative way for a wide spectrum of numerical program users, without special training, to gauge the sensitivity of their program output to perturbations in their input and to roundoff errors. Running a program multiple times with MCA yields a distribution of sample values. Statistical analysis of the distribution then can give a rough intuitive sense, or rigorously established confidence intervals, about the roundoff error and the instability of the program (i.e., its sensitivity to rounding errors).

Specific other points that can be argued about MCA include:

- MCA is a simple way to bring some of the benefits of the Monte Carlo method [38] to floating-point computation. The Monte Carlo method offers simplicity; it replaces exact computation with random sampling, and replaces exact analysis with statistical analysis.

- MCA is a probabilistic way to detect occurrences of catastrophic cancellation in numeric computations. This gives an optional *'idiot light'* for numeric computations, that can be used at fairly low cost, and can be used without changing existing programs. While large standard deviation values are not guaranteed to reflect numerical instability, or vice versa, they are a warning signal that strongly recommends further analysis.

- MCA gives a way to maintain information about the number of significant digits in conventional floating-point values. We believe many users can appreciate the significance perspective on error analysis, though they find backward analysis hard to understand.

- MCA is a way to formally circumvent some anomalies of floating-point arithmetic. Although floating-point summation is not associative, for example, Monte Carlo summation is 'statistically associative' (i.e., associative up to the standard error of the result).

- MCA can be used like any other rounding method, and thus it can be used without changing existing programs. Thus MCA should have wide applicability.

Perhaps the strongest argument for MCA is that it is a way to make numerical computing more *empirical.* Thus it may encourage consumers of numerical software to investigate the quality of the results they are getting. We suspect that many certified numerical algorithms are giving inaccurate results in practice, mainly because they are being misused (being applied to ill-conditioned or stiff problems, or resting on assumptions that do not hold). An experimental attitude definitely helps in getting high-quality numerical results.

References

[1] American National Standards Institute, *ANSI/IEEE Std 754-1985: IEEE standard for binary floating-point arithmetic*, New York, 12 Aug. 1985.

See also: "An American National Standard, IEEE standard for binary floating-point arithmetic", *SIGPLAN Notices*, **22**:2, 9–25, Feb. 1987.

[2] R. Alt, J. Vignes, "Validation of results of collocation methods for ODEs with the CADNA library", *Appl. Numer. Math.* **21**:2, 119–139, June 1996.

[3] R.L. Ashenhurst, N. Metropolis, "Unnormalized Floating Point Arithmetic", *J. ACM* **6**:3, 415–428, July 1959.

[4] R.L. Ashenhurst, N. Metropolis, "Error Estimation in Computer Calculation", in *Computers and Computing*, AMM Slaught Memorial Papers, *American Mathematical Monthly* **72**:2, 47–48, February 1965.

[5] J.-C. Bajard, D. Michelucci, J.-M. Moreau, J.-M. Muller, Introduction to the Special Issue on "Real Numbers and Computers", *Journal of Universal Computer Science* **1**:7, page 438, Jul 28, 1995. Available on the internet at many sites.

[6] J.L. Barlow, E.H. Bareiss, "On Roundoff Error Distributions in Floating Point and Logarithmic Arithmetic", *Computing* **34**:4, 325–347, 1985.

[7] R.P. Brent, "Static and Dynamic Numerical Characteristics of Floating-Point Arithmetic", *IEEE Trans. Comput.* **C-22**:6, 598–607, June 1973.

[8] M.-C. Brunet, F. Chatelin, "CESTAC, a tool for a stochastic round-off error analysis in scientific computing", pp. 11–20 in *Numerical Mathematics and Applications*, R. Vichnevetsky and J. Vignes (eds.), Elsevier/North-Holland, 1986.

[9] W. Buchholz, *Planning a Computer System: Project Stretch*, NY: McGraw-Hill, 1962.

[10] A.C. Callahan, "Random rounding: Some principles and applications", *Proc. 1976 IEEE International Conference on Acoustics, Speech and Signal Processing*, Philadelphia, PA, USA, 12-14 April 1976, 501–504, 1976.

[11] S.L. Campbell, C.D. Meyer, Jr., *Generalized Inverses of Linear Transformations*, NY: Pitman, 1979; reprinted by Dover Publications, 1991.

[12] F. Chatelin, M.-C. Brunet, "A probabilistic round-off error propagation model. Application to the eigenvalue problem", in [16], 139–160, 1990.

[13] F. Chatelin, V. Frayssé, "Elements of a Condition Theory for the Computational Analysis of Algorithms", in *Iterative Methods in Linear Algebra*, R. Beauwens and P. de Groen (eds.), Elsevier North-Holland, 15–25, 1992.

