Assessing The Reliability of Statistical Software: Part I

B. D. McCullough

Entry-level tests of the accuracy of statistical software, such as Wilkinson's *Statistics Quiz*, have long been available, but more advanced collections of tests have not. This article proposes a set of intermediate-level tests focusing on three areas: estimation, both linear and nonlinear; random number generation; and statistical distributions (e.g., for calculating p-values). The complete methodology is described in detail. Convenient methods for summarizing the results are presented, so that an assessment of numerical accuracy can easily be incorporated into a software review.

KEY WORDS: Accuracy; Benchmarks; Random number generator; Software testing; StRD.

1. INTRODUCTION

Familiar to all is the tale of the researcher who solves the same problem using two software packages and obtains two different results. Of course the researcher has no idea which result, if either, is correct. Barring further inquiry, the output of neither package can be considered reliable. A second researcher who solved the problem using only one of the two packages would have no such doubts about the accuracy of the results, this researcher's blissful ignorance relying on (1) the software vendor's assurance that every effort has been made to verify the accuracy of the program and (2) the fact that no review of the software had ever mentioned any problem with accuracy. Relying upon the vendor's assurance and the reviewer's thoroughness amounts to a willful suspension of disbelief.

With respect to vendor assurances, the existence of well-known benchmarks for simple statistical procedures and the failure of many programs to pass these tests suggests that many vendors are not making an effort to verify the accuracy of their programs. Consider calculating the sample variance of three observations: 90000001, 90000002, 90000003 (the correct answer is unity). Because single precision is only accurate to six or seven digits, a single precision program with a good algorithm will fail this test. A double precision program with a good algorithm will pass, but with a bad algorithm can fail. An extended precision program with a bad algorithm might pass, but in such a case the extended precision does not deliver evidence of more accurate software—it only hides poor programming. The purpose of benchmarking is to assess the quality of the algorithms implemented, not to see how many registers can be stacked.

Wilkinson and Dallal (1977) applied this test to several mainframe statistical packages and found that many of them failed. Even today some packages cannot pass this simple test. A well-known collection of simple tests is Wilkinson's (1985) *Statistics Quiz*, which presents a set of problems designed to uncover common flaws in statistical programs. Wilkinson's tests may be characterized as a collection of entry-level tests, to be conducted before more serious tests of numerical accuracy are applied, and are discussed in detail by Sawitzki (1994a). Sawitzki (1994b) applied these tests to several statistical packages and no package passed them all. Wilkinson (1994), Bankhofer and Hilbert (1997), and McCullough (in press) also have applied *Statistics Quiz* and documented many failures, including negative variances and correlation coefficients greater than unity. The need for benchmarking is clear.

Will reviewers alert readers to numerical inaccuracies? No. A casual survey of software reviews published in statistical journals attests to this. This stands in stark contrast to the statistical profession's long history of concern about the reliability of its software, as evidenced by numerous books and articles (e.g., Francis, Heiberger, and Velleman 1975; Francis 1981; and Eddy, Howe, Ryan, Teitel, and Young 1991), various sessions of the COMPSTAT proceedings, and the ASA Statistical Computing Section. In defense of reviewers and software review editors, until recently there has been no authoritative or comprehensive source available to reviewers. In the absence of such a source, editors could not insist that reviewers address numerical accuracy. Reviewers, lacking such a source, rarely addressed numerical accuracy. Exceptions such as Vinod (1989), Veall (1991), and McCullough (1997) were limited in scope. Many users, relying on reviewers to mention problems with software, had no reason to question the accuracy of their software since few of them used different packages to solve the same problem.

B. D. McCullough is Senior Economist, Federal Communications Commission, Washington, DC 20554 (E-mail:bmccullo@fcc.gov). Thanks to R. Beardsley, R. Cavazos, C. Cummins, S. Hunka, W. J. Kennedy, L. Knusel, P. L'Ecuyer, J. G. MacKinnon, G. Marsaglia, J. Prisbrey, B. D. Ripley, M. R. Veall, and the referees, as well as to various persons who wish to remain anonymous, for comments and useful suggestions. Special thanks are due to M. Lovell, who commented extensively on early versions. The views expressed herein are those of the author and do not necessarily reflect those of the commission.

Benchmarks do exist, though. In addition to Wilkinson's tests, Lachenbruch (1983) published a suite of tests for general statistical software on microcomputers, as did Lesage and Simon (1985) and Elliott, Reisch, and Campbell (1989). Simon and Lesage (1988, 1989) also presented various benchmarks for univariate statistics and analysis of variance. For linear regression there is the Wampler (1970) suite of benchmarks and additional test problems (Wampler 1980), and the famous Longley (1967) benchmark. Longley (1967) calculated by hand the coefficients to a linear regression problem, and then compared his answer to those provided by several mainframe linear regression programs. He found that many programs produced coefficients that were not accurate to more than one or two digits, and some produced zero accurate digits. Although today most any regression package can accurately compute the Longley coefficients to several digits, passing one or even several benchmarks is no assurance that the program is error-free. The numerical accuracy of statistical and econometric software cannot be taken for granted.

The above-mentioned benchmarks all are for linear procedures, and the lack of nonlinear benchmarks has forestalled efforts to assess this crucial aspect of statistical software. Recently, the National Institutes of Standards and Technology (NIST) remedied this deficiency by compiling benchmarks for over 20 nonlinear least squares problems. Adding these to existing benchmarks and refining the latter, NIST produced the *Statistical Reference Datasets* (StRD) website (http://www.nist.gov/itl/div898/strd). The website has four suites of benchmarks: univariate summary statistics, analysis of variance, linear regression, and nonlinear regression, with test data in ASCII format and "certified values" for the parameters which are accurate to several digits.

