Journal of Economic Literature
Vol. XXXVII (June 1999), pp. 633–665

The Numerical Reliability of Econometric Software

B. D. McCULLOUGH
and
H. D. VINOD[1]

1. *Introduction*

> Numerical software is central to our computerized society; it is used . . . to analyze future options for financial markets and the economy. It is essential that it be of high quality; fast, accurate, reliable, easily moved to new machines, and easy to use. (Ford and Rice 1994)

APART FROM COST considerations, economists generally choose their software by its user-friendliness or for specialized features. They rarely worry whether the answer provided by the software is correct (i.e., whether the software is reliable). The economist, whose degree is not in computer science, can hardly be faulted for this: is it not the job of the software developer to ensure reliability? Caveat emptor. Would a reviewer notice if the software is inaccurate? We think not. We surveyed five journals that regularly publish reviews of econometric software. For the years 1990–97, over 120 reviews appeared. All but three paid no attention to numerical accuracy, and only two applied more than a single test of numerical accuracy (Michael Veall 1991, and McCullough 1997, but see also Vinod 1989 and Colin McKenzie 1998). Since computation is the raison d'etre of an econometric package, this lacuna is all the more puzzling given the failure of many statistical packages to pass even rudimentary benchmarks for numerical accuracy (James Lesage and Stephen Simon 1985; Bernhard, Herbold, and Meyers 1988; Günther Sawitkzi 1994b; Udo Bankhofer and Andreas Hilbert 1997). One would think that similar assessments of econometric software have been conducted, but the economics profession has no history of benchmarking econometric software.

1.1 *Econometric Software Has Bugs*

Consider full information maximum likelihood (FIML) estimation of Klein's Model I using Klein's original data. The consumption (C_t), investment (I_t), and wage (W_t) equations are given by

$$C_t = \alpha_0 + \alpha_1(W_t^p + W_t^g) + \alpha_2 P_t + \alpha_3 P_{t-1}$$

$$I_t = \beta_0 + \beta_1 P_t + \beta_2 P_{t-1} + \beta_3 K_{t-1}$$

$$W_t = \gamma_0 + \gamma_1 E_t + \gamma_2 E_{t-1} + \gamma_3 (t - 1931).$$

[1] McCullough: Federal Communications Commission. Vinod: Fordham University. For comments and useful suggestions, thanks to Reginald Beardsley, Robert Cavazos, Francisco Cribari, Douglas Dacy, Jerry Duvall, William Greene, David Kendrick, Robert Kieschnick, David Letson, Charles Renfro, and Janet Rogers. We are especially indebted to Frank Wolak and two referees, who made substantial contributions. The views expressed herein are the authors', and not necessarily those of the Commission. Email: bmccullo@fcc.gov and vinod@murray.fordham.edu

TABLE 1
VARIOUS FIML RESULTS FOR KLEIN'S MODEL I
(standard errors in parentheses, asterisk denotes 5% significance)

	α_0	a_1	a_2	α_3
Berndt	17.165*	0.791*	−0.062	0.310
	(7.363)	(0.066)	(1.09)	(0.629)
Greene	17.8*	0.853*	−0.214	0.351*
	(2.12)	(0.047)	(0.096)	(0.101)
C & P	18.34*	0.8018*	−0.2324	0.3857
	(2.485)	(0.0359)	(0.3120)	(0.2174)
	β_0	β_1	b_2	β_3
Berndt	29.837	−0.625	1.020	−0.173
	(23.77)	(1.48)	(0.999)	(0.106)
Greene	17.2*	0.130	0.140)	−0.136*
	(6.47)	(0.137)	(0.613)	(0.038)
C & P	27.26*	−0.8010	1.052*	−0.1481*
	(7.938)	(0.4914)	(0.3525)	(0.0299)
	γ_0	γ_1	g_2	γ_3
Berndt	4.790	0.278*	0.257*	0.213*
	(4.82)	(0.084)	(0.052)	(0.078)
Greene	1.41	0.498*	0.087*	0.403*
	(0.943)	(0.018)	(0.015)	(0.021)
C & P	5.794*	0.2341*	0.2847*	0.2348*
	(1.804)	(0.0488)	(0.0452)	(0.0345)

William Greene (1997, p. 760), Ernst Berndt (1990, p. 553), and Giorgio Calzolari and Lorenzo Panattoni (1988) present FIML parameter estimates and standard errors as found in Table 1. Note that the magnitude, significance, and even sign of the parameter estimates differ. Which set of estimates, if any, is correct? How much faith can be placed in the FIML estimates reported in journal articles?