[14] F. Chaitin-Chatelin, V. Frayssé, *Lectures on Finite Precision Computations*, Philadelphia: SIAM, 1996.

[15] W.J. Cody, J.T. Coonen, D.M. Gay, K. Hanson, et al. "A proposed radix- and word-length-independent standard for floating-point arithmetic", *SIGNUM Newsletter* **20**:1, 37–51, Jan. 1985.

[16] M.G. Cox, S. Hammarling, *Reliable Numerical Computation*, NY: Oxford University Press, 1990.

[17] J.W. Demmel, "The probability that a numerical analysis problem is difficult", *Mathematics of Computation* **50**:182, 449–480, April 1988.

[18] W. Feller, *An Introduction to Probability Theory and its Applications*, NY: J. Wiley & Sons, 1968.

[19] G.S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, NY: Springer-Verlag, 1996.

[20] G.E. Forsythe, R.A. Leibler, "Matrix inversion by a Monte Carlo method.", *Mathematical Tables and Other Aids to Computation* **4**, 127–129, 1950.

[21] G.E. Forsythe, "Round-off errors in numerical integration on automatic machinery. Preliminary report", *Bull. AMS* **56**, 61, 1950.

[22] G.E. Forsythe, "Note on rounding-off errors" (review by J. Todd), *Math. Rev.* **12**, 208, 1951.

[23] G.E. Forsythe, "Reprint of a note on rounding-off errors", *SIAM Review* **1**:1, 66–67, 1959. Originally written June 1950 at the National Bureau of Standards, Los Angeles, CA, and abstracted in [22].

[24] G. Forsythe, "Solving a quadratic equation on a computer", *The Mathematical Sciences*, MIT Press, 1969.

[25] C.F. Gauss, *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae (Theory of the Combination of Observations Least Subject to Errors)*, translated by G.W. Stewart, Philadelphia, PA: SIAM, 1995.

[26] P.E. Gill, W. Murray, M.H. Wright, *Numerical Linear Algebra and Optimization*, Addison-Wesley, 1991.

[27] D. Goldberg, "What every computer scientist should know about floating-point arithmetic", *ACM Computing Surveys* **23**:1, 5–48, March 1991.

[28] H.H. Goldstine, J. von Neumann, "Numerical inverting of matrices of high order. II", *Proc. Amer. Math. Soc.* **2**, 188–202, 1951.

[29] H.H. Goldstine, *A History of Numerical Analysis from the 16th Through the 19th Century*, NY: Springer-Verlag, Studies in the History of Mathematics and Physical Sciences 2, 1977.

[30] H.H. Goldstine, *The Computer: From Pascal to von Neumann*, Princeton University Press, 1993.

[31] G.H. Golub, C.F. Van Loan, *Matrix Computations: Second Edition*, Baltimore: Johns Hopkins University Press, 1989.

[32] J.H. Halton, "A Retrospective and Prospective Survey of the Monte Carlo Method", *SIAM Review* **12**:1, 1–63, 1970.

[33] J.M. Hammersley, D.C. Handscomb, *Monte Carlo Methods*, NY: Chapman and Hall, 1965.

[34] P. Henrici, *Error Propagation for Difference Methods*, NY: J. Wiley & Sons, 1963.

[35] P. Henrici, "Tests of Probabilistic Models for the Propagation of Roundoff Errors", *Comm. ACM* **9**:6, 409–410, June 1966.

[36] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, Philadelphia, PA: SIAM, 1996.

[37] H. Hotelling, "Some new methods in matrix inversion", *Ann. Math. Stat.* **14**, 1–34, 1943.

[38] A.S. Householder, *Principles of Numerical Analysis*, NY: McGraw-Hill, 1953. Reprinted by Dover Publications, 1974.

[39] T.E. Hull, J.R. Swenson, "Tests of Probabilistic Models for Propagation of Roundoff Errors", *Comm. ACM* **9**:2, 108–113, February 1966.

[40] H.D. Huskey, "On the precision of a certain procedure of numerical integration", *J. Research National Bureau of Standards* **42**, 57–62, 1949.
Includes the appendix: D.R. Hartree, "Note on Systematic Rounding-off Errors in Numerical Integration", p.62.

[41] W. Kahan, "A survey of error analysis," invited paper, *Proc. IFIP Congress 1971*, 200–206, August 1971.

[42] W. Kahan, "The programming environment's contribution to program robustness," *SIGNUM Newsletter*, Oct. 1981, p.10.