Benchmarking is a time-consuming process, and most users will not wish to spend the time checking their software, preferring instead to read reviews which include an assessment of numerical accuracy. However, the output from benchmarking a single package is voluminous, requiring more space than a journal can allocate to a survey, let alone a software review. Presenting all of the output is impractical, as is merely reporting "pass" or "fail" for each test. The output from the tests must be condensed, perhaps to a single page, with discussion of the results and anomalies encountered limited to one or two pages. Such a method provides several advantages: first, it means that assessing numerical reliability can be part of any software review; second, it provides the reader with a summary of the numerical strengths and weaknesses of a package; and third, it facilitates comparison of packages.

Estimation is not the only aspect of statistical software prone to numerical error. Both the random number generator (RNG) and the various statistical distributions (e.g., for calculating p-values) typically are presented as black boxes: the RNG will provide as many truly random numbers as a user desires, and the output of a statistical distribution is accurate to all displayed decimals, regardless of the input parameters. Of course, these last two assertions are false. Vendors rarely provide the algorithms underlying these procedures, let alone the limits within which they can safely be relied upon or any evidence of their accuracy. Therefore, both the RNG and the statistical distributions must be assessed for reliability, or lack thereof.

Section 2 is a brief introduction to the limitations of a computer's numerical accuracy. Section 3 shows how to measure the accuracy of a calculated number. Section 4 describes the four suites of tests in the NIST StRD website. Section 5 discusses problems with RNGs and introduces Marsaglia's (1996) "DIEHARD Battery of Randomness Tests" for RNGs. Section 6 shows how to assess the accuracy of statistical distributions. Section 7 offers a summary. In Part II, to appear in a future issue, the methodology outlined here will be applied to three popular statistical packages: SAS, SPSS, and S-Plus.

2. ERRORS IN NUMERICAL COMPUTATION

Computers generally commit two types of errors when engaging in numerical computation, *rounding error* and *truncation error*. [There are many definitions of truncation and roundoff error. The definitions given here follow Higham (1996).] Part of the error inherent in numerical computation is due to the computer's *binary representation* of numbers with *finite precision*. Each of these concepts merits brief mention, more detailed accounts appearing in Kennedy and Gentle (1980, chaps. 2–3), Thisted (1988, chap. 2), Stewart (1996, chaps. 6–8), Gentle (1998a), and the classic article by Goldberg (1991) which, despite its title, is not just for computer scientists.

2.1 Binary Representation and Finite Precision

A computer represents numbers in binary form. The binary representation of the decimal .1 is .0001100110$\underline{0011}$, where underline indicates an infinitely repeating sequence. Computers, having finite precision, cannot represent an infinite number of digits. Assume single precision (it is easier than using double precision, and all the arguments follow, *mutatis mutandis*). The computer represents numbers in binary form with 24 binary digits (exclusive of leading zeroes) with a decimal somewhere. The single-precision binary representation (spbr) of the decimal .1 is .000110011001100110011001100110. Converting this binary number back into decimal notation yields .099999964, which is accurate to seven decimal places. This is what the computer "sees" when the number .1 is entered. The spbr of the number 100,000 is 11000011010100000.000000 which converts back into 100,000 exactly. Adding these two numbers, it is apparent that only seven binary places to the right of the decimal point are available to represent the decimal .1, so the sum has spbr 11000011010100000.000110. Reconverted to decimal, this number is not 100,000.1 but 100,000.09375, which is accurate only to two decimal places. Thus, adding a number accurate to seven decimals to an exact number produces a sum less accurate than either of its summands. Moreover, subtracting away the large part does not help, for in single precision the computer "sees" $(100,000.1 - 100,000.0)$ as $.09375$. This demonstrates that for very large numbers which differ only in the decimals,

the computer might not be able to represent accurately the differences between the numbers due to cancellation error.

2.2 Rounding Error

Rounding error is a function of hardware, and is primarily due to finite precision—that is, a computer has only so many bits with which to represent any number. For example, by the IEEE-754 standard for computer arithmetic which is implemented in the hardware of most every PC and workstation and many mainframes, single precision has about six or seven digits of accuracy, while double precision has about 15 or 16 digits of accuracy. Both $x = 1{,}000{,}000$ and $y = .000001$ can be represented in single precision, but their sum, $z = x + y$, cannot. In single precision the result will be $z = 1{,}000{,}000$ with the least significant digits being lost to rounding error. In double precision, the sum can be accurately represented, $z = 1{,}000{,}000.000001$. One type of rounding error which merits mention is *cancellation error*, which occurs when two nearly equal numbers are subtracted from each other, leaving only the rightmost digits, those most susceptible to rounding error. Successive rounding errors in a series of calculations do not cancel but, rather, accumulate, and the bound on the total error is proportional to the number of calculations. Sometimes this total error affects only the rightmost digits of the final answer. Sometimes the total error can be so large as to completely swamp the answer, resulting in no accurate digits; many such examples will be provided in Part II.

Another consequence of finite precision and rounding error is that two formulas, while algebraically equivalent, might not be numerically equivalent. Consider the following two formulas:

$$\sum_{n=1}^{10000} n^{-2} \qquad \sum_{n=1}^{10000} (10001 - n)^{-2}$$

The former sums the numbers in ascending order, the latter in descending order. In the latter method, the tiniest numbers are accurately represented and summed. In the former method, by the time the tiniest numbers are reached, they are all lost to rounding error when added to the existing sum. In fact, the numerical error in the former is 650 times greater than the error in the latter (Dahlquist and Björk 1974). Even with an infinite number of digits, which effectively eliminates rounding error, computers sometimes produce only approximate answers due to *truncation error*.