As another example, Michael Lovell and David Selover (1994) attempted to fit a Cochrane-Orcutt AR(1) correction to three data sets by means of four different packages. For the first data set, estimates of ρ ranged from 0.36 to −0.79, with slope estimates for the parameter of interest ranging from −35.1 to −30.96. For the second data set, ρ estimates ranged from 0.31 to 0.93, with slope estimates ranging from −2.83 to 3.06. For the third data set, ρ estimates ranged from 0.84 to 1.001, with slope estimates ranging from 0.26 to 0.92. Sometimes these differences could be traced to differences in algorithms, in which case the differences are acceptable. Other times, they could not. Paul Newbold, Christos Agiakloglou and John Miller (1994) used 15 packages to fit ARMA models to five separate time series, with similar results. They also documented cases in which two packages generate the same parameter estimates, but markedly different forecasts. This has important implications for estimates of the long-run persistence of macroeconomics shocks (i.e., cumulative impulse response functions), as in John Campbell and N. Gregory Mankiw (1987). Such

important results might be quite sensitive to the choice of software, a possibility overlooked in textbooks.

When discussing the solution to nonlinear estimation problems, textbooks invariably mention that for a given problem, one algorithm might yield a solution, while another algorithm might fail to produce a solution. Just as invariably, no mention is made that when both algorithms solve the problem, one solution is likely to be more accurate than the other. As our FIML example makes clear, there is a distinct possibility that neither "solution" is correct, but this issue is never raised. Neither do textbooks warn students that even simple linear procedures, such as calculation of the correlation coefficient, can be horrendously inaccurate.

Suppose a multiple regression produced a high R^2 with low t-statistics. A first thought might be "multicollinearity." Standard practice in this situation is to compute the correlation matrix of the independent variables. Any conclusion regarding the correlation of the independent variables would be critically dependent upon the package used. Leland Wilkinson (1985, test II-D) has a test for the accuracy of computing a correlation matrix. His six variables (X, BIG, LITTLE, HUGE, TINY, and ROUND) all are linear transformations of each other, and so are perfectly correlated. Therefore the correlation matrix should be all units. The standard deviation of each variable should be 2.738 raised to some (possibly negative) power of ten. In Table 2 the output of four popular econometric packages is presented.[2] While package "X1" gives the correct answer, programs "X2" and "X3" do not. Package "X4" even manages to report correlation coefficients in excess of unity! We also note that programs "X2" and "X4" do not correctly calculate all the standard deviations for the variables in question.

1.2 *Why Has No One Noticed?*

It is understandable that economists have paid little attention to whether or not econometric software is accurate. Until recently, econometrics texts rarely discussed computational aspects of solving econometric problems (but see Russell Davidson and James MacKinnon 1993, §1.5; Greene 1997, §5.2). Many textbooks convey the impression that all one has to do is use a computer to solve the problem, the implicit and unwarranted assumptions being that the computer's solution is accurate and that one software package is as good as any other. Statisticians are more likely to be versed in statistical computing and numerical analysis and to know that accuracy cannot be taken for granted. Yet even reviews of statistical software pay little attention to accuracy. In some quarters, complaints are raised against purveyors of inaccurate software, but as economists we take a more sanguine view. While the purveyance of inaccurate software is perhaps regrettable, it is predictable: the market provides us not necessarily with what we need, but with what we want; and we want speed, user-friendliness, and the latest econometric features.

The market forces that militate against the supply of accurate software are twofold. First, how often do advertisements for econometric software

[2] We have elected not to identify software packages by name for two reasons. First, we regard published software reviews as a more suitable vehicle for providing full and fair assessments of individual packages. Second, some developers are remarkably quick to respond to reports of errors, and many of the errors we recount were fixed even before this article went to press.

