Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic

D. Stott Parker
stott@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 includes random rounding — which forces roundoff errors to be randomly distributed — and precision bounding — which limits the number of significant digits in a given value by random perturbation. Random rounding can be used to produce roundoff errors that are truly random and uncorrelated, and that have zero expected bias. Precision bounding can be used to vary precision dynamically, to implement inexact values (values known to only a few significant digits), and most importantly to detect catastrophic cancellation, which is the primary way that significant digits are lost in numerical computation.

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. Unlike much previous work in this area, MCA makes no assumptions about the resulting roundoff error distributions, such as that they are normal. By running a program multiple times, one directly measures the sensitivity of particular outputs to random perturbations of particular inputs. MCA thus gives a way to implement roundoff analysis, using random variables for roundoff errors, so that the roundoff distributions can be studied explicitly. It encourages an empirical approach to evaluating numerical quality, and gives a way to exploit the Monte Carlo method in numerical computation.

MCA also generally gives a different perspective on the study of error. For example, while floating-point summation is not associative, Monte Carlo summation is "statistically associative" up to the standard error of the sum. A statistical approach avoids anomalies of floating-point arithmetic.

This work summarizes ways in which MCA has promise as a tool in numerical computation. It seems particularly promising as a way for the person on the street to estimate the number of significant digits in a floating-point value, and to experiment with the effect of changing the precision used in numerical computation. Numerical modeling is becoming a part of life for more and more people, and few of these people either enjoy or are skilled at formal error analysis; MCA gives them a way to estimate the quality of their numerical models.

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

Contents

1 Motivation **4**

2 Overview **5**
2.1 A simple example . . . 5
2.2 Two startling examples . . . 7
2.3 Monte Carlo Arithmetic . . . 9
2.4 Why Monte Carlo Arithmetic works . . . 10
2.5 What Monte Carlo Arithmetic has to offer . . . 11

3 Problematic Issues in Floating-Point Arithmetic **13**
3.1 Floating-point arithmetic . . . 13
3.2 Fundamental problems with floating-point arithmetic . . . 14
3.2.1 Floating-point arithmetic retains no information about significance . . . 15
3.2.2 Significance can be completely lost by catastrophic cancellation . . . 15
3.2.3 Catastrophic cancellation produces major floating-point anomalies . . . 16
3.3 Perspectives on error . . . 17

4 Measuring Significance via Randomness and Statistics **19**
4.1 Formalizing accuracy, error, and number of significant digits . . . 19
4.2 Exact and inexact values . . . 20
4.3 Monitoring significance of computed results . . . 21
4.4 Modeling inexactness with randomness . . . 22
4.5 Determining the number of significant digits via statistics . . . 23

5 Previous Work relating Statistics with Numeric Computation **24**
5.1 The statistical theory of error . . . 24
5.2 Monte Carlo methods . . . 26
5.3 Statistical roundoff analysis . . . 28
5.4 Random rounding . . . 28
5.5 Stochastic computer arithmetic . . . 29

6 Monte Carlo Arithmetic **31**
6.1 Basic objects . . . 31
6.2 Implementing inexact values with randomization . . . 32
6.3 Precision bounding . . . 32
6.4 Random rounding . . . 33
6.5 Modeling roundoff error with randomization . . . 36
6.5.1 Perspectives on roundoff error . . . 36
6.5.2 Rephrasing error perspectives with precision bounding and random rounding 36
6.6 Monte Carlo Arithmetic . . . 37
6.7 Which instance of Monte Carlo Arithmetic is best? . . . 38
6.8 Transforming floating-point computations into Monte Carlo computations . . . 40
6.8.1 Transformation of programs . . . 40
6.8.2 Handling multiple references properly . . . 41
6.8.3 Exploratory statistical roundoff error analysis . . . 42

7 Benefits of Monte Carlo Arithmetic **43**
7.1 Variable precision is supported . . . 43
7.2 Roundoff errors actually do become random . . . 43
7.3 Two-sided error analysis is supported . . . 45
7.4 Addition becomes statistically associative . . . 46
7.5 Other floating-point anomalies are avoided statistically . . . 48
7.6 Open-ended statistical analysis is supported . . . 48
7.7 Limitations of CESTAC and CADNA are avoided . . . 49

8 Example Applications of Monte Carlo Arithmetic **52**
8.1 Basic numerical analysis . . . 52
8.1.1 Cosines . . . 52
8.1.2 Tchebycheff polynomials . . . 52
8.1.3 Horrific examples used in the literature . . . 54
8.2 Gaussian elimination . . . 55
8.2.1 Small examples . . . 56
8.2.2 The Turing-Wilkinson matrix . . . 57
8.2.3 The Hilbert matrix . . . 57
8.2.4 Perspective on error bounds . . . 58
8.3 Differential equations . . . 60
8.3.1 Random rounding can reduce global error in numerical integration . . . 60
8.3.2 Chaotic differential equations . . . 61

9 Conclusion **65**

Acknowledgements **66**

Appendix A: Explanations for the Startling Examples **67**

Appendix B: Answers to Some Frequently Asked Questions **69**

Appendix C: Basic Results in Probability Theory and Statistics **73**
C.1 Results involving density functions . . . 73
C.2 The Tchebycheff and Chernoff Bounds . . . 74
C.3 The Central Limit Theorem . . . 75

Appendix D: Statistical Error Analysis of Gaussian Elimination **77**
D.1 Solving linear systems by Gaussian elimination . . . 77
D.2 Error analysis . . . 77
D.3 Backward error analysis . . . 78
D.4 Forward error analysis . . . 79
D.5 Two-sided error analysis . . . 79
D.6 Statistical error analysis . . . 79

References **80**

1 Motivation

Several forces are currently bringing the discipline of numerical computing to an important crossroads. First, today it is within the reach of anyone to perform significant large-scale computations at tremendous computation speeds and memory capacities. Second, as the appetite for accuracy in models grows, numerical computations are getting increasingly more complex. Third, existing numerical packages have been designed primarily for speed and require judicious application, but this judiciousness can be lacking in the new generation of numerical consumers.

In other words, while numerical computing has never been intended for mass consumption, numerical models are increasingly needed now and market forces are putting them in the reach of everyone. This raises the need for methodologies and tools that allow non-specialists to assess the quality of the results of numeric computations. Inevitably it will become the duty of numerical packages to detect and notify less expert users about problem ill-conditioning, algorithm instability, and so forth.

This paper develops an approach for assessing the accuracy of a numeric computation called *Monte Carlo Arithmetic* (MCA). It is complementary to existing approaches such as running in higher precision, using interval analysis, performing a formal error analysis, etc. These approaches are all useful, but are not always adequate. For example, it is often impractical to rewrite programs to run in higher precision, since changes in precision tend to be painfully error-prone, and programs are also often written in double precision to begin with and that is the highest precision that most systems support. Interval analysis, although given lip service, appears little used in practice; the oft-cited reason for this is that its bounds tend to be overly pessimistic [111]. Formal error analysis is usually focused on basic algorithms, and not on large-scale models; it is significant that Wilkinson concludes his 1971 John von Neumann speech [111] by remarking that he expects error analysis will expand beyond its preoccupation with basic linear algebra problems, while Higham's comprehensive 1996 book on error analysis [54] goes no further. With this in mind, Pierce [85, §7.1] likens the problem of formal error analysis to program verification, and argues that proofs of numeric quality of programs of any reasonable scale are unachievable in practice.

In a sense, MCA gives an numerical error 'idiot light', i.e., a warning signal about numerical quality that should be comprehensible to a very broad audience of numerical program users. Unlike formal error bounds and interval analysis, MCA does not necessarily guarantee numeric quality. Instead, as we will show, it detects loss of accuracy due to catastrophic cancellation in floating-point subtraction, and more generally detects high sensitivity of computed values to minor perturbations in the computational process. Furthermore, in the hands of a user experienced in statistics, MCA can sometimes deliver tighter bounds than either formal (worst-case) error analysis or interval analysis. These features make us believe that MCA can be an important tool in numerical computing.

MCA has other interesting features that need further investigation. It gives an empirical flavor (a Monte Carlo flavor) to numerical analysis; this seems healthy, and even necessary, in a world of increasingly realistic models. It produces numeric results that can be interpreted statistically. Also, it gives an alternative perspective on roundoff error that may be more natural to many users than backward error analysis. While backward error analysis has desirable and important properties, many find it difficult to understand, and it is potentially irrelevant for the user who is interested only in the number of significant digits in the results. Briefly, MCA offers some new ideas at a time when it seems important to consider new perspectives on error in numerical computing.

2 Overview

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 $x \oplus y = fl[\, x + y \,] = (x + y)\,(1 + \delta)$, 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, 25, 62]), 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.

2.1 A simple example

Kahan (e.g., [40, 60, 61]) 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

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

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). When we instead randomize the inputs and outputs of floating-point operations, the results vary each time we execute the program.

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.

Running it 5 times with single precision[1] 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.

Notice that, in Table 2, the standard deviation estimates the absolute error in the computed roots. That is, the roots in each run are within a few standard deviations of the exact solution.

[1] We used single-precision for simplicity in the implementation. Similar results can be produced in any precision.

The standard error (also called "the standard deviation of the average") is the standard deviation divided by the square root of the number of samples, and is a commonly-used estimate of the absolute error in the computed average. As the number of samples n increases, the standard error decreases like $1/\sqrt{n}$, as taking more samples usually yields a more accurate computed average.

The results here were obtained with 'uniform absolute' input and output precision bounding (defined on p.33) giving 'random nearness' rounding (see p.34). The coefficients a, b, c and the constants 2 and 4 were not randomized, as they are exact (precisely representable in floating-point format). We used `gcc` version 2.7.2 with no options (except '`-lm`') on a Sun SPARCstation 20 model 501-2324 running SunOS 5.5. While our implementation of MCA used the IEEE single precision representation for floating-point numbers in order to prove its concept, a higher-precision implementation is easily developed. The source code for MCA and all examples here is available from the author.

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.

Again, we run the program 5 times with MCA, obtaining Table 4. The large standard deviation reflects high error in the computed second root values.

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 that estimate the absolute error (and number of significant digits) in the average. Here the standard error suggests the computed average for `r2` has one significant digit.

2.2 Two startling examples

Consider the following two marvelous examples adapted from [6]. 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	3.000000000000000000000000
20 digits	3.00000000000000000000
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 its enormous standard deviation of x_{30}. With 10 samples, again using uniform input and output precision bounding, 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. Also, the sample distributions obtained for $k = 8, 9, 10$ are most likely not normal (Gaussian).

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). The underlined values failed a skewness/excess test for normality of the underlying distribution.

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

$$\frac{99017602587022271797 0867}{16487411721593453990 9207} = 6.00564868877\ldots$$

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). The underlined values failed a skewness/excess test for normality of the underlying distribution.

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.

These results are further explained in Appendix A.

2.3 Monte Carlo Arithmetic

Monte Carlo Arithmetic can be viewed as making a distinction between exact and inexact values, and implementing this distinction by modeling inexact values as random variables. We create an inexact 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 that can be discrete or continuous, and can depend on x, generating values from the interval $(-\frac{1}{2}, \frac{1}{2})$.

The basic Monte Carlo approach is well-understood [34, 49, 50], so implementing a Monte Carlo arithmetic on these values is straightforward. The real challenge comes from implementing this real arithmetic with floating-point hardware.

Our key idea is that the information lost by using finite-precision computation can *also* be modeled as inexactness, and can also be implemented as random error. 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 real value x and a desired precision t, the **precision bounding** of x to t digits is

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

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.

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 $t_\mathsf{digit_precision}\,(x)$ 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.

Pursuing this idea through implementation gives a Monte Carlo Arithmetic of exact and inexact values, which is implemented as a straightforward extension of floating-point arithmetic. If '$\circledcirc$' is the floating-point approximation to the '$\bullet$' operation, then in Monte Carlo Arithmetic

$$x \;\circledcirc\; y \quad = \quad \mathsf{round}\,(\; t_\mathsf{digit_precision}\,(\;\; t_\mathsf{digit_precision}\,(x) \;\bullet\; t_\mathsf{digit_precision}\,(y) \;\;)\;)$$

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.

Optionally, the input or output precision bounding in this definition can be omitted; this omission loses certain properties. Input precision bounding detects cancellation in any subtraction, as we explain in the next section. Output precision bounding implements random rounding, and with the right choice of randomization can guarantee zero expected rounding bias. If z is an inexact value (random variable), and '$\mathsf{E}[\,z\,]$' denotes the expected value (average value) of z, then $\mathsf{E}[\; t_\mathsf{digit_precision}\,(z)\;] \;=\; \mathsf{E}[\,z\,]$ and $\mathsf{E}[\; \mathsf{round}\,(t_\mathsf{digit_precision}\,(z))\;] \;=\; \mathsf{E}[\,z\,]$ when properly

implemented. The effect of rounding is eliminated, as far as expected value is concerned. So, for a bounded arithmetic expression (possibly involving inexact values), the expected value of its computed Monte Carlo average will be its expected real value. Furthermore, the accuracy of this average is estimated by its standard error. With the right statistical caveats, then, we can prove that Monte Carlo arithmetic actually avoids many floating-point 'anomalies', i.e., important properties of real arithmetic that floating-point arithmetic fails to satisfy.

2.4 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	+3.495683 $\times 10^0$
operand 2	+3.495681 $\times 10^0$
difference	+0.000002 $\times 10^0$
normalized difference	+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	+3.495683 $\times 10^0$	
$x' = t_\mathsf{digit_precision}(x)$		+3.4956832039169594160088 4...
operand y	+3.495681 $\times 10^0$	
$y' = t_\mathsf{digit_precision}(y)$		+3.4956819187079542083546 3...
difference $x' - y'$		+0.0000022852090052076542 1...
rounded difference $\mathsf{round}(x' - y')$	+2.2852090 $\times 10^{-6}$	

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

In MCA, randomization is used in all arithmetic operations. Input precision bounding detects catastrophic cancellation. Output precision bounding 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.

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., [95, 107]), and it is traditional to write

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

This notation is misleading since it does not give hard bounds on the error, but instead gives ‘one standard deviation’ bounds. Better practice is to give confidence intervals, as discussed on p.23.

The ratio of the standard deviation of a result of a program to its average also measures its sensitivity to roundoff perturbations (in its input and numerical operations), and can be used to measure instability and ill-conditioning. By **stability** of a program here we mean the insensitivity of the computed values to perturbation of the particular input data given (estimated by their standard deviations as compared with the machine precision). This is much like forward stability of the program [42, p.47], and less like backward stability. In this sense instability of the algorithm is essentially the same as **ill-conditioning** of the problem, i.e., high sensitivity of some computed values to minor changes in the input values. This instability was estimated in each the examples above.

2.5 What Monte Carlo Arithmetic has to offer

Summarizing all the arguments we will make about MCA:

- MCA probabilistically detects 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, they are a warning signal that strongly recommends further analysis.

- MCA gives an alternative method for analyzing roundoff error in numerical computations. All useful methods (e.g., running in higher precision, computing condition numbers, using interval arithmetic, performing a formal roundoff analysis) have strengths and weaknesses. MCA does not replace any of these methods, but rather gives a complementary method. It can be used in any precision arithmetic, and in any computation.

- MCA encourages the introduction of statistical inference into numerical computing, and the use of confidence intervals as certifications of numerical quality. If the roundoff errors can be shown to follow normal distributions, for example, tight bounds on the exact results can be obtained with a few samples. These bounds can be tighter than worst-case roundoff bounds, and tighter than bounds obtained with interval arithmetic.

- MCA lets one change the precision of floating-point computation dynamically. One can thus experiment with the effect of precision change on a particular program, and evaluate both its stability and its actual precision requirements for a given input.

- MCA gets higher-accuracy numerical results in some cases, because individual roundoff errors are *forced* to be random and uncorrelated.

- MCA formally circumvents 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 maintains information about the significance (number of significant digits) of conventional floating-point values, without resorting to unnormalized arithmetic or other significance arithmetic schemes. Focus on loss of significance in numerical computations is a perspective on roundoff error that most users immediately understand and are concerned about. Backward error analysis, on the other hand, is hard for users to understand or appreciate, and gets unwieldy in large computations.

- MCA makes numerical computing more empirical, and makes detection of numerical error easier for the person on the street. Giving numerical computing an experimental flavor makes it more accessible.

- MCA implements a distinction between real values that are **exact** (i.e., completely known with certainty, and accurately represented by a floating-point value), and those that are **inexact** (values that are either not completely known with certainty, or whose floating-point representation is inaccurate). This is an important distinction, since for example most quantities in scientific computation are inexact: constants like Avogadro's number 6.0225×10^{23} and Planck's constant 1.0545×10^{-19} are known only to a few digits [68, §4.2.2.B]. The inexactness can be due to ignorance, uncertainty, estimation, measurement or computational error.

- MCA generalizes and improves upon previous approaches for integrating randomization into computer arithmetic described in Section 5, including the random rounding of Forsythe and others, the significance arithmetic of Ashenhurst and Metropolis, the stochastic arithmetic of Vignes, and even interesting curiosities like the noisy mode of the Stretch supercomputer.

3 Problematic Issues in Floating-Point Arithmetic

We review basic issues in floating-point arithmetic, and highlight some of the most important problems. Goldberg's tutorial [43] and Higham's encyclopedia [54, chs.1–5] are great references; here we develop only what is essential.

3.1 Floating-point arithmetic

Any floating-point arithmetic system defines a certain set $\mathcal{F}$ of possible floating-point numbers. For example, in nonextended IEEE 754 floating-point arithmetic with denormalized numbers, the minimum and maximum positive floating-point numbers are $f_{\min} = 2^{E_{\min}} b_{\min}$ and $f_{\max} = 2^{E_{\max}} b_{\max}$, with values derived from [1, §3.2] in the following table:

floating-point system	$E_{\min}$	$E_{\max}$	$b_{\min}$	$b_{\max}$
IEEE *single precision*	-126	$+127$	2^{-23}	$2 - 2^{-23}$
IEEE *double precision*	-1022	$+1023$	2^{-52}	$2 - 2^{-52}$

Relative to a given $\mathcal{F}$, we can define a subset of the real numbers $\Re$

$$\Re_{\mathcal{F}} \quad = \quad \{0\} \quad \cup \quad \{\ x \in \Re \mid r_{\min} < |x| < r_{\max}\ \}.$$

In IEEE single precision floating-point and the default rounding (round to nearest) [1, §4.1], the first real value not in $\Re_{\mathcal{F}}$ is $r_{\max} = 2^{127} \times (2 - 2^{-24}) \simeq 3.4 \times 10^{38}$ (which rounds to $+\infty$ with an overflow). Similarly [1] permits denormalized values above $r_{\min} = 2^{-126} \times 2^{-24} \simeq 1.4 \times 10^{-44}$ (which can round to 0 with an underflow). Real numbers in $\Re_{\mathcal{F}}$ can be *approximated* by the floating-point numbers in $\mathcal{F}$, and numbers outside this set are not representable within $\mathcal{F}$.

With rounded arithmetic, one can define the floating-point approximation

$$fl[\ x\ ] \quad = \quad \begin{array}{l}\text{the value } \widehat{x} \in \mathcal{F} \text{ that is nearest to } x, \text{ where} \\ \text{ties are resolved in some well-defined way}\end{array}$$

provided that $x \in \Re_{\mathcal{F}}$. Ignoring denormalized values, the accuracy of the rounded approximation $fl[\ x\ ]$ to x can be summarized by the relative error bound

$$fl[\ x\ ] \quad = \quad x\ (1 + \delta), \qquad \text{where } |\delta| \ \leq\ \mathbf{u}$$

and $\mathbf{u}$ is the **unit roundoff** or **machine epsilon** of $\mathcal{F}$. On a machine with p-digit, base-β rounded floating-point arithmetic, $\mathbf{u}$ is $\frac{1}{2}\beta$ times the **machine precision** β^{-p}, i.e.

$$\mathbf{u} \quad = \quad \tfrac{1}{2}\ \beta^{1\,-\,p}\ .$$

For example, the IEEE 754 floating-point standard uses rounded arithmetic with $\beta = 2$ and either $p = 24$ for single precision or $p = 53$ for double precision; so $\mathbf{u} = 2^{-24} \approx 5.96 \times 10^{-8}$ in single precision, and $\mathbf{u} = 2^{-53} \approx 1.11 \times 10^{-16}$ in double precision.

With rounded arithmetic and $x \in \Re_{\mathcal{F}}$, we can therefore write $fl[\ x\ ] = \mathsf{round}\,(x)$, where $\mathsf{round}\,(x)$ is a suitable function. This function is usually tricky to define. The essence of the definition for rounding of a positive real value x away from zero to p base-β digits can be written as

$$\mathsf{round}\,(x) \quad = \quad \beta^{e\,-\,p}\ \left\lfloor\ \beta^{p-e}\ x\ +\ \tfrac{1}{2}\ \right\rfloor \qquad \text{whenever} \quad \beta^{e-1}\ \leq\ x\ <\ \beta^{e}\ (1\ -\ \tfrac{1}{2}\beta^{-p}).$$

This is not the complete definition: it fails to cover the situations where: rounding carry-out occurs (i.e., $\beta^{e}\ (1\ -\ \frac{1}{2}\beta^{-p}) \leq x < \beta^{e}$), or $x = 0$, or $x < 0$, or $x \notin \Re_{\mathcal{F}}$. Normalization is also implicit in this definition. However it does capture the essence.

The traditional notion of rounding x to $\widehat{x}$ follows the convention of rounding away from zero defined above, and thus resolves ties simply by always following this definition. In the 1960s it became appreciated that the resolution of ties among nearest floating-point approximations was significant, because it arises frequently in floating-point addition. 'Unbiased rounding' was developed to avoid this problem [92, §6.4]. Round to Nearest, the default IEEE method, resolves ties by choosing the nearest value $\widehat{x}$ whose least significant bit is zero, except that the magnitude of any value out of range rounds to ∞.

Section 4 of the IEEE standard actually defines four user-selectable rounding methods: Round to Nearest (the default), Round to Zero, Round to $+\infty$, and Round to $-\infty$. Programmers are allowed to select a specific rounding method at any point in a computation, and this method remains in force until the next selection; thus the rounding operation $\mathsf{round}\,(x)$ and the meaning of $fl[\ x\ ]$ can even vary during the execution of a single program on a given machine.

The p-digit floating-point arithmetic operations $\oplus$, $\ominus$, $\otimes$, $\oslash$ can then be defined for floating-point values $x, y \in \mathcal{F}$ by

$$\begin{aligned} x \oplus y &= \mathsf{round}\,(x + y) \\ x \ominus y &= \mathsf{round}\,(x - y) \\ x \otimes y &= \mathsf{round}\,(x \times y) \\ x \oslash y &= \mathsf{round}\,(x \ / \ y) \end{aligned}$$

provided that no exponent overflow occurs. As a result, if '$\circledcirc$' is the floating-point approximation to the '$\bullet$' operation, then there exists a real value δ so that $|\delta| \leq \mathbf{u}$ and

$$x \circledcirc y \quad = \quad (x \bullet y) \times (1 + \delta)$$

whenever x and y are floating-point values in $\mathcal{F}$ and $(x \bullet y) \in \Re_{\mathcal{F}}$.

Below we will use these operators instead of '$fl[\ldots]$' because in the literature the latter is used ambiguously: as a function that maps the real value z to the floating-point value $\widehat{z}$; as an evaluator that maps expressions to floating-point values, so $fl[\ x + y + z\ ]$ denotes the value of the floating-point expression $(x \oplus y) \oplus z$ where x, y, and z are floating-point values in $\mathcal{F}$; and as an evaluator like that just described, but where x, y, and z are *real values in* $\Re_{\mathcal{F}}$. These alternatives are inequivalent. Specifically, the statement above that there exists a real value δ such that $|\delta| \leq \mathbf{u}$ and $x \oplus y = (x + y) \times (1 + \delta)$ is false if x and y are permitted to be arbitrary reals in $\Re_{\mathcal{F}}$. In other words, a statement such as

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

implicitly requires x and y to be floating-point values.

3.2 Fundamental problems with floating-point arithmetic

Despite fifty years of development, the great contributions of Wilkinson, Knuth, Kahan and others, and the great advance made by the IEEE floating-point standards [1, 25], floating-point arithmetic is still an arcane and labyrinthine subject. In general, finding good error bounds for numerical algorithms is difficult, and determining whether a numerical computation is 'stable' is an art. Worse, users of most floating-point packages really do not know the quality of the results they are getting, and their options for getting a precise understanding of that quality are limited.

The quality of a numeric result is usually measured by its **accuracy**, which we equate with **significance**: its number of significant digits. Here we stress three fundamental problems with floating-point arithmetic, which all have to do with accuracy.

3.2.1 Floating-point arithmetic retains no information about significance

The normalization operation of floating-point arithmetic destroys the type distinction between **accurate values** (values that have some significant digits) and **completely inaccurate values** (values that have no significant digits). The user is given responsibility for interpreting the significance of all floating-point results.

Since the very early work by Bauer and Samelson in 1953 [9], various authors have stressed that unnormalized arithmetic can improve on normalized arithmetic in some cases for retaining information about significance. See for example Knuth [68, §4.2.2.B], the fixed-point arithmetic analyses of Wilkinson in [109], and the arguments for unnormalized arithmetic of Ashenhurst and Metropolis [4, 5], who show that it retains important information about the number of significant digits in the computed results. The basic idea is that unnormalized numbers can, by convention, represent their significant digits (although in practice this approach tends to over- or understate significance somewhat [68]).

In later work, Metropolis [75] proposes maintaining a distinction between two kinds of numbers: precise and imprecise. Precise values are represented with complete accuracy, and their preciseness is monitored by Metropolis' significant digit arithmetic (SDA). Imprecise values are implemented essentially with unnormalized floating-point arithmetic.

3.2.2 Significance can be completely lost by catastrophic cancellation

Because of catastrophic cancellation and the normalization following it, floating-point arithmetic can lose *all* significant digits of a computed value, yielding a result that is completely inaccurate. This is almost certainly its most serious problem.

However, because floating-point arithmetic retains no information about the number of significant digits in the values it produces, it is not so obvious that it should assume the blame for this problem. In fact, the case is often made that floating-point arithmetic is blameless!

First, many texts note that when catastrophic cancellation arises, the result of the floating-point operation is **exact**, under the nontrivial assumption that the subtraction operands are also exact (e.g., Higham [54, p.11]). That is, whenever the operands have zero error, the result also has zero error. Under this assumption, then, floating-point arithmetic is not to blame for the loss of significance. It has produced exact results, with zero error.

Second, some texts point out that "cancellation is not *always* a bad thing" [54, p.12] (cf. also [43, Thm.11]). If the operands are exact, then the result is also. If the cancelled value is insignificant relative to the rest of the expression it appears it, then the cancellation affects nothing. When nothing is known about the exactness or inexactness of the operands, no blame for loss of it can be assigned.

With this outlook, catastrophic cancellation is often viewed as a consequence of poor choice in algorithms, and it is the programmer who usually gets the blame. An excellent example is offered by Goldberg [43, pp.10–11], who explains that computing $x^2 - y^2$ with the expression

$$(x \otimes x) \ominus (y \otimes y)$$

can yield terrible results because of catastrophic cancellation, while the 'equivalent' expression

$$(x \ominus y) \otimes (x \oplus y)$$

gives better results when $x^2 \approx y^2$, assuming x and y are exact, since then $(x \ominus y)$ and $(x \oplus y)$ are both exact (or almost exact). Programmers are supposed to know this. They also should know that when x and y are *inexact* and $|x| \approx |y|$ to the machine precision, neither expression yields many significant digits, and $x^2 - y^2$ cannot be computed accurately.

Unfortunately, many people are unaware of the problems of cancellation. For example, the formula for computing the sample standard deviation from $\sum x_i^2$ and $\sum x_i$

$$\hat{\sigma} \quad = \quad \sqrt{\frac{1}{n-1}\left(\left(\sum_{i=1}^{n} x_i^2\right) \quad - \quad \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right)}$$

is in widespread use. Yet it routinely produces results of *terrible* quality [18, 19]. In fact, it can produce such complete cancellation that it raises floating-point exceptions, because the argument of the square root goes negative. (This actually happened in the initial test programs for this work!) It seems no one-pass algorithm for computing standard deviations avoids cancellation, although with some ingenuity the problem of exceptions can be avoided [69].

Even computing ordinary *sums* in floating-point arithmetic is problematic because of the possibility of catastrophic cancellation. See [53] or [54, ch.4] for an illuminating survey of the many tricks and techniques for dealing with summation.

3.2.3 Catastrophic cancellation produces major floating-point anomalies

Floating-point arithmetic is widely (and correctly) viewed as a rat's nest of **anomalies**, i.e., important properties of real arithmetic it fails to satisfy, and which render it difficult to formalize and to reason about. Knuth [68, §4.2.2] has made a careful enumeration of axioms that *are* satisfied by floating-point operations and shows that they make it possible to prove basic arithmetic properties. As an important application, he shows how to build extended-precision arithmetic using these operations. The work of Kulisch and Miranker [71, 72] has made the axiomatic approach more popular, but it has hardly been received with universal acclaim, and formal methods still have far to go in influencing implementations of computer arithmetic (e.g., [87]).