2.3 Truncation and Algorithmic Error

Truncation error is a function of software, and may be considered an approximation error. Iterative algorithms for nonlinear least squares problems are subject to truncation error, since the algorithm only yields the correct answer for an infinite number of iterations, yet in practice the number of iterations is finite. As a more concrete example, consider calculating the sine of x, $\sin(x)$.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \tag{1}$$

Clearly, the computer cannot carry the sum out to infinity, and must stop summing after some finite number of terms, k. Assuming infinite precision, the difference between the true value of $\sin(x)$ and that achieved by summing k terms is truncation error.

Yet another source of inaccuracy is the algorithm itself. There can exist many different ways to compute a quantity, and some are better than others. Ling (1974) analyzed five different ways to calculate the sample variance, and found that not all are equally reliable. The least accurate of the five is the "calculator formula" often presented in elementary statistical texts as a computational shortcut. As an algorithm, this formula is known to be numerically unstable. Yet, it is possible to find software packages that use this formula. Similarly, there are many ways to obtain the least squares coefficients in a linear regression, for example Gaussian elimination, the LU decomposition and the singular value decomposition (SVD). The former methods are not robust to collinear data, while the SVD is; therefore the SVD is the method of choice for solving linear least squares problems (Hammarling 1985; Press, Teukolsky, Vetterling, and Flannery 1992, p. 51), especially when the tradeoff between speed and accuracy is weighted 100% for accuracy and 0% for speed.

[Note: The QR decomposition, while not as robust to collinear data as the SVD, is more robust than the other methods mentioned, and much faster than the SVD. Software vendors, who must balance the various needs of diverse users, do well to implement the QR decomposition. The present author admits that, when bootstrapping, he generally prefers the QR to the SVD.]

Roundoff, truncation, and algorithmic errors all can contaminate a computed solution, thus degrading accuracy. It is useful to have some method for measuring the accuracy of a computed solution, say, q, by comparing it to a known or more reliably computed solution, say, c.

3. MEASURING ACCURACY

The significant digits in a number are the first nonzero digit and all succeeding digits. Thus, 3.1415 has five significant digits while .0012 has only two significant digits. The number of correct significant digits, while seemingly a very intuitive notion, actually is fraught with many pitfalls (see Higham 1996, sec. 1.2 for details). An oft-used measure of the number of correct significant digits, is the base-10 logarithm of the relative error

$$\mathrm{LRE} = -\log_{10}[|q - c|/|c|], \tag{2}$$

where q is the estimated value and c is the correct value. In the event that $q = c$ exactly, LRE is undefined, in which case it should be set equal to the number of digits in c. In a some cases, as in the standard errors of some of the linear regression benchmarks, the certified value is zero, so that the LRE is undefined. In those instances, the base-10 logarithm of the absolute error should be used

$$\mathrm{LAR} = -\log_{10}[|q|]. \tag{3}$$

For expositional ease, in the sequel, no distinction will be made between LRE and LAR, referring to both or either as LRE. The symbol λ with an appropriate subscript will denote the LRE of a computed quantity.

A fractional LRE has a specific interpretation: $\lambda_q = 2.70$ means that q agrees with the first 9 ($\approx 2.7/\log_{10} 2$) bits of c. Note that, as a function, the LRE is a measure of the number of correct significant digits only when q is "close" to c. To see this, let $q = 165.89$ and $c = 2.7070$; then LRE $= 1.78$. Values of q far from c are not infrequent, especially for nonlinear benchmarks. Therefore, each estimated quantity must be compared to its certified value to make sure that they differ by a factor of less than two; otherwise simply set λ_q to zero. It is possible for an LRE to exceed the number of digits in the c; for example, it is possible to calculate an LRE of 11.4 even though c contains only 11 digits. In part, this is because a double precision computer will "pad" the 11 digits with zeroes. Correcting for this is not worth the effort, so in such a case λ_q should be set to the number of digits in c. Finally, any λ_q less than unity should be set to zero.

What LRE is acceptable varies from user to user, but for coefficients from low-difficulty linear procedures it should be at least nine. This is because a decent implementation of a decent algorithm for a linear procedure should, with well-behaved data, return 10 accurate digits in double precision. This is not to suggest that users often need ten digits of accuracy. Rather, the idea is that a program which cannot deliver 10 digits of accuracy for an easy problem is more likely to deliver inaccurate results in practice. Guidelines for coefficients from nonlinear procedures are much less clear, though four or five digits of accuracy does not appear to be unreasonable.

4. THE NIST STRD BENCHMARKS

The StRD has four suites of benchmarks: univariate summary statistics, analysis of variance, linear regression, and nonlinear regression. Clearly the benchmarks would be of little value if all packages could solve them, or if no package could solve them. Within each suite, NIST has created the problems according to level of difficulty: lower, average, and higher. For the linear problems, data were read in and represented with 500 decimal digits of accuracy, as opposed to the usual double precision. Simple algorithms were used to perform each analysis and 500 digits were carried through all calculations using Bailey's multiple precision FORTRAN preprocessor and subroutine library (available at NETLIB). The final 500-digit answer was rounded to 15 significant digits. The nonlinear problems were solved using quadruple precision (128 bits) and two public domain programs with different algorithms and different implementations; the convergence criterion was residual sum of squares (RSS) and the tolerance was 1E–36. Certified values were obtained by rounding the final solutions to 11 significant digits. Each of the two public domain programs, using only double precision, could achieve 10 digits of accuracy for every problem. Complete information on problem selection, difficulty rating, method of solution, and computational details can be found in Rogers et al. (1998). A brief description of each suite follows.