TABLE 2
MATRICES OF CORRELATION COEFFICIENTS

	X	BIG	LITTLE	HUGE	TINY	ROUND	st. dev.
X	1.0						2.738
BIG	1.0	1.0					2.738
LITTLE	1.0	1.0	1.0				2.738E–8
HUGE	1.0	1.0	1.0	1.0			2.738E+12
TINY	1.0	1.0	1.0	1.0	1.0		2.738E–12
ROUND	1.0	1.0	1.0	1.0	1.0	1.0	2.738
			"Package X1" (correct answer)				
X	1.0						2.738
BIG	0.867	1.0					4.216
LITTLE	0.863	0.614	1.0				3.518E–8
HUGE	1.0	0.866	0.836	1.0			2.738E+12
TINY	1.0	0.866	0.836	1.0	0.0		2.738E–12
ROUND	1.0	0.866	0.836	1.0	1.0	1.0	2.738
			"Package X2"				
X	0.999						2.738
BIG	0.645	1.0					2.738
LITTLE	0.819	0.577	1.0				2.738E–8
HUGE	1.0	0.645	0.820	1.0			2.738E+12
TINY	1.0	0.645	0.820	1.0	1.0		2.738E–12
ROUND	0.999	0.645	0.820	1.0	1.0	0.999	2.738
			"Package X3"				
X	1.0						2.738
BIG	1.129	1.127					2.424
LITTLE	1.007	1.137	1.013				2.87E–8
HUGE	1.0	1.130	1.007	1.0			2.738E+12
TINY	1.0	1.130	1.007	1.0	0.0		2.738E–12
ROUND	1.0	1.130	1.007	1.0	1.0	1.0	2.738
			"Package X4"				

feature "speed of solution"[3] as opposed to "accuracy of solution"? Frequently there is a trade-off between speed and accuracy, and often the fastest way to compute is not the most accurate way to compute. William Kahan (1997) has noted the deleterious effects of this overarching emphasis on speed at the expense of accuracy, and in a field where one might hope that consumers would know better: computer science. If the same affliction bedevils the profession of computer science, the economics profession can hardly be faulted. Second, not only consumers' emphasis on speed, but consumers' emphasis on new features militates against more accurate econometric software. Much of software development is the incorporation of the latest econometric procedures. Designing and testing software for accuracy is extremely labor intensive, and the developer who seeks to ensure accuracy may have to delay implementation of new features. Thus, an accuracy-enhanced upgrade would lack features possessed by other products.

[3] Some common measures of "speed" as found in advertisements and software reviews are misleading, such as "the number of seconds required to invert a 100x100 matrix 1000 times." Such looping procedures can exaggerate the influence of the cache dramatically, especially when the cache hit rate reaches 100 percent after the first loop. See Weicker (1984).

In a market that demands speed and new features with little emphasis on accuracy, the developer who attempts to enhance accuracy could lose out to the competition. When numerical accuracy is not a criterion for evaluation (e.g., Jeffrey MacKie-Mason 1992), a clear signal is sent to software developers that allocating resources toward numerical accuracy is not a profit-maximizing strategy. Developers are merely satisfying demand—if we do not demand some essential feature like numerical accuracy, it is no one's fault but our own.

We believe that if the consumers of econometric software were aware of the extent of numerical inaccuracies in econometric software, developers would have the incentive to spend more time ensuring accuracy, and could do so without losing market share. To this end, then, a goal of this article is to show consumers of econometric software that accuracy cannot be taken for granted, and that conversations about econometric software should begin with accuracy, and only then turn to speed and user-friendliness. This, in turn, will provide developers the incentive to supply that accuracy.

1.3 *Benchmarking*

Thirty years ago James Longley (1967) worked out by hand the solution to a regression of "total employment" on GNP, the GNP deflator, Unemployment, Size of the Armed Forces, Population, and Time, for the sixteen years 1947–62, and he did so to ten significant digits. He compared these results with those from a variety of mainframe regression packages and discovered that most programs produced drastically incorrect answers: "With identical inputs, all except four programs produced outputs which differed from each other in every digit" (Longley 1967, p. 822). One program gave one digit of accuracy, two gave four-digit accuracy, and another gave either zero or one-digit accuracy for each coefficient. Longley traced the source of many failures to poor choices of algorithms. While Longley wrote thirty years ago, the lesson learned remains: software reliability cannot be taken for granted.

The statistical literature has a long history of concern for software reliability (Ivor Francis, Richard Heiberger, and Paul Velleman 1975; Albert Beaton, Donald Rubin, and John Barone 1976; Francis 1981, 1983; Eddy et al. 1981), and this concern has produced many benchmarks. In addition to the Longley Benchmark, Roy Wampler (1980) proposed an entire suite of linear regression benchmarks. Lesage and Simon (1985) and Simon and Lesage (1988) constructed benchmark tests for univariate summary statistics and the analysis of variance. Finally, Alan Elliott, Joan Reisch and Nancy Campbell (1989) and P. Lachenbruch (1983) provided benchmarks for elementary statistical software packages. This trend most recently culminated in Sawitzki (1994), who proposed a testing strategy for assessing the reliability of statistical software. Recognizing that stringent testing is worthwhile only after entry-level tests are passed, he proposed L. Wilkinson's (1985) tests as a set of minimal standards. Sawitzki (1994a), Wilkinson (1994), and Bankhofer and Hilbert (1997) applied the Wilkinson tests to several well-known statistical packages, and the results were less than impressive: all products failed some of these entry-level tests. McCullough (1999b) applied the Wilkinson Tests to several econometric packages, which fared about as well as the statistical packages did. That is to say, there is definite

room for improvement in the numerical accuracy of econometric software.