A major anomaly is that *floating-point addition is not associative*. Knuth gives the following example for eight-digit decimal arithmetic [68, p.196]:

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

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.

It is important to realize that modeling the significance (number of significant digits) of the input operands can evade problems of catastrophic cancellation. Knuth [68, §4.2.2.B] shows that the unnormalized arithmetic discussed above obeys an approximate associative law.

Catastrophic cancellation is at the root of another major anomaly in floating-point arithmetic. Although floating-point values x, y with sum in $\Re_{\mathcal{F}} - \{0\}$ always satisfy the relative error bound

$$\left|\frac{(x+y) \quad - \quad (x \oplus y)}{(x+y)}\right| \quad \leq \quad \mathbf{u}$$

Wilkinson [109, p.17] points out that there is no useful bound on the relative error

$$\left|\frac{(x+y+z) \quad - \quad ((x \oplus y) \oplus z)}{(x+y+z)}\right|$$

when x, y, and z are floating-point values with sum in $\Re_{\mathcal{F}} - \{0\}$. No bound can be obtained in the event of catastrophic cancellation. Thus, it is generally impossible to obtain forward relative error bounds on sums, and this encourages the use of backward error analysis.

3.3 Perspectives on error

Formal roundoff analyses always rest on assumptions about what the analysis is to accomplish. These can go far beyond basic framing assumptions (such as the input domain), and usually rest on some established perspective about error in numerical computation.

Let x and y be nonzero p-digit floating-point values with floating-point exponents e_x and e_y. There are at least three different, useful perspectives about the error in $x \circledcirc y$:

1. **Backward Error**
   Error arises in the *representation of numbers*, i.e., in the *operands* in numerical computation. Backward error is usually modeled by perturbation of these operands:

   $$x \circledcirc y \quad = \quad ((x + \varepsilon_x) \bullet (y + \varepsilon_y))$$

   where $|\varepsilon_x| \leq \frac{1}{2}\beta^{e_x - p}$ and $|\varepsilon_y| \leq \frac{1}{2}\beta^{e_y - p}$.

2. **Forward Error**
   Error arises in *operations on numbers*, i.e., in the *operators* in numerical computation. Forward error is usually modeled by perturbation of the results of these operations:

   $$x \circledcirc y \quad = \quad (x \bullet y) + \varepsilon_{xy}$$

   where $|\varepsilon_{xy}| \leq \frac{1}{2}\beta^{e_{xy} - p}$ and e_{xy} is the floating-point exponent of $(x \bullet y)$.

3. **Loss of Significance**
   Error occurs in both the operands and operators in numerical computation, and measures the *erosion of significant digits* in computed values. A p-digit floating-point value x is specified to some number s of significant digits (which need not be integral [92, §3.1]), and this specification can denote *any* number $x + \varepsilon$ where $|\varepsilon| \leq \frac{1}{2}\beta^{e-s}$, if e is the floating-point exponent of x. We permit $s < p$, so the number of significant digits of a value is not necessarily the precision of the machine.

   The significance of a computed value is reflected by a (*forward and backward*) expression

   $$x \circledcirc y \quad = \quad ((x + \varepsilon_x) \bullet (y + \varepsilon_y)) \quad + \quad \varepsilon_{xy}$$

   where $|\varepsilon_x| \leq \frac{1}{2}\beta^{e_x - s_x}$, $|\varepsilon_y| \leq \frac{1}{2}\beta^{e_y - s_y}$, $|\varepsilon_{xy}| \leq \frac{1}{2}\beta^{e_{xy} - s_{xy}}$. Notice that the difference of significance levels

   $$\min(\, s_x,\, s_y \,) \quad - \quad s_{xy}$$

   measures *ill-conditioning* of the problem, i.e., sensitivity of the output to nonsignificant changes in the input.

These perspectives give fundamentally different interpretations of 'quality of a solution' and can differ from the intuitive sense of solution quality entertained by numerical program users.

A nice example illustrating this point is the ill-conditioned problem adapted from [42, p.50] of computing a root of $(x-1)^4 = 0$ (i.e., solving $x^4 - 4x^3 + 6x^2 - 4x + 1 = 0$), When working with eight-digit precision, an algorithm that produces the solution $x = 1.01$ has terrible forward error (10^6 times the machine precision). Also, this solution has only $s = 2$ significant digits, a huge loss of significance if we assume the coefficient 1 has eight significant digits. However, beauty lies in the perspective of the beholder. Because 1.01 is the exact solution of the slightly perturbed problem $(x-1)^4 = 10^{-8}$ (i.e., $x^4 - 4x^3 + 6x^2 - 4x + 0.99999999 = 0$), in the backward error perspective the algorithm has produced a very good solution!

Certainly Wilkinson is right in stating for backward error analysis that "it seems highly improbable that any method of solving any problem which is restricted to the use of t-digit arithmetic will, in general, do better than give the solution to a t-digit approximation to the original problem" [109, p.32]. If we accept the limitations of floating-point arithmetic, and therefore ignore the notion of significance, backward error bounds are essentially the best one can hope for. However, interpreting backward error bounds correctly requires education. While we are not arguing against education, we are skeptical that many numerical package users truly understand backward error results. Even Wilkinson felt they do not understand backward error results:

> It is, however, interesting that in two of his last five papers Jim [Wilkinson] returned to a discussion of his theories of the famous 'backward error analysis', exhibiting touches of sorrow and irritation that so many colleagues, particularly those with a training in classical mathematics, had failed to understand it, or found it difficult to adopt, or had even exhibited a sort of moral disapproval in its use. —— L. Fox [29, Epilogue p.333]

We suspect that many users see loss of significance as the most natural error perspective, but this is a sociological question, and will not be resolved here.

Let us conclude with an important point: *the dominance of backward error analysis in the modern numerical analysis literature is a consequence of the dominance of floating-point arithmetic in modern computers.* Backward error bounds are natural algebraic bounds derivable for floating-point summations. Forward error bounds on floating-point expressions involving any addition can be quite pessimistic. The significance of numeric results is not tracked by floating-point arithmetic, since significance information is lost in normalization.

4 Measuring Significance via Randomness and Statistics

In this section we develop basic notions of numeric accuracy, and formally define what we mean by error, number of significant digits, and inexactness. We also show how to use randomness to model inexactness in floating-point computation, and how to use statistics (specifically, standard deviations) to measure the number of significant digits in an inexact value.

4.1 Formalizing accuracy, error, and number of significant digits

Sterbenz [92, §3.1] discusses why it can be difficult to define what is meant by "the number of significant digits of a value", i.e., "the number of digits to which a value is accurate". First, the number of significant digits of a value is usually thought of as being an integer, but this convention gives a crude measure of accuracy and usually appraises values as less accurate than they actually are. Second, intuitive notions of the number of significant digits can give counterintuitive results.

For example, suppose $x = 3.14159\ldots$. If $\widehat{x} = 3.1415$, many people will argue that $\widehat{x}$ has *five* significant digits. However, if $\widehat{x} = 3.1416$, some will say that $\widehat{x}$ has only *four* significant digits, even though it is closer to x than 3.1415.

More strikingly, suppose $x = 1.0000$. If $\widehat{x} = 1.0007$, it is intuitively clear to most people that $\widehat{x}$ has *four* significant digits. However, if $\widehat{x} = 0.9997$, some people will argue that $\widehat{x}$ has *zero* significant digits, even though it is much closer to x than 1.0007. Finally, if $\widehat{x} = 1.0001$, some will say that $\widehat{x}$ has *four* significant digits, while others state that $\widehat{x}$ has *five* significant digits.

Despite these quirks of intuition, people's definitions usually agree closely. *We believe that most people measure the quality of a numeric result by its "number of significant digits".* A formal definition for this term is important, then, and numerical packages should be able to present results in terms of significant digits. Let us now compare two different definitions — via relative error and a human algorithm of our own devising — and prove that they are close.

The number of significant digits is often taken as a simple measure of relative error [92, p.72]. Given numeric values x and $\widehat{x}$ such that $x \neq 0$,[2] we say $\widehat{x}$ approximates x with

$$\textbf{relative error} \quad \delta \;=\; (\widehat{x} - x)/x \qquad \text{so} \quad \widehat{x} \;=\; x\,(1 + \delta\,).$$

It is then common to define the **number of (relative) significant digits** of $\widehat{x}$ as

$$-\log_\beta |\delta|, \qquad \text{or} \qquad \lfloor -\log_\beta |\delta| \rfloor \quad \text{if an integer is desired.}$$

This definition is pretty clearly not what is actually used by human beings, since they do not compute relative errors. However, it gives results close to those given by humans, as we now show.

A human algorithm for the number of significant digits (ours, actually) is to find the leftmost digit position s in which x and $\widehat{x}$ differ by less than one-half unit (i.e., less than a rounding error). We call this the $\frac{1}{2}$**ulsp algorithm**, where '**ulsp**' abbreviates 'units in the last significant place', a measure of *absolute* error. The idea is that if $x = 29$, then $\widehat{x} = 37$ has zero significant digits, but $\widehat{x} = 31$ has one significant digit (since 29 and 31 differ by less than one-half unit in the first place). We formalize this algorithm as follows. In base-β arithmetic we say $\widehat{x}$ approximates x with

$$\begin{aligned} \textbf{absolute error} \quad \varepsilon &= (\widehat{x} - x) && \text{so} \quad \widehat{x} = x + \varepsilon \\ \textbf{scaled absolute error} \quad \eta &= (\widehat{x} - x)/\beta^e && \text{so} \quad \widehat{x} = x + \beta^e\,\eta \end{aligned}$$

where e is **the base-β order of magnitude of x (βoom of x)**

$$e \;=\; \beta\text{oom}(x) \;=\; \lfloor \log_\beta |x| \rfloor \;+\; 1.$$

[2] When $x = \widehat{x} = 0$, we define $\delta = 0$. When $x = 0$ and $\widehat{x} \neq 0$, we define $\delta = \infty$.

This definition merely reexpresses the bounds[3] $\beta^{e-1} \leq |x| < \beta^e$ when e is an integer.

Thus x and $\widehat{x}$ agree to s digits, and differ by less than one-half unit in the s-th position, iff

$$|\eta| \quad \leq \quad \tfrac{1}{2}\,\beta^{-s}$$

or equivalently

$$s \quad \leq \quad -\log_\beta(\,2\,|\eta|\,).$$

Therefore we define the **number of (absolute) significant digits** of $\widehat{x}$ as

$$-\log_\beta(\,2\,|\eta|\,), \qquad \text{or} \qquad \lfloor -\log_\beta(\,2\,|\eta|\,)\,\rfloor \quad \text{if an integer is desired.}$$

This definition differs from the relative definition. For example, consider $x = 12.345$ and $\widehat{x} = 12.3$. Then $\widehat{x}$ has *two* relative significant digits, but has *three* absolute significant digits:

$\lvert\delta\rvert$	=	0.36×10^{-2}	$\lvert\eta\rvert$	=	0.45×10^{-3}	$(e = 2)$
$-\log_{10}\lvert\delta\rvert$	=	2.44	$-\log_{10}(\,2\,\lvert\eta\rvert\,)$	=	3.35	
$\lfloor -\log_{10}\lvert\delta\rvert\,\rfloor$	=	2	$\lfloor -\log_{10}(\,2\,\lvert\eta\rvert\,)\,\rfloor$	=	3.	

Nevertheless, the two definitions generally give values that are quite close. If $|x| = \frac{1}{2}\,\beta^e$, then $2\,|\eta| = |\delta|$ and the two definitions agree. In fact, because $\beta^{e-1} \leq |x| < \beta^e$ the absolute number of significant digits and the relative number of significant digits differ by less than 1:

$$\log_\beta(\,2\,|\eta|\,) \;-\; \log_\beta|\delta| \quad = \quad \log_\beta\left(\,\frac{2\,|x|}{\beta^e}\,\right) \quad < \quad \log_\beta(\,2\,) \quad \leq \quad 1,$$

$$\log_\beta|\delta| \;-\; \log_\beta(\,2\,|\eta|\,) \quad = \quad \log_\beta\left(\,\frac{\beta^e}{2\,|x|}\,\right) \quad \leq \quad \log_\beta\left(\frac{\beta}{2}\right) \quad < \quad 1.$$

In summary, there are many definitions for the number of significant digits. These definitions differ by small amounts, so they are all interchangeable, provided we permit slight divergence from human intuition. The $\frac{1}{2}$ulsp algorithm seems to do a better job of modeling what people intuitively mean by the "number of significant digits" of $\widehat{x}$ than relative error, and we will emphasize its use below.

4.2 Exact and inexact values

Henceforth, we draw a distinction between real values that are **exact** (i.e., completely known with certainty, and accurately represented by a floating-point value), and those that are **inexact** (values that are either not completely known with certainty, or whose floating-point representation contains some inaccuracy). Furthermore, *inexactness is 'contagious'*: except for degenerate expressions like $(0 \times z)$, any arithmetic operation involving an inexact operand yields an inexact value.

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 [68, §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.

The distinction between exact and inexact values has been important in the implementation of significance arithmetics. In Analyzed Binary Computing [75], for example, Metropolis maintained the distinction between *precise* and *imprecise* values, and represented precise values with

[3] When $x = 0$, we define $e = \beta\text{oom}(x) = 0$.
Note: $e = \beta\text{oom}(x)$ can differ from the effective floating-point exponent E of x, depending on the normalization convention. If $x = f \times \beta^E$ where $1/\beta \leq f < 1$, so $f = 0\,.\,f_1\,f_2\,f_3\,\cdots\,f_p$ where $f_1 \neq 0$, then $E = \beta\text{oom}(x)$. However, when $x = f \times \beta^E$ where $1 \leq f < \beta$, so $f = f_1\,.\,f_2\,f_3\,\cdots\,f_p$ where $f_1 \neq 0$, then $E = \beta\text{oom}(x) - 1$. The IEEE 754 standard follows the second convention.

conventional floating-point format, and imprecise values with unnormalized format. The Stretch supercomputer [15] also maintained this distinction. The design was inspired in part by the work of Ashenhurst and Metropolis, and was implemented in the IBM 7030 delivered to Los Alamos National Labs in April 1961. In ways the Stretch floating-point system was more flexible than the IEEE standards: it had both normalized and unnormalized arithmetic, both rounding and chopping, support for extended precision, and representation of singular values ($\pm\infty$, infinitesimals, and order-of-magnitude zero). An **order-of-magnitude zero** [15, p.108–111] is a value $0 \times \beta^e$, which we say **has base-β order of magnitude** e (has βoom e), and represent it by a floating-point value with fraction 0 and exponent e. Thus Stretch distinguished between exact and inexact zero. This distinction is useful in absolute error analysis, since absolute error is defined for an order-of-magnitude zero but not for exact zero.

By contrast, the convention common in numerical analysis is to view the input or output values to a floating-point computation as 'exact', and to then characterize a way in which the floating-point computation behaves like exact computation. Backward error analysis assumes the output is exact and seeks perturbed inputs corresponding to that output, while forward error analysis does the reverse. Although this approach is reasonable and entirely consistent with mathematical analysis, it is inconsistent with the inexactness inherent in physical reality.

Why is this distinction missing in all major computer programming languages? There seem to be several reasons, but a primary reason is lack of hardware support for monitoring inexactness.

4.3 Monitoring significance of computed results

Computer scientists have long sought to include some automatic monitoring of accuracy in numerical computations [92, ch.7]. Interval arithmetic and significance arithmetic, for example, were considered by Turing as early as 1946 [111, p.566]. Interval computation [77, 78] has never really caught on because of the pessimism of its error bounds. (See the commentary by Wilkinson in Appendix B, for example.) Significance arithmetic has also failed to gain a large following.

Significance arithmetic was popularized by Ashenhurst and Metropolis from the late 1950s through the 1970s. They wrote a number of papers arguing the need for numerical systems to track the accuracy (significant digits) of their results [4, 5, 11, 75]. Significance arithmetic faced several difficult problems, which it did not really overcome. First, there is the basic problem just discussed of defining what is meant by the "number of significant digits" [92, §3.1]. Second, there are serious problems of implementation. Implementations based on unnormalized arithmetic make error analysis tricky and can overstate or understate accuracy [68, §4.2.2B]. Implementing numbers of significant digits as integers gives results that are crude. Finally, implementing numbers of significant digits as real values (such as negative base-β logarithms of relative error) is expensive.

Project Stretch offered an ingenious alternative implementation called **noisy mode**:

> 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. —— [15, p.25]

> After an extensive search, the most effective technique [for monitoring significance loss] turned out to be both elegant and remarkably simple. ... During addition the n-digit operand that is not preshifted is extended with n zeros, so as to provide the extra positions to which the preshifted operand can be added. Any operand or result that is shifted left to be normalized requires a corresponding number of zeros to be shifted in at the right. Both sets of zeros tend to produce numbers smaller in absolute value than they would have been if more digits had been carried. 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. —— [15, p.102]

The designers of Stretch had no illusions about its flaws, but realized that noisy mode gave a good way to monitor loss of significance:

> 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. —— [15, p.102]

Although noisy mode is very clever, it is not difficult to come up with examples that subvert its intent. Moreover, it can require some care in programming; for example, in noisy mode apparently the difference of two successive values in any iteration will never be zero. Even more surprisingly, the result of computing $x - x$ will apparently never be zero. Kahan [63, p.19] declares noisy mode to be the "worst way" to randomly perturb rounding errors. This strong statement ignores the mode's modest intent in Stretch, but Kahan's basic point is right. There is a better way to represent inexactness and track numeric significance.

4.4 Modeling inexactness with randomness

Finally, we model inexact values as random variables — the fundamental idea of statistical probability theory [33]. An inexact value $\widetilde{x}$, which is known to have value x to s significant digits (in base-β arithmetic), can be represented as a random variable defined by a **random $\frac{1}{2}$ulsp error**

$$\widetilde{x} \quad = \quad x \quad + \quad \beta^{e-s}\ \xi$$

where e is the base-β order of magnitude of $\widetilde{x}$, and ξ is a random variable with some distribution on the interval $(-\frac{1}{2}, \frac{1}{2})$, so the error is less than one half unit in the s-th place. Individual samples of $\widetilde{x}$ then give estimates for x, with an error determined by ξ.

This definition permits modeling of inexact zeroes, since their order of magnitude e can be specified even when $x = 0$. The definition also permits modeling of 'totally inexact' values — values with no significant digits — when we permit negative values to be chosen for s. Vignes calls such values **informatical zeroes** [104].

It is often easier to analyze relative errors than absolute errors. Provided that x is nonzero, the equation above is equivalent to an equation using a **random relative error**

$$\widetilde{x} \quad = \quad x\ (\ 1\ +\ \beta^{-s'}\ \xi\ )$$

when we take $s' = s\ -\ e\ +\ \log_\beta |x|$. Since $e = \beta\mathrm{oom}(x) = \lfloor \log_\beta |x| \rfloor\ +\ 1$ when $x \neq 0$, we also find $s' \approx s$, and moreover s' and s differ by less than 1. So varying the value for s (resp. s') by a small amount still produces an expression that can be said to agree with x to s (resp. s') digits. In fact, if we simply take $s' = s$ then we get a relative error that is close to the $\frac{1}{2}$ulsp error. Thus for nonzero inexact values, these two definitions are in a sense interchangeable, provided we permit slight divergence from human intuition about what is meant by 'the number of significant digits'.

With the relative error definition, if x is zero then either $\widetilde{x}$ is also zero, or the error is unbounded. *Given any inexact zero, its scaled absolute error is bounded, but its relative error is unbounded.* It is possible to ignore this point in problems where zeroes never arise, but otherwise relative errors cannot be used to get error bounds. Instead, absolute ($\frac{1}{2}$ulsp) errors must be used to get bounds.

4.5 Determining the number of significant digits via statistics

Adapting standard statistical notation, we write $\mathsf{E}[\,\xi\,]$ to denote the **expected value** (mean) of ξ, $\mathsf{V}[\,\xi\,]$ to denote its **variance**, and $\mathsf{S}[\,\xi\,]$ to denote its **standard deviation**. Then [30, §15]:

$$\begin{aligned}
\mathsf{E}[\,g(\xi)\,] &= \int g(t)\,dF(t) \qquad \text{(where } F \text{ is the distribution underlying } \xi\text{)} \\
\mathsf{V}[\,\xi\,] &= \mathsf{E}\Big[\,(\xi - \mathsf{E}[\,\xi\,])^2\,\Big] \;=\; \mathsf{E}\Big[\,\xi^2\,\Big] \;-\; \mathsf{E}[\,\xi\,]^2 \\
\mathsf{S}[\,\xi\,] &= \sqrt{\mathsf{V}[\,\xi\,]}\,.
\end{aligned}$$

We can then prove the following important result:

Theorem 1 *If* $\mathsf{E}[\,\xi\,] = 0$ *and* $\mathsf{S}[\,\xi\,] = 1$*, then the difference in the orders of magnitude of the mean* μ *and standard deviation* σ *of* $\widetilde{x}$ *measures the number of significant digits of* $\widetilde{x}$ *(if* $\mu \neq 0$*,* $\sigma \neq 0$*).*

Proof
When $\mu = x \neq 0$, the $\frac{1}{2}$ulsp error definition $\widetilde{x} = x + \beta^{e-s}\,\xi$ gives

$$\begin{array}{rcccccl}
\mu &=& \mathsf{E}[\,\widetilde{x}\,] &=& x &+& \beta^{e-s}\,\mathsf{E}[\,\xi\,] \;=\; x \\
\sigma &=& \mathsf{S}[\,\widetilde{x}\,] &=& & & \beta^{e-s}\,\mathsf{S}[\,\xi\,] \;=\; \beta^{e-s}.
\end{array}$$

where $e = \lfloor \log_\beta |x| \rfloor + 1$ (so $\mu = x \neq 0$). When $\sigma \neq 0$ the number of absolute significant digits s can then be obtained directly from the mean and standard deviation of $\widetilde{x}$:

$$s \;=\; e - (e-s) \;=\; \lfloor \log_\beta(\,\mathsf{E}[\,\widetilde{x}\,]\,) \rfloor + 1 \;-\; \log_\beta(\,\mathsf{S}[\,\widetilde{x}\,]\,) \;=\; \lfloor \log_\beta(\mu) \rfloor + 1 \;-\; \log_\beta(\sigma).$$

Similarly, using the relative error definition $\widetilde{x} = x\,(1 + \beta^{-s'}\,\xi)$, we find

$$\begin{array}{rcccl}
\mathsf{E}[\,\widetilde{x}\,] &=& x\,(\,1 \;+\; \beta^{-s'}\,\mathsf{E}[\,\xi\,]\,) &=& x \\
\mathsf{S}[\,\widetilde{x}\,] &=& x\,\beta^{-s'}\,\mathsf{S}[\,\xi\,] &=& x\,\beta^{-s'}.
\end{array}$$

Then deriving the number of relative significant digits is particularly natural:

$$s' \;=\; \log_\beta(\,\mathsf{E}[\,\widetilde{x}\,]\,) \;-\; \log_\beta(\,\mathsf{S}[\,\widetilde{x}\,]\,) \;=\; \log_\beta\left(\frac{\mathsf{E}[\,\widetilde{x}\,]}{\mathsf{S}[\,\widetilde{x}\,]}\right) \;=\; \log_\beta\left(\frac{\mu}{\sigma}\right).$$

Thus with either definition the difference in the orders of magnitude of μ and σ measures the number of significant digits. □

The essence of this result was exploited repeatedly in all the examples in the Overview, where we produced estimates $\widehat{\mu}$ and $\widehat{\sigma}$ of μ and σ, and then compared their orders of magnitude. Assuming the orders of magnitude of these estimates are accurate (which is not assuming very much), the theorem reduces determining the number of significant digits to computing a few statistics — the average and standard deviation.

The accuracy of the estimates $\widehat{\mu}$ and $\widehat{\sigma}$ increases with the number of samples n. For example, the accuracy of $\widehat{\mu}$ is measured by the standard error $\sigma/\sqrt{n}$, so the relative error in $\widehat{\mu}$ is on the order of $\widehat{\sigma}/(\widehat{\mu}\,\sqrt{n})$. Thus $\widehat{\mu}$ has something like $s' + \log_\beta(\sqrt{n})$ significant digits.

If we wish, we can also get **statistical confidence intervals** on the number of significant digits estimated from $\widehat{\mu}$ and $\widehat{\sigma}$. Although x itself may not be normally distributed, the average $\widehat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ involves a sum of independent random variables x_i (random samples), so for moderate values of n, if the distribution for x is not too unusual, the distribution of $\widehat{\mu}$ will follow a normal distribution by the Central Limit Theorem (Appendix C.3). Except for truly pathological x, enough random samples will make $\widehat{\mu}$ normally distributed. We then have the confidence interval

$$\mathsf{Pr}[\;|\,\widehat{\mu} \;-\; \mu\,| \;\leq\; t_{n-1}(\alpha/2)\;\;\widehat{\sigma}\,/\,\sqrt{n}\;\;] \;=\; 1 - \alpha$$

where t_{n-1} is the **Student** t **distribution** with $n-1$ degrees of freedom [30].

5 Previous Work relating Statistics with Numeric Computation

5.1 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 [93]. 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. ...
>
> His conclusion, the earliest statistical advice from mathematician to experimental scientist of which I am aware, was sweeping:
>
> > *Upon the whole ... it appears, that taking of the Mean of a number of observations, greatly diminishes the chances for all the smaller errors, and cuts off almost all possibility of any great ones: which last consideration, alone, seems sufficient to recommend the use of the method, not only to astronomers, but to all others concerned in making of experiments of any kind (to which the above reasoning is equally applicable). And the more observations or experiments there are made, the less will the conclusion be liable to err, provided they admit of being repeated under the same circumstances.*
> >
> > —— *Simpson [90, pp.93-94]*
>
> This statement of a law of large numbers was unsupported by proof, and Simpson's advice was long anticipated by at least the better astronomers. Nevertheless, the idea of basing this law upon mathematical calculations was new. Simpson had seen that the concept of error distributions permitted a back-door access to the measurement of uncertainty. —— Stigler [93, 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 [93, 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 [93, p.143]. He spent much of the 1770s working on the idea of an error curve, and focused particularly on the probability density $\phi(x) = (m/2)\, e^{-m|x|}$ [93, 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's initial theory of errors [107, Chs.6–7] took errors to be normally distributed, which was

[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. [45, p.210], [93, pp.145-146]).

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 [41, §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 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 [93, pp.140-141].

Gauss used his method to sensational effect:

> In January of 1801 an astronomer named G. Piazzi briefly observed and then lost a 'new planet' (actually this 'new planet' was the asteroid now known as Ceres). During the rest of 1801 astronomers and other scientists tried in vain to relocate this 'new planet' of Piazzi. The task of finding this 'new planet' on the basis of a few observations seemed hopeless.
>
> Astronomy was one of the many areas in which Gauss took an active interest. In September of 1801, Gauss decided to take up the challenge of finding the lost planet. Gauss hypothesized an elliptical orbit rather than the circular approximation which previously was the assumption of the astronomers of that time. Gauss then proceeded to develop the method of least squares. By December, the task was completed and Gauss informed the scientific community not only where to look, but also predicted the position of the lost planet at any time in the future. [On December 7, 1801, the first clear fall night, von Zach found it exactly] where Gauss had predicted it would be.
>
> This extraordinary feat of locating a tiny, distant heavenly body from apparently insufficient data astounded the scientific community. Furthermore, Gauss refused to reveal his methods. These events directly lead to Gauss' fame throughout the entire scientific community (and perhaps most of Europe) and helped to establish his reputation as a mathematical and scientific genius of the highest order. Because of Gauss' refusal to reveal his methods, there were those who even accused Gauss of sorcery.
> Gauss waited until 1809 when he published his *Theoria Motus Corporum Coelestium In Sectionibus Conicis Dolem Ambientium* to systematically develop the theory of least squares and his methods of orbit calculation. This was in keeping with Gauss' philosophy to publish nothing but well polished work of lasting significance. —— Campbell & Meyer [17, pp.39–40]

Goldstine, in his history of numerical analysis [45], translates the remarks of Gauss on how having more samples can increase the accuracy of an estimated value:

> But in such a case, if it is proposed to aim at the greatest precision, we shall take care to collect and employ the greatest possible number of accurate places. Then, of course, more data will exist than are required: but all these data will be liable to errors, however small, so that it will generally be impossible to satisfy all perfectly. Now as no reason exists, why, from among those data, we should consider any six as absolutely exact, but since we must assume, rather, upon the principles of probability, that greater or less errors are equally possible in all, promiscuously; since, moreover, generally speaking, small errors oftener occur than large ones; it is evident, that an orbit which, while it satisfies precisely six data, deviates more or less from the others, must be regarded as less consistent with the principles of the calculus of probabilities, than one which, at the same time that it differs a little from those six data, presents so much the better an agreement with the rest. The investigation of an orbit having, strictly speaking, the *maximum* probability, will depend upon a knowledge of the law according to which the probability of errors decreases as the errors increase in magnitude: but that depends upon so many vague and doubtful considerations — physiological included — which cannot be subjected to calculation, that it is scarcely, and indeed less than scarcely, possible to assign properly a law of this kind, in any case of practical astronomy. Nevertheless, an investigation of the connection between this law and the most probable orbit, which we will undertake in its utmost generality, is not to be regarded by any means a barren speculation. —— C.F. Gauss, *Theoria Motus*, p.253; in [45, p.213].