Table 1. Results of ANOVA Data Set AgWt

source	*df*	*ss*	*ms*	*F*
			StRD	
between	1	3.64E-09	3.64E-09	1.59467335677930E+01
within	46	1.05E–08	2.28E–10	
			Package X	
between	1	3.61E–09	3.61E–09	1.5670329670329700E+00
within	46	1.06E–08	2.3E–10	

4.1 Univariate Summary Statistics

This suite has nine data sets, with levels of difficulty six lower, two average, and one higher, with the number of observations ranging from 3 to 5,000. For each data set, certified values to 15 digits are provided for the mean, μ; the standard deviation, σ; the first-order autocorrelation coefficient, ρ. Since only three certified values are given, the LRE for each can be presented, λ_μ, λ_s, and λ_ρ.

The simplest or most obvious method should be used to calculate each statistic. This corresponds to what a user might do if seeking to estimate only that statistic. For example, if "MEAN" and "STDEV" commands exist, these should be used rather than a "STATS" command which calculates the mean and the standard deviation in addition to several other statistics. (In a single product review it can be interesting to use different commands to compute the same quantity and observe the resulting differences.)

As an example, consider the data set Michelso, 100 observations from an experiment on the speed of light. [Note: The StRD uses UNIX-style file names, which are case-sensitive. The names are mnemonic, as perusing the StRD website will show. For example, SmnLsg01 is the first of the nine Simon-Lesage data sets for analysis of variance.] Its difficulty rating is "average." The certified value for the standard deviation is .0790105478190518. Package W calculates .078614502891384, so $\lambda_s = 2.30$. This suggests that the package uses one of the less efficient algorithms.

4.2 ANOVA Tests

This suite has 11 data sets, with levels of difficulty four lower, four average, and three higher. Each is a one-way analysis of variance problem. The number of replicates per cell ranges from 5 to 2001. For each data set, certified values to 15 digits are provided for "between treatment" degrees of freedom; "within treatment" degrees of freedom; sums of squares; mean squares; the F-statistic; the R^2; and the residual standard deviation.

Since most of the certified values are used in calculating the F-statistic, only its LRE is presented, λ_F. Again, the simplest or most obvious command should be used. For example, a package might have "ONEWAY", "ANOVA", "MANOVA" and "GLM" commands, all of which are capable of handling these problems. The "ONEWAY" command should be used. In a single product review, it may be useful to compare all the various methods. For packages that do not have an analysis of variance procedure per se,

Table 2. Results of Longley Data Set for Package Y

Variable	Coefficient LRE	Standard error LRE
constant	6.84	7.15
GNP deflator	4.87	6.44
GNP	5.81	6.26
unemployment	6.45	6.36
military employment	6.94	6.86
population	5.14	6.23
time	6.87	7.17

these tests can be performed using linear regression with the appropriate dummy variables for treatments.

As an example, consider the ANOVA data set AgWt, which has a difficulty rating of "average." Table 1 shows the certified values calculated by NIST and the results of a popular spreadsheet, Package X. The usual calculation produces $\lambda_F = 1.76$, which is sufficiently low that the reliability of the spreadsheet is called into question.

4.3 Linear Regression

This suite has 11 data sets, with levels of difficulty two lower, two average, and seven higher, the number of observations ranging from 3 to 82, and the number of coefficients ranging from 1 to 11. For each data set, certified values to 15 digits are provided for coefficient estimates; standard errors of coefficients; the residual standard deviation; R^2; and the usual analysis of variance for linear regression table, which includes the residual sum of squares.

Table 2 shows the Longley benchmark LREs for the coefficients and standard errors produced by Package Y, a statistical package. This surfeit of information can be conveniently summarized by appealing to the "weakest link in the chain" principle: use the minimum of the coefficient LREs and the minimum of the standard error LREs. Thus, for this package and this data set, $\lambda_\beta = 4.9$ and $\lambda_\sigma = 6.2$. These comparatively low LREs indicate that Package Y might use one of the less robust solution methods, for example the Cholesky decomposition.

4.4 Nonlinear Regression

This suite has 27 data sets, with levels of difficulty 8 lower, 11 average, and 8 higher, the number of observations ranging from 6 to 250, and the number of coefficients ranging from two to nine. For each data set, certified values to 11 digits are provided for coefficient estimates; standard errors of coefficients; the residual sum of squares; the residual standard deviation; the degrees of freedom; and the number of observations.

Also provided are two sets of starting values ("Start I" and "Start II") with which to initialize the nonlinear routine. Start I values are "far" from the solution, and Start II values are "close" to the solution. As in the case of linear regression, the LRE is presented for the least accurate coefficient, λ_β, and the least accurate of the standard errors of the coefficients, λ_σ. Table 3 presents results of the data set Chwirut2 for Package Z using various options for the calculation of first derivatives and the convergence tolerance. Clearly, these options can make a large difference in the accuracy of the solution obtained.

In the process of assessing numerical accuracy, the existence of nonlinear benchmarks can shed light on two important questions concerning nonlinear estimation:

- Should default options be relied upon? The computed solution depends on the convergence tolerance (e.g., 1E–8), the method of solution (e.g., Gauss–Newton or Levenberg–Marquardt), and the convergence criterion (e.g., residual sum of squares (RSS) or square of the maximum of the parameter differences (PARAM)). Frequently, varying these options changes the computed solution
- Are analytic derivatives worth the effort? Analytic derivatives deliver more accuracy than their finite-difference approximations. Yet, user-supplied gradients and Hessians often are viewed as more trouble than they are worth, and users frequently rely on numerical derivatives when they could supply analytic derivatives (Dennis 1984). Monte Carlo evidence in Donaldson and Schnabel (1987) shows that such an approach leads to a loss of accuracy. Specific examples will be given in Part II.