With respect to accuracy and economics, a common sentiment is that economic data are accurate only to a few digits, and more accuracy than that is not necessary, so worrying about 10 digits of accuracy in econometric calculations is pointless. We partially agree with this view, but make an important distinction. As expressed above, this sentiment conflates how output is calculated and to what end the output is used. A better expression is: since economic data are accurate only to a few digits, reporting 10 digits of a final answer is pointless; however, all intermediate calculations should be carried out to as many digits as possible. The dichotomy between calculation of output and use of output is important. It may well be the case that there is no practical difference between two packages' estimates of a correlation coefficient, $\hat{\rho}_1 = 0.95$ and $\hat{\rho}_2 = 0.92$. However, if the exact result is 0.95 and a good implementation of a good algorithm will achieve that result, then package two's correlation routine is atrocious. This is because it produces only one accurate digit in a setting where even a fair implementation of a fair algorithm will produce seven digits of accuracy in double precision. Specifically, $\hat{\rho}_2 = 0.92$ is evidence of bad software.

Some readers might be dismayed at becoming aware of the extent of inaccuracies in econometric software, and their confidence in the reliability of numerical computation might be shaken. The inaccuracies we recount have long existed in all manner of computational software, econometric and other. Only recently have the tools for diagnosing and remedying many of these deficiencies become available, so only recently are users and developers becoming more aware of these problems. The foreword to a recent text on the subject (Francoise Chaitin-Chatelin and Valérie Frayssé 1996) addresses this precise point: "In some sense, that awareness [of inaccuracy] is no bad thing, so long as the positive aspects of the understanding of finite precision computation are appreciated."

Before finite precision computation can be appreciated, some acquaintance with how computers handle numbers is necessary. Therefore, Section Two discusses computer arithmetic and errors in computation. It emphasizes: (1) that two algebraically equivalent methods may have drastically different effects when implemented on a computer; and (2) that "small" differences in the input or the algorithm can produce "large" changes in output. Section Three documents inaccuracies in existing econometric software, and suggests useful benchmark collections for testing general routines. Section Four argues that specialized routines, specific to economics, need benchmarks. Section Five discusses random number generators and how to test them. Section Six discusses statistical distributions (e.g., for calculating p-values) and how to test them. Section Seven offers the conclusions.

2. *Computers and Software*

Computers are exceedingly precise and can make mistakes with exquisite precision. Certain tasks which are frequently repeated, such as rounding a sum, need to be exceedingly accurate. Consider rounding to three decimals a number whose magnitude is about one thousand. Should 1000.0006 be rounded up to 1000.001 or rounded down to 1000.000? Perhaps the answer is obvious, but what if the number in question was 1000.0005? Both rounding up and rounding down will inject a bias

into the final result, so perhaps the answer is not so obvious.[4]

Improper attention to the method of rounding can produce disastrous results. *The Wall Street Journal* (November 8, 1983, p. 37) reported on the Vancouver Stock Exchange, which created an index much like the Dow-Jones Index. It began with a nominal value of 1,000.000 and was recalculated after each recorded transaction by calculation to four decimal places, the last place being truncated so that three decimal places were reported. Truncating the fourth decimal of a number measured to approximately 10^3 might seem innocuous. Yet, within a few months the index had fallen to 520, while there was no general downturn in economic activity. The problem, of course, was insufficient attention given to the method of rounding. When recalculated properly, the index was found to be 1098.892 (*Toronto Star*, November 29, 1983).