Gauss, according to Goldstein, was also first to systematically discuss the effect of rounding errors on the accuracy of numerical computation. Moreover, these discussions in *Theoria Motus* (excerpted and translated by Goldstine [45, pp.258–260]) are disconcertingly modern. He also first described Gaussian elimination in the *Theoria Motus*.

The impact of the method of least squares was immense; it was widely adopted in astronomy and geodesy [93, 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 [41].

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 [41, §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 [41, §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. [95]), and is fundamental to experimental design. Random errors are much easier to analyze, both formally and experimentally.

The central idea behind the approaches of Simpson and Gauss 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.

5.2 Monte Carlo methods

Monte Carlo methods [34, 49, 50] 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 $\hat{\sigma}$ 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 [50, 76]. By 1949 the method worked impressively on the ENIAC and von Neumann, Ulam, Fermi, Metropolis and others had produced elegant theoretical results [74].

Metropolis remarks that he invented the name 'Monte Carlo' for the method because Ulam had a gambling uncle who borrowed money from relatives "because he just had to go to Monte Carlo" [76, p.127]. He credits Fermi with having first conceived the Monte Carlo idea circa 1932:

> Fermi took great delight in astonishing his Roman colleagues with his remarkably accurate, "too-good-to-believe" predictions of experimental results. After indulging himself, he revealed that his "guesses" were really derived from the statistical sampling techniques that he used to calculate with whenever insomnia struck in the wee morning hours! And so it was that nearly fifteen years earlier Fermi had independently developed the Monte Carlo method. —— Metropolis [76, p.128]

In [50, ch.1], Lord Kelvin is also credited with developing and using modern Monte Carlo techniques in his 1901 paper [65].

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 [35] (elaborating an idea of von Neumann and Ulam), and in Householder's book [56]. 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 [34, 79]. It is worth mentioning that a number of available commercial Operations Research software packages for linear and nonlinear programming include Monte Carlo computation as a basic feature for sensitivity analysis. Among others, Paragon Decision Technology's AIMMS, Palisade's @RISK, and Sunset Software Technology's XA'llence rely on Monte Carlo analysis. In addition, there are many statistical packages (particularly those aimed at forecasting), combinatorial optimization packages, and even fuzzy system and neural network packages that perform similar functions with randomization. One feature common to such packages is that they strive to be comprehensible to ordinary users, allowing them to visualize the sensitivity (uncertainty, inexactness) of the results in an intuitive way. Techniques useful in visualizing sensitivity of matrix computations are described in [23] and [113].

5.3 Statistical roundoff analysis

The use of statistical methods in numerical analysis arguably began with the pioneering works of von Neumann and Goldstine in 1947 [80] and 1951 [44], who decided to show that the 1943 gloomy exponential forward error bounds for Gaussian elimination of Hotelling [55, p.7] were not reflective of practice. An excellent summary of these papers is given by Wilkinson [111], and also by Higham [54, §9.6], who reproduces the following remark by Goldstine about this work with von Neumann:[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 [46, p.290]

In [80, 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 [44], 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 actually made an analysis for random matrices also, remarking that normally distributed random matrices have expected spectral norm on the order of $\sqrt{n}$, and that "random matrices are only slightly ill-conditioned" [98, p.299].

Early statistical analyses of rounding methods, leading digit frequencies, etc., are surveyed by Knuth [69] and Sterbenz [92, §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., [21, 22], and very recently [23]). 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 [23].

5.4 Random rounding

Random rounding arises from allowing floating-point rounding to be nondeterministic, rounding up or down with some probability that can depend on the value to be rounded. Although it has not been developed fully, random rounding has been used a number of times.

Random rounding was considered in the early 1950s by George Forsythe [36, 37]. Inspired by the Monte Carlo approach, and the work of von Neumann and Goldstine, he noted [38] that roundoff errors in some problems (specifically citing Huskey's results for numerical integration [58]) are not distributed like independent random variables (a point stressed by Kahan [63] and Higham [54, §1.17,§2.6]), better error estimates could be obtained if they *were*. He suggested that they can be forced to be random using the method we call (see p.34) `round_random_nearness`:

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

> 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 [36] 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 $v(1 - v) \leq \frac{1}{4}$ 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 [38]

In concluding [36], 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."

Prompted by the same results of Huskey for numerical integration that led to Forsythe's random rounding proposal, Henrici [51, ch.5] gives a good review of early statistical analyses of roundoff error in numerical integration, and argues that it can be analyzed statistically. 'Probabilistic rounding' (`round_random_nearness` again) was used by Hull and Swenson [57] in order to test the hypothesis that ordinary rounding can be modeled statistically as a random process. Their experiments agreed with the results of Henrici, that the cumulative roundoff errors obtained with random rounding (resp. chopping) were similar to those obtained with ordinary rounding (resp. chopping), and that probabilistic models of roundoff propagation are generally valid. Henrici was intrigued by this experimental approach, and showed their results could be predicted analytically to within an error of 10% [52].

We have also found the idea of random rounding in a paper of Callahan [16], who demonstrated its use in signal processing applications (perhaps the most well-developed domain for roundoff analysis). He also proposes `round_random_nearness`. Callahan generally investigated the uses of randomness in signal processing, but like Forsythe, in this paper he stresses the usefulness of truly random roundoff errors. He points out that the often-used simple recursive integrator whose filter is implemented by the difference equation

$$y_n \quad = \quad (1 \; - \; 1/M)\, y_{n-1} \quad + \quad x_n/M$$

where the x_n are an input time series, $M = 2^m$, can have "dramatic correlated quantization error effects unless random rounding is employed" [16, p.501].

Finally, a twist on the idea of random rounding was developed by Yoshida, Goto, and Ichikawa [115], permitting them to implement floating-point multipliers with many fewer gates than usual. They omit computation of the less significant part of the product of two floating-point numbers, instead relying on the fact that "the average of that neglected part is compensated for by using the statistical properties of medium significant bits", since they "are regarded as random in pseudorandom number generators" [115, p.1065–1066]. Extraction of the medium significant bits from a sequence of squared values — the 'midsquare' method of Metropolis and von Neumann [68, §3.1] — is how random numbers were produced in the original Monte Carlo computations on ENIAC, but this method was quickly discarded in favor of congruential methods [50, p.27].

5.5 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 [99]. The paper [73] 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 [99] '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` [100]. Vignes and Ung patented the idea in Europe in 1979 [101], and in the USA in 1983 [102]. The retrospective survey [103] 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 [14], CESTAC works on numerical programs in which each floating-point expression '$X \bullet Y$' has been preprocessed to $\texttt{PER}(X \bullet Y)$, so for example $X+Y+Z$ becomes $\texttt{PER}(\texttt{PER}(X+Y)+Z)$, 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 [105] accompanies an entire journal issue with papers describing applications of CESTAC.

Both `PER` and the perturbation mode of `PEPER` work identically [14, 105]: 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 [106]. 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' [104], and when tested by a comparison operator these zeroes produce exceptions.

Kahan [63] 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" [63, p.7]. He also remarks that the CESTAC patents can be circumvented with IEEE standard arithmetic by randomly toggling the IEEE directed rounding control bits [63, p.19].

Other statistical studies of computer arithmetic include [7, 13, 32, 70, 81]. In [81], 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).

Having said all this, it is also worth saying that the use of stochastic or statistical methods in computer arithmetic is seldom discussed in contemporary numerical analysis literature. For example, the reference text [47] by Golub and Van Loan does not mention Vignes, and the only text we know of that mentions Vignes is Higham's encyclopedia [54, p.53], which allots two sentences to CESTAC. Generally, the coverage of significance arithmetic is similar. Interval arithmetic is mentioned in most texts, but is dismissed quickly.

6 Monte Carlo Arithmetic

With all of the preceding in mind, our goal is to exploit randomness in order to produce better floating-point arithmetic systems that model inexactness and significance, and resolve basic anomalies of conventional floating-point arithmetic.

6.1 Basic objects

As discussed in Section 4.4, the distinction between exact and inexact values can be realized by modeling inexact values as random variables. Using this approach, our goal becomes the implementation of arithmetic on real numbers (exact values) and random variables (inexact values), and somehow this is to be done with floating-point hardware.

We assume that the following are available, relative to a given hardware architecture (with its own floating-point format and integer format):

- **Random variables**
  We assume that (real-valued) random variables are available, and they are sampled whenever referenced. Since confusion should not result, we simply use the single symbol ξ for a random variable, and let this also denote a sampled value.

- **Program variables**
  Program variables are 'names' used to refer to specific values. Here program variables are represented with symbols like X, Y, Z, and they respectively denote numeric values x, y, z.

- **Values**
  Here all values are random variables, including the reals (exact values) as a special case. The uncertainty inherent in inexact values is modeled with randomness beyond a specified number of significant digits. Thus, values can belong to the following overlapping classes:

  - **exact values**: real values with arbitrarily many significant digits.
  - **inexact values**: real values with only a finite number of significant digits.
  - **floating-point values**: real values representable in the machine floating-point format.
  - **machine integer values**: integer values representable in the machine integer format.

  Symbols like x, y, z denote values.

The distinction between **exact floating-point values** and **inexact floating-point values** is important. Inexact floating-point values have a specified number of significant digits (at most p, the floating-point hardware precision).

Analyzing the distinction between exact and inexact machine integer values is also worthwhile. However, for simplicity we will discuss only floating-point computation below.

Inexactness is 'contagious': any arithmetic operation involving an inexact operand yields an inexact value, except for degenerate expressions like $(0 \times z)$. Ideally the distinction between exact and inexact floating-point values would be implemented in hardware, e.g., by a bit in the representation of floating-point values [75, 85].[6] This was done in the NORC calculator in the early 1950s [92, p.196]. The distinction can still be implemented in an approximate way in software, using techniques described below in Section 6.8.1.

[6]The IEEE 754 standard defines an INEXACT flag, which is raised whenever the result of a floating-point computation does not fit within the floating-point format. However, the standard does not otherwise track inexactness of values. Including the INEXACT flag in the floating-point format instead would cost an additional bit, but would permit tracking of inexactness, and would eliminate the expense of monitoring this flag, which is controversial and has resulted in abuse [62].

6.2 Implementing inexact values with randomization

Let x be a numeric value (exact or inexact), and ξ be a random variable. We transform x into a new inexact value with **randomization**, as developed in Section 4.4:

$$\mathsf{inexact}\,(x,\, s,\, \xi) \quad = \quad x \quad + \quad \beta^{e-s}\ \xi \qquad \text{where } e = \beta\mathrm{oom}(x).$$

Here s is a real value (typically a positive integer), and ξ is a random variable that can be discrete or continuous, and can depend on x. [7]

If ξ is restricted to the interval $(-\frac{1}{2}, \frac{1}{2})$, then s gives the accuracy (number of digits) to which the result agrees with x. The error is less than one half unit in the s-th place. Then the effect of $\mathsf{inexact}\,(x,\, s,\, \xi)$ is to add random digits below the s-th digit in the representation of the value of x, and the result is significant to (at most) s digits:

- An interesting choice is $s = 0$, so that $\mathsf{inexact}\,(x,\, s,\, \xi)$ generates random numbers about x.
- A more relevant choice is $s = t$, where t is the virtual precision. The effect is to extend a t-digit floating-point number with 'noise', as illustrated in Figure 3.

sampled value of ξ	`-0.14415412810232532...`
value x	`+7.5111111`
value of $10^{1-8}\,\xi$	`-0.00000001441541281023253 2...`
value of $\mathsf{inexact}\,(x,\, 8,\, \xi)$	`+7.51111108558458718976746 8...`

Figure 3: Example of randomization using eight-digit decimal arithmetic ($s = t = 8$, $\beta = 10$). Notice $\mathsf{inexact}\,(x,\, s,\, \xi)$ yields *real values*.

6.3 Precision bounding

Our key idea is that the information that is lost by using finite-precision computation can *also* be modeled as inexactness, and can also be implemented with random error. 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 precision bounding of x to t digits is implemented as

$$t_\mathsf{digit_precision}\,(x),$$

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

The **virtual precision** t is the precision to which arithmetic values are represented (in memory). If the precision of floating-point registers is p, it is natural to require $t \leq p$. When $t = p$, the memory and register precision agree. By varying t in the definition of $t_\mathsf{digit_precision}\,(x)$ we implement arithmetic of any desired precision $t \leq p$. The ability to vary the virtual precision can be very useful, so we allow t to differ from p.

[7] Because the order of magnitude of an exact 0 is unspecified, we define $\mathsf{inexact}\,(0,\, s,\, \xi) = 0$. On computers like Stretch [15] with an order-of-magnitude zero $(0 \times \beta^e)$, define $\mathsf{inexact}\,((0 \times \beta^e),\, s,\, \xi) = \beta^{e-s}\ \xi$.

We define **precision bounding** of a value x to t digits as randomizing x if it is inexact, and leaving x undisturbed otherwise:

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

Here the values for t and ξ are left implicit; they take default values if unspecified. We have implemented many varieties of randomization, with default $t = p$, including the following:

- `identity`
  The random variable ξ used in randomization is the constant zero.

- `uniform_absolute`
  The random variable ξ is uniformly distributed over $(-\frac{1}{2}, \frac{1}{2})$, with mean 0 and standard deviation $1/\sqrt{12}$. Thus $t_\mathsf{digit_precision}(x)$ has mean x and standard deviation $\beta^{e-t}/\sqrt{12}$ when x is a real value. More generally, when x is a random variable, $t_\mathsf{digit_precision}(x)$ has mean $\mathsf{E}[x]$, and in the usual case that $e = \beta\mathrm{oom}(x)$ is constant[8] over the distribution of x, its standard deviation is $\sqrt{\mathsf{S}[x]^2 \;+\; \beta^{2(e-t)}/12}$.

- `uniform_relative`
  Here $\xi = (x/\beta^e)\,\delta$, where δ is a random variable that is uniformly distributed over $(-\frac{1}{2}, \frac{1}{2})$. Thus $t_\mathsf{digit_precision}(x) \;=\; x\,(1 \;+\; \beta^{-t}\,\delta)$ has mean x and standard deviation $x\,\beta^{-t}/\sqrt{12}$ when x is a real value. More generally, when x is a random variable, it has mean $\mathsf{E}[x]$ and has standard deviation[9] $\mathsf{S}[\; x\,(1+\delta)\;] \;=\; \sqrt{\mathsf{S}[x]^2\,(1 \;+\; \beta^{-2t}/\,12) \;+\; \mathsf{E}[x]^2\,\beta^{-2t}/\,12}$.

6.4 Random rounding

Analysis of the rounding operation by itself is very complicated. However, combining it with randomization makes its analysis much simpler.

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

$$\mathsf{random_round}(x) \quad = \quad \mathsf{round}(\; t_\mathsf{digit_precision}(x)\;).$$

In this work, random rounding uses `uniform_absolute` randomization in $t_\mathsf{digit_precision}(x)$, because the resulting rounding method has zero expected bias, as we now explain.

When x is exact (i.e., x can be expressed exactly with t digits), notice that

$$\mathsf{random_round}(x) \quad = \quad \mathsf{round}(\; x\;) \quad = \quad x.$$

Thus random rounding not only generalizes on ordinary rounding, but is also consistent with the basic assumption about the nature of rounding that *rounding an exact value yields an exact value.* When x is inexact, then $t_\mathsf{digit_precision}(x) \;=\; x \;+\; \beta^{e-t}\,\xi$ where $e = \beta\mathrm{oom}(x)$, and expanding the definition of $\mathsf{random_round}(x)$ with the definition of rounding gives

$$\mathsf{random_round}(x) \quad = \quad \beta^{e\,-\,p}\;\left\lfloor\; \beta^{p-e}\;x \;+\; \beta^{p-t}\;\xi \;+\; \tfrac{1}{2}\;\right\rfloor \qquad \text{where} \quad e \;=\; \beta\mathrm{oom}(x)$$

[8] When $e = \beta\mathrm{oom}(x)$ varies over the distribution of x, which occurs for example when x is uniform over the interval $(.999\cdots999,\; 1.00\cdots001)$, then in the standard deviation formula $\sqrt{\mathsf{S}[x]^2 \;+\; \beta^{2(e-t)}/12}$ the factor β^{2e} is replaced by another factor that is upper-bounded by β^{2E}, where E is the maximum value of $\beta\mathrm{oom}(x)$.

[9] This formula follows from $\mathsf{S}[\,x\,y\,] \;=\; \sqrt{\mathsf{S}[x]^2\,(\mathsf{E}[y]^2 \;+\; \mathsf{S}[y]^2) \;+\; \mathsf{E}[x]^2\,\mathsf{S}[y]^2}$.

ignoring details such as handling of carry-out. Again, we let ξ be an implicit parameter, a random variable that can be discrete or continuous, and can depend on the value of x.

In the important case $t = p$, natural rounding methods result from choices about the distribution of ξ. These are definable in terms of the fractional part of x to be rounded away, which we call θ:

$$\theta \quad = \quad \theta(x) \quad = \quad (\ \beta^{p-e} \ x \quad - \quad \lfloor \ \beta^{p-e} \ x \ \rfloor \ ) \qquad \text{where} \quad \beta^{e-1} \ \leq \ x \ < \ \beta^{e}.$$

- `round_to_nearest`:
  In the degenerate case where ξ is the constant 0 we obtain the ordinary round function.

- `round_random_up_or_down`:
  Round x up or down with probability $\frac{1}{2}$. This is achieved if ξ is a continuous random variable that is uniform on $(\ -\theta, \ \ (1-\theta) \ )$. Then ξ has mean $-\theta$ and standard deviation $1/\sqrt{12}$.

  This method is easily implemented with the existing IEEE standard through random toggling of the directed rounding bits, as suggested by Kahan in 1996 [63, p.19].

- `round_random_nearness`:
  Round x up or down with probability proportional to the distance from x to either of these two alternatives. The desired effect is achieved if ξ is a `uniform_absolute` variable, i.e., is uniform on $\left(-\frac{1}{2}, \frac{1}{2} \right)$. Then ξ has mean 0 and standard deviation $1/\sqrt{12}$.

  This method was implemented and used successfully by Forsythe in 1959 [38], Hull and Swenson (with coaching by Kahan) in 1966 [57], Callahan in 1976 [16], and probably others.

The final two random rounding methods above satisfy the worst-case forward error bound

$$| \ \mathsf{random_round}\,(x) \ - \ x \ | \quad \leq \quad \beta^{e-p}$$

when x is a nonzero floating-point value, $e = \beta\mathrm{oom}(x)$, and again $t = p$. This bound is twice as large as that for `round_to_nearest`, because rounding is not always to the nearest value.

Despite this, there are two compelling reasons for using `uniform_absolute` randomization in

$$\mathsf{random_round}\,(x) \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(x)\ ).$$

First, the uniform absolute distribution is independent of x, which permits simpler statistical analysis. Second, *random rounding then has zero expected bias*:

Theorem 2 *For* `uniform_absolute` *randomization, if x is a random variable then*

$$\begin{aligned}
\mathsf{E}[\ \mathsf{round}\,(t_\mathsf{digit_precision}\,(x))\ ] &= \mathsf{E}[\ x\ ] \\
\mathsf{V}[\ \mathsf{round}\,(t_\mathsf{digit_precision}\,(x))\ ] &= \mathsf{V}[\ x\ ] \ + \ \beta^{2\,(e\,-\,t)}\ \varsigma^2 \\
\mathsf{S}[\ \mathsf{round}\,(t_\mathsf{digit_precision}\,(x))\ ] &= \sqrt{\mathsf{S}[\ x\ ]^2 \ + \ \beta^{2\,(e\,-\,t)}\ \varsigma^2}
\end{aligned}$$

where $\varsigma^2 < 1$ depends on the distribution of x.

Proof
We prove first the case $t = p$, which was essentially done by Forsythe [38] (see the reproduced paragraph on p.29). The case $t < p$ is a straightforward extension. Again, let

$$\theta(x) \quad = \quad \beta^{p-e}x \quad - \quad \lfloor \ \beta^{p-e}x \ \rfloor$$

so that $0 \leq |\theta(x)| \leq 1$. Then for `uniform_absolute` randomization with $t = p$ we derive:

$$\begin{aligned}
\mathsf{E}[\ \mathsf{random_round}(x)\ ] &= \mathsf{E}\Big[\ \beta^{e-p}\ \big\lfloor\ \beta^{p-e}\ x\ +\ \xi\ +\ \tfrac{1}{2}\ \big\rfloor\ \Big] \\
&= \beta^{e-p}\ \mathsf{E}\left[\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ \beta^{p-e}\ x\ +\ z\ +\ \tfrac{1}{2}\ \big\rfloor\ dz\ \right] \\
&= \beta^{e-p}\ \mathsf{E}\left[\ \int_{-\frac{1}{2}}^{\frac{1}{2}-\theta(x)} \big(\ \beta^{p-e}\ x\ -\ \theta(x)\ \big)\ dz\ +\ \int_{\frac{1}{2}-\theta(x)}^{\frac{1}{2}} \big(\ \beta^{p-e}\ x\ -\ \theta(x)\ +\ 1\ \big)\ dz\ \right] \\
&= \beta^{e-p}\ \mathsf{E}\Big[\ (1\ -\ \theta(x))\ \big(\ \beta^{p-e}\ x\ -\ \theta(x)\ \big)\ \ +\ \ \theta(x)\ \big(\ \beta^{p-e}\ x\ -\ \theta(x)\ +\ 1\ \big)\ \Big] \\
&= \beta^{e-p}\ \mathsf{E}\Big[\ \beta^{p-e}\ x\ \Big] \\
&= \mathsf{E}[\ x\ ].
\end{aligned}$$

$$\begin{aligned}
\mathsf{E}\big[\ \mathsf{random_round}(x)^2\ \big] &= \beta^{2(e-p)}\ \mathsf{E}\Big[\ (1\ -\ \theta(x))\ \big(\ \beta^{p-e}\ x\ -\ \theta(x)\ \big)^2\ \ +\ \ \theta(x)\ \big(\beta^{p-e}\ x\ -\ \theta(x)\ +\ 1\ \big)^2\ \Big] \\
&= \mathsf{E}\big[\ x^2\ \big]\ \ +\ \ \beta^{2(e-p)}\ \mathsf{E}[\ \theta(x)\ (1\ -\ \theta(x))\ ].
\end{aligned}$$

$$\begin{aligned}
\mathsf{V}[\ \mathsf{random_round}(x)\ ] &= \mathsf{E}\big[\ \mathsf{random_round}(x)^2\ \big]\ \ -\ \ \mathsf{E}[\ \mathsf{random_round}(x)\ ]^2 \\
&= \mathsf{E}\big[\ x^2\ \big]\ \ +\ \ \beta^{2(e-p)}\ \mathsf{E}[\ \theta(x)\ (1\ -\ \theta(x))\ ]\ \ -\ \ \mathsf{E}[\ x\ ]^2 \\
&= \mathsf{V}[\ x\ ]\ \ +\ \ \beta^{2(e-p)}\ \mathsf{E}[\ \theta(x)\ (1\ -\ \theta(x))\ ].
\end{aligned}$$

Defining $\varsigma^2 = \mathsf{E}[\ \theta(x)\ (1\ -\ \theta(x))\ ]$ completes the proof, since $0 \leq \theta(x) \leq 1$ implies $\varsigma^2 \leq \frac{1}{4}$.

For the case $t < p$, we have

$$\mathsf{E}[\ \mathsf{random_round}(x)\ ]\ =\ \mathsf{E}\Big[\ \beta^{e-p}\ \big\lfloor\ \beta^{p-e}\ x\ +\ \beta^{p-t}\ \xi\ +\ \tfrac{1}{2}\ \big\rfloor\ \Big]\ =\ \beta^{e-p}\ \mathsf{E}\left[\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ \beta^{p-e}\ x\ +\ \beta^{p-t}\ z\ +\ \tfrac{1}{2}\ \big\rfloor\ dz\ \right]$$

and the stated expected values can be derived from the fact that for $B = \beta^{p-t}$,

$$\begin{aligned}
\int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ B\ z\ +\ \tfrac{1}{2}\ \big\rfloor\ dz &= \frac{1}{B}\ \int_{-\frac{1}{2}B}^{\frac{1}{2}B} \big\lfloor\ X\ +\ t\ +\ \tfrac{1}{2}\ \big\rfloor\ dt \\
&= \frac{1}{B}\ \sum_{k=1}^{B}\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ \big(\ k\ -\ \tfrac{1}{2}B\ -\ \tfrac{1}{2}\ \big)\ +\ u\ +\ \tfrac{1}{2}\ \big\rfloor\ du \\
&= \frac{1}{B}\ \sum_{k=1}^{B}\ \left(\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ u\ +\ \tfrac{1}{2}\ \big\rfloor\ du\ +\ \big(\ k\ -\ \tfrac{1}{2}B\ -\ \tfrac{1}{2}\ \big)\ \right) \\
&= \left(\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ u\ +\ \tfrac{1}{2}\ \big\rfloor\ du\ \right)\ +\ 0.
\end{aligned}$$

$$\begin{aligned}
\int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ B\ z\ +\ \tfrac{1}{2}\ \big\rfloor^2\ dz &= \frac{1}{B}\ \sum_{k=1}^{B}\ \left(\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ u\ +\ \tfrac{1}{2}\ \big\rfloor^2\ du\ +\ \big(\ k\ -\ \tfrac{1}{2}B\ -\ \tfrac{1}{2}\ \big)^2\ \right)\ +\ 0 \\
&= \left(\ \int_{-\frac{1}{2}}^{\frac{1}{2}} \big\lfloor\ X\ +\ u\ +\ \tfrac{1}{2}\ \big\rfloor^2\ du\ \right)\ +\ \frac{(B^2-1)}{12}.
\end{aligned}$$

□

Theorem 2 does not hold for `uniform_relative` randomization. If $s' = p$ in

$$\mathsf{inexact}(x, p, \xi)\ \ =\ \ x\ (\ 1\ +\ \beta^{-s'}\ \xi\ )$$

then $\mathsf{E}[\ \mathsf{random_round}\,(x)\ ] \neq \mathsf{E}[\ x\ ]$, i.e., random rounding becomes biased. It can be made unbiased, however, by choosing $s' = p - e + \log_\beta |x|$ where $e = \beta\mathrm{oom}(x)$, so that this formulation is equivalent to the `uniform_absolute` formulation. Because Theorem 2 does not hold for all randomization schemes, below 'random rounding' will mean rounding with `uniform_absolute` randomization.

6.5 Modeling roundoff error with randomization

Roundoff error in computer arithmetic operations is usually studied with either forward or backward error analysis. These analyses assume the input or output is exact, and thus defer error analysis of the input or output. Here we define backward & forward error, which does no such deferral, and shows how these analyses can be expressed in terms of randomization. Although we formulate roundoff in terms of absolute error, the same conclusions could be reached with relative error.

6.5.1 Perspectives on roundoff error

Let '$\circledbullet$' be the floating-point approximation to the '$\bullet$' operation. Then there are several perspectives on the error in $(x \circledbullet y)$, where x and y are floating-point values in $\mathcal{F}$ and $(x \bullet y) \in \Re_{\mathcal{F}}$:

1. **forward error: error is attributed to output inaccuracy**
   Assuming the inputs x and y are exact, there exists a real value ε so that

$$x \circledbullet y \quad = \quad (x \bullet y) \ + \ \varepsilon.$$

2. **backward error: error is attributed to input inaccuracy**
   Assuming the computed output is exact, there exist real values ε_x and ε_y so that

$$x \circledbullet y \quad = \quad (\ x \ + \ \varepsilon_x\ ) \ \bullet \ (\ y \ + \ \varepsilon_y\ ).$$

3. **backward & forward error: error is attributed to limited precision**
   There exist real values ε_x, ε_y, ε so that

$$x \circledbullet y \quad = \quad (\ (\ x \ + \ \varepsilon_x\ ) \ \bullet \ (\ y \ + \ \varepsilon_y\ )\ ) \ + \ \varepsilon.$$

   Error is attributed to the inexactness of t-digit arithmetic on both input and output. This assumption-free approach emphasizes simplicity at the cost of possibly doubling the error.