A related question concerns packages which offer default initial conditions for nonlinear estimation: should they be relied upon? Default initial conditions rarely coincide with initial conditions crafted to fit the problem at hand, yet as Press et al. (1992, p. 341) have noted, "It cannot be overemphasized, however, how crucially success depends on having a good first guess for the solution[.]"

In view of these considerations, the nonlinear benchmarks should be run several ways to determine a "preferred combination" of options. Having determined the preferred combination, run all benchmarks using Start I. Report the LREs of any solution produced. When no solution is produced (e.g., failure to converge or abnormal end), tinker with the preferred combination in order to find a solution from Start I (e.g., vary the tolerance, switch from Gauss–Newton to Levenberg–Marquardt, if default derivatives are numerical then change to analytic derivatives, etc.). If none of these produces an answer, then switch procedures completely; for example, from unconstrained nonlinear estimation to constrained nonlinear estimation (but with null constraints); if this produces a solution, it can be footnoted. In sum, make an all-out effort to obtain a solution from Start I. The only proscription is that the objective function cannot be reparameterized (though it may be necessary to reparameterize analytic derivatives to achieve a solution). Only if all these methods fail should Start II be used. In particular, if Start I yields a solution with zero accurate

Table 3. Results of Chwirut2 Data Set for Package Z

Derivatives	Tolerance	λ_β	λ_σ
numerical	1E–6	6.0	3.8
numerical	1E–12	6.1	3.9
analytic	1E–6	6.6	7.2
analytic	1E–12	10.7	11

digits and Start II yields several accurate digits, report the Start I LREs. In general, a Start I solution is preferred to a Start II solution, and a higher LRE is preferred to a lower LRE with the following exception: reporting Start II with a nonzero LRE is preferable to reporting Start I with a zero LRE, because in the former case at least it is obvious that no correct solution was produced by Start I.

For packages that allow user access to computed results, such as SAS and S-Plus, running so many nonlinear regressions is not burdensome. However, it is a burdensome task for a package which requires the user to save output as ASCII and cut-and-paste to another program in order to calculate LREs. In a more comprehensive assessment, as might be undertaken when focusing on only a single package, the StRD can also be applied to other procedures. For example, by formulating an objective function as the negative of a sum of squared residuals, the nonlinear benchmarks can be applied to function maximization routines. Some of the nonlinear problems also are amenable to testing partially linear routines of the Golub–Pereyra type. Often it may be instructive to present nonlinear results for both default options and a preferred combination. Results from default options typically have fewer digits of accuracy, and more "zero digits of accuracy" cases.

Attention should be paid to the method of calculating the covariance matrix. The StRD standard errors are computed using the product of the gradient and contain a degrees of freedom adjustment, and so can be used to assess only packages which employ a similar method. In particular, if a package uses another method for calculating the standard errors based, say, on the inverse of the Hessian, it would be inappropriate to calculate LREs for those standard errors. Finally, it is important to run the tests not only for Start I and Start II (footnoting those cases where a solution is achieved for Start I but not Start II), but also for Start III: using the solution as a starting value. This can produce anomalous behavior, of which mention should be made should it occur.

5. TESTING THE RANDOM NUMBER GENERATOR

The use of random number generators (RNGs) has surged in recent years. Good introductions are Knuth (1997) or L'Ecuyer (1994, 1998). A more in-depth, yet still nontechnical treatment is Gentle (1998b). Bootstrapping, Monte Carlo, and other such techniques place ever-increasing demands on a statistical package's RNG. Theoretical advances in random number generation have kept pace with these demands (L'Ecuyer 1994). Whether these theoretical advances have been practically implemented in statistical software is an open question.

An RNG is said to be *reproducible* if the same sequence of random numbers can be produced at will, perhaps by specifying a seed which initializes the sequence. An RNG should be reproducible for debugging purposes. The *period* of an RNG is the number of calls which can be made to the RNG before it begins to repeat itself. The period should be long because the period should be a least an order of magnitude larger than the square of the number of values used (Ripley 1987, p. 26); that is, if p is the period and n is the number of calls to the RNG, then $p \gg 200n^2$ (Ripley 1990). The reason that only a fraction of the RNG's period should be used is because the discrepancy between the RNG's output and true randomness increases as $n \to p$ (L'Ecuyer 1998, sec. 4.2.3). Thus, if $n = 1000$ then $p \approx 2^{31}$ is acceptable, whereas if $n =$ one million then p should be at least 2^{50}. Knuth (1997, p. 195) recommended a more modest approach, suggesting that at most $p/1000$ calls be made to the RNG. Even then, if $n =$ one million, p should be 2^{30}, which nearly exhausts $p \approx 2^{31}$ generators common in PC software.

The implications of period length for applied research are enormous. A modest double bootstrap with 1999 first-stage and 250 second-stage resamples for a sample size of 100 requires 50 million calls to the RNG, implying a period of at least 500 quadrillion (by Ripley's rule) or 50 billion (by Knuth's rule). The situation is even more stark for Monte Carlo studies. The Monte Carlo of the double bootstrap by Letson and McCullough (1998) required 45 billion random numbers, and MacKinnon's (1996) investigation of unit root tests required nearly 100 billion.