2.1 *Computer Arithmetic*

To a certain degree, all computers produce "incorrect" answers due to the computer's lack of an infinite word length to store numbers. A computer's arithmetic is different from the one people apply with paper and pencil. While people calculate using decimal (base-10) representations for numbers, computers calculate using base-2. As an example, the decimal number 23 has the base-2 representation 10111 since $1(2^4) + 0(2^3) + 1(2^2) + 1(2^1) + 1(2^0) = 23$, which is denoted $\phi_2(23) = 10111$. Similarly, $\phi_2(10) = 1010$ and the decimal 0.5 has an exact base-2 representation $\phi_2(0.5) = 0.1$ since $0.5 = 1(2^{-1})$. The decimal 0.1 has an infinite (but periodic) binary representation with period four, $\phi_2(0.1) = 0.0001\overline{0011}$ where an overbar indicates infinite repetition. However, a computer has finite storage. If it has 23 bits of storage to the right of the decimal, it will hold the decimal 0.1 in memory as the binary number $\hat{\phi}_2(0.1) = 0.00011001100110011001100$ where $\hat{\phi}_2(\cdot)$ is the stored version of $\phi_2(\cdot)$. Since $\varphi_2(0.1)$ is an infinitely repeating number, the stored version $\hat{\phi}_2(0.1)$ is not exactly equal to the decimal 0.1 it represents. If the stored binary number $\hat{\phi}_2(0.1)$ is reconverted to decimal, it becomes 0.09999999403953. Thus, the computer "sees" 0.1 as something slightly less than 0.1. This has some interesting implications. First, it implies that rescaling a number by 10 can cause a loss of precision, since the exponent is stored base-2 rather than base-10. Second, reading data and performing a units conversion is different from reading already-converted data, a most discomfiting situation for those who encounter it. These problems arise not due to the use of base-2 per se, but due to the combination of base-2 and finite precision.

A *floating point* binary number has a fixed number of places, say 24, with a decimal point which can be placed anywhere. For example, $\hat{\phi}_2(10) =$ 000000000000000000001010.0, $\hat{\phi}_2(0.1) =$ 0.000110011001100110011001 and the base-2 representation of their sum is $\hat{\phi}_2(10.1) =$ 1010.000110011001100110011. When this sum is reconverted to decimal, it becomes 10.0999985. A real number is represented in the floating point format: $s \times M \times B^E$; where s is the sign (zero for positive or unity for negative), B is the base of the representation (usually 2), E is the exponent, and M is a positive integer mantissa. A useful way to view this is as M being a string of zeroes and ones, with B^E placing a decimal point somewhere in the string (or

[4] A common solution is to round to the nearest even digit, e.g., 1.005 becomes 1.00 while 1.015 becomes 1.02. This scheme is called "round-to-even."

to the left or right of the string, implicitly padding with zeroes). If the leading digit is non-zero, the number is said to be *normalized*. The representations of $\hat{\phi}_2(10)$ and $\hat{\phi}_2(0.1)$ above are not normalized, whereas that of $\hat{\phi}_2(10.1)$ is. While under certain circumstances intermediate calculations might involve non-normalized numbers, results are stored in normalized format. Since the first digit in normalized format is always unity, we get an extra bit for the mantissa, referred to as the "hidden bit." [5]

For single-precision, a 32-bit word might be partitioned as follows: one bit for the sign, eight bits for the exponent (which can take integer values from −126 to 127) and 23 bits for the mantissa which, with the hidden bit, yields a 24-bit mantissa, sometimes referred to as "23 + 1" to indicate that the 24th bit comes from the hidden bit. The magnitudes of such single-precision floating point numbers are constrained to lie between $2^{-126} \approx 1.2 \times 10^{-38}$ and $(2 - 2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$, the *range of representable numbers*, with the *underflow threshold* and *overflow threshold* as lower and upper limits, respectively.

Let $\mathbb{R}$ be the familiar real number line, and let $\mathbb{F}$ be the representable numbers. Let $fl(x)$ be the floating point representation of x. That is, $fl(x)$ is the number in $\mathbb{F}$ that corresponds to some number in $x \in \mathbb{R}$. For example, if $x = 0.1$ then $fl(x) = 0.09999999403953$. Thus it is not always true that $x = fl(x)$, i.e., that $x \in \mathbb{R} \Rightarrow x \in \mathbb{F}$. Even when $x, y \in \mathbb{F}$ it does not follow that $(x + y) \in \mathbb{F}$. This is a first suggestion that computer arithmetic is different from pencil-and-paper arithmetic. Let us define *floating point addition* by $fl(x + y)$. Ideally, $fl(x + y)$ is the number in $\mathbb{F}$ that is closest to $x + y \in \mathbb{R}$. The difference, $fl(x + y) - (x + y)$, is *rounding error*, and this difference depends in part on the values of x and y since the representable numbers are not uniformly distributed in base-10. In single precision, there are 8,388,607 floating point numbers between 1 and 2, while between 1023.0 and 1024.0 there are 8,191 floating point numbers. Thus, it can be expected that numbers with larger magnitudes are more susceptible to rounding error than numbers with smaller magnitudes; this supports the recentering and scaling of numbers to mitigate the adverse effects of floating point arithmetic, as is frequently suggested in discussions of collinear data. While frequently it is true that $fl(x + y) \in \mathbb{F}$, much less frequently is it true that $fl(x \cdot y) \in \mathbb{F}$ for floating point multiplication, since the product involves $2s$ or $2s - 1$ significant digits, which extend beyond the s-digit mantissa. Therefore, floating point multiplication is much more likely to be contaminated by rounding error than floating point addition. All this implies that computer arithmetic, contrary to the familiar pencil-and-paper arithmetic, is neither associative nor distributive, though it is commutative. One consequence of this, shown in subsection 2.3, is that absent special precautions, it is possible for the logical statement $(x = y)$ to evaluate "false" while $(x - y = 0)$ evaluates "true." In such a case, logical tests of equality cannot be trusted.