6.5.2 Rephrasing error perspectives with precision bounding and random rounding

Each of the rounding errors ε above can be replaced by a random variable, and each expression $(z + \varepsilon)$ can be replaced by $t_\mathsf{digit_precision}\,(z)$. This gives the following transcription:

1. **forward error: output precision bounding**

$$x \circledbullet y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(\ x \ \bullet \ y\ )\ ).$$

2. **backward error: input precision bounding**

$$x \circledbullet y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(x) \ \bullet \ t_\mathsf{digit_precision}\,(y)\ ).$$

3. **backward & forward error: input & output precision bounding**

$$x \circledbullet y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(\ t_\mathsf{digit_precision}\,(x) \ \bullet \ t_\mathsf{digit_precision}\,(y)\ )\ ).$$

This transcription is essentially direct. It has the advantage of identifying all errors with bona-fide random variables. In this view a roundoff error for a program of arithmetic operations is an expression involving random variables, defining a distribution that can be sampled by executing the program. See Section 6.8.3.

Notice that by Theorem 2, the rounded output precision bounding transcriptions of forward error and backward & forward error have zero expected bias, as they implement random rounding.

6.6 Monte Carlo Arithmetic

Finally, we define **Monte Carlo Arithmetic** (MCA) to be a floating-point arithmetic system including exact and inexact values, random rounding, and precision bounding. In MCA, if x and y are floating-point values and '$\circledbullet$' is the floating-point implementation of the '$\bullet$' operator, then

$$x \circledbullet y \quad = \quad \mathsf{round}\,(\ \textsc{outbound}(\ \textsc{inbound}(x)\ \bullet\ \textsc{inbound}(y)\ )\ )$$

where precision bounding can be selectively employed or suppressed:

$$\begin{array}{rcl@{\qquad}rcl} \textsc{inbound}(x) & = & x & \text{or} & \textsc{inbound}(x) & = & t_\mathsf{digit_precision}\,(x) \\ \textsc{outbound}(x) & = & x & \text{or} & \textsc{outbound}(x) & = & t_\mathsf{digit_precision}\,(x). \end{array}$$

MCA allows differing methods to be used in random rounding and precision bounding, and allows them to be changed during execution. The choices determine many instances of MCA, including:

- **Random rounding**
  With random rounding alone, only the output is randomized, modeling forward error:

  $$x \circledbullet y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(\ x\ \bullet\ y\ )\ ).$$

  Random rounding eliminates the expected error from rounding, as shown by Theorem 2.

- **Input precision bounding**
  An alternative approach is to randomize only the inputs:

  $$x \circledbullet y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(x)\ \bullet\ t_\mathsf{digit_precision}\,(y)\ ).$$

  Input precision bounding detects catastrophic cancellation: the random lower-order digits it introduces record the loss of significance resulting from cancellation, as illustrated in Figure 4. The only drawback to this approach is bias: the expected rounding error is occasionally nonzero, for example when two positive values are added and a carry-out occurs.

operand x	`+11111113.`	
$x' = t_\mathsf{digit_precision}\,(x)$		`+11111113.20391695941600884...`
operand y	`-11111111.`	
$y' = t_\mathsf{digit_precision}\,(y)$		`-11111110.91870795420835463...`
sum $x' + y'$		`+2.28520900520765421...`
rounded sum $\mathsf{round}\,(x' + y')$	`+2.2852090`	

Figure 4: Example of input precision bounding (in a sum producing catastrophic cancellation) using eight-digit decimal arithmetic ($t = p = 8$, $\beta = 10$). Boxed values are floating-point values.

- **Random unrounding**
  An elegant extension of input precision bounding was developed by Pierce [85]. He showed that it can be explained naturally in terms of **random unrounding**, which is the random conversion of a floating-point value to an inexact real value. Pierce also developed an 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. Maintaining values in registers in this way reduces the roundoff error incurred in the course of a computation. Effectively, Pierce proposes implementing $p \gg t$, where p is the 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.

- **Full MCA: precision bounding & random rounding**
  With both input and output precision bounding, we have the full power of MCA:

  $$x \circledcirc y \quad = \quad \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(\ t_\mathsf{digit_precision}\,(x)\ \bullet\ t_\mathsf{digit_precision}\,(y)\ )\ ).$$

  By randomizing both the inputs and the output, full MCA achieves two important properties. First, input precision bounding detects catastrophic cancellation, which is the primary way that significant digits are lost in numerical computation. Second, output precision bounding 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 all executed with full MCA.

With MCA, although in theory all intermediate computations in the ALU before the final rounding are done to infinite precision (as illustrated in Figure 4), in practice this is not necessary of course. As suggested by Forsythe in the quotation on p.29, it is sufficient is to generate the random digits incrementally, as required by cancellation and renormalization in these computations.[10]

There are other 'partial' instances of Monte Carlo Arithmetic. First, since cancellation arises only in addition/subtraction, input precision bounding might be omitted in multiplication/division. Second, if input precision bounding is used and the output of an addition/subtraction operation has the same exponent as an inexact input, then output precision bounding is unnecessary — the randomization for that input will eliminate any expected rounding bias. Other 'optimizations' of this kind can be used to reduce both the overhead and the variance of the roundoff errors of MCA. In the interest of simplicity, though, for the rest of this work we will emphasize full MCA.

6.7 Which instance of Monte Carlo Arithmetic is best?

Monte Carlo Arithmetic is 'open', in the sense that the MCA definition is parametrized in several ways. There seems to be no single best instance; it seems best left parametric for several reasons.

First, it is evident that different numerical problems produce values of different kinds and different significance. Thus, for example, varying the virtual precision t may be important in some situations.

Second, there are multiple perspectives on error, and the stability of any numerical problem can be defined in different ways, depending on what one views as the input and as the output. If our goal is flexibility in evaluating the sensitivity of one to the other, and support of experimentation, then user control over randomization is necessary.

[10] Another strategy is simply to carry these computations to $2p$ digits, i.e., to use p guard digits. These guard digits are adequate, except sometimes in the case when $t = p$ and $t_\mathsf{digit_precision}\,(x) \ominus t_\mathsf{digit_precision}\,(y)$ is computed for two inexact values such that $x = y$; but this case can be detected and handled specially.

Third, different problems give most accurate results with different instances of MCA. As an example, let us compare the performance of the different rounding and input precision bounding methods on the quadratic equation tests mentioned in the Introduction.

In all of these results, we used $t = p = 24$, and the virtual precision was single precision. When $t = p$, rounded output precision bounding is just random rounding. Therefore the results below simply tabulate random rounding methods against input precision bounding methods.

In each test, we computed the number of significant digits derived from the relative error (the negative base-10 log of the relative error of the average of `r2` with respect to the actual second root) and derived from the standard error (the negative base-10 log of the ratio of the standard error of `r2` over the actual second root).

For the first equation

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

we found the second root `r2` using the standard quadratic formula. As a simple test, we compared the results of using the randomization methods introduced above, over 100 samples. The results are in Table 9. The latter value is presented in square brackets. The final value is omitted from the table because with CESTAC ($b^2 - 4ac$) occasionally goes negative.

output	*input precision bounding*			
precision bounding	`identity`	`uniform_absolute`	`uniform_relative`	`CESTAC`
`identity`	`4.1`	`6.4 [5.2]`	`5.2 [5.2]`	`4.5 [4.8]`
`uniform_absolute`	`6.1 [5.3]`	`6.7 [5.2]`	`4.9 [5.1]`	`4.5 [4.8]`
`uniform_relative`	`5.8 [5.3]`	`5.3 [5.2]`	`5.4 [5.1]`	`4.5 [4.8]`
`CESTAC`	`5.9 [5.0]`	`4.7 [5.0]`	`5.6 [5.0]`	

Table 9: Average number of significant decimal digits in `r2` and [the number of significant decimal digits in `r2` estimated from its standard error] for the first equation (100 samples).

For the second root of the second equation

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

computing these significance estimates again gave Table 10.

output	*input precision bounding*			
precision bounding	`identity`	`uniform_absolute`	`uniform_relative`	`CESTAC`
`identity`	`0.7`	`2.2 [1.7]`	`2.3 [1.7]`	`1.6 [1.5]`
`uniform_absolute`	`1.7 [1.9]`	`1.5 [1.7]`	`1.6 [1.7]`	`1.6 [1.4]`
`uniform_relative`	`2.1 [1.8]`	`1.6 [1.6]`	`2.0 [1.6]`	`1.6 [1.4]`
`CESTAC`	`1.8 [1.5]`	`1.4 [1.6]`	`1.5 [1.5]`	`1.0 [1.3]`

Table 10: Average number of significant decimal digits in `r2` and [the number of significant decimal digits in `r2` estimated from its standard error] for the second equation (100 samples).

These tables are inconclusive; they reflect the results for only two small computations, and the results depend strongly on the number of samples. However, they do show that the second computation is much more ill-conditioned than the first. Also, at least for these tiny problems, the standard error usually gives a good estimate of the number of significant digits in a result, but can be pessimistic (or slightly optimistic): the average value can be a good estimate, even when

the standard deviation is high. Finally, they show that the differences between methods are not enormous, but can be consistent, and furthermore that the effectiveness of any given method is problem-dependent.

6.8 Transforming floating-point computations into Monte Carlo computations

It is not difficult to use MCA in implementing numeric programs. In this section we sketch a way to transform programs written in an algorithmic language into similar programs that use Monte Carlo Arithmetic.

There are several points to sketching the transformation explicitly. First, doing this shows how to implement MCA in software. Even if no hardware support is available, we can still implement the bulk of MCA without great effort. Second, it shows how different instances of MCA can be implemented with variations on one basic approach. Finally, it sketches how one can implement executable roundoff analyses. By having the transformation generate code that makes random perturbations in specific inputs and then analyzes the effect of these perturbations on specific outputs, we produce a program that can be used to explore output roundoff near those inputs.

6.8.1 Transformation of programs

Using t_digit_precision$(\cdot)$ as just extended, we can transform arbitrary floating-point computations into randomized (Monte Carlo) computations in a natural way. Suppose a program is defined using the following grammar:

PROGRAM	→	STATEMENT
PROGRAM	→	STATEMENT ; PROGRAM
STATEMENT	→	VARIABLE := EXPR
EXPR	→	EXACT_EXPR
EXPR	→	INEXACT_EXPR
EXPR	→	(EXPR)
EXACT_EXPR	→	EXACT_VALUE
EXACT_EXPR	→	INTEGER_VARIABLE
INEXACT_EXPR	→	INEXACT_VALUE
INEXACT_EXPR	→	FLOAT_VARIABLE
INEXACT_EXPR	→	EXACT_EXPR OPERATOR EXACT_EXPR
INEXACT_EXPR	→	EXACT_EXPR OPERATOR INEXACT_EXPR
INEXACT_EXPR	→	INEXACT_EXPR OPERATOR EXACT_EXPR
INEXACT_EXPR	→	INEXACT_EXPR OPERATOR INEXACT_EXPR.

Then we can transform a program into a 'Monte Carlo' program with a transformation $MC[\,\cdot\,]$, defined as follows:

$$
\begin{aligned}
MC[\,\text{STATEMENT ; PROGRAM}\,] &= MC[\,\text{STATEMENT}\,]\ ;\ MC[\,\text{PROGRAM}\,] \\
MC[\,\text{VARIABLE := EXPR}\,] &= \text{VARIABLE} := MC[\,\text{EXPR}\,] \\
MC[\,(\ \text{EXPR}\ )\,] &= (\ MC[\,\text{EXPR}\,]\ ) \\
MC[\,\text{EXACT_VALUE}\,] &= \text{EXACT_VALUE} \\
MC[\,\text{INTEGER_VARIABLE}\,] &= \text{INTEGER_VARIABLE} \\
MC[\,\text{INEXACT_VALUE}\,] &= \text{INEXACT_VALUE} \\
MC[\,\text{FLOAT_VARIABLE}\,] &= \text{FLOAT_VARIABLE} \\
MC[\,E_1\ \text{OPERATOR}\ E_2\,] &= \text{round}\left(t\text{_digit_precision}\left(\widetilde{E_1}\ \text{OPERATOR}\ \widetilde{E_2} \right) \right) \\
\widetilde{E} &= \begin{cases} E & \text{if} \quad E = \text{EXACT_EXPR} \\ t\text{_digit_precision}(E) & \text{if} \quad E = \text{INEXACT_EXPR.} \end{cases}
\end{aligned}
$$

The transformation simply replaces every subexpression with its randomization. It can be adapted for any standard algorithmic language. The transformation currently assumes that integer values are exact, but inexact integers could be properly implemented, as remarked in Section 6.1.

As an example of the transformation, consider the program P

$$\begin{aligned} Y &:= 2 \times (Z + 3.1416) \;\; ; \\ X &:= Y \uparrow 2 \end{aligned}$$

where '2' is an exact value, '3.1416' is inexact, Z takes integer values, and X and Y take floating-point values. The transformation $MC[\,P\,]$ then yields the following Monte Carlo program:

$$\begin{aligned} Y &:= \mathsf{round}(t_\mathsf{digit_precision}(2 \times \mathsf{round}(t_\mathsf{digit_precision}((Z + t_\mathsf{digit_precision}(3.1416)))))) \;\; ; \\ X &:= \mathsf{round}(t_\mathsf{digit_precision}(t_\mathsf{digit_precision}(Y) \uparrow 2)). \end{aligned}$$

If we wanted to indicate that '3.1416' is accurate only to 5 digits, we could extend the grammar to permit some declaration of this, such as

$$(\; 3.1416 \ldots \;)$$

and extend the transformation so that, for example,

$$MC[\,(\; 3.1416 \ldots \;)\,] \quad = \quad \mathsf{inexact}(3.1416,\; 5,\; \mathsf{randerr})$$

where randerr generates random values over $(-\frac{1}{2}, \frac{1}{2})$.

6.8.2 Handling multiple references properly

An important extension of the definition of $\mathsf{inexact}(\cdots)$ is to treat *program variables* (i.e., all references) specially. The idea is that for a program variable X that is referenced multiple times in an expression, the implementation of $\mathsf{inexact}(X, \ldots)$ should yield the same value for each such reference. This is a subtle point, and for simplicity in the presentation we deferred its discussion, presenting it now as an 'extension'.

The extension is clearly necessary for numeric expressions such as '$X - X$', or comparisons such as '$X \oslash X$', if the computed results are to match one's intuition. Assuming that the value x of X does not change, then having multiple references to X yield the same value of $\mathsf{inexact}(X, \ldots)$ give the semantics one would expect.

This extension is not that difficult to implement. By taking the name X and its stored value x into account, $t_\mathsf{digit_precision}(X)$ and $\mathsf{inexact}(X, \ldots)$ can be made to yield a consistent value for all references to X while it has that value. Essentially, the randomization discussed up to this point must be extended to a *hash function.*

Some analysts may see this extension as inessential. If the axiom $X - X = 0$ is expendable for inexact values, the extension is not needed. Thus although formally both sides of the test

$$(X \otimes X) \ominus (Y \otimes Y) \quad < \quad (X \ominus Y) \otimes (X \oplus Y)$$

denote the same value, it may or may not be significant that the test can succeed.

Another consideration with this extension is that it affects expected values. If we *do* implement the extension, then

$$\mathsf{E}[\; X \otimes X \;] \quad = \quad \mathsf{E}\Big[\; X^2 \;\Big].$$

However, if we *do not* implement it, then each reference to X is randomized independently, and

$$\mathsf{E}[\; X \otimes X \;] \quad = \quad \mathsf{E}[\,X\,] \times \mathsf{E}[\,X\,] \quad = \quad \mathsf{E}[\,X\,]^2.$$

Depending on the context, one or the other expected value may be desired. If accurate modeling of correlated error is wanted, then the extension should be implemented. If accuracy in computing with inexact values is wanted, dropping the extension may give better results.

6.8.3 Exploratory statistical roundoff error analysis

The transformation currently implements backward & forward error, as it introduces a random error on each input (backward error) and on each output (forward error). The resulting output values reflect both backward and forward errors. While this potentially doubles the measured error, it avoids error being overlooked.

With minor modifications to the transformation we can implement other kinds of statistical roundoff error analysis. Specifically, for forward-, backward-, or backward & forward roundoff analysis, we can define a transformation $MC[\,\cdot\,]$ such that each roundoff error of a program P appears as a random variable in $MC[\,P\,]$, and each worst-case error bound for P should imply a statistical error bound for $MC[\,P\,]$. Notice this result requires multiple references to a program variable X to be implemented properly by $t_\mathsf{digit_precision}(X)$ in the transformed program $MC[\,P\,]$.

It is natural to view inexact floating-point values as random variables that are sampled, and floating-point computations as Monte Carlo computations. We can equate the distinction between exact and inexact values with the distinction between non-random and random values because inexactness is 'contagious' in the same way that randomness is. Both inexactness and randomness are preserved through computation in an identical way. This view also reflects the fact that exact values are degenerate cases of the inexact values just as non-random values are degenerate cases of the random values, which underlies a basic Monte Carlo principle: replacing any estimate by an exact value in a Monte Carlo computation reduces the sampling error in the final result [50, p.54].

Monte Carlo Arithmetic seems quite close in spirit to what Metropolis had in mind throughout his career. As one of the founders of the Monte Carlo school [74] (see also [34, 49, 50]), it seems very likely that Metropolis thought of MCA-like schemes at some point.

7 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.

7.1 Variable precision is supported

Figure 5 shows the result of varying the virtual precision while computing the Tchebycheff polynomial $T_{20}(0.75)$, using the factored representation discussed on p.54. 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.

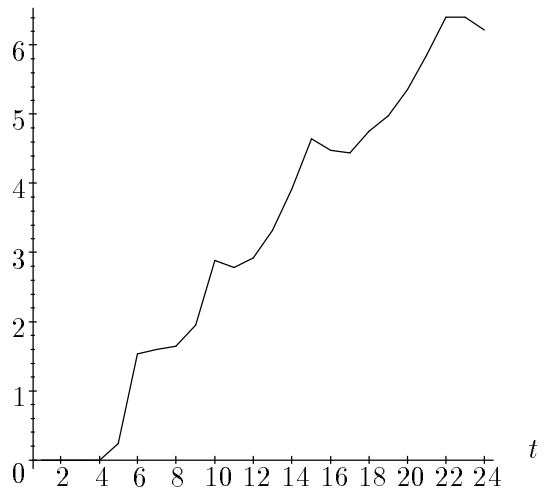


Figure 5: Number of significant decimal digits (negative base-10 log of the relative error) in the value $T_{20}(0.75)$ computed with full MCA (100 samples), at the indicated binary virtual precision t.

7.2 Roundoff errors actually do become random

Kahan [63] and others (e.g. Higham [54, §1.17,§2.6]) argue that statistical analyses of roundoff error are improperly founded because they assume roundoff errors are random. For functions like

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

which are sensitive to perturbation, they show its roundoff errors are not randomly distributed at all with a plot like that in Figure 6.

With MCA, this argument does not work. This is demonstrated with the equivalent plot, Figure 7, produced with MCA using uniform input precision bounding and IEEE default rounding. Randomization forces the roundoff errors to be random. Forsythe predicted randomization would have this effect [36], but because this problem has considerable cancellation, we have used precision bounding, and not just the random rounding proposed by Forsythe. Figure 8 also shows the

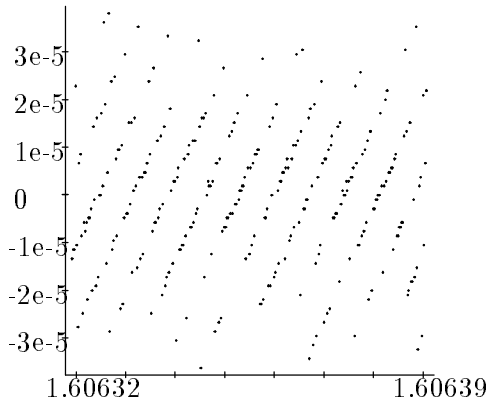


Figure 6: $rp(x) - rp(u)$, for $u = 1.60631924$ and $x = u, (u + \epsilon), \ldots, (u + 300\epsilon)$ where $\epsilon = 2^{-24}$ —— computed with single precision IEEE arithmetic.

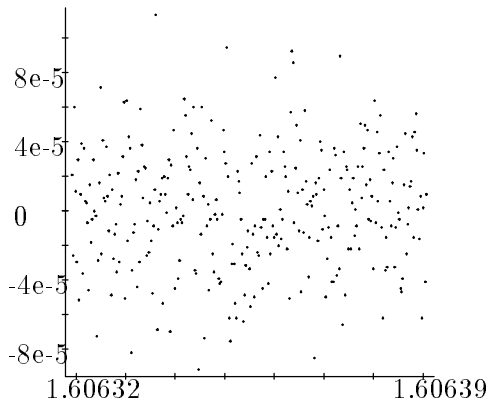


Figure 7: $rp(x) - rp(u)$, for $u = 1.60631924$ and $x = u, (u + \epsilon), \ldots, (u + 300\epsilon)$ where $\epsilon = 2^{-24}$ —— computed with single precision MCA (uniform input precision bounding, round to nearest).

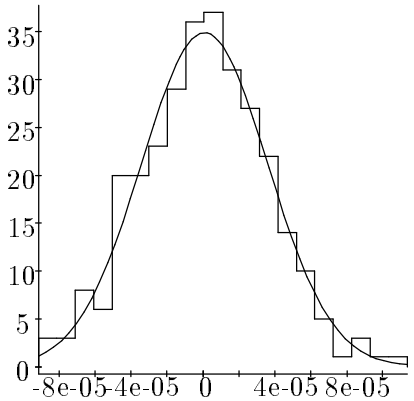


Figure 8: Histogram of values of $rp(x) - rp(u)$ computed with MCA in the previous figure. The normal density (for the average and standard deviation of these values) is superimposed.

distribution of these values. The computation of rp involves 16 arithmetic operations, and the resulting distribution is quite close to normal.

Note also that the randomization used for Figure 7 was done at the machine (single) precision, with $t = p$. If we had chosen to randomize at some level above the machine precision (i.e., carry some number $r = p - t > 0$ of random bits in every computed value), then at the price of losing these bits for representing significant digits, we make the distribution of roundoff errors as statistically ideal as we wish. In other words, in inexact values it can be worthwhile to carry a nontrivial number r of random least-significant bits.

7.3 Two-sided error analysis is supported

In Section 3.3 we discussed the 'Loss of Significance' perspective on error. Unlike forward and backward analysis alone, this perspective encourages **two-sided error analysis**: one should appreciate the sensitivity of the input to the output (backward analysis) *and* the output to the input (forward analysis). Specifically, the number of significant digits in the output is bidirectionally related to the number of significant digits in the input. It is our belief that numerical program users have an intuitive understanding of this bidirectional relationship, and rely on a two-sided error perspective.

Two-sided analysis is inherent in the notion of ill-conditioning. A condition number κ_f for a computation $y = f(x)$ usually is defined to be a quantity like

$$\kappa_f \quad = \quad \sup_{y=f(x),\, y'=f(x')} \left| \frac{y \;-\; y'}{x \;-\; x'} \right|,$$

which neatly measures the sensitivity of input to output and vice-versa.

With MCA, the sensitivity of output to input can be measured by comparing their significance. Recalling Theorem 1, we can define an estimate of the sensitivity a variable z by

$$\mathsf{sensitivity}(z) \quad = \quad (\ \widehat{\sigma}_z \ / \ \widehat{\mu}_z\ ).$$

This then suggests measuring condition as

$$\kappa_f \quad = \quad \frac{\mathsf{sensitivity}(y)}{\mathsf{sensitivity}(x)}.$$

Variations yield other information of interest, such as *covariance* of the input and output. Using standard errors instead of standard deviations can be important when the number of samples is nontrivial. Wilkinson [109, p.29] gives an interesting abstract development of condition numbers, pointing out that the problem being solved determines which aggregate condition number bounds one can establish. He gives examples such as

$$\left|\sum \delta y_j^2\right| \quad \le \quad \kappa \sqrt{\sum \delta x_i^2}, \qquad |\delta y_j| \quad \le \quad \kappa_j \sqrt{\sum \delta x_i^2}, \qquad \frac{|\delta y_j|}{\sqrt{\sum y_j^2}} \quad \le \quad \kappa_j \sqrt{\frac{\sum \delta x_i^2}{\sum x_i^2}}.$$

This observation should hold for the sensitivity statistics $\delta z = \mathsf{sensitivity}(z)$ as well, because of the similar properties satisfied by variances of aggregates.

Two-sided analysis is also inherent in statistical hypothesis testing. In statistics, most questions are reduced to statistical inequalities relating different parameters [30, §34], of a form like

$$\mathsf{Pr}[\ |\ y \;-\; y'\ |\ \ >\ \ \epsilon(y,\, y')\ ] \quad \le \quad \delta(\ |\ x \;-\; x'\ |\ )$$

where ϵ is an arbitrary threshold function and δ is a function producing a confidence level. For example, the **two-sided relative error condition** is

$$\mathsf{Pr}[\ |\ y\ -\ y'\ |\ \ >\ \ \epsilon\ y\ ]\quad \leq\quad \delta$$

where $\epsilon > 0$ and $0 < \delta < 1$ are constants; see e.g. Fishman [34, §2.5]. It would be useful if numerical libraries could guarantee error conditions like this, promising to deliver a specific output accuracy for a given input accuracy, with a confidence that can be set as high as desired.

7.4 Addition becomes statistically associative

Revisiting Knuth's example (p.16), let '?' denote a random digit resulting from precision bounding that becomes part of the floating-point result after catastrophic cancellation:

$$\begin{array}{lllllllll}
(\ 11111113.\ \oplus\ -11111111.\ )\ \oplus\ 7.5111111 & = & 2.???????\ \oplus\ 7.5111111 & = & 10.??????; \\
11111113.\ \oplus\ (\ -11111111.\ \oplus\ 7.5111111\ ) & = & 11111113.\ \oplus\ -11111103. & = & 10.??????.
\end{array}$$

If we ignore the nonsignificant random digits, addition is associative. Table 11 gives a computational demonstration, showing that standard errors decrease with increasing numbers of samples.

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

Theorem 3 *Let $T = (x_1 + x_2 + \cdots + x_m)$ where the x_i are distinct exact or inexact values, and let $\widehat{T}$ and $\widehat{\sigma}$ be average and standard deviation obtained from computing this sum n times with Monte Carlo addition with random rounding and input precision bounding in some fixed order, where t is the virtual precision. Then*

$$\mathsf{E}\Big[\ \widehat{T}\ \Big]\quad =\quad T$$

and

$$\mathsf{S}\Big[\ \widehat{T}\ \Big]\quad \leq\quad \beta^{E-t}\ \sqrt{2\,m}\ /\ \sqrt{n}$$

where E is the largest order of magnitude, over the n samples, of any x_i or computed partial sum.

Proof
In Monte Carlo addition

$$\begin{array}{rcl}
y\ \oplus\ z & = & \mathsf{round}\,(\ t_\mathsf{digit_precision}\,(\ t_\mathsf{digit_precision}\,(y)\ +\ t_\mathsf{digit_precision}\,(z)\ )\ ) \\
& = & \mathsf{round}\,\Big(\ y\ +\ \beta^{e_y-t}\,\xi_y\ +\ z\ +\ \beta^{e_z-t}\,\xi_z\ +\ \beta^{e_{yz}-t}\,\xi_\oplus\ \Big) \\
& = & y\ +\ z\ +\ (\ \varepsilon_y\ +\ \varepsilon_z\ +\ \varepsilon_\oplus\ )
\end{array}$$

where ε_y, ε_z, and $\varepsilon_{\oplus}$ are random variables with expected value zero, by Theorem 2. Thus the sum used in computing $\widehat{T}$ has form (reflecting each of the $m-1$ partial sums)

$$x_1 + x_2 + \cdots + x_m \quad + \quad (\varepsilon_{y_1} + \varepsilon_{z_1} + \varepsilon_{\oplus_1}) \quad + \quad \cdots \quad + \quad (\varepsilon_{y_{m-1}} + \varepsilon_{z_{m-1}} + \varepsilon_{\oplus_{m-1}})$$

and the expected value of each of the random variables is zero. Again by Theorem 2,

$$\mathsf{V}[\ \mathsf{round}\,(t_\mathsf{digit_precision}\,(x))\ ] \quad = \quad \mathsf{V}[\ x\ ] \quad + \quad \beta^{2(e-t)}\ \varsigma^2$$

where $\varsigma < 1$ and $e = \beta\mathrm{oom}(x)$. Notice that necessarily $e < E = \beta\mathrm{oom}(\ \tau\ )$, where τ is the largest absolute value of any x_i or partial sum produced over the n samples. Since for independent random variables $\mathsf{V}[\ \sum_i \varepsilon_i\ ] = \sum_i \mathsf{V}[\ \varepsilon_i\ ]$, we obtain the stated bound for the standard error $\mathsf{S}\left[\ \widehat{T}\ \right]$. □

Although this bound is not that tight, it is enough to show that Monte Carlo addition is statistically associative. That is, if $\widehat{T}_1$ and $\widehat{T}_2$ are two Monte Carlo estimates obtained for a sum of distinct values $T = (x_1 + x_2 + \cdots + x_m)$ by performing the operations in different fixed orders, then

$$\mathsf{E}\left[\ \widehat{T}_1\ \right] \quad = \quad \mathsf{E}\left[\ \widehat{T}_2\ \right].$$

Furthermore, although the standard errors $\mathsf{S}\left[\ \widehat{T}_1\ \right]$ and $\mathsf{S}\left[\ \widehat{T}_2\ \right]$ may differ substantially, both are bounded and can be made as small as desired by taking sufficiently many samples n.