Since the RNG is intended to produce uniform numbers, its output should pass tests for uniformity. Sawitzki (1985) tested the RNG for IBM PC BASIC and found not only that it has a very short period ($p \approx 2^{16} = 65,536$; per Ripley's suggestion useful only for 20 calls!) but that its output was decidedly not uniform. As Marsaglia (1968) showed, some RNGs produce sequences which are correlated in k-space. Therefore, even if the RNG passes univariate tests for uniformity (Stephens 1986), it should be subjected to multivariate tests as well (Tezuka 1995; Marsaglia 1993), though for practical reasons the number of dimensions is usually eight or fewer. The infamous RANDU (IBM 1968, p. 77) generator distributed with the IBM 360 mainframe computer also was decidedly nonrandom. Complicating matters for users, RANDU was widely imitated, and recommended in textbooks long after its faults were well-known (Park and Miller 1988, p. 1198).

Ripley (1990) listed the desiderata of an RNG:

1. be reproducible from a simply specified starting point;
2. have a very long period;
3. produce numbers which are a very good approximation to a uniform distribution;
4. produce numbers very close to independent output in a moderate number of dimensions.

An RNG which met these criteria only a few years ago might now be deemed completely unacceptable, due primarily to deficient period length. If an RNG has a deficient period, its period often can be increased by shuffling (Press et al. 1992), but before a user can know be aware of this situation, the vendor must make known the RNG's period. However, if an RNG fails empirical tests of the "birthday spacings" or "random walk" types, its shuffled output will not be appreciably more random (Knuth 1997, p. 34).

Coding statistical tests for RNGs is an arduous task. DIEHARD is automated (with executables for DOS and

C, and source code provided), thus greatly facilitating the testing of RNGs. DIEHARD presumes that the RNG to be tested can produce 32-bit random integers, but a good $p \approx 2^{31}$ RNG will pass almost all the tests. It includes 18 randomness tests, some with many variations.

Details of the tests are described in the DIEHARD documentation, as is interpretation of the tests. Some of the tests are also discussed in Knuth (1997, sec. 3.3), Marsaglia (1993), and Gentle (1998b). Marginal rejections are of no interest; with so many tests statistically significant rejections are bound to occur. Instead, of interest are p-values that are zero or unity to all decimal places reported by DIEHARD, which is evidence of catastrophic failure. Many DIEHARD tests produce two-level statistical tests with Kolmogorov–Smirnoff (K–S) statistics, others produce a series of p-values. Of course, if the null is rejected according to the K–S test, it fails. Even if the K–S test does not reject the null, the RNG fails if one of the p-values upon which the K–S test is based is zero or unity. Similarly, for those tests which report several p-values: if just one of them is zero or unity, the RNG fails. Only the uniform RNG is tested, since other distributions (e.g., normal and Laplacian random numbers) typically are created by calls to a uniform RNG.

6. STATISTICAL DISTRIBUTIONS

The accuracy of statistical distributions (e.g., tail areas and percentiles) cannot be taken for granted. Inaccuracies have been documented in Gauss v3.2.6 (Knüsel 1995) which were still uncorrected in Gauss v3.2.13 (Knüsel 1996), and also in Excel97 (Knüsel 1998). Functions for evaluating statistical distributions typically do not have closed form expressions, and can be approximated in a variety of ways (Kennedy and Gentle 1980, ch. 5), some more accurate than others. Brown and Levy (1994), for example, assessed several algorithms for the incomplete beta function, and found only one of them to be reliable. Too, the algorithms for some approximations converge faster in the central region than in the tails, and conversely for other distributions. Details of numerically stable computation of statistical functions are explored in Knüsel (1986) and Knüsel and Bablok (1996). Crude approximations accurate to two or three digits may suffice for calculating p-values, but for most other applications several digits of accuracy are required. A program should compute the desired probability to at least six significant digits, with relative error smaller that 1E–6 (so that the exact answer rounds to the displayed result). If not, the result should be set to zero or an error message should be displayed. Further details on the accuracy a user ought to expect are in Knüsel (1995).

Vendors rarely provide any information about their statistical distributions, presenting them to users as black boxes, whose output is implicitly represented as reliable to all displayed digits. It is not uncommon, though, for packages to display several digits for crude approximations which are accurate to only a couple digits. Two more common problems are worth noting. First, numerical underflow varies from distribution to distribution, and the user is left to discover this limit for himself for each distribution. For example, if a probability p is reported as zero for the Poisson distribution, this might mean $p < 1E-16$, whereas it might mean $p < 1E-308$ for the binomial distribution. This should be noted in the reference manual. Second, many packages compute only lower tail probabilities, $P(X \leq x)$ leaving upper tail probabilities, $P(X > x)$ to be calculated via complementation. With highly accurate approximations this usually is not a problem for the central region or tail of a distribution (for crude approximations it is a problem), but the extreme tail is another matter. When the desired probability in the upper tail is near zero and its complement is near unity, cancellation errors can produce "answers" which are off by an order of magnitude or more.

To see this, consider calculating $p = P(X > 265)$ for a χ^2 distribution with 100 degrees of freedom. SAS 6.12, SPSS 7.5, and S-Plus 4.0 will compute only $P(X \leq 265)$ so the requisite calculation is $p = 1 - P(X \leq 265)$. SAS, SPSS, and S-PLUS all calculate $p = 1.1102$E–16 while the correct answer is 7.2119E–17. With such programs, small probabilities in the tails ought not be calculated by complementation, and it would be preferable if such upper tail probabilities were computed directly.