Let $\oplus$ denote any of the four arithmetic operations: $+ - \times \div$ and let $fl(x \oplus y)$ be the floating point representation of $x \oplus y$. Then machine precision, ε, is the smallest value satisfying

$$|s(x \oplus y) - (x \oplus y)| \leq \varepsilon |x \oplus y| \qquad (1)$$

i.e., it is the smallest value for which (1) holds for all $\oplus$ and for all x and y such that the magnitude of $x \oplus y$ is neither greater than the overflow threshold nor

[5] Not all computers use the hidden bit, but it is part of IEEE-754.

less than the underflow threshold. For standard single precision, machine epsilon is $\varepsilon_S = 2^{-24} \approx 5.96 \cdot 10^{-8}$ and for double precision, which uses a 64-bit word-length, it is $\varepsilon_D = 2^{-53} \approx 1.11 \cdot 10^{-16}$. *Accuracy* refers to the error of an approximation, whereas *precision* refers to the accuracy with which basic arithmetic operations are performed. They are the same for scalar computation, e.g., $c = a \cdot b$, but precision can be better than accuracy for nonscalar operations such as matrix inversion; since precision refers to the result of a single calculation, while accuracy can refer to the result of several calculations.

2.2 *Errors in Computation*

When two floating point numbers are added, the smaller (in magnitude) number is right-shifted until the exponents of the two numbers are equal, and then added (this procedure is referred to as *normalization*). While this method does not waste any bits of the mantissa and hence preserves accuracy, the least significant bits of the smaller number are lost when it is right-shifted. This is an example of *roundoff error* or *rounding error*. Roundoff error is a function of hardware and exists because a computer has a finite number of significant digits with which to represent real numbers. For given numbers x, y, and z, when the computer is asked to calculate $xy + z$ what it actually returns is $w = (xy(1+\alpha)+z)(1+\beta)$ where α and β are rounding errors. Usually there are bounds for the rounding errors, such as $|\alpha| < 2^{-53}$ and $|\beta| < 2^{-53}$, and these can be used to bound the total error between the computed w and the actual value of $xy + z$. Such bounds are necessary because clearly $w \neq xy + z$ due to rounding error, and in fact $E[w] \neq xy + z$. Unknown distributional forms and correlations hamper statistical analysis of rounding errors.[6] It makes sense that there are only bounds: if α and β were actually known, they could be subtracted off to yield an exact result.

Even when two numbers are precise to several digits, a single arithmetic operation can introduce sufficient error to ensure that the result is precise to no significant digits. One such occasion is the special case of rounding error called *cancellation error*, which occurs when two nearly equal numbers are subtracted. Nicholas Higham (1996, p. 10) provides an illustrative example. For $f(x) = (1 - \cos x)/x^2$ let $x = 1.2 \times 10^{-5}$. To ten significant digits, $\cos x = 0.9999999999$, so $1 - \cos x = 0.0000000001$ and $f(x) = 0.6944 \ldots$, though by the definition of $\cos x$ the following inequality is true: $0 \leq |f(x)| \leq 0.5$ for all x. The subtraction is exact, but the error in the sole nonzero digit in 0.0000000001 is of the same order as the true answer, and thus the truth is not visible in the final answer. Determining when such adverse results can and cannot happen is the field of *error analysis*: what proportion of the final answer is truth and what proportion is error. Underlying the notion of error analysis is the idea of how accurately each calculation is performed, and how the error from each calculation propogates through subsequent calculations.