[43] W.V. Kahan, "Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic" (work in progress), Dept. of Elect. Eng. & Computer Science, UC Berkeley, dated May 31, 1996. Currently available as:
`http://http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps`

[44] W.V. Kahan, "The Improbability of PROBABILISTIC ERROR ANALYSES for Numerical Computations", lecture notes prepared for the UC Berkeley Statistics Colloquium, 28 February 1996, and subsequently revised (4 March 1996). (An earlier version of this lecture was presented at the third ICIAM Congress, 3–7 July, 1995.) Currently available as:
`http://http.cs.berkeley.edu/~wkahan/improber.ps`

[45] D.E. Knuth, *The Art of Computer Programming. Vol. II: Seminumerical Algorithms*, Addison-Wesley, 1969.

[46] D.E. Knuth, *The Art of Computer Programming. Vol. II: Seminumerical Algorithms*, 2nd edition, Addison-Wesley, 1981.

[47] D. Kuck, D.S. Parker, A.H. Sameh, "Analysis of Floating-Point Rounding Methods", IEEE Trans. Comput. **C-26**:7, 643–650, July 1977.

[48] M. La Porte, J. Vignes, "Méthode numerique de détection de la singularité d'une matrice", *Numer. Math* **23**:1, 73–81, 1974.

[49] N. Metropolis, S. Ulam, "The Monte Carlo method", *J. Amer. Stat. Assoc.* **44**, 335, 1949.

[50] N. Metropolis, "Analyzed Binary Computing", *IEEE Trans. Comput.* **C-22**:6, 573–576, June 1973.

[51] R.E. Moore, *Interval Analysis*, Englewood Cliffs, NJ: Prentice-Hall, 1966.

[52] R.E. Moore, *Methods and Applications of Interval Analysis*, Philadelphia, PA: SIAM, 1979.

[53] N. Metropolis, "The Beginning of the Monte Carlo method", *Los Alamos Science* **15**, 125–130, 1987.

[54] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, Philadelphia: SIAM, 1992.

[55] J. von Neumann, H.H. Goldstine, "Numerical inverting of matrices of high order", *Bull. Amer. Math. Soc.* **53**, 1021–1099, 1947.

[56] D.S. Parker, "The Statistical Theory of Relative Errors in Floating-Point Computation", M.S. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1976.

[57] D.S. Parker, "Monte Carlo Arithmetic: systematic random improvements upon floating-point arithmetic", UCLA Computer Science Department, Technical Report CSD-970002, February 1997.

[58] B.A. Pierce, *Applications of randomization to floating-point arithmetic and to linear systems solution*, Ph.D. dissertation, UCLA Computer Science Department, December 1996.

[59] P.H. Sterbenz, *Floating-Point Computation*, Englewood Cliffs, NJ: Prentice-Hall, 1974.

[60] S.M. Stigler, *The History of Statistics: The Measurement of Uncertainty before 1900*, Cambridge, MA: Belknap/Harvard U. Press, 1986.

[61] J.R. Taylor, *An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*, Mill Valley, CA: University Science Books (Oxford University Press), 1982.

[62] A.M. Turing, "Rounding-Off Errors in Matrix Processes", *Quart. J. Mech.* **1**, 287–308, 1948.

[63] J. Vignes, M. La Porte, "Error Analysis in Computing", *Proc. IFIP 1974*, North-Holland, 610–614, 1974.

[64] J. Vignes, "New methods for evaluating the validity of the results of mathematical computations", *Mathematics and Computers in Simulation* **XX**, 227–249, 1978.

[65] J. Vignes, V. Ung, Procédé et ensemble de calcul aléatoirement par défaut ou par excès, pour fournir des résultats de calcul avec le nombre de chiffres significatifs exacts, European Patent No. 7902784 (1979).

[66] J. Vignes, V. Ung, Arrangement for determining number of exact significant figures in calculated results, U.S. Patent 4,367,536 (1983).

[67] J. Vignes, R. Alt, "An Efficient Stochastic Method for Round-off Error Analysis", in *Accurate Scientific Computations* (LNCS #235), W.L. Miranker and R.A. Toupin (eds.), NY: Springer-Verlag, 183–205, 1985.

[68] J. Vignes, "Zéro mathématique et zéro informatique", *Comptes Rendus de l'Academie des Sciences, Serie I (Mathématique)* **303**:20, 997–1000, 21 Dec. 1986.

[69] J. Vignes, "Review on stochastic approach to round-off error analysis and its applications," *Mathematics and Computers in Simulation* **30**:6, 481–491, December 1988.

[70] J. Vignes, "A stochastic arithmetic for reliable scientific computation", *Mathematics and Computers in Simulation* **35**, 233–261, 1993.

[71] B.L. van der Waerden, *Mathematical Statistics*, 2nd edition, NY: Springer-Verlag, 1969.

[72] J.H. Wilkinson, "Modern Error Analysis", *SIAM Review* **13**:4, 548–568, October 1971.