The first step in assessing statistical distributions is obtaining a program for calculating exact values. Two noncommercial programs are Knüsel's (1989) ELV program which is available as a DOS executable, and Brown's (1998) DCDFLIB, which is available in C and FORTRAN-77 tar files. ELV offers a greater variety of functions and inverses. Output from these programs is referred to as "exact" percentiles or critical values. Output from the package being assessed is referred to as "estimated" percentiles or critical values. It is impossible to test every possible input for a statistical distribution, and every algorithm will break down for very extreme inputs. Together, these two points suggest a useful testing strategy. Use ELV or DCDFLIB to obtain critical values for the following basic sequence of percentiles (BSP): $\{.0001, .001, .01, .1, .2, \ldots, .9, .99, .999, .9999\}$ (probabilities outside this range are referred to as the "extreme tails"). Feed the exact critical values into the statistical package's distribution procedure to see if the estimated percentile is correct. If the package has an inverse function, feed the BSP into it to determine whether the estimated critical values are correct. If all seems well, then check the extreme tails to determine the limits within which the package's output is reliable. It is important to check the extreme tails, as many sophisticated statistical methods make use of them, sometimes requiring accurate evaluation of probablilities as small as 1E–10 and smaller.

This strategy is easily implemented for the normal distribution. For the chi-square and Student's-t distribution, it might suffice to try for $k = 1, 5, 10, 50, 100, 1000$ where k is the degrees of freedom. For distributions with more parameters, such a semi-thorough approach quickly becomes infeasible, and the existence of an inverse function greatly facilitates the search for inaccuracies. For a variety of different parameters, use the BSP and the statistical package's inverse function to obtain estimated critical values and im-

mediately feed these into the function itself. If the original BSP is not returned, a fruitful area for more thorough investigation has been uncovered. Reasonable parameter values for the F distribution and some of the noncentral distributions can be obtained from *Biometrika* tables.

The relative error (relerr) is used to measure the accuracy probabilities. Essentially, if relerr $<$ 1E–4, then the first four significant digits of the estimated percentile agree with the exact percentile. Critical values are measured by reporting the number of accurate digits. Considering statistical functions such as the F, beta, and the various noncentral distributions, even with an inverse function available it is obvious that testing statistical distributions is an extremely tedious exercise—as tedious as it is important. Useful templates for what an exhaustive investigation might look like are the studies of Gauss and Excel97 by Knüsel (1995, 1998).

7. CONCLUSIONS

Until quite recently, the tools for assessing the reliability of statistical software were quite limited in scope and not readily available. Generally, only entry level tests were available, and even these were not widely used. The release of the NIST StRD, together with programs by Marsaglia, Knüsel, and Brown, changes all that. The present article proposes a methodology based on these tools. In Part II, this methodology will be applied to SAS, SPSS, and S-PLUS.

REFERENCES

Bankhofer, U., and Hilbert, A. (1997), "Statistical Software Packages for Windows: A Market Survey," *Statistical Papers*, 38, 393–407.

Brown, B. W. (1998), DCDFLIB v1.1 (Double Precision Cumulative Distribution Function LIBrary), available at ftp://odin.mdacc.tmc.edu/pub/source

Brown, B. W., and Levy, L. B. (1994), "Certification of Algorithm 708: Significant Digit Computation of the Incomplete Beta," *ACM Transactions on Mathematical Software*, 20, 393–397.

Dahlquist, G., and Björck, A. (1974), *Numerical Methods*, New York: Prentice-Hall.

Dennis, J. E. (1984), "A User's Guide to Nonlinear Optimization Algorithms," in *Proceedings of the IEEE*, 72, 1765–1776.

Donaldson, J. R., and Schnabel, R. B. (1987), "Computational Experience With Confidence Regions and Confidence Intervals for Nonlinear Least Squares," *Technometrics*, 29, 67–82.

Eddy, W. F., Howe, S. E., Ryan, B. F., Teitel, R. F., and Young, F. (1991), *The Future of Statistical Software: Proceedings of a Forum*, Washington, DC: National Academy Press.

Elliott, A. C., Reisch, J. S., and Campbell, N. P. (1989), "Benchmark Data Sets for Evaluating Microcomputer Statistical Programs," *Collegiate Microcomputer*, 11, 289–299.

Francis, I. (1981), *Statistical Software: A Comparative Review*, New York: North-Holland.

Francis, I., Heiberger, R. M., and Velleman, P. F. (1975), "Criteria and Considerations in the Evaluation of Statistical Software," *The American Statistician*, 29, 52–56.

Gentle, J. (1998a), *Numerical Linear Algebra with Applications in Statistics*, New York: Springer.

——— (1998b), *Random Number Generation and Monte Carlo Methods*, New York: Springer.

Goldberg, D. (1991), "What Every Computer Scientist Should Know about Floating-Point Arithmetic," *ACM Computing Surveys*, 23, 5–48.

Hammarling, S. (1985), "The Singular Value Decomposition in Multivariate Statistics," *ACM Special Interest Group on Numerical Mathematics*, 20, 2–25.

Higham, N. J. (1996), *Accuracy and Stability of Numerical Algorithms*, Philadelphia: SIAM.

IBM (1968), *System/360 Scientific Subroutine Package, Version III, Programmer's Manual*, White Plains, NY: author.

Kennedy, W. J., and Gentle, J. E. (1980), *Statistical Computing*, New York: Marcel-Dekker.

Knüsel, L. (1986), "Computation of the Chisquare and Poisson Distribution," *SIAM Journal on Scientific and Statistical Computing*, 7, 1022–1036.

——— (1989), "Computergestützte Berechnung Statistischer Verteilungen," Oldenburg, München-Wien (an English version of the program can be obtained at www.stat.uni-muenchen.de/~knuesel/elv)

——— (1995), "On the Accuracy of Statistical Distributions in Gauss," *Computational Statistics and Data Analyis*, 20, 699–702.