Even if computers had an infinite number of significant digits and so had no roundoff error, another type of error

[6] Make the simplifying assumption that rounding errors are independent of x, y and z (they are not, because the floating point numbers are not uniformly distributed; the rounding error is smaller when the magnitude of the number is smaller, as seen in Section 2.1). Trivially, if $E[\alpha] = E[\beta] = 0$ then $E[w] = xy + z + xyE[\alpha\beta]$, and $E[\alpha\beta] \neq 0$ because rounding errors are neither independent nor uncorrelated. Asymptotic theory for dependent and correlated sequences is of little help, because rounding errors routinely violate the Lindberg condition: often a few rounding errors dominate the final error.

would exist as a function of software: *truncation error*. It exists because a program uses finite term approximations. The analysis of truncation error can be said to constitute much of the field of *numerical analysis*. Consider a power series in x whose infinite sum is $\sin(x)$. A computer will truncate the infinite sum by including only a finite number of terms, and in doing so will commit a truncation error. We note in passing that some power series for $\sin(x)$ converge quickly while others do not, thus the choice of algorithm can be crucial. Also, the rate of convergence can depend upon x, converging faster for some values than for others. Indeed, even the order in which summation is undertaken can affect the quality of the result. Germund Dahlquist and Ake Bjorck (1974) showed that in calculating $\Sigma_{n=1}^{10,000} n^{-2}$, reversing the order leads to an error 650 times smaller than summing over increasing n (see also Higham 1996, §4.2–4.5). This is because rounding error accumulates more slowly when small terms are added first. Summing a very large number of very small quantities is not just an exercise for testing software; numerical integration plays an important role in econometrics. Numerical integration becomes particularly difficult when higher moments are involved, because the numerical error becomes more severe. See Vinod and Shenton (1996) for a discussion.

Whether via truncation or roundoff, error is introduced into most any result of an arithmetic operation. As a general rule, successive errors do not offset each other; they accumulate, and this cumulative error is an increasing function of the number of operations. Generally, the total error is of order $N\varepsilon$ where N is the number of operations. In the special case that errors tend to be of opposite sign, the cumulative effect of such errors does not disappear, but produces a total error of order $\sqrt{N}\varepsilon$. We note that there are *multiple precision* arithmetic routines available, which can carry out calculations to 500 digits, thus effectively eliminating roundoff error for many types of problems. However, they are quite specialized and not generally used by economists.

Two methods of solving the same problem can differ dramatically in the number of operations. Consider solving n equations in n unknowns. We all know how to solve such a problem using Cramer's Rule. It can be shown that the number of multiplications and divisions necessary to solve the system is $(n^2-1)n!+n$. The method of Gaussian elimination requires only $\frac{n}{6}(2n^2+9n-5)$ such operations. For $n = 5$ Cramer's Rule requires 2885 multiplications whereas Gaussian elimination requires only 75. With noncancelling errors and machine precision 2^{-22}, the orders of the approximations are 0.0006874 and 0.00001788, respectively. However, if $n = 10$, Gaussian elimination requires 475 operations with error order 0.0001132 while Cramer's rule requires approximately 360 million operations with error order 85.831. Clearly, systems of equations should be solved by Gaussian elimination rather than Cramer's rule (though for econometric work, other methods are preferred to Gaussian elimination). Prima facie it is intuitively desirable to employ methods with fewer operations, not only for the sake of speed but also for the sake of accuracy. This is not always so—sometimes more operations are desired not for sake of speed but for sake of accuracy, e.g., in the calculation of the sample variance (see subsection 2.5). Moreover, not only are some algorithms preferable to others, at an even more fundamental level, some methods of performing calculations are preferable to others.

When a calculation produces a number that is too large, the result is *overflow*, and a number that is too small produces *underflow*. To see overflow, on a handheld calculator with eight digits and without scientific notation, square 99,999,999. Dividing unity by 99,999,999 yields underflow. Overflow is a very stark process, usually resulting in noticeable program failure. Underflow can be pernicious, because when handled improperly it can result in sensible-looking answers that are completely inaccurate. One method of handling underflow is known as *abrupt underflow* (a.u.), in which numbers that are sufficiently small but nonzero often are automatically set to zero. One unfortunate consequence of abrupt underflow is that $x - y = 0$ does not imply $x = y$ (David Goldberg 1991), and logical tests for equality cannot be trusted. To see how this can happen, it is easiest to use decimal arithmetic rather than base-2.