It is very reasonable to assume that $\widehat{T}$ is normally distributed. Even for small m the distribution of a sum of m uniform distributions converges very rapidly to a normal distribution (Appendix C.3). Because the expected value of the errors in Monte Carlo addition with random rounding is zero, only widely differing orders of magnitude among the x_i and/or catastrophic cancellation can prevent $\widehat{T}$ from being normally distributed. This being the case, we can obtain confidence intervals:

Theorem 4 (statistical confidence intervals for sums)
Let $T = (x_1 + x_2 + \cdots + x_m)$ where the x_i are distinct exact or inexact values, and let $\widehat{T}$ and $\widehat{\sigma}$ be average and standard deviation obtained from computing this sum n times with Monte Carlo addition with random rounding and input precision bounding in some fixed order. Assume that $\widehat{T}$ is normally distributed. Then for each $\alpha \in (0, 1]$ there is a finite value $k(n, \alpha)$ such that with probability $1 - \alpha$,

$$\left|\ T \ - \ \widehat{T}\ \right| \quad \leq \quad k(n,\alpha)\ \ \frac{\widehat{\sigma}}{\sqrt{n}}.$$

Proof
If $\widehat{T}$ follows a normal distribution, the theorem follows from the elementary statistical inequality

$$\mathsf{Pr}\left[\ -t_{n-1}(\alpha/2) \quad \leq \quad \frac{\widehat{T} \ - \ T}{\widehat{\sigma}/\sqrt{n}} \quad \leq \quad t_{n-1}(\alpha/2)\ \right] \quad = \quad 1 - \alpha$$

where t_{n-1} is the Student t distribution with $n-1$ degrees of freedom. Defining $k(n, \alpha) = t_{n-1}(\alpha/2)$ and simplifying gives the stated result. □

Under the assumption that $\widehat{T}$ is normally distributed, the Student t statistic gives the following bounds. If $\alpha = 0.05$ and $n = 5$ then $k(n, \alpha) = t_4(0.025) = 2.776$. Reducing n to 3 gives $k(n, \alpha) = t_2(0.025) = 4.303$. For most values of n and α of interest, $k(n, \alpha)$ is between 1 and 10.

Even if we do not assume T is normally distributed, it is still possible to get bounds involving the standard error. For example, if we assume only that all random variables have uniform symmetric distributions, then we can employ the Chernoff bound (Appendix C.2) which gives only slightly larger confidence intervals than the Student t distribution. Even if we assume nothing about the distribution of $\widehat{T}$ except the bound on its variance in the previous theorem, we can still derive a confidence interval from the Tchebycheff bound (Appendix C.2).

7.5 Other floating-point anomalies are avoided statistically

A benefit of Theorem 2 is that division-free floating-point expressions involving independent variables behave, in expected value, like real expressions:

Theorem 5 *With* MCA *using random rounding, for independent random variables x, y,*

$$\begin{aligned}
\mathsf{E}[\ x \oplus y\ ] &= \mathsf{E}[\ x\ ] + \mathsf{E}[\ y\ ] \\
\mathsf{E}[\ x \ominus y\ ] &= \mathsf{E}[\ x\ ] - \mathsf{E}[\ y\ ] \\
\mathsf{E}[\ x \otimes y\ ] &= \mathsf{E}[\ x\ ] \times \mathsf{E}[\ y\ ] \\
\mathsf{E}[\ x \oslash y\ ] &= \mathsf{E}[\ x\ ] \times \mathsf{E}[\ 1 / y\ ]
\end{aligned}$$

provided that no exponent overflow or underflow occurs.

Using a relative error analysis, we can go further and get statistical confidence intervals on the standard deviations produced by MCA for specific expressions. See [81].

7.6 Open-ended statistical analysis is supported

The easiest way to make strong assertions about the accuracy of computed values is the parametric approach — first to verify that the sample distribution follows a given distribution (such as the normal distribution), and then to apply statistical tests available for that distribution, ultimately obtaining strong confidence intervals.

A sample distribution can be verified to follow a given distribution with explicit tests. For example, normality can be verified with the χ^2 **test**, which compares the histogram of the samples with the histogram expected from a normal distribution, or with a **Kolmogorov-Smirnov test**, which compares the cumulative distribution of the samples with a normal distribution [68, §3.3]. These tests are not difficult to implement, but require maintaining a histogram for each sampled variable.

A simpler test, which we implemented and seems to do fairly well, checks for large coefficients of *skewness* and *excess (kurtosis)* (cf. [30, §29.3] and/or [107, pp.234–235]). These measure higher moments of the samples, and moments can be computed incrementally (on-line) [91]. Skewness measures asymmetry of a distribution, while excess measures its flatness or peakedness. For $k > 2$, the **k-th central moment** of x is the expected value $\mu_k = \mathsf{E}\left[\ (x - \mu)^k\ \right]$, estimated by

$$m_k \quad = \quad \frac{1}{n} \sum_{i=1}^{n} (x_i - \widehat{\mu})^k .$$

Then the **coefficients of skewness** g_1 and **excess** g_2 are defined by:

$$\begin{aligned}
g_1 &= m_3\ m_2^{-3/2} & \qquad g_2 &= m_4\ m_2^{-2} - 3 \\
G_1 &= \tfrac{\sqrt{n(n-1)}}{n-2}\ g_1 & \qquad G_2 &= \tfrac{n-1}{(n-2)(n-3)}\ ((n+1)\ g_2 + 6) \\
\sigma_1^2 &= \tfrac{6\ n\ (n-1)}{(n-2)\ (n+1)\ (n+3)} & \qquad \sigma_2^2 &= \tfrac{24\ n\ (n-1)^2}{(n-3)\ (n-2)\ (n+3)\ (n+5)} .
\end{aligned}$$

The coefficients g_1 and g_2 are both asymptotically normally distributed, and have unbiased estimates G_1 and G_2 with mean zero and variances σ_1^2 and σ_2^2. So, for example, using the error function (p.25) we can check $|G_1| \leq 2.57\,\sigma_1$ and $|G_2| \leq 2.57\,\sigma_2$, to test the hypothesis that the sampled distribution is normal with confidence level 99%.

As an example, the sample distribution of the x_k sequence defined in the Overview failed this skewness/excess test. A quick look at Appendix A explains why: there are 4 fixed points of the iteration, and a sample distribution will therefore have 4 peaks.

Hull and Swenson [57] found that, for a Runge-Kutta code with probabilistic rounding, when several test differential equations were integrated the errors followed a normal distribution. Chatelin and Brunet [21] discuss the situation for eigenvalue computations with the QR algorithm and present a clear and well-reasoned set of assumptions about the distributions of errors in 'sufficiently long' numeric programs in which the output is an analytic function of a perturbation of the problem. They then experimentally verify the normality assumption using the CESTAC perturbation method, and use this in finding t-test confidence intervals for specific eigenvalues. Both Hull and Swensen [57, pp.110–111] and Chatelin and Brunet used the χ^2 method to verify normality of the distribution. Recently Chaitin-Chatelin and Frayssé [23, p.101] showed that Hull and Swensen's results hold only for regular points; normality does not hold in neighborhoods of singular points.

MCA is not committed to any assumption of normality. Sampling is an art, and the right approach to sampling can settle very tricky problems [26]. The book by Fishman [34] surveys many sampling techniques for Monte Carlo analysis. Nonparametric statistical analysis is also useful: one need not assume any particular sample distribution, but can instead develop and test hypotheses using other tools, particularly for order statistics [107]. Open-endedness is vital.

7.7 Limitations of CESTAC and CADNA are avoided

MCA generalizes and improves upon existing approaches for random rounding or random perturbation of numerical results. Specifically, in a number of ways it improves on the stochastic arithmetic of CADNA and CESTAC ([106], updated recently in [3]).[11] The CADNA environment includes:

- randomization with PER(x), which is *ad hoc* but can be implemented very efficiently on almost all machines;
- type declarations of stochastic program variables, causing 2 or 3 samples of that variable to be saved;
- stochastic arithmetic operators, one execution of which produces 2 or 3 random values from the 2 or 3 values of its operands, without the need for repeated sampling of (any segment of) the program;
- stochastic comparison operators, which perform statistical inference over the population of 2 or 3 values of their operands;
- program control based on a notion of numerical significance, essentially the logarithm of an average value divided by its standard deviation.

Specific differences between MCA from CADNA include:

- MCA is not committed to a statistical inference model. CADNA rests on the assumptions that the values it computes have normal distributions, that the means of these distributions

[11] Actually, CADNA and MCA have different goals. CADNA is intended as a programming environment in which programmers can take advantage of predefined stochastic arithmetic operations and statistical inference mechanisms defined on stochastic data types. MCA, on the other hand, is not committed to the nature of the environment. It is intended as a new mode for floating-point arithmetic, with which existing programs will still work (without change). It allows statistical inference (interpretation of the sampling results) also, but the nature of this inference is left open. Ideally, MCA should be a substrate on which systems such as CADNA can be built.

are the true values, that 2 or 3 samples are sufficient, and that the resulting Student t values give meaningful confidence intervals that can be used in comparison operations ("hypotheses which generally hold in real problems" [3, p.131]).

Chatelin gives a critical review of these assumptions in [20], and reworks them into rigorous conditions under which the results of CESTAC will be right 95% of the time. (Under assumptions that Chatelin disputes, taking only 2 or 3 samples gives a confidence level of 95%. It is illuminating to think of CESTAC as being incorrect about 5% of the time.) These criticisms led to refinements by Vignes in [24] and [106, pp.241–242]. Unfortunately the paper [24], given as a formal basis for these assumptions, makes further assumptions and dismisses the possibility of exception to them in a cavalier way as "pratiquement jamais observé" (translated as "quasi never satisfied"). Kahan [63] exhibits problems for which these assumptions are invalid, and on which a CESTAC-based program named ProSolveur gives incorrect answers.

- MCA has precision bounding t_digit_precision(x), while CADNA and CESTAC have $\texttt{PER}(x)$, a random perturbation by ± 1 in the least significant bit of x. First, $\texttt{PER}(x)$ is clearly less likely to satisfy theoretical properties. In particular, it has no beneficial effects on bias (expected rounding error), while Theorem 2 shows how precision bounding gives zero expected bias.

  Second, with a nontrivial probability CADNA will overstate the number of significant digits in computed values. For example, with probability 20/64, computing $x \ominus y$ as $\texttt{PER}(x - y)$ three times will yield the average $(x - y)$ with zero average perturbation. In other words, even with three computations, about 1/3 of the time the random perturbation has no effect. Furthermore, if $x = y + \varepsilon$, and $x \ominus y$ is computed as $\texttt{PER}(x - y) = \texttt{PER}(\varepsilon)$, then the catastrophic cancellation will always slip by unnoticed. The probabilities of these occurrences in MCA are zero if t_digit_precision(x) is implemented correctly.

- In MCA, we recommend defining comparisons in terms of the underlying arithmetic operations, as programmers often assume this [43]:

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

  In CADNA, comparisons are based on statistics [3, p.133]:

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

  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) [95].

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

  Chatelin and Brunet warn [21] 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 lacks. 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.

- As the inequality above suggests, CADNA treats values whose standard deviation exceeds their computed mean as 'zero' [104]. In a sense, statistical inference is tied to the number zero itself, and this inference rests on assumptions about the distribution of errors that may not hold (e.g., as they are proved not to hold for eigenvalue computations in [21, Theorem 8.2; cf. also p.158]). The only way that MCA treats zero specially is that it does not have an order of magnitude.

Summarizing, CADNA is a compromise between theory and practice. Its strength is its simplicity, its demonstrated results, and its two decades of experience in solving real problems. The basic idea on which it and CESTAC are founded is very powerful, and Vignes should be credited as its pioneer. Its weakness is its concern more with computational efficiency than with theoretical soundness, and its unnecessary investment in questionable assumptions that inspire doubt about the validity of its results.

By contrast, MCA is infrastructure, aimed at encouraging the experimental outlook of Monte Carlo computation in numerical computing, and at extending standard hardware floating-point implementations to support this outlook. It makes a point of avoiding statistical interpretation of its results, leaving this interpretation to the user, or to statistical analysis software. Its weakness is in its lack of experience in solving real problems. Its strength lies in its flexibility, and its theoretical and practical advantages discussed in this section.

8 Example Applications of Monte Carlo Arithmetic

Several examples illustrate what MCA can do. A large number of basic examples have been developed by Vignes and his coworkers: rank computations and linear system solution [73], polynomial root finding [2], finding eigenvalues [21], solving ODEs [3], etc. The survey papers by Vignes (e.g., [100, 103, 105, 106]) have many references with both new and classical numerical analysis examples that show interesting results with randomization, and should be consulted for perspective. Here we restrict ourselves to a few sketchy examples that try to illustrate the points made above.

8.1 Basic numerical analysis

8.1.1 Cosines

A classic problem is to evaluate the cosine function

$$\cos(z) \quad = \quad 1 \; - \; \frac{1}{2!}\, z^2 \; + \; \frac{1}{4!}\, z^4 \; - \; \frac{1}{6!}\, z^6 \; + \; \frac{1}{8!}\, z^8 \; - \; \frac{1}{10!}\, z^{10} \; + \; \ldots$$

Evaluating cosines accurately is important; a large number of papers continue to be published every year on roundoff error in Fourier transforms and filtering applications. Although there are better methods than just summing terms in a power series, it is instructive and we start with this.

Table 12 reports the results of summing this series using full MCA (averaged over 4 samples). MCA gives results comparable to those with IEEE arithmetic (here using single precision), but gives information about their significance. The results are very good, perhaps because of the way the powers z^k are accumulated in the evaluation of the series, which randomized each instance of z independently. Another way that produces greater correlated rounding error could easily produce worse results.

z:	0	$\pi/6$	$\pi/3$	$\pi/2$
C Library $\cos(z)$	1.000000000000	0.866025403784	0.500000000000	0.000000000000
IEEE double $\cos(z)$	1.000000000000 (2)	0.866025403784 (9)	0.500000000000 (11)	0.000000000000 (19)
IEEE single $\cos(z)$	1.000000 (2)	0.866025 (6)	0.500000 (7)	-0.00000006 (11)
MCA single avg. $\cos(z)$	1.000000 (2)	0.866025 (7)	0.500000 (7)	-0.000000004 (12)
MCA single standard error	0.00000001	0.00000004	0.000000007	0.00000004

Table 12: Values of $\cos(z)$ computed from its power series using single precision full MCA; numbers in parentheses give the number of power series terms needed for convergence (4 samples).

8.1.2 Tchebycheff polynomials

For another example, consider the Tchebycheff polynomial studied by Wilkinson

$$\begin{aligned} T_{20}(z) \quad &= \quad \cos(20 \; \cos^{-1}(z)) \\ &= \quad 524288\, z^{20} - 2621440\, z^{18} + 5570560\, z^{16} - 6553600\, z^{14} + 4659200\, z^{12} \\ &\quad -2050048\, z^{10} + 549120\, z^8 - 84480\, z^6 + 6600\, z^4 - 200\, z^2 + 1 \end{aligned}$$

Wilkinson points out [109, pp.46–47] that the zeroes of this polynomial are at $\cos((2k+1)\pi/40)$ for $k = 0, 1, \ldots, 19$, and that the polynomial is moderately ill-conditioned at the roots near 1, but is otherwise well-conditioned. Table 13 reports the results of computing the Horner series using single precision full MCA (averaged over 4 samples), and demonstrates the ill-conditioning: in the final column, the results for MCA are significantly worse than those for single precision. This reflects

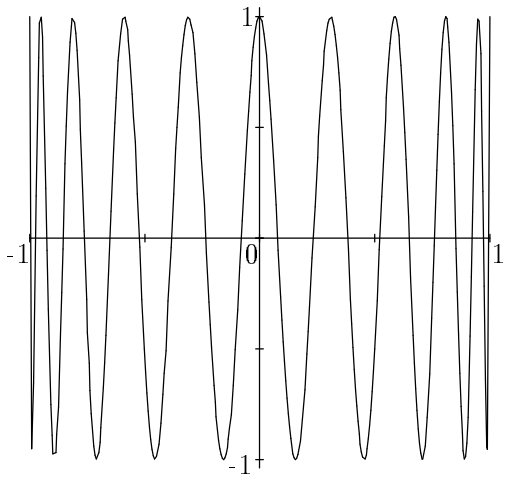


Figure 9: $T_{20}(z)$

the catastrophic cancellation among the coefficients of $T_{20}(z)$ when $z = 1$. The single precision computation here has been 'lucky', since all the coefficients are all representable exactly within IEEE 24-bit single precision (just: the binary representation of 6553600 is 23 digits long). If IEEE single precision provided 2 fewer bits of precision, the results above would be quite different. Figure 10 shows the increasing ill-conditioning as z nears 1.

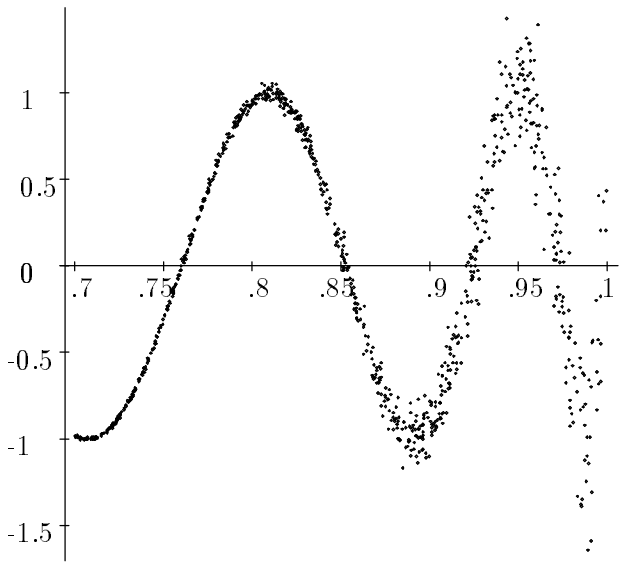


Figure 10: results with MCA at random points, showing instability of the Horner $T_{20}(z)$ computation.

Several aspects of this example are interesting. First, in the results for MCA above the constant coefficients in the definition of $T_{20}(z)$ were treated as exact values, since they all are representable exactly within IEEE 24-bit single precision, and were *not* randomly rounded to floating-point values. If these constants *had* been randomly rounded initially, the errors reported would have been uniformly worse (not much worse, but worse). The same was true for the constants 2 and 4 in the quadratic equation example discussed earlier. Exact values need not be randomized. Currently, our implementation does not implement the distinction between exact and inexact values; this distinction is hand-coded.

z:	0	1/3	2/3	1
C Library: `cos(20*arccos(z))`	1.000000000000000	0.871004566880877	-0.441604476192560	1.000000000000000
IEEE double Horner $T_{20}(z)$	1.000000000000000	0.871004566880885	-0.441604476198411	1.000000000000000
IEEE single Horner $T_{20}(z)$	1.000000	0.871006	-0.443183	1.000000
MCA single avg. Horner $T_{20}(z)$	1.000000	0.870998	-0.441314	0.778383
MCA standard error	0.00000004	0.00000447	0.00180201	0.126928

Table 13: Values for $T_{20}(z)$ computed with its Horner series expansion (4 samples).

Second, while the Horner series used above

$$T_{20}(z) = (((((((((524288\,z^2 - 2621440)\,z^2 + 5570560)\,z^2 - 6553600)\,z^2 + 4659200)\,z^2 \\ -2050048)\,z^2 + 549120)\,z^2 - 84480)\,z^2 + 6600)\,z^2 - 200)\,z^2 + 1$$

is efficient in terms of the number of floating-point operations required, for $T_{20}(z)$ it actually leads to much greater roundoff error than is obtained by using the factored representation

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

To see the improvement, compare Table 14 with Table 13.

z:	0	1/3	2/3	1
C Library: `cos(20*arccos(z))`	1.000000000000000	0.871004566880877	-0.441604476192560	1.000000000000000
IEEE double factored $T_{20}(z)$	1.000000000000000	0.871004566880876	-0.441604476192562	1.000000000000000
IEEE single factored $T_{20}(z)$	1.000000	0.871004	-0.441604	1.000000
MCA single avg. factored $T_{20}(z)$	1.000000	0.871006	-0.441603	0.999985
MCA standard error	0.00000002	0.00000007	0.00000075	0.00001145

Table 14: Values for $T_{20}(z)$ computed with its factored series expansion (4 samples).

8.1.3 Horrific examples used in the literature

Most horrific examples of roundoff error result from catastrophic cancellation. For example, the problem suite of Kulisch and Miranker [72] produces severe cancellation errors. Kahan [63, p.23] also gives the problem of computing the ratio of areas of two narrow triangles

$$r = \sqrt{\frac{(x+y+z)\cdot(z-(x-y))\cdot(z+(x-y))\cdot(x+(y-z))}{(x+y+2z)\cdot(2z-(x-y))\cdot(2z+(x-y))\cdot(x+(y-2z))}}$$

where $x = 1234567$, $y = 1234567$, and $z = 1.043e-8$. IEEE double precision floating-point gets the value 0.5. This result is 'correct' only assuming x and y are exact, so that the catastrophic cancellation $(x-y)$ 'correctly' yields 0. MCA gets this result also when x and y are exact, or when input precision bounding is not used. Any inexactness in x or y after the seventh digit would invalidate this correctness, and would be detected by input precision bounding.[12] Furthermore,

[12] Kahan shows that ProSolveur (i.e., CESTAC) gets poor results for this problem. Apparently ProSolveur computes $(x-y)$ as PER$(x-y)$, where sometimes PER$(0) \neq 0$. If instead PER$(0) = 0$ holds always (like t_digit_precision(0) in MCA), the problem disappears. When Kahan changes the input values a bit, ProSolveur complains about attempting to compute the square root of a negative number. Kahan derides this as ignorance of a basic theorem of floating-point arithmetic (that the result of catastrophic cancellation is 'exact', e.g., [54, p.12]). However, this theorem again rests on the assumption that the arguments of the cancellation are 'exact', and Kahan's argument depends crucially on assumptions about error in catastrophic cancellation.

when x and y are not identical, floating-point arithmetic gives horrific results: if values of x, y, and z had been used that are not exactly representable within IEEE's precision, like $x = 1.2345670000000000 02e18$, $y = 1.2345670000000000 01e18$, $z = 1.043$, then IEEE arithmetic gives the completely incorrect result 0.5 (the correct value is 0.16190323914071253796 28...), while MCA detects its results have no significant digits by producing a standard deviation larger than its sample average.

Another example often used in the interval analysis literature is Rump's problem of evaluating

$$f(x,y) \quad = \quad 333.75y^6 \quad + \quad x^2\left(11\,x^2y^2 - y^6 - 121\,y^4 - 2\right) \quad + \quad 5.5y^8 \quad + \quad \frac{x}{2\,y}$$

at $(x,y) = (77617, 33096)$ [88]. The results at various precisions are terrifying: while the correct answer is about -0.8273960599, IEEE single and double precision obtain the values `-6.33825e+29` and `-1.18059162071741e+21`, respectively. At the indicated point it happens that $x^2 = 5.5y^2 + 1$, giving the tremendous cancellation

$$333.75y^6 \quad + \quad x^2\left(11\,x^2y^2 - y^6 - 121\,y^4 - 2\right) \quad + \quad 5.5y^8 \quad = \quad -2.$$

Since the largest term in this expression is $5.5\,y^8 = 7.9 \times 10^{36}$, computing with fewer than 37 digits produces inaccurate results. Also, because the terms differ by many orders of magnitude, the results are highly sensitive to the machine precision. With full MCA we obtained an average value of `8.65602e+29` over 5 samples, with standard error `2.23405e+29`, detecting the cancellation and complete loss of accuracy.

8.2 Gaussian elimination

Gaussian elimination is a basic procedure for matrix decomposition and linear systems solution. It is the workhorse commonly used to solve the important special case of the least squares problem where the system is not overdetermined. In its basic form, Gaussian elimination performs a sequence of transformations to an $n \times n$ square matrix $A = (a_{ij})$, reducing it to upper-diagonal form in n steps. It can be defined equationally, with the initial assignment $a_{ij}^{(1)} = a_{ij}$ and the recursive definition of $a_{ij}^{(k+1)}$ for $1 \le k \le n-1$:

$$a_{ij}^{(k+1)} \quad = \quad \begin{cases} 0 & i \ge k+1,\ j = k \\ a_{ij}^{(k)} \quad - \quad \left(a_{ik}^{(k)} \,/\, a_{kk}^{(k)}\right)\ a_{kj}^{(k)} & i \ge k+1,\ j \ge k+1 \\ a_{ij}^{(k)} & \text{otherwise.} \end{cases}$$

Often this definition is viewed as summarizing a program – an implementation of Gaussian elimination. Gaussian elimination is typically implemented with **for**–loops in a program requiring $n(n^2-1)/3$ assignments altogether:

```
for k = 1 to n − 1 do
    for i = k + 1 to n do
        begin
            m_ik := a_ik / a_kk;                    { a_ik^(k) / a_kk^(k) }
            for j = k + 1 to n do
                a_ij := a_ij − m_ik · a_kj          { a_ij^(k) − a_ik^(k) / a_kk^(k) · a_kj^(k) }
        end
```

Starting with $A^{(1)} = A$, each iteration of the outer loop computes $A^{(k+1)} = (a_{ij}^{(k+1)})$ for $1 \leq k \leq n-1$, except that as written above the program does not zero the elements of A below the diagonal in the first k columns. (These elements are in precisely the same positions as the multipliers m_{ik}, and so many implementations store m_{ik} in a_{ik}.) The program forms the triangular matrices

$$L = \begin{pmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} a_{11}^{(n)} & a_{12}^{(n)} & a_{13}^{(n)} & \cdots & a_{1n}^{(n)} \\ & a_{22}^{(n)} & a_{23}^{(n)} & \cdots & a_{2n}^{(n)} \\ & & a_{33}^{(n)} & \cdots & a_{3n}^{(n)} \\ & & & \ddots & \vdots \\ & & & & a_{nn}^{(n)} \end{pmatrix}$$

such that $A = LU$. The computation reduces a general linear problem $A\mathbf{x} = \mathbf{b}$ to a pair of back-substitution problems $L\mathbf{y} = \mathbf{b}$, $U\mathbf{x} = \mathbf{y}$. For a good introduction see [39]; [59] gives excellent perspective on roundoff analyses for this problem.

Statistical analyses have been performed for Gaussian elimination by various authors, including Goldstine and von Neumann [44] (nice distillations of which are [111] and [54, p.187]), Barlow and Bareiss [8], Trefethen and Schreiber [97], Yeung and Chan [114], and Chaitin-Chatelin and Frayssé [23]. The statistical analysis of Barlow and Bareiss [8] is reproduced in Appendix D.

8.2.1 Small examples

Let us begin with some nice examples of Kahan. These examples show that, at least for small linear systems, MCA can detect ill-conditioning.

For the linear system $A\mathbf{x} = \mathbf{b}$ given by [59, p.772]

$$\begin{pmatrix} 0.2161 & 0.1441 \\ 1.2969 & 0.8648 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 0.1440 \\ 0.8642 \end{pmatrix}$$

the vector

$$\hat{\mathbf{x}} = \begin{pmatrix} 0.9911 \\ -0.4870 \end{pmatrix}$$

has the smallest nonzero residual $A\hat{\mathbf{x}} - \mathbf{b}$ with four-digit floating-point arithmetic, yet contains no significant digits! The exact solution is

$$\mathbf{x} = \begin{pmatrix} 2 \\ -2 \end{pmatrix}.$$

The problem here is that the matrix A is ill-conditioned, having the extremely large condition number[13] $\kappa_2(A) \approx 2.5 \times 10^8$. This is large enough to render inaccurate *all* of the digits in single precision, and half the digits in double precision. When Gaussian elimination with MCA single precision (with or without pivoting), each entry of the computed solution $\hat{\mathbf{x}}$ is found to have a standard deviation larger than the entry itself. In other words, MCA detects the ill-conditioning by finding the computed solution entries not to be significant.

A similar story holds for the linear system $A\mathbf{x} = \mathbf{b}$ defined by [59, p.788]

$$\begin{pmatrix} 0.8647 & 0.5766 \\ 0.4322 & 0.2882 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 0.2885 \\ 0.1442 \end{pmatrix}.$$

[13] Here $\kappa_2(M) = \|M\|_2 \, \|M^{-1}\|_2$, the condition number of M using the spectral norm $\|\cdot\|_2$.

with condition $\kappa_2(A) \approx 6.75 \times 10^7$. Kahan notes [59, p.788] the exact solution is

$$\mathbf{x} \quad = \quad \begin{pmatrix} -1 \\ 2 \end{pmatrix}.$$

Kahan stresses that the only clue of ill-conditioning is the cancellation in the first Gauss transformation, but single precision full MCA (with 5 samples) detects the ill-conditioning by again finding the computed solution entries not to be significant.