——— (1996), "Telegrams," *Computational Statistics and Data Analysis*, 21, 116.

——— (1998), "On the Accuracy of Statistical Distributions in Microsoft Excel 97," *Computational Statistics and Data Analysis*, 26, 375–377.

Knüsel, L., and Bablok, B. (1996), "Computation of the Noncentral Gamma Distribution," *SIAM Journal on Scientific Computing*, 17, 1224–1231.

Knuth, D. E. (1997), *The Art of Computer Programming* (vol. 2, 3rd ed.), Reading, MA: Addison-Wesley.

Lachenbruch, P. A. (1983), "Statistical Programs for Microcomputers," *Byte*, 8, 560–570.

L'Ecuyer, P. (1994), "Uniform Random Number Generation," *Annals of Operations Research*, 53, 77–120.

——— (1999), "Random Number Generation," in *Handbook on Simulation*, ed. J. Banks, New York: Wiley, pp. 93–138.

Lesage, J. P., and Simon, S. D. (1985), "Numerical Accuracy of Statistical Algorithms for Microcomputers," *Computational Statistics and Data Analysis*, 3, 47–57.

Letson, D., and McCullough, B. D. (1998), "Better Bootstrap Confidence Intervals: The Double Bootstrap with No Pivot," *American Journal of Agricultural Economics*, 80, 552–559.

Ling, R. F. (1974), "Comparison of Several Algorithms for Computing Sample Means and Variances," *Journal of the American Statistical Association*, 69, 859–866.

Longley, J. W. (1967), "An Appraisal of Computer Programs for the Electronic Computer from the Point of View of the User," *Journal of the American Statistical Association*, 62, 819–841.

Marsaglia, G. (1968), "Random Numbers Fall Mainly in the Planes," in *Proceedings of the National Academy of Sciences of the United States of America*, 60, pp. 25–28.

——— (1993), "Monkey Tests for Random Number Generators," *Computers and Mathematics with Applications*, 26, 1–10.

——— (1996), "DIEHARD: A Battery of Tests of Randomness," http://stat.fsu.edu/~geo

MacKinnon, J. G. (1996), "Numerical Distribution Functions for Unit Root and Cointegration Tests," *Journal of Applied Econometrics*, 11, 601–618.

McCullough, B. D. (1997), "Benchmarking Numerical Accuracy: A Review of RATS v4.2," *Journal of Applied Econometrics*, 12, 181–190.

——— (in press), "Wilkinson's Tests and Econometric Software," *Journal of Economic and Social Measurement*.

Park, S. K., and Miller, K. W. (1988), "Random Number Generators: Good Ones are Hard to Find," *Communications of the ACM*, 31, 1192–1201.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992), *Numerical Recipes in FORTRAN* (2nd ed.), New York: Cambridge University Press.

Ripley, B. D. (1987), *Stochastic Simulation*, New York: Wiley.

——— (1990), "Thoughts on Pseudorandom Number Generators," *Journal of Computational and Applied Mathematics*, 31, 153–163.

Rogers, J., Filliben, J., Gill, L., Guthrie, W., Lagergren, E., and Vangel, M. (1998), "StRD: Statistical Reference Datasets for Assessing the Numerical Accuracy of Statistical Software," NIST TN# 1396, National Institute of Standards and Technology.

Sawitzki, G. (1985), "Another Random Number Generator Which Should Be Avoided," *Statistical Software Newsletter*, 11, 81–82.

——— (1994a), "Testing Numerical Reliability of Data Analysis Systems," *Computational Statistics and Data Analysis*, 18, 269–286.

——— (1994b), "Report on the Reliability of Data Analysis Systems," *Computational Statistics and Data Analysis (SSN)*, 18, 289–301.

Simon, S. D., and Lesage, J. P. (1988), "Benchmarking Numerical Ac-

curacy of Statistical Algorithms," *Computational Statistics and Data Analysis*, 7, 197–209.

——— (1989), "Assessing the Accuracy of ANOVA Calculations in Statistical Software," *Computational Statistics and Data Analysis*, 8, 325–332.

Stephens, M. A. (1986), "Tests for the Uniform Distribution," in *Goodness-of-Fit Techniques*, eds. R. D'Agostino and M. Stephens, New York: Marcel-Dekker, pp. 331–366.

Stewart, G. W. (1996), *Afternotes on Numerical Analysis*, Philadelphia, PA: SIAM Press.

Tezuka, S. (1995), *Uniform Random Numbers: Theory and Practice*, Boston: Kluwer.

Thisted, R. A. (1988), *Elements of Statistical Computing*, New York: Chapman and Hall.

Veall, M. R. (1991), "Shazam 6.2: A Review," *Journal of Applied Econometrics*, 6, 317–320.

Vinod, H. D. (1989), "A Review of Soritec 6.2," *American Statistician*, 43, 266–269.

Wampler R. H. (1970), "A Report on the Accuracy of Some Widely-Used Least Squares Computer Programs," *Journal of the American Statistical Association*, 65, 549–565.

——— (1980), "Test Procedures and Test Problems for Least Squares Algorithms," *Journal of Econometrics*, 12, 3–22.

Wilkinson, L. (1985), *Statistics Quiz*, Evanston, IL: SYSTAT, Inc. (available at http://www.tspintl.com/benchmarks)

——— (1994), "Practical Guidelines for Testing Statistical Software," *Computational Statistics*, eds. P. Dirschedl and Rüdiger Ostermann, Berlin: Physica-Verlag, pp. 111–124.

Wilkinson, L., and Dallal, G. E. (1977), "Accuracy of Sample Moments Calculations Among Widely Used Statistical Programs," *The American Statistician*, 31, 128–131.