Finally, if the value of a_{22} is changed to be 0.2822 (as in the typo in [59, p.787]), MCA obtains the solution

$$\text{average}(\hat{\mathbf{x}}) \quad = \quad \begin{pmatrix} 0.333648 \\ -0.0000101273 \end{pmatrix}, \qquad \text{standard error}(\hat{\mathbf{x}}) \quad = \quad \begin{pmatrix} 0.00000104104 \\ 0.00000157369 \end{pmatrix}$$

with first entry correct to five significant digits, and the second entry correctly detected not to be significant (the correct value is about -7.7×10^{-6}).

8.2.2 The Turing-Wilkinson matrix

Now, let us consider some classic examples of linear systems, that appear in virtually every treatment of Gaussian elimination. In [98, p.307], Turing remarks that the matrices defined by

$$T_n \quad = \quad \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 0 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix}, \qquad T_n^{-1} \quad = \quad \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 2 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 2^{n-3} & 2^{n-4} & 2^{n-5} & \cdots & 1 & 0 \\ 2^{n-2} & 2^{n-3} & 2^{n-4} & \cdots & 1 & 1 \end{pmatrix}$$

have high condition number. It is plain from the definition here that $\kappa_1(T_n) = n\,2^{n-1}$ where $\kappa_1(M) = \|M\|_1 \, \|M^{-1}\|_1$ is the condition number of M using the column-sum norm $\|\cdot\|_1$. Turing's colleague Wilkinson [110, p.212] later adapted T_n to give a matrix W_n with worst-case error in LU decomposition with partial pivoting, where $L = T_n$:

$$W_n \quad = \quad \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix} \quad = \quad T_n \quad \cdot \quad \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & 0 & 2 \\ 0 & 0 & 1 & \cdots & 0 & 4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 2^{n-1} \\ 0 & 0 & 0 & \cdots & 0 & 2^n \end{pmatrix}$$

We applied Gaussian elimination to solve $A\mathbf{x} = \mathbf{b}$ where $A = T_n$ and $\mathbf{b} = \langle 1, 1, 1, \ldots, 1 \rangle$, which has correct solution $\mathbf{x} = \langle 1, 2, 4, \ldots, 2^{n-1} \rangle$. With $n = 24$ and partial pivoting, single precision full MCA (with 5 samples) produced a solution $\hat{\mathbf{x}}$ whose entries were all 100 times larger than their standard deviations, and were in fact correct to 2 decimal digits. With no pivoting, on the other hand, the solution $\hat{\mathbf{x}}$ had entries that were all 10^6 times larger than their standard deviations (roughly the machine precision), and were in fact correct to 6 digits.

8.2.3 The Hilbert matrix

Finally, let us define the $n \times n$ **floating-point Hilbert matrix** $A = (a_{ij})$ by

$$a_{ij} \quad = \quad fl\left[\frac{1}{i + j - 1} \right].$$

The exact result of Gaussian elimination without pivoting, on the 8-digit 8×8 floating-point Hilbert matrix A, is (rounded to 8 digits on output):

$$A^{(8)} = \begin{pmatrix} 1.0000000 & 0.5000000 & 0.3333333 & 0.2500000 & 0.2000000 & 0.1666667 & 0.1428571 & 0.1250000 \\ 0.5000000 & 0.0833333 & 0.0833333 & 0.0750000 & 0.0666667 & 0.0595238 & 0.0535714 & 0.0486111 \\ 0.3333333 & 1.0000001 & 0.0055556 & 0.0083333 & 0.0095238 & 0.0099206 & 0.0099206 & 0.0097222 \\ 0.2500000 & 0.9000000 & 1.5000011 & 0.0003571 & 0.0007143 & 0.0009921 & 0.0011905 & 0.0013257 \\ 0.2000000 & 0.8000001 & 1.7142854 & 2.0000760 & 0.0000226 & 0.0000566 & 0.0000927 & 0.0001262 \\ 0.1666667 & 0.7142857 & 1.7857158 & 2.7778576 & 2.5021274 & 0.0000014 & 0.0000042 & 0.0000079 \\ 0.1428571 & 0.6428572 & 1.7857149 & 3.3334487 & 4.0949531 & 3.0387345 & 0.0000000 & 0.0000002 \\ 0.1250000 & 0.5833333 & 1.7500012 & 3.7122455 & 5.5743811 & 5.7543651 & 4.4617154 & 0.0000000 \end{pmatrix}.$$

Of course, floating-point implementation of the program above finds approximations $\widehat{a}^{(k)}$ to $a^{(k)}$, and computes approximations $\widehat{L}$ and $\widehat{U}$ to L and U. Although the Hilbert matrix is positive-definite, it is spectacularly ill-conditioned. The ill-conditioning is such that when Gaussian elimination is applied to the 8×8 matrix A with single precision computation, all significant digits are eroded from some entries of the matrix. In [83] we discuss this erosion further.

Below is the result, over 5 samples, of applying Gaussian elimination without pivoting to the 8×8 matrix A, using single precision full MCA to compute the average of $\widehat{A}^{(8)}$:

$$\begin{pmatrix} 1.00000 & 5.00000\text{e}{-01} & 3.33333\text{e}{-01} & 2.50000\text{e}{-01} & 2.00000\text{e}{-01} & 1.66667\text{e}{-01} & 1.42857\text{e}{-01} & 1.25000\text{e}{-01} \\ .500000 & 8.33333\text{e}{-02} & 8.33333\text{e}{-02} & 7.50000\text{e}{-02} & 6.66667\text{e}{-02} & 5.95238\text{e}{-02} & 5.35714\text{e}{-02} & 4.86111\text{e}{-02} \\ .333333 & 1.00000 & 5.55556\text{e}{-03} & 8.33332\text{e}{-03} & 9.52380\text{e}{-03} & 9.92062\text{e}{-03} & 9.92062\text{e}{-03} & 9.72221\text{e}{-03} \\ .250000 & .900000 & 1.50000 & 3.57154\text{e}{-04} & 7.14274\text{e}{-04} & 9.92060\text{e}{-04} & 1.19047\text{e}{-03} & 1.32575\text{e}{-03} \\ .200000 & .800000 & 1.71429 & 2.00002 & 2.27051\text{e}{-05} & 5.67126\text{e}{-05} & 9.27830\text{e}{-05} & 1.26279\text{e}{-04} \\ .166667 & .714286 & 1.78571 & 2.77780 & 2.49892 & 1.46552\text{e}{-06} & 4.37966\text{e}{-06} & 8.21931\text{e}{-06} \\ .142857 & .642857 & 1.78571 & 3.33332 & 4.08867 & 2.98893 & 8.93965\text{e}{-08} & 3.22636\text{e}{-07} \\ .125000 & .583333 & 1.75000 & 3.71210 & 5.56428 & 5.63757 & -8.37310 & 2.85102\text{e}{-06} \end{pmatrix}.$$

The exponential error growth present in the lower triangle results from catastrophic cancellation in the expressions $(a_{ij} - m_{ik} \cdot a_{kj})$, which is tracked by input precision bounding. For each of the matrix entries the standard error value in $\widehat{A}^{(8)}$ is shown below:

$$\begin{pmatrix} 1.19209\text{e}{-08} & 5.96046\text{e}{-09} & 5.96046\text{e}{-09} & 3.65002\text{e}{-09} & 0.00000\text{e}{+00} & 2.98023\text{e}{-09} & 3.65002\text{e}{-09} & 0.00000\text{e}{+00} \\ 1.19209\text{e}{-08} & 1.34524\text{e}{-08} & 1.12007\text{e}{-08} & 4.34440\text{e}{-09} & 5.57550\text{e}{-09} & 1.82501\text{e}{-09} & 3.61180\text{e}{-09} & 2.52661\text{e}{-09} \\ 7.30005\text{e}{-09} & 1.02548\text{e}{-07} & 2.07471\text{e}{-08} & 1.39170\text{e}{-08} & 1.13362\text{e}{-08} & 8.50211\text{e}{-09} & 6.38754\text{e}{-09} & 7.20798\text{e}{-09} \\ 2.98023\text{e}{-09} & 1.62798\text{e}{-07} & 3.97570\text{e}{-06} & 3.08420\text{e}{-08} & 2.88167\text{e}{-08} & 2.77717\text{e}{-08} & 3.66413\text{e}{-08} & 3.31846\text{e}{-08} \\ 4.71216\text{e}{-09} & 1.22733\text{e}{-07} & 4.10312\text{e}{-06} & 9.15819\text{e}{-05} & 4.34014\text{e}{-08} & 7.52915\text{e}{-08} & 8.51134\text{e}{-08} & 9.90333\text{e}{-08} \\ 3.65002\text{e}{-09} & 1.32478\text{e}{-07} & 3.88282\text{e}{-06} & 1.61856\text{e}{-04} & 1.57049\text{e}{-03} & 3.98876\text{e}{-08} & 8.13126\text{e}{-08} & 1.33117\text{e}{-07} \\ 5.57550\text{e}{-09} & 1.28669\text{e}{-07} & 5.06557\text{e}{-06} & 1.99351\text{e}{-04} & 3.76214\text{e}{-03} & 3.23460\text{e}{-02} & 8.25043\text{e}{-08} & 1.53945\text{e}{-07} \\ 2.98023\text{e}{-09} & 7.99680\text{e}{-08} & 4.64846\text{e}{-06} & 2.34731\text{e}{-04} & 6.12322\text{e}{-03} & 7.66732\text{e}{-02} & 1.31922\text{e}{+01} & 2.99854\text{e}{-06} \end{pmatrix}.$$

This example shows not only the inaccuracy of the computation, but also that a small number of samples suffice for standard errors to estimate the number of significant digits in entries of $\widehat{A}^{(8)}$.

If Gaussian elimination with partial pivoting is used here, the effect of partial pivoting is to reduce the standard deviations somewhat (some by an order of magnitude), but the computation still ultimately loses all significant digits in some entries of $\widehat{A}^{(n)}$.

8.2.4 Perspective on error bounds

Wilkinson established the following well-known backward error bound for partial pivoting:

Theorem 6 *(Wilkinson [109])*
Suppose A is an $n \times n$ nonsingular matrix. Then the matrices $\widehat{L}$ and $\widehat{U}$ computed by Gaussian elimination with pivoting in floating-point with unit roundoff $\mathbf{u}$ *satisfy $\widehat{L}\widehat{U} = A + E$, where E is a matrix of roundoff errors such that, if $\widehat{a}_{ij}^{(k)}$ is the computed floating-point approximation to $a_{ij}^{(k)}$,*

$$\| E \|_\infty \quad \leq \quad 2\, n^2 \quad \mathbf{u} \quad \frac{\max_{i,j,k} \left| \widehat{a}_{ij}^{(k)} \right|}{\max_{i,j} \left| \widehat{a}_{ij} \right|} \quad \| A \|_\infty$$

where the matrix infinity-norm is defined by $\| E \|_\infty = \max_{i=1}^{n} \sum_{j=1}^{n} |e_{ij}|$.

Because the **growth factor** $\left(\max_{i,j,k} \left|\widehat{a}_{ij}^{(k)}\right|\right)/(\max_{i,j} |a_{ij}|)$ rarely exceeds 10, Gaussian elimination with partial pivoting has long been acknowledged as "stable in practice", albeit unstable in theory. (The matrix W_n studied above has growth factor 2^{n-1}, which is worst possible.)

By Theorem 6 Gaussian elimination has outstandingly good backward error bounds for the Hilbert matrix, since its growth factor is 1 for this matrix. When the matrix product $\widehat{L}\widehat{U}$ is computed with MCA, this theorem is borne out: the results are all correct to full precision, and the magnitudes of the standard deviations produced are considerably *smaller* than the worst-case error bounds stated in the theorem (the largest was 6×10^{-8} for the $[1,1]$-entry).

This example underscores the difference between the loss of significance and backward error perspectives. $\widehat{A}^{(k)}$ diverges completely from $A^{(k)}$ for the Hilbert matrix, yet the product $\widehat{L}\widehat{U}$ will be very close to A.

A modern error bound for the solution of a linear system by Gaussian elimination is as follows:

Theorem 7 *([47, p.106])*
Suppose A is an $n \times n$ nonsingular matrix, and $\mathbf{b}$ is a nonzero vector of length n, and the matrices $\widehat{L}$ and $\widehat{U}$ computed by Gaussian elimination in floating-point with unit roundoff $\mathbf{u}$ are used to solve the linear system $A\mathbf{x} = \mathbf{b}$ with the back-substitution computations $\widehat{L}\widehat{\mathbf{y}} = \mathbf{b}$, $\widehat{U}\widehat{\mathbf{x}} = \widehat{\mathbf{y}}$. Then $(A + E)\widehat{\mathbf{x}} = \mathbf{b}$ where E is a matrix of roundoff errors bounded by

$$|E| \quad \leq \quad n \quad \left(3\,|A| \;+\; 5\,\left|\widehat{L}\right|\,\left|\widehat{U}\right|\right)\mathbf{u} \quad + \quad O(\mathbf{u}^2)$$

where $|M|$ is the matrix whose entries are the absolute values of the corresponding entries in M.

This is an excellent backward error bound. For example, with the exact Hilbert matrix, the exact vector $\mathbf{b} = (1/1, 1/2, 1/3, \ldots, 1/n)$ has exact solution $\mathbf{x} = (1, 0, 0, \ldots, 0)$. However, in single precision MCA, using Gaussian elimination without pivoting we get the following results from 5 samples:

$$\text{average}(\widehat{\mathbf{x}}) \quad = \quad \begin{pmatrix} 0.999548 \\ 0.00198615 \\ 0.144277 \\ -1.37692 \\ 4.81751 \\ -7.95838 \\ 6.27661 \\ -1.90534 \end{pmatrix}, \qquad \text{standard error}(\widehat{\mathbf{x}}) \quad = \quad \begin{pmatrix} 0.00049622 \\ 0.0192774 \\ 0.289198 \\ 1.82883 \\ 5.48997 \\ 8.37467 \\ 6.28928 \\ 1.84686 \end{pmatrix}.$$

Despite the bound of Theorem 7, the standard errors show that the computed solution $\widehat{\mathbf{x}}$ is good only in the first entry, and then only to about three digits. Even though it is the exact solution of a nearby problem, this solution is *terrible* as far as the number of significant digits in its entries. That is, from the backward error perspective it is a good solution, while in the significance loss perspective it is a terrible solution.

This is not a result of failure to use pivoting, either. If we use complete pivoting instead, we reduce the standard deviations, but not by enough to save the significance of the results. Nor is this a result of choosing a problematic value for $\mathbf{b}$. On the contrary, it is easy to produce examples with worse behavior. This example has the benefit of having a simple exact solution.

Of course, none of this would have surprised Wilkinson; for an eloquent discussion with great historical perspective see [109, p.118] and his John von Neumann lecture [111]. Gaussian elimination is just a very compelling example that has persuaded many numerical analysts to adopt the backward perspective, giving a way to account for its success in practice.

We argue that statistical analysis gives a better way to appreciate Gaussian elimination. Standard errors of the results measure ill-conditioning of the problem and instability of the computation,

and can be computed relatively inexpensively. Backward error bounds, good as they may be, do not guarantee that computed results will have any significant digits. The Monte Carlo approach allows an exploratory significance analysis of the problem, and (like floating-point arithmetic and unlike interval analysis) it gives results that are consistent with the theorems above.

The empirical attitude is very important. It pays to question the quality of numerical results. Recently a number of researchers have begun to question the stability of Gaussian elimination with partial pivoting in practice. For example, S.J. Wright [112] finds exponential error growth for a specific family of matrices arising in two-point boundary value problems. Although Gaussian elimination is of great importance, there remain a number of mysteries about the quality of its results [96]. Many of these revolve around its average- and worst-case roundoff error behavior, which is more amenable to direct computation than to formal analysis [83, 97]. Despite textbook assurances to the effect that partial pivoting is almost always adequate, it is not expensive to perform a statistical analysis in order to evaluate the quality of Gaussian elimination's results. This analysis is no more complicated than the *a posteriori* computation needed to determine the growth factor in the backward error bounds.

8.3 Differential equations

It is fitting to conclude with ordinary differential equations for several reasons. First, numerical integration was the application that helped inspire Simpson with the idea of using a mean instead of a single observation in order to increase accuracy. Second, accurate solution of ordinary differential equations sparked the initial interest in statistical computations of Rademacher, Huskey, Forsythe, Henrici, Hull and Swensen, and others. Third, one of the original uses of Monte Carlo methods was in the integration of differential equations. Whenever a problem can be phrased in terms of random walks or Markov chains, Monte Carlo techniques are sophisticated tools for finding qualitative solutions; see [34, Ch.5].

In the numerical solution of differential equations, rounding error often plays a significant and complex role. Numerical instability can arise in multiple ways: unstable solutions, unstable methods, and stiff problems. The texts by Golub and Ortega [48] or Parker and Chua [84] have excellent tutorials.

8.3.1 Random rounding can reduce global error in numerical integration

Experimenting on ENIAC in 1948, Huskey [58] solved the simple system

$$x'(t) \;=\; y(t), \qquad y'(t) \;=\; -x(t)$$

(having exact solution $x(t) = \sin(t)$, $y(t) = \cos(t)$) with Heun's method:

$$\begin{aligned} x_i^* &= x_{i-1} + h\, y_{i-1} & \qquad x_i &= x_{i-1} + \tfrac{h}{2}\,(\, y_i^* + y_{i-1}\,), \\ y_i^* &= y_{i-1} - h\, x_{i-1} & \qquad y_i &= y_{i-1} - \tfrac{h}{2}\,(\, x_i^* + x_{i-1}\,). \end{aligned}$$

ENIAC provided 10-digit decimal arithmetic, and with a stepsize $h = 2 \cdot 10^{-5}$ and t near $0.5225 \approx \pi/6$, Huskey noticed the integration value $\frac{h}{2}\, y(t)$ repeatedly had the same final digit 4, which was rounded away. Thus the roundoff errors were not random, contradicting earlier arguments made by Rademacher [86]. Repeated rounding down made the resulting $x(t)$ values inaccurate. In the appendix to [58], Hartree explains Huskey's phenomenon as a consequence of properties of $y = \cos(t)$ and the ENIAC word length $k = 10$.

As an initial test, we reimplemented Huskey's computation. Interestingly, Huskey's phenomenon does not seem to be easily producible in IEEE arithmetic with the explanation given by Hartree.[14] However, the single precision IEEE implementation does yield high errors in $y(t)$ when $h = 2^{-12}$ with $t \approx \pi/6$. Notice that then h is the square root of the machine precision, so terms of order $O(h^2)$ are precisely at the level of roundoff, and since in this problem $y''(t) = -y(t)$, it makes sense that these second-order terms produce unusual roundoff at particular values of t.

Random rounding as proposed by Forsythe [36], and implemented in MCA, avoids Huskey's phenomenon, because identical values are rounded randomly. Hull and Swensen [57] demonstrated this explicitly, and our full MCA implementation did too. Also, MCA avoided the high errors incurred when $h = 2^{-12}$. For $T = 0.5234375$ (near $\pi/6$), $\cos(T) = 0.866106030320657$, where IEEE single precision integration obtained 0.866113 — accurate to only four digits — and full MCA single precision integration (over 4 samples) obtained 0.866106 with a standard error of $9.2 \cdot 10^{-7}$. Again, however, it is important to bear in mind that Chaitin-Chatelin and Frayssé [23, p.101] showed that Hull and Swensen's results hold only for regular points.

8.3.2 Chaotic differential equations

A recent paper by Alt and Vignes [3] shows dramatic results of using CESTAC on standard example differential equations. Rather than consider such problems further, we elected to explore the effects of MCA in chaotic differential equations. Chaotic equations are particularly interesting, since they are ill-conditioned (a key aspect of the definition of chaos is sensitivity to initial conditions), and their behavior can be argued to be 'pseudorandom' or 'statistical'.

Recently some researchers have shown that roundoff errors in differential equations corresponding to chaotic dynamical systems can have surprising and highly significant effects [10, 12, 27, 94]. These results are sometimes sensationalized, and predict disaster.

In actual experience, on the other hand, numerical integration of these equations generally seems to do well, and give results that are consistent with physical experiments. Consider the following assessment:

> The integration of a chaotic system poses a special problem. Sensitive dependence on initial conditions implies that an arbitrarily small error eventually affects the macroscopic behavior of the system. Error is inherent in any integration algorithm. Does it follow that the integration of a chaotic system is meaningless? Perhaps there are a few researchers who would answer yes, but the impressive contributions of numerical simulations in chaotic research show the answer is definitely no. ... What *is* true is that simulations of chaotic systems require careful interpretation and should always be verified. —— Parker & Chua [84, p.110]

In order to explain why numerical integration gives suprisingly good results, numerical analysts have taken different strategies. Parker and Chua explain that solutions one obtains often are quite good because they approach an attractor (attracting limit set), and even if they are not in precise agreement with the intended trajectory, they at least follow the attractor. The paper [28] by Corless (in the edited volume [67], which gives a good overview of numerical effects in chaotic dynamics) argues that the situation is similar to what arises in Gaussian elimination, and that the backwards error perspective is useful for chaotic systems (although the forward perspective is needed for stiff systems).

We suspect that statistical analysis gives better explanations. To test this suspicion, we ran the popular workhorse Runge-Kutta-Fehlberg RKF45 program [89] with MCA on both linear and nonlinear differential equations defining oscillators. RKF45 estimates error by comparing the results

[14] Hartree's explanation seems slightly incorrect. He argues the final digit of $\lfloor 10^{k+1} h y \rfloor$ is constant if the final digit of $\lfloor 10^{k+1} h^2 |y'| \rfloor$ is zero, or equivalently $10n - \frac{1}{2} < 10^{k+1} h^2 |y'| < 10n + \frac{1}{2}$ where n is an integer. However, Huskey observed this for $\frac{h}{2}\, y$ and not $h\, y$. Using $y' = -\sin(t)$ and solving for t, Hartree finds $\sin(t) \approx 1/2$ (so $t \approx \pi/6$), $n = 1$, and argues that the final digit will not change for $1/(10^{k+1} \frac{1}{2} h^3 |y''|)$ steps, predicting Huskey's results excellently.

of fourth- and fifth-order Runge-Kutta algorithms, and uses these estimates to adjust the integration stepsize. We considered the **Duffing oscillator**, defined by

$$\frac{d^2x}{dt^2} \;=\; -\,r\,\frac{dx}{dt} \;-\; \alpha^2\,x \;-\; \beta\,x^3 \;+\; f\,\cos(\omega\,t), \qquad x(0) \;=\; 0, \qquad \frac{dx}{dt}(0) \;=\; A,$$

in the trivial case $\alpha = \omega = 1$, $\beta = 0$, $A = f/r$, $f = 12$, $r = 0.1$, which has solution

$$x(t) \;=\; A\,\cos(\,t \,-\, \pi/2\,),$$

and the chaotic case $\alpha = 0$, $\beta = \omega = 1$, which is known as the Ueda equation. We also considered the **Van der Pol oscillator**

$$\frac{d^2x}{dt^2} \;=\; k\,(1 \,-\, x^2)\,\frac{dx}{dt} \;-\; x, \qquad x(0) \;=\; 2, \qquad \frac{dx}{dt}(0) \;=\; 0$$

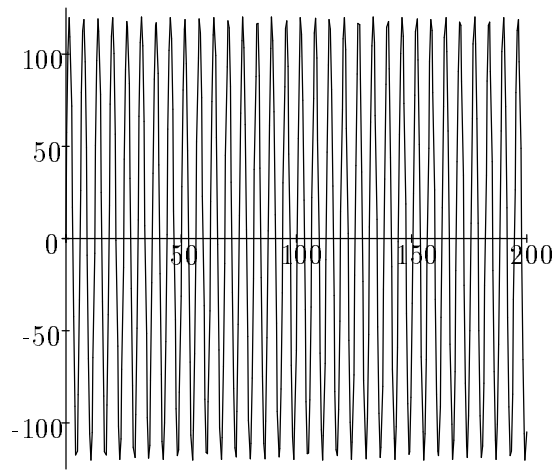
in the standard and extreme overdamped cases $k = 3$ and $k = 10$. Except for the trivial Duffing oscillator, none of these has an analytic solution. For more on chaotic oscillators, see e.g. [64].

Arithmetic	*Problem*	`relerr`	`abserr`	*step*
IEEE double	trivial Duffing	`1.0d-11`	`1.0d-9`	`0.25`
IEEE double	chaotic Duffing	`1.0d-11`	`1.0d-10`	`0.25`
IEEE double	standard van der Pol	`1.0d-11`	`1.0d-11`	`0.25`
IEEE double	extreme van der Pol	`1.0d-11`	`1.0d-11`	`0.25`
IEEE single	trivial Duffing	`2.0e-7`	`2.0e-5`	`0.25`
IEEE single	chaotic Duffing	`2.0e-7`	`2.0e-6`	`0.25`
IEEE single	standard van der Pol	`2.0e-7`	`2.0e-7`	`0.25`
IEEE single	extreme van der Pol	`2.0e-7`	`2.0e-7`	`0.25`
MCA single	trivial Duffing	`2.0e-7`	`2.0e-5`	`0.25`
MCA single	chaotic Duffing	`2.0e-7`	`2.0e-6`	`0.25`
MCA single	standard van der Pol	`2.0e-7`	`2.0e-7`	`0.25`
MCA single	extreme van der Pol	`2.0e-7`	`2.0e-7`	`0.25`

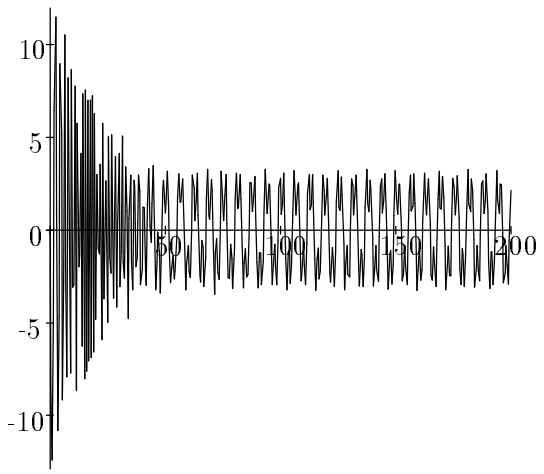
Figure 11: Parameters used with RKF45

With RKF45, we used the local relative and absolute error specifications in Table 11. The results are shown in Figures 12 and 13. MCA results reflect 5 samples obtained with ordinary rounding (`round_to_nearest`) and `uniform_absolute` input precision bounding. They agree closely with the single and double precision solutions, much as was observed by Hull and Swensen [57]; overall the magnitude of the global error is often less than the standard deviation of the samples, and was smaller than we had anticipated. The perspective of Parker and Chua quoted above was borne out. For example, with the trivial Duffing oscillator, the final average value computed for $t = 200$ was -104.795, correct to six digits with a standard deviation of 0.003.

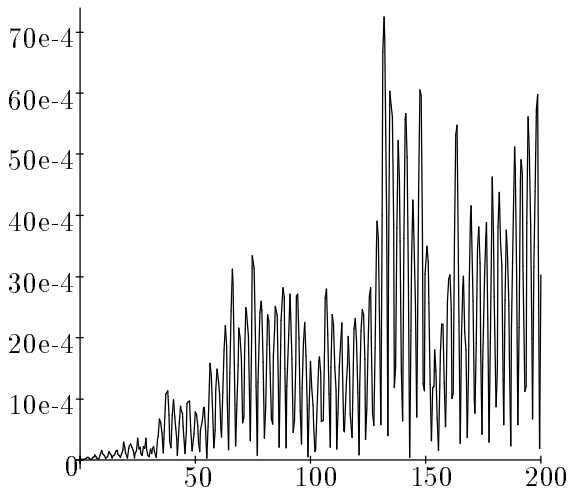
Statistics also appear useful in evaluating the quality of numerical integrations. With nonlinear equations the primary tools for evaluating error are running in higher precision and using another integration package. MCA yields a statistical appreciation of the error. Figures 12 and 13 show that, for these oscillators, the error incurred is 'bursty', and this is particularly pronounced for the extremely overdamped Van der Pol oscillator. Although with traditional methods one can estimate the local roundoff error at any given step, it is difficult to estimate the global roundoff error incurred, i.e., it is usually difficult to estimate the accuracy of results obtained after a long integration. Furthermore there is a basic interaction between roundoff error and discretization (stepsize) error that is difficult to quantify. The standard deviation plots in Figures 12 and 13 shows how standard deviation can be used to estimate problem condition, and suggests that standard error could be used as an estimate of global error.



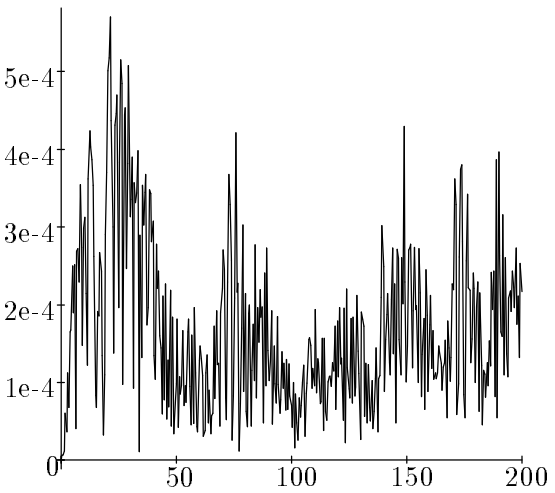
12.1 trivial Duffing oscillator (sine curve)



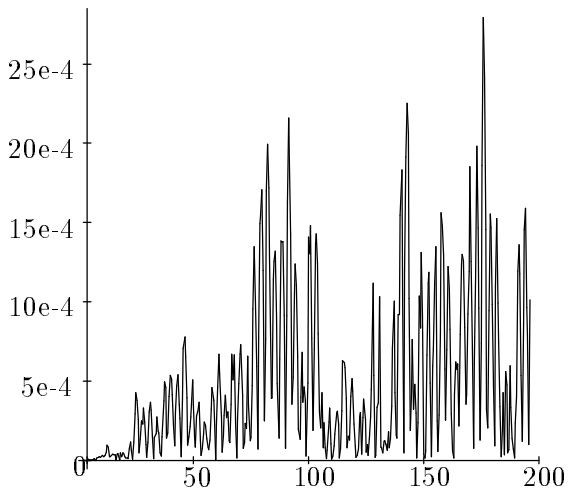
12.2 chaotic Duffing oscillator



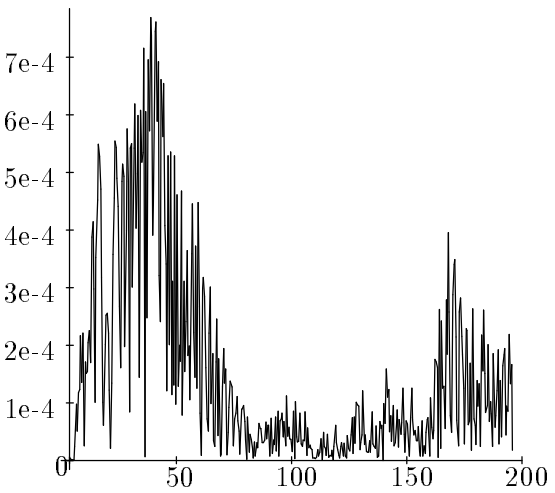
12.3 trivial Duffing: std. deviations (5 samples)



12.4 chaotic Duffing: std. deviations (5 samples)

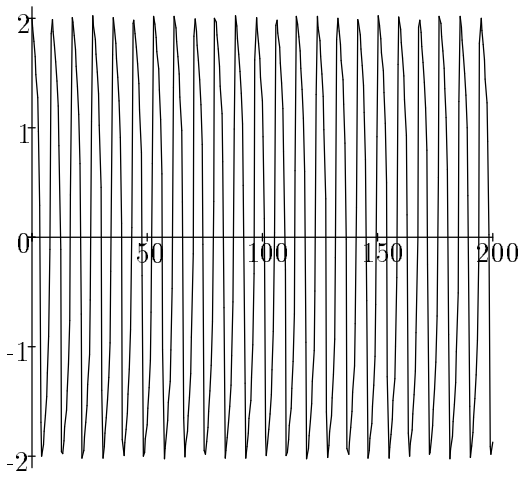


12.5 trivial Duffing: global error

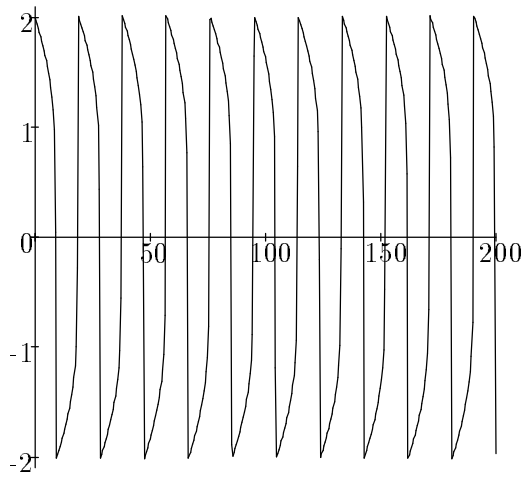


12.6 chaotic Duffing: global error

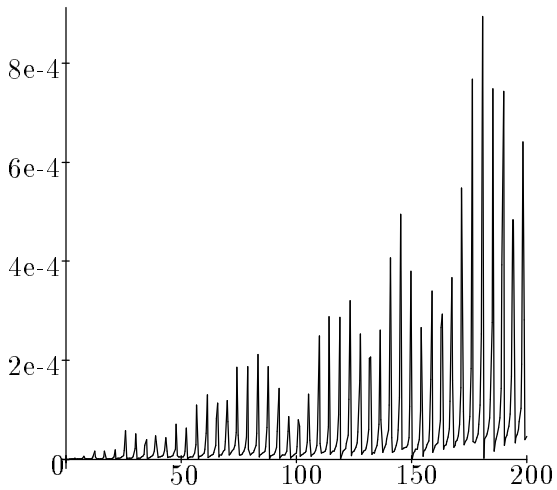
Figure 12: Duffing oscillator, standard deviation (estimating local error), and global error



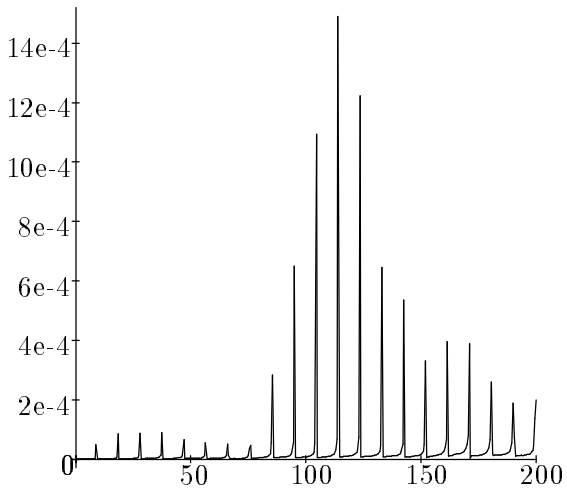
13.1 standard van der Pol oscillator



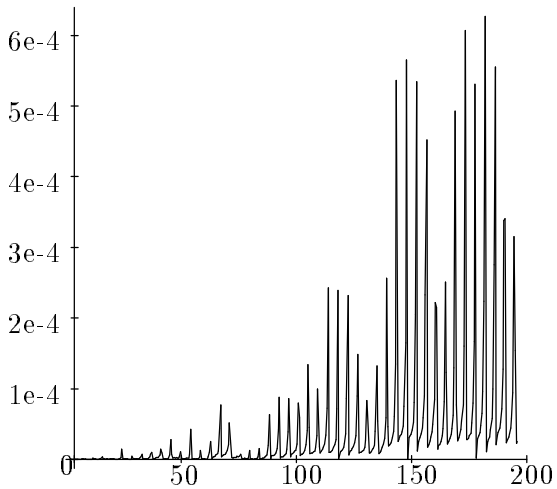
13.2 extreme van der Pol oscillator



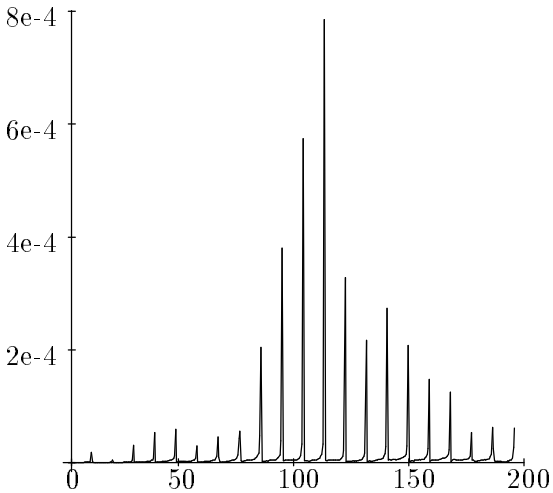
13.3 standard van der Pol: std. deviations (5 samples)



13.4 extreme van der Pol: std. deviations (5 samples)



13.5 standard van der Pol: global error



13.6 extreme van der Pol: global error

Figure 13: Van der Pol oscillator, standard deviation (estimating local error), and global error

9 Conclusion

We have argued that Monte Carlo Arithmetic (MCA) has interesting practical uses in numerical computations. With both a theoretical and case-study analysis, we sketched how MCA can give useful results without imposing a great deal of overhead. Although it is certainly no panacea, it 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 about MCA we have tried to argue in this work include:

- MCA is a simple way to bring some of the benefits of the Monte Carlo method [56] 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 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. These anomalies render floating-point arithmetic difficult to formalize and to reason about, and recently have been attacked by some as putting into question the foundations on which numerical analysis is based. 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. In the words of Nick Metropolis [76, p.130]:

> It is, in fact, the coupling of the subtleties of the human brain with rapid and reliable calculations, both arithmetical and logical, by the modern computer that has stimulated the development of experimental mathematics. This development will enable us to achieve Olympian heights.

Acknowledgements

Paul Eggert has contributed heavily to the ideas behind this work, and was the catalyst for it happening in the first place; over breakfast in June 1996 he suggested the idea of random rounding. He also developed the first version of the C implementation of MCA, which implemented several interesting random rounding methods not currently described in §6.4, and made other conceptual contributions as well. He would have been a co-author of this work but for ill-conditioning in his schedule, and unexpected random arrivals of chaotic phenomena.

I am indebted to Brad Pierce for his voluminous perceptive comments that have improved this work in a thousand ways, and for his many talks with me during the development of this work. If it were not for Brad, my writing would be even less concise and organized than it is, and the ideas would certainly be less well worked out than they are. I have been fortunate to be associated with him while he was at UCLA.

Walter Karplus pointed out that the ability to change the precision used in a computation, and do this dynamically, is important. He remarked that in special-purpose numeric hardware, one generally wants to use the least precision possible, so it would be nice to be able to simulate the hardware under various precisions first. He also provided important perspective about the nature of numeric computation.

Many of the examples in this work were worked out with Maple. The interactive, experimental flavor of environments like this makes numerical analysis an enjoyable affair, and has been part of the impetus leading to the development of MCA.

This work is dedicated to W. Kahan, whose efforts have been a continuing source of inspiration.

Appendix A: Explanations for the Startling Examples

The first sequence (x_k) was defined by

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

The reason that floating-point implementations find this sequence converging to 1, 2, 3, or 4 is that the rational function

$$x \;-\; \frac{(3x^4 \;-\; 20x^3 \;+\; 35x^2 \;-\; 24)}{(4x^3 \;-\; 30x^2 \;+\; 70x \;-\; 50)} \;=\; \frac{(x \;-\; 1)\,(x \;-\; 2)\,(x \;-\; 3)\,(x \;-\; 4)}{4\left(x \;-\; \frac{5-\sqrt{5}}{2}\right)\left(x \;-\; \frac{5}{2}\right)\left(x \;-\; \frac{5+\sqrt{5}}{2}\right)}$$

has roots at 1, 2, 3, and 4, yet if the iteration strays near the singularities then it can easily jump from one root's basin of attraction to another. See Figure 14. The starting value is cleverly chosen near the first singularity, and the results of the iteration are extraordinarily sensitive to this value. For example, changing x_0 in the final place to 1.5100050721318 changes the results of Table 5 for precisions 16 or greater, and IEEE double precision converges to 1 instead of 3.

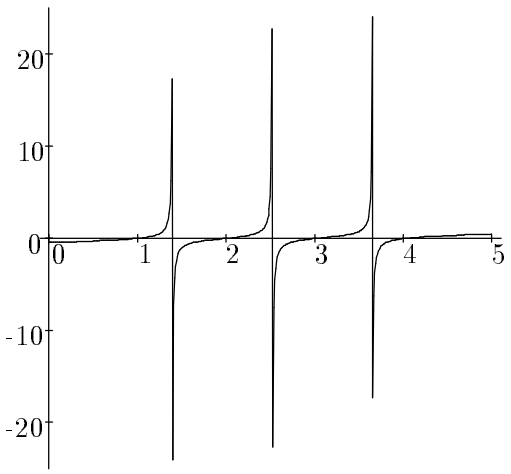


Figure 14: Plot of $x \;-\; \frac{(3x^4 \;-\; 20x^3 \;+\; 35x^2 \;-\; 24)}{(4x^3 \;-\; 30x^2 \;+\; 70x \;-\; 50)}$ for $0 \le x \le 5$.

The sequence (u_k) was defined by

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

All fixed-precision evaluations of u_k drift toward 100, even though $\lim_k u_k = 6$, because the iteration fixed points are roots of the polynomial

$$u^3 \;-\; 111\,u^2 \;+\; 1130\,u \;-\; 3000 \;=\; (u \;-\; 5)\,(u \;-\; 6)\,(u \;-\; 100)$$

which has a dominant root at 100. So, once the iteration strays from the fixed point at 6, it quickly converges to the fixed point 100. Table 15 clarifies how this works, by giving the first 30 iterates u_k computed in rounded decimal arithmetic at several machine precisions. Greater precision allows the iteration to hold on longer near 6, but there is a linear erosion of accuracy, and it always will eventually arrive at 100. Sterbenz [92, §7.5] gives other nice examples of anomalies that can arise when re-running programs in higher precision.

k	IEEE single	IEEE double	30 digit decimal computation	exact value of u_k (to 30 digits)
1	-4.	-4.	-4.	-4.
2	18.5	18.5	18.5	18.5
3	9.37838	9.37837837837838	9.378378378378378378378378784	9.378378378378378378378378378838
4	7.8011*6*	7.801152737752*261*	7.80115273775216138328530259*40*	7.801152737752161383285302596
5	7.154*54*	7.154414480976*556*	7.1544144809752493535278906*583*	7.154414480975249353527890653
6	6.80*856*	6.8067847369*41964*	6.80678473692363298394175*583*	6.8067847369236329839417565
7	6.6*1870*	6.592632768*974369*	6.592632768704438392742003*6893*	6.5926327687044383927420027
8	6.*84388*	6.44946593*7891006*	6.4494659337902879688684*930887*	6.4494659337902879688684792
9	*12.11769*	6.3484521*20305296*	6.34845205665435714710*9068265*	6.3484520566543571471006915
10	*53.92212*	6.274439*601630107*	6.27443859821632791383*27719786*	6.2744385982163279138293784
11	*94.63513*	6.2187*11741399645*	6.2186957398023977883*752685191*	6.218695739802397788321151
12	*99.64730*	6.176*0947362141*96	6.17583730492123012*0733967456*2	6.1758373049212301198633398
13	*99.97813*	6.14*6528757389504*	6.1423590812383559*534484911051*	6.1423590812383559393461401
14	*99.99866*	6.1*837401046771*84	6.1158830665510*9934007358046*	6.1158830665510807637442006
15	*99.99992*	*7.192332789799011*	6.09473943933368*4884249535155*1	6.0947394393336811283200392
16	*99.99999*	*21.3410907280212*00	6.07772230484*3043732093419757*	6.0777223048472427362818281
17	*100.00000*	*77.595467629816824*	6.0639403225*0082304925511873*78	6.0639403224998087552661669
18	*100.00000*	*98.2489186958330*69	6.0527217610*3288083957861570*19	6.0527217610161521933562840
19	*100.00000*	*99.892112375951569*	6.043552110*4656780547404219166*	6.0435521101892688677774736
20	*100.00000*	*99.993472190644496*	6.03603188*5655844156461642706*8	6.0360318810818567800106436
21	*100.00000*	*99.999605927688080*	6.029847*4008069413690742610341*	6.0298473250239018567130113
22	*100.00000*	*99.999976236658568*	6.024750*9092336562406588170602*	6.0247496523668478987435204
23	*100.00000*	*99.999998568253702*	6.0205*6084671840022262540220*92	6.0205399840615160624311381
24	*100.00000*	*99.999999913797950*	6.017*4047936456141510556288952*	6.0170582573289876111771827
25	*100.00000*	*99.999999994813010*	6.01*9933991097041023220444911*2	6.0141749145508189625309787
26	*100.00000*	*99.999999999688043*	6.*1074537932882993488170997518*	6.0117845878713336733840379
27	*100.00000*	*99.999999999981242*	*7.576267846759217472144778097*0	6.0098012392984845677868805
28	*100.00000*	*99.999999999998877*	*26.6844832280932568274499343378*	6.0081543789122289278208216
29	*100.00000*	*99.999999999999929*	*83.4923782553687487356215046220*	6.0067860930312057585305540
30	*100.00000*	*100.000000000000000*	*98.8123592273140258417713360972*	6.0056486887714202678924919

Table 15: Values of u_k $(1 \leq k \leq 30)$ computed with arithmetic of various fixed precisions. Incorrect digits are in italics.

Appendix B: Answers to Some Frequently Asked Questions

The idea of MCA provokes a number of fundamental questions, and we devote some space to trying to answer them.

- **Won't MCA be expensive to implement in hardware?**
  MCA is more expensive to implement in hardware than established floating-point rounding, requiring both more logic and potentially more clock cycles than conventional rounding methods. We are skeptical the hardware cost or computation time would be excessive; in particular, fast random number generators can be implemented efficiently with feedback shift registers (e.g., [66]; a nice recent survey is in [34, §7.15–16]).

  Actually, the reverse can be argued! Though we are not endorsing their method, Yoshida, Goto, and Ichikawa [115] have shown that floating-point multipliers with random rounding can be implemented more efficiently than they can without it. They avoid computing the less significant part of the product, arguing that the middle bits in the product are pseudorandom.

- **Won't MCA give pessimistic error bounds?**
  Generally, the answer to this question is: no. If the inputs are exact values, the MCA computes the same values computed by ordinary floating-point arithmetic. If the inputs are inexact, MCA simply tracks the extent of numeric precision. If the errors produced are large, this will be because the computation is ill-conditioned, and not because MCA is somehow pessimistic.

- **Won't MCA be useless in many common situations?**
  First, it doesn't really matter. Although MCA is certainly not *guaranteed* to be useful, if it is helpful in some situations, it is worth having.

  Specific situations where MCA is unnecessary include:

  – the input and output values are representable within the machine precision, and the problem is stable;

  – the accuracy of intermediate values increases, rather than decreases (as with Newton's method);

  – reproducibility of the results is imperative.

  None of these situations *preclude* the use of MCA. When a computation is well-conditioned, any reasonable rounding scheme will do, and precision bounding will not hurt. Also, reproducibility can be attained by providing user access to random number initialization (allowing user specification of a seed value at any point). Furthermore, the 'jitter' introduced by MCA can actually be useful in computations like equation-solving where degeneracy causes problems [82]. So while MCA is not needed in some situations, it may not harm to use it there.

- **Won't MCA give incorrect answers?**
  All finite-precision arithmetics give incorrect answers. It is important to be clear here about what one means by 'correct'. It is not enough to simply validate MCA against existing floating-point arithmetic, since that rests on compromises about exactness, error, and divergence from real arithmetic that warrant careful consideration.

  The very idea of randomization seems strange or suspicious at first to some people, perhaps because of the natural concern over injecting randomness into intricate logic and algorithms that were designed with conventional floating-point arithmetic in mind.

  One answer to this question is to notice that MCA can improve on ordinary floating-point in certain situations (like catastrophic cancellation). Another is that MCA is the adaptation

of Monte Carlo methods to floating-point arithmetic (where Monte Carlo is a well-developed branch of statistics [34]). This adaptation has various theoretical advantages, actually, such as resolution of floating-point anomalies.

Another answer is to appeal to authority — C.F. Gauss, surely one of the most accomplished numerical analysts of all time. Gauss developed his own Monte Carlo foundation for computational errors over a thirty-year span of his life, and the resulting theory has become a basic part of scientific method. MCA can be viewed as little more than a way to engineer Gauss' theory into numerical computing practice.

- **Will MCA really have any effect?**
  One line of reasoning goes, for example, that "the lower-order bits in the mantissa of computed values should be essentially random anyway, so random rounding and MCA will just be mixing randomness with randomness". The concern is that random rounding may not differ in any significant way from ordinary rounding.

  First, in Section 7.2, we recalled that several authors have shown that the lower-order bits are sometimes not random at all, and that MCA can eliminate this nonrandomness.

  Second, the simplest answer to this question is: the proof of the pudding is in the eating. The examples in this work show random rounding is good at detecting catastrophic cancellation, ill-conditioning, and poor numeric quality of results obtained with floating-point arithmetic.

  In the final analysis, the fundamental difference between MCA and deterministic arithmetic is that with MCA we can perform a computation multiple times, acquiring ever more information about the sensitivity of the computation. Deterministic computation does not have this property at all.

- **Why do we need another option for analyzing roundoff?**
  MCA gives another way to determine the accuracy of numerical computation. This is useful because the currently available options are limited:

  – **Trying both single and double precision**
    First, in at least some situations (like the u_k iteration in the Overview), double precision can fail to detect the inaccuracy of single precision computations.

    Second, comparing the results of a program when run at different precision levels (typically single and double precision) is not easy in many numerical computing environments. Both FORTRAN and C make this option annoying and error-prone (without enforced typing at the procedure call level, type polymorphism, or language features that make precision a parameter), and numerical packages often do little to support it.

    Third, while environments like Maple and Mathematica do allow run-time specification of desired precision, these environments are significantly slower than FORTRAN or C, and require effort to interface with existing programs and libraries.

    MCA *can be used in any precision.* So, even when a computation uses double precision arithmetic, it can make sense to also use MCA. Using randomization and higher precision are complementary strategies.

  – **Numeric quality indicators in existing numerical packages**
    Although formal roundoff analysis will identify some metric of instability, such as a condition number, numerical packages may only compute an estimate of this measure in the interest of speed. Many numerical packages return no measure of the quality of their computed values. There is an inherent conflict of interest between computational speed and run-time error measurement.

– **Formal roundoff analysis**
Roundoff analysis of a particular algorithm is usually hard to derive, amounting to determination of bounds on appropriate symbolic derivatives and/or on suitable aggregate norms. Also, and importantly, numerical algorithms often differ from their implementations in significant ways that affect the quality of the results dramatically. The paper by Shampine et al. [89] stresses, for example, that the ways ODE methods are implemented can be more significant than the formal differences between methods. Also note the opinion of Wilkinson [111, p.567]:

> There is still a tendency to attach too much importance to the precise error bounds obtained by an a priori analysis. In my opinion, the bound itself is usually the least important part of it. The main object of such an analysis is to expose the potential instabilities, if any, of an algorithm so that hopefully from the insight thus obtained one might be led to improved algorithms. Usually the bound itself is weaker than it might have been because of the necessity of restricting the mass of detail to a reasonable level and because of the limitations imposed by expressing the errors in terms of matrix norms. A priori bounds are not, in general, quantities that should be used in practice. Practical error bounds should usually be determined by some form of a posteriori error analysis, since this takes full advantage of the statistical distribution of rounding errors and of any special features, such as sparseness, in the matrix.

– **Interval analysis**
Interval computation [77, 78] has failed to gain widespread popularity. The most commonly given reason for this is that the intervals it produces are typically very pessimistic, and therefore not really useful in assessing the quality of the results. An excellent summary is offered by Wilkinson [111, pp.566–567]:

> If this mode of arithmetic is applied, for example, to the elimination method for solving linear equations, it is found that after a few steps the intervals become very wide, the intervals representing the solution being so wide that virtually no information is obtained. This is not surprising, since the computer is effectively performing numerically a strict forward analysis of the type that was used before the [von Neumann-Goldstine] paper.
> As far as the eigenvalue problem is concerned the situation is even less favorable. Even in a stable algorithm based on orthogonal transformations, it may well happen that in one single intermediate step the computed matrix diverges completely from the true matrix. The intervals must be large enough to contain the 'correct' matrix and all matrices which might be obtained when rounding errors intervene. The elements obtained by ordinary digital computation with rounding will consist of a set of values drawn from the intervals which are specially correlated so that the eigenvalues are well-preserved. Interval arithmetic has no means of recognizing this favorable correlation.
> This does not imply that interval arithmetic is useless, but it does place severe restrictions on the way it can be applied. In general it is best in algebraic computations to leave the use of interval arithmetic as late as possible so that it effectively becomes an a posteriori weapon.

A new option for analyzing roundoff can only help. As Kahan remarks [63, p.34]:

> Competent engineers rightly distrust all numerical computations and seek corroboration from alternative numerical methods, from scale models, from prototypes, from experience, ...

- **Won't running the program more than once be too expensive?**
Running the program more than once is clearly overhead. However, there are reasons why this overhead may not be that significant.

  First, one does not *always* have to use MCA; instead, one can use MCA when one needs it. In other words, MCA can be used in validating the result of important computations.

  Second, even today the cycles are often available for burning. Intel's Pentium Pro processor already provides sufficient parallelism to permit multiple program evaluations to be done in parallel, and everyone expects the availability of cycles to increase.

  Finally, the overhead of running the program a few times may be negligible compared to the many other costs facing computer modeling tasks.

- **Instead of implementing MCA in hardware, why not simply implement greater machine precision?**
  There is no such thing as 'enough' precision. Any sufficiently unstable computation can overwhelm any fixed amount of finite precision. A major intent of MCA is to be able to detect exactly when and where the available precision has been overwhelmed.

  MCA can be used in any hardware precision; it is 'orthogonal' to the computational precision. No matter how precise one's floating-point arithmetic, one can always use an 'idiot light' — an *a posteriori* analysis of the accuracy of the results that measures the sensitivity of the computation to roundoff.

- **Is MCA really an alternative way to obtain higher accuracy?**
  MCA is a very inefficient way to get greater accuracy from floating-point arithmetic. In the limit, as the number of samples n goes to infinity, the number of significant digits in the computed average value of a simple arithmetic function does increase gradually to any desired level. However, assuming the number of significant digits is measured by the log of the relative error, which is the approximately the ratio of the computed average to the standard error, it increases like $O(\log(n))$, so doubling the number of significant digits requires squaring the number of samples. Although it *is* therefore possible to look at MCA as a way to get greater accuracy, it is an extremely inefficient way to do this.

Copyright ©1996, 1997 D. Stott Parker

Appendix C: Basic Results in Probability Theory and Statistics

C.1 Results involving density functions

Recall that a **probability density** is a function $f : \Re \rightarrow \Re$ satisfying

$$\begin{array}{lll} 1. & f(t) \;\geq\; 0, & \text{for all } t \in \Re \\ 2. & \int_{-\infty}^{\infty} f(t)\,dt \;=\; 1. & \end{array}$$

The **impulse function at** x is a probability density $\uparrow_x$ defined by

$$\int_{-\infty}^{c} \uparrow_x (z)\,dz \;=\; \begin{cases} 0 & c < x \\ 1 & c \geq x. \end{cases}$$

It is a density with a spike (all of its support) at x.

The **convolution** $f \star f_2$ of two probability densities f_1 and f_2 is the density defined by

$$(f_1 \star f_2)(x) \;=\; \int_{-\infty}^{\infty} f_1(x - z)\, f_2(z)\,dz.$$

Theorem 8 *If ξ_1 and ξ_2 are independent random variables with corresponding densities f_1 and f_2, then the sum $\xi = \xi_1 + \xi_2$ corresponds to the convolution density $f = f_1 \star f_2$.*

Proof
Since f is the density that gives the probability at each point x that $\xi_1 + \xi_2 = x$,

$$\mathsf{Pr}[\; \xi \;\leq\; x \;] \;=\; \int\int_{y+z \leq x} f_1(y)\, f_2(z)\,dy\,dz \;=\; \int_{-\infty}^{\infty} \int_{-\infty}^{x-z} f_1(y)\, f_2(z)\,dy\,dz.$$

Differentiating with respect to x, $f(x) = \int_{-\infty}^{\infty} f_1(x - z)\, f_2(z)\,dz = (f_1 \star f_2)(x)$. □
Notice that the identity under convolution is $\uparrow_0$:

$$(\uparrow_0 \star f)(x) \;=\; (f \star \uparrow_0)(x) \;=\; f(x).$$

Theorem 9 *If ξ_1 and ξ_2 are independent random variables with corresponding densities f_1 and f_2, then the product $\xi = \xi_1\,\xi_2$ corresponds to the density f defined by*

$$f(x) \;=\; \int_{-\infty}^{\infty} f_1(x\,/\,z)\, f_2(z)\,\frac{dz}{z}.$$

Proof
Very similar to that of Theorem 8. □

The **n-th moment** M_n of ξ (or of its density f) is

$$M_n \;=\; \int_{-\infty}^{\infty} z^n\, f(z)\,dz.$$

For any arbitrary complex-valued function g on $\Re$, **the expected value of** $g(\xi)$ is

$$\mathsf{E}[\, g(\xi)\,] \;=\; \int_{-\infty}^{\infty} g(z)\, f(z)\,dz$$

so $M_n = \mathsf{E}[\, \xi^n\,]$. The **mean** of ξ (or of its density f) is its first moment:

$$\mu \;=\; \mathsf{E}[\, \xi\,] \;=\; M_1.$$

The **variance** of ξ (or of its density f) is given by

$$\sigma^2 \;=\; \mathsf{V}[\, \xi\,] \;=\; \mathsf{E}\Big[\, (\xi - \mu)^2\,\Big] \;=\; \mathsf{E}\Big[\, \xi^2 - \mu^2\,\Big] \;=\; M_2 - M_1^2.$$

C.2 The Tchebycheff and Chernoff Bounds

Theorem 10 *(Tchebycheff Bound)*
If ζ is a random variable with underlying density $p(z)$, having mean 0 and variance σ^2, then

$$\mathsf{Pr}[\ |\zeta|\ \geq\ \lambda\,\sigma\ ]\ \leq\ 1/\lambda^2.$$

Proof

$$\sigma^2\ =\ \mathsf{E}\Big[\ \zeta^2\ \Big]\ =\ \int_{-\infty}^{\infty} z^2\,p(z)\,dz\ \geq\ \int_{|z|\,\geq\,\lambda\,\sigma} z^2\,p(z)\,dz\ \geq\ (\lambda\,\sigma)^2\int_{|z|\geq\lambda\,\sigma} p(z)\,dz\ =\ (\lambda\,\sigma)^2\ \mathsf{Pr}[\,|\zeta|\ \geq\ \lambda\,\sigma\,].$$

Dividing both sides by $(\lambda\,\sigma)^2$ gives the stated inequality. □

This inequality says that we are guaranteed

$$\mathsf{Pr}[\ |\zeta|\ \geq\ \lambda\,\sigma\ ]\ \leq\ 0.0028$$

provided we take $1\ /\ \lambda^2 \leq 0.0028$, or in other words

$$\lambda\ \geq\ 18.89.$$

Clearly this is not very tight, since $\lambda = 3$ is sufficient for normally distributed variables. *Much* tighter limits can be placed on λ.

Theorem 11 *(Chernoff Bound)*
Let $\zeta = \sum_{i=1}^{N} \zeta_i$, where each ζ_i is a random variable with symmetric uniform density. Then

$$\mathsf{Pr}[\ |\zeta|\ \geq\ \lambda\,\sigma\ ]\ \leq\ 2\ \exp(\,-\lambda^2\,/\,2\,).$$

Proof
Define the threshold function

$$d(z)\ \ =\ \ \begin{cases} 0 & z\ <\ \lambda\,\sigma \\ 1 & z\ \geq\ \lambda\,\sigma. \end{cases}$$

Then for every positive value L, $d(z) \leq \exp(L\,(z-\lambda\,\sigma))$, so

$$\begin{aligned} \mathsf{Pr}[\ |\zeta|\ \geq\ \lambda\,\sigma\ ]\ &=\ \mathsf{E}[\,d(\zeta)\,] \\ &\leq\ \mathsf{E}[\,\exp(L\,(\zeta-\lambda\,\sigma))\,] \\ &=\ \mathsf{E}[\,\exp(L\,\zeta)\,]\ \ \mathsf{E}[\,\exp(-\,L\,\lambda\,\sigma)\,] \\ &=\ \mathsf{E}[\,\exp(L\,\zeta)\,]\ \ e^{-\,L\,\lambda\,\sigma}. \end{aligned}$$

Now, if $p(z)$ is the density associated with ζ, then

$$\begin{aligned} \mathsf{E}[\,\exp(L\,\zeta)\,]\ &=\ \int_{-\infty}^{\infty}\ \exp(L\,z)\ p(z)\ dz \\ &=\ \sum_{k=0}^{\infty}\ \frac{L^k}{k!}\ \int_{-\infty}^{\infty}\ z^k\ p(z)\ dz \\ &=\ \sum_{k=0}^{\infty}\ \frac{L^k\ M_k}{k!} \end{aligned}$$

where M_k is the k-th moment of $p(z)$. Since $p(z)$ is symmetric, $M_k = 0$ for odd k, and

$$\mathsf{E}[\,\exp(L\,\zeta)\,]\ \ =\ \ \sum_{k=0}^{\infty}\ \frac{L^{2k}\ M_{2k}}{(2k)!}.$$

It can be shown that for sums of uniform densities $p(z)$,

$$\frac{M_{2k}}{M_2^k} \quad \leq \quad \frac{(2k)!}{k!\,2^k}$$

using Fourier transforms on both sides of the equation $p = p_1 \star \cdots \star p_N$. (In fact, if $p(t)$ were normal with mean zero, this bound would be satisfied with equality.) It now follows that

$$\mathsf{E}[\,\exp(L\,\zeta)\,] \quad \leq \quad \sum_{k=0}^{\infty} \frac{1}{k!} \left(\frac{L^2\ M_2}{2}\right)^k \quad = \quad \exp(\,L^2\,\sigma^2\,/\,2\,).$$

Therefore

$$\mathsf{Pr}[\ |\zeta|\ \geq\ \lambda\,\sigma\ ] \quad \leq \quad \exp(\,L^2\,\sigma^2\,/\,2\ -\ L\,\lambda\,\sigma\,)$$

for all positive L. The Chernoff bound results from choosing L so that the right side of this expression is minimized. Differentiating shows that $L = \lambda/\sigma$ is the right choice, and gives the stated bound. □

The Chernoff bound is much tighter than the Tchebycheff bound, and is almost as good as having a normal distribution. For comparison we tabulate the values of λ obtained from each.

Desired probability $\mathsf{Pr}[\ \lvert\zeta\rvert \geq \lambda\sigma\ ]$	*Confidence interval value for* λ (ζ is normal)	*Chernoff bound value for* λ (ζ sum of symmetric uniform)	*Tchebycheff bound value for* λ (ζ arbitrary)
.10	1.645	2.448	3.162
.05	1.960	2.716	4.472
.02	2.326	3.035	7.072
.01	2.576	3.255	10.00
.002798	3.000	3.625	18.90
.002	3.090	3.717	22.36
.001	3.290	3.899	31.62
.0002	3.719	4.292	70.71
.00002	4.265	4.799	223.6
.000002	4.753	5.257	707.2

C.3 The Central Limit Theorem

Theorem 12 *Given densities $f_1, \ldots, f_N$ where f_i has mean μ_i and variance σ_i^2, then the convolution density $f_1 \star \cdots \star f_N$ has mean $\mu = \sum_{i=1}^N \mu_i$ and variance $\sigma^2 = \sum_{i=1}^N \sigma_i^2$.*

This can be proven by taking the Fourier transform $\mathsf{E}[\,\exp(\mathrm{i}\omega\zeta)\,]$ of both sides of $f = f_1 \star \cdots \star f_N$, and using basic facts about the Fourier transform of a convolution.

Theorem 13 *(Central Limit Theorem)*
In the limit as $N \to \infty$, the sum of N random variables $\xi = \xi_1 + \ldots + \xi_N$ has a density $f = f_1 \star \ldots \star f_N$, that converges to a normal density provided the individual probability densities $f_1, \ldots, f_N$, of the random variables $\xi_1, \ldots, \xi_N$ meet certain conditions.

Proof of this theorem is dependent of course on the conditions surrounding the summand variables and the type of convergence used. An excellent survey may be found in [33]; see also [30, 107] for formal treatments. For a history of early developements, see [93]. Fishman [34] lists a number of recently-derived variants of the Central Limit Theorem for Monte Carlo applications.

For the special case of sums of roundoff errors, whose densities are convolutions of roundoff densities, the convergence to a normal distribution is strikingly rapid. In Figures 15–17 the convergence is shown for sums of one to three errors that are uniform on the interval $(-\frac{1}{2}, \frac{1}{2})$.

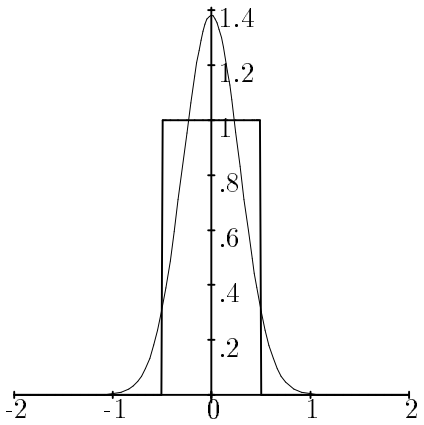


Figure 15: Uniform roundoff density vs. normal density ($\mu = 0$, $\sigma^2 = \frac{1}{12}$).

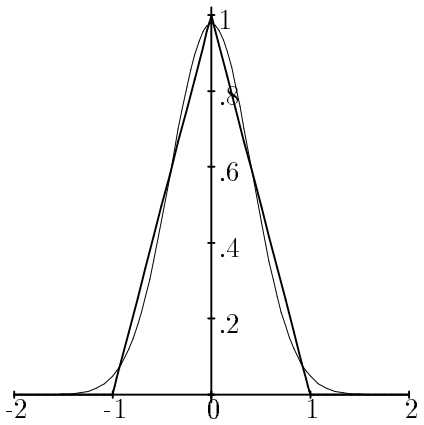


Figure 16: Sum of 2 uniform roundoff errors vs. normal density ($\mu = 0$, $\sigma^2 = \frac{1}{6}$).

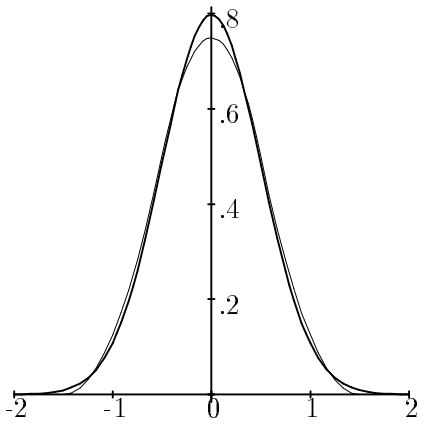


Figure 17: Sum of 3 uniform roundoff errors vs. normal density ($\mu = 0$, $\sigma^2 = \frac{1}{4}$).

Appendix D: Statistical Error Analysis of Gaussian Elimination

D.1 Solving linear systems by Gaussian elimination

A system of n linear equations $A\mathbf{x} = \mathbf{b}$ can be solved by Gaussian elimination as follows:

/* LU decomposition */
  for $k = 1$ **to** $n - 1$ **do**
    for $i = k + 1$ **to** n **do**
      begin
        $m_{ik} := a_{ik}^{(k)} \;/\; a_{kk}^{(k)}$
        for $j = k + 1$ **to** n **do** $a_{ij}^{(k+1)} := a_{ij}^{(k)} - m_{ik} \cdot a_{kj}^{(k)}$
        $b_i^{(k+1)} := b_i^{(k)} - m_{ik} \cdot b_k^{(k)}$
      end
  $\widehat{U} := A^{(n)}$
  $\mathbf{c} := \mathbf{b}^{(n)}$
/* Back substitution */
  for $k = n$ **downto** 1 **do**
    begin
      $K := n - k + 1$
      $x_k := c_k^{(K)} \;/\; u_{kk}$
      for $i = 1$ **to** $k - 1$ **do** $c_i^{(K+1)} := c_i^{(K)} - u_{ik} \cdot x_k$
    end

The LU decomposition phase starts with $A^{(1)} = A$ and $\mathbf{b}^{(1)} = \mathbf{b}$, and each iteration of the outer loop then computes $A^{(k+1)} = (a_{ij}^{(k+1)})$ and $\mathbf{b}^{(k+1)} = (b_i^{(k+1)})$ for $1 \le k \le n - 1$. This phase reduces the problem $A\mathbf{x} = \mathbf{b}$ to $A^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$, where $A^{(n)} = \widehat{U}$, an upper-triangular matrix. If $\mathbf{b}^{(n)} = \mathbf{c} = \mathbf{c}^{(1)}$, the second phase then solves the triangular system $\widehat{U}\widehat{\mathbf{x}} = \mathbf{c}$ by back substitution.

D.2 Error analysis

We follow the error analysis of Barlow and Bareiss [8], which differs from that of Wilkinson [108] to support statistical analysis. Corresponding directly to the algorithm, define the computed errors

$$e_{ik}^{(k+1)} = \begin{cases} - \left(a_{ik}^{(k)} - m_{ik}\, a_{kk}^{(k)} \right) & i \ge k+1 \\ 0 & \text{otherwise} \end{cases}$$

$$e_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k+1)} - \left(a_{ij}^{(k)} - m_{ik}\, a_{kj}^{(k)} \right) & i \ge k+1,\ j \ge k+1 \\ 0 & \text{otherwise} \end{cases}$$

$$e_{i}^{(k+1)} = \begin{cases} b_{i}^{(k+1)} - \left(b_{i}^{(k)} - m_{ik}\, b_{k}^{(k)} \right) & i \ge k+1,\ j \ge k+1 \\ 0 & \text{otherwise.} \end{cases}$$

If M_k is the matrix whose entries are zero except in the k-th column below the diagonal, where they are m_{ik}, then one iteration of the outermost loop of LU decomposition performs the computation

$$(I - M_k) \left(A^{(k)} \;\middle|\; \mathbf{b}^{(k)} \right) \;=\; \left(A^{(k+1)} \;\middle|\; \mathbf{b}^{(k+1)} \right) \;+\; \left(E^{(k+1)} \;\middle|\; \mathbf{e}^{(k+1)} \right)$$

where $E^{(k+1)} = (e_{ij}^{(k+1)})$ and $\mathbf{e}^{(k+1)} = (e_i^{(k+1)})$. Because we have $(I - M_k)^{-1} = (I + M_k)$ and $(I + M_k) \left(E^{(k+1)} \;\middle|\; \mathbf{e}^{(k+1)} \right) = 0$, we can reverse this equality to express $A^{(k)}$ in terms of $A^{(k+1)}$:

$$\left(A^{(k)} \;\middle|\; \mathbf{b}^{(k)} \right) \;=\; (I + M_k) \left(A^{(k+1)} \;\middle|\; \mathbf{b}^{(k+1)} \right) \;+\; \left(E^{(k+1)} \;\middle|\; \mathbf{e}^{(k+1)} \right).$$

If we further define the sums $E = \sum_{k=1}^{n} E^{(k)}$, $\mathbf{e} = \sum_{k=1}^{n} \mathbf{e}_k$, and

$$\widehat{L} \;=\; (I + M_1)\,(I + M_2) \cdots (I + M_{n-1}) \;=\; I \;+\; \sum_{k=1}^{n-1} M_k,$$

so that $\widehat{L}$ is the unit lower triangular matrix (m_{ik}), an induction on k on the reversed equality then produces

$$\left(A^{(1)} \;\middle|\; \mathbf{b}^{(1)} \right) \;=\; \widehat{L} \left(A^{(n)} \;\middle|\; \mathbf{b}^{(n)} \right) \;+\; (E \mid \mathbf{e})$$

which in simplified notation becomes

$$(A \mid \mathbf{b}) \;=\; \widehat{L} \left(\widehat{U} \mid \mathbf{c} \right) \;+\; (E \mid \mathbf{e})$$

or equivalently

$$\begin{aligned} A &= \widehat{L}\,\widehat{U} + E \\ \mathbf{b} &= \widehat{L}\,\mathbf{c} + \mathbf{e}. \end{aligned}$$

Similarly, back substitution produces errors $f_i^{(K+1)}$ which we define by

$$x_k \;=\; c_k^{(K)} \,/\, u_{kk} \;+\; f_k^{(K+1)} \,/\, u_{kk} \qquad 1 \le k \le n \;\; (K = n - k + 1),$$

$$c_i^{(K+1)} \;=\; c_i^{(K)} \;-\; u_{ik}\, x_k \;+\; f_i^{(K+1)} \qquad 1 \le k \le n, \quad 1 \le i \le k - 1 \;\; (K = n - k + 1).$$

Thus for $1 \le k \le n$

$$\sum_{i=1}^{k} u_{ik}\, x_k \;=\; c_i^{(1)} \;+\; \sum_{K=1}^{n-i+1} f_i^{(K+1)}$$

or equivalently

$$\widehat{U}\,\widehat{\mathbf{x}} \;=\; \mathbf{c} \;+\; \mathbf{f}.$$

D.3 Backward error analysis

The analysis above shows $(A - E) = \widehat{L}\,\widehat{U}$. This gives the backwards analysis result that Gaussian elimination yields an exact LU decomposition of the matrix $(A - E)$.

From $(A - E) = \widehat{L}\,\widehat{U}$, $\mathbf{b} = \widehat{L}\,\mathbf{c} + \mathbf{e}$, and $\widehat{U}\,\widehat{\mathbf{x}} = \mathbf{c} + \mathbf{f}$, we derive

$$\begin{aligned} (A - E)\,\widehat{\mathbf{x}} &= \widehat{L}\,\widehat{U}\,\widehat{\mathbf{x}} \\ &= \widehat{L}\,(\mathbf{c} + \mathbf{f}) \\ &= (\mathbf{b} - \mathbf{e} + \widehat{L}\,\mathbf{f}). \end{aligned}$$

This summarizes the effect of the algorithm in a backward way: the output solution $\widehat{\mathbf{x}}$ is the exact solution of the problem $A'\widehat{\mathbf{x}} = \mathbf{b}'$, where $A' = (A - E)$ and $\mathbf{b}' = (\mathbf{b} - \mathbf{e} + \widehat{L}\,\mathbf{f})$.

D.4 Forward error analysis

If $\mathbf{x}$ is the exact solution to $A\mathbf{x} = \mathbf{b}$, from

$$(A - E)\,\hat{\mathbf{x}} \quad = \quad \mathbf{b} - \mathbf{e} + \hat{L}\,\mathbf{f}$$

we obtain the forward error

$$(\hat{\mathbf{x}} - \mathbf{x}) \quad = \quad A^{-1}\,\left(\, E\,\hat{\mathbf{x}} - \mathbf{e} + \hat{L}\,\mathbf{f}\,\right).$$

When $\|\,A^{-1}\,\|$ is large, as it is for the Hilbert matrix, the output solution $\hat{\mathbf{x}}$ can differ considerably from the exact solution $\mathbf{x}$.

D.5 Two-sided error analysis

From $(A - E)\hat{\mathbf{x}} = \mathbf{b} - \mathbf{e} + \hat{L}\,\mathbf{f}$, and $(A - E)\,\mathbf{x} = \mathbf{b} - E\ \mathbf{x}$ we obtain the two-sided error expression

$$(\hat{\mathbf{x}} - \mathbf{x}) \quad = \quad (A - E)^{-1}\,\left(\, E\,\mathbf{x} - \mathbf{e} + \hat{L}\,\mathbf{f}\,\right)$$

or equivalently

$$(\hat{\mathbf{x}} - \mathbf{x}) \quad = \quad (A - E)^{-1}\,\left(\, E\,A^{-1}\,\mathbf{b} - \mathbf{e} + \hat{L}\,\mathbf{f}\,\right).$$

D.6 Statistical error analysis

Finally, the errors defined above can be evaluated statistically. Barlow and Bareiss [8, pp.357–360] give explicit formulas for the means $\mathsf{E}[\,\varepsilon\,]$ and variances $\mathsf{V}[\,\varepsilon\,]$, where ε is any of the error variables $e_{ik}^{(k)}$, $e_{ij}^{(k)}$, $e_{i}^{(k)}$, $f_{i}^{(K)}$, as they have been constructed so as to be independent random variables. For example,

$$\begin{aligned}
\mathsf{E}\left[\, e_{ik}^{(k)} \,\right] &= -a_{kk}^{(k-1)}\ \ \mathrm{sgn}(m_{ik})\ \ \beta^{G}\ \ \mathsf{E}[\,\varepsilon_{\otimes}\,] \\
\mathsf{V}\left[\, e_{ik}^{(k)} \,\right] &= \left(a_{kk}^{(k-1)}\right)^{2}\ \ \mathrm{sgn}(m_{ik})\ \ \beta^{2G}\ \ \mathsf{V}[\,\varepsilon_{\otimes}\,]
\end{aligned}$$

where $G = \lceil \log_{\beta}(|m_{ik}|) \rceil$ and $\varepsilon_{\otimes}$ is the error incurred in a multiplication or division operation. When rounding is used in t-digit base-β arithmetic, then [7, p.336]:

$$\begin{aligned}
\mathsf{E}[\,\varepsilon_{\otimes}\,] &= \frac{\beta^{-2t}}{48}\,\left(\, 12\,(1+\beta) \;-\; \frac{(\beta+1)\,(\beta^{2}+1)}{\ln(\beta)} \,\right) \;+\; O(\beta^{-3t}) \\
\mathsf{V}[\,\varepsilon_{\otimes}\,] &= \frac{\beta^{-2t}}{12} \;+\; O(\beta^{-3t}).
\end{aligned}$$

Barlow and Bareiss go on to derive 99.5% confidence bounds for $|E| = \sum_{i,j} \left| \sum_{k} e_{ij}^{(k)} \right|$ for the Hilbert matrix with $n = 4, 5, 6, 7, 8, 9$.

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, "The use of the CESTAC Method in the Parallel Computation of Roots of Polynomials", pp. 3–9 in *Numerical Mathematics and Applications*, R. Vichnevetsky and J. Vignes (eds.), Elsevier/North-Holland, 1986.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] J.L. Barlow, E.H. Bareiss, "Probabilistic analysis of Gaussian elimination in Floating Point and Logarithmic Arithmetic", *Computing* **34**:4, 349–364, 1985.

[9] F.L. Bauer, K. Samelson, "Optimale Rechengenauigkeit bei Rechenanlagen mit gleitendem Komma", *Zeitschrift für angewandte Mathematik und Physik* **4**, 312–316, 1953.

[10] C. Beck, "Effects of roundoff errors on chaotic dynamics", in: *Signal Processing VI - Theories and Applications: Proc. EUSIPCO-92, Sixth European Signal Processing Conference* (Brussels, Belgium, 24-27 Aug. 1992), vol.1, p. 183–186. J. Vandewalle, R. Boite, M. Moonen, A. Oosterlinck (eds.), Amsterdam: Elsevier, 1992.

[11] R.L. Bivins, N. Metropolis, "Significance Arithmetic: Application to a Partial Differential Equation", *IEEE Trans. Comput.* **C-26**:7, 639–642, July 1977.

[12] M. Blank, "Pathologies generated by round-off in dynamical systems", *Physica D* **78**:1-2, 93-114, 1 Nov. 1994.

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

[14] 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.

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

[16] 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.

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

[18] T.F. Chan, J.G. Lewis, "Computing standard deviations: Accuracy", *Comm. ACM* **22**:9, 526–531, 1979.

[19] T.F. Chan, G.H. Golub, R.J. LeVeque, "Algorithms for computing the sample variance: Analysis and recommendations", *Amer. Statist.* **37**:3, 242–247, 1983.

[20] F. Chatelin, "Sur la taux de fiabilité général de la méthode CESTAC", *Comptes Rendus de l'Academie des Sciences, Serie I (Mathématique)* **307**, 851–854, 1988.

[21] F. Chatelin, M.-C. Brunet, "A probabilistic round-off error propagation model. Application to the eigenvalue problem", in [29], 139–160, 1990.

[22] 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.

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

[24] J.-M. Chesneaux, J. Vignes, "Sur la robustesse de la méthode CESTAC", *Comptes Rendus de l'Academie des Sciences, Serie I (Mathématique)* **307**, 855–860, 1988.

[25] 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.

[26] W.G. Cochran, *Sampling Techniques*, 3rd edition, NY: J. Wiley & Sons, 1977.

[27] J.-F. Colonna, "The Subjectivity of Computers", *Comm. ACM* **36**:8, August 1993.

[28] R.M. Corless, "Error Backward", in [67], 31–62, 1994.

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

[30] H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, 1946.

[31] J.W. Demmel, "Underflow and the reliability of numerical software", *SIAM J. Sci. Stat Comput.* **5**:4, 887–919, December 1984.

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

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

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

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

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

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

[38] 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 [37].

[39] G.E. Forsythe, C.B. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, 1967.

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

[41] 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.

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

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

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

[45] 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.

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

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

[48] G.H. Golub, J.M. Ortega, *Scientific computing and differential equations: an introduction to numerical methods*, Boston: Academic Press, 1992.

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

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

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

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

[53] N.J. Higham, "The accuracy of floating point summation", *SIAM Journal on Scientific Computing* **14**:4, 783–799, July 1993.

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

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

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

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

[58] 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.

[59] W. Kahan, "Numerical Linear Algebra", *Canadian Mathematical Bulletin* **9**:6, 756–801, 1966.

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

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

[62] 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`

[63] 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`

[64] T. Kapitaniak (ed.), *Chaotic Oscillators: Theory and Applications*, River Edge, NJ: World Scientific, 1992.

[65] Lord Kelvin, "Nineteenth century clouds over the dynamical theory of heat and light", *Phil. Mag.* (6th series) **2**, 1–40, 1901.

[66] S. Kirkpatrick, E.P. Stoll, "A very fast shift-register sequence random number generator", *J. Comp. Phys.* **40**:2, 517–526, April 1981.

[67] P.E. Kloeden, K.J. Palmer, eds., *Chaotic Numerics*, Contemporary Mathematics series #172, Providence, RI: AMS, 1994.

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

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

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

[71] U. Kulisch, "Mathematical Foundation of Computer Arithmetic", *IEEE Trans. Comput.* **C-26**:7, 610–621, July 1977.

[72] U. Kulisch, W.L. Miranker, "The Arithmetic of the Digital Computer: A New Approach", *SIAM Review* **28**:1, 1–40, March 1986.

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

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

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

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

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

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

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

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

[81] 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.

[82] D.S. Parker, "Random Butterfly Transformations with Applications in Computational Linear Algebra", UCLA Computer Science Department, Technical Report CSD-950023, July 1995.

[83] D.S. Parker, "Explicit Formulas for the Results of Gaussian Elimination", UCLA Computer Science Department, Technical Report CSD-950025, 1995.

[84] T.S. Parker, L.O. Chua, *Practical Numerical Algorithms for Chaotic Systems*, NY: Springer-Verlag, 1989.

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

[86] H. Rademacher, "On the accumulation of errors in processes of integration", *The Annals of the Computation Laboratory of Harvard University* **16**, 176–185, 1948.

[87] F. Ris, E. Barkmeyer, C. Schaffert, P. Farkas, "When floating-point addition isn't commutative", *SIGNUM Newsletter* **28**:1, 8–13, Jan. 1993.

[88] S. M. Rump, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, NY: Academic Press, 1988.

[89] L. Shampine, H. Watts, S. Davenport, "Solving Nonstiff Ordinary Differential Equations – the State of the Art", *SIAM Review* **18**:3, 376–411, July 1976.

[90] T. Simpson, "A letter to the Right Honorable George Earl of Macclesfield, President of the Royal Society, on the advantage of taking the mean of a number of observations, in practical astronomy," *Proc. Royal Society of London* **49**, 82–93, 1755.

[91] C.C. Spicer, "Calculation of Power Sums of Deviations About the Mean", *Applied Statistics* **21**:2, 226–227, June 1972.

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

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

[94] G.G. Szpiro, "Cycles and circles in roundoff errors", *Physical Review E*, **47**:6, 4560–4563, June 1993.

[95] 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.

[96] L.N. Trefethen, "Three mysteries of Gaussian elimination", *ACM SIGNUM Newsletter* **20**:4, 2–5, October 1985.

[97] L.N. Trefethen, R.S. Schreiber, "Average-Case Stability of Gaussian Elimination", *SIAM J. Matrix Anal. Appl.* **11**:3, 335–360, July 1990.

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

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

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

[101] 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).

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

[103] 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.

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

[105] 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.

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

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

[108] J.H. Wilkinson, "Error analysis of direct methods of matrix inversion", *J. ACM* **8**:3, 281–330, July 1961.

[109] J.H. Wilkinson, *Rounding Errors in Algebraic Processes*, NJ: Prentice-Hall, Inc., 1963.

[110] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, London: Oxford University Press, 1965.

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

[112] S.J. Wright, "A Collection of Problems for which Gaussian Elimination with Partial Pivoting is Unstable", *SIAM J. Sci. Comput.* **14**:1, 231–238, January 1993.

[113] C.M. Wittenbrink, A.T. Pang, S.K. Lodha, "Glyphs for Visualizing Uncertainty in Vector Fields", *IEEE Trans. Vis. Comput. Graph.* **2**:3, 266–279, September 1996.

[114] M.-C. Yeung, T.F. Chan, "Probabilistic Analysis of Gaussian Elimination without Pivoting", Technical report CAM-95-29, Group in Computational and Applied Mathematics, Department of Mathematics, University of California, Los Angeles, Los Angeles,CA 90024-1555, 1995.

[115] N. Yoshida, E. Goto, S. Ichikawa, "Pseudorandom Rounding for Truncated Multipliers", *IEEE Trans. Comput.* **40**:9, 1065–1067, September 1991.