Suppose, then, that the base is 10, the mantissa has three digits, and the exponent ranges from –16 to +15, so that the smallest representable floating point number, say fp_{min}, is 1.00×10^{-16}. Consider two numbers, $x = 5.28 \times 10^{-15}$ and $y = 5.23 \times 10^{-15}$, both greater than fp_{min} by a factor of 10. A test of $x = y$ will return false. Paradoxically, a test of $x - y = 0$ will return true. The reason for this is that $x - y = 0.06 \times 10^{-15} = 6.0 \times 10^{-17}$ when normalized, which is smaller than fp_{min} and hence is set flush to zero, i.e., subjected to *abrupt underflow* (a.u.). In the presence of a.u., no theorems requiring $x - y = 0 \Longleftrightarrow x = y$ can be used to prove anything, since this relation is then only sometimes true, not always true.

As a means of handling the inaccuracy of a.u., I. Goldberg (1967) proposed the method of *gradual underflow* (g.u.). Continuing the above example, when the exponent is at its minimum, –16, fp_{min} no longer is the smallest representable number, because 0.99×10^{-16} is smaller than fp_{min}. Note, however, that 0.99×10^{-16} is not normalized, so such floating point numbers are referred to as subnormals. Subnormal numbers are part of IEEE-754 (IEEE 1985), the rules for computer arithmetic on which hardware has standardized.[7] Virtually all PC and workstation hardware supports the use of subnormal numbers, though some software does not take advantage of this feature. Prior to Demmel (1981), it was commonly thought that a.u. is "harmless" and that the choice of either a.u. or g.u. was innocuous. One of the advantages of IEEE-754 is that it supports g.u., which is a necessary condition for $x - y = 0 \Longleftrightarrow x = y$. Perhaps more importantly, g.u. achieves greater numerical reliability for solving linear systems of equations than a.u. does.

To see this, consider the LU decomposition of a matrix (another way to solve for the least squares coefficients), which produces a lower triangular (L) and upper triangular (U) factorization of a matrix A. Recall from matrix theory that if A is non-singular then the main diagonals of both L and U are non-zero. Demmel (1981) gives the following example. Let

$$A = \lambda \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \tag{2}$$

which is clearly well-conditioned for matrix inversion. Using g.u. produces

[7] As of this writing, some conforming computers are: PCs based on Intel 386, 387, 486, Pentium, and P6 processors and associated clones by Cyrix, IBM, AMD, and TI; Macintosh based on Motorola 68020 + 68881/2 or 68040; IBM RS/6000; PowerPC based PCs and Macintoshes; Suns based on M 68020 + 68881/2 or SPARC chips; DEC Alpha based on DEC 21064 and 21164 chips; Cray T3D based on DEC 21064; and HP based on PA-RISC chips. Exceptions are: Cray X-MP, Y-MP, C90, J90; IBM /370 and 3090; and DEC VAX.

$$L^{gu}U^{gu} = \begin{bmatrix} 1 & 0 \\ 1/2 & 1 \end{bmatrix} \cdot \lambda \cdot \begin{bmatrix} 2 & 3 \\ 0 & 1/2 \end{bmatrix} = A \quad (3)$$

whereas a.u. yields

$$L^{au}U^{au} = \begin{bmatrix} 1 & 0 \\ 1/2 & 1 \end{bmatrix} \cdot \lambda \cdot \begin{bmatrix} 2 & 3 \\ 0 & 0 \end{bmatrix} \neq A \quad (4)$$

Thus, g.u. produces the correct factorization while a.u. incorrectly attributes singularity to the decidedly non-singular matrix A. It is known that a computer that does not support g.u. cannot satisfy some of the LAPACK (Linear Algebra PACKage) benchmarks. To the extent that nonlinear solvers are based on linear approximations, it can be expected that, ceteris paribus, g.u. is also better than a.u. for solving nonlinear equations.

Though many chips fully support IEEE-754 and many compilers and languages support some aspects of IEEE-754, no compiler (as of this writing) fully supports IEEE-754, nor does any language (though FORTRAN 90 is more 754–compliant than FORTRAN 77, and similarly for FORTRAN 95 with respect to FORTRAN 90).[8] Nonetheless, there are many languages and compilers that support g.u., while there are many that do not. Further results on underflow and numerical reliability, in addition to more examples, can be found in Demmel (1984).

2.3 *Misconceptions of Floating-Point Arithmetic*

Having discussed some of the intricacies of computer arithmetic and floating point calculations, it is instructive to dispel what Higham (1996) calls "Misconceptions of Floating Point Arithmetic," of which we mention only three. We mention these not only to illuminate the discussion of floating-point arithmetic, but to drive home the fundamental point that computer math is not at all like pencil-and-paper math.[9]

> Misconception Number One: A short computation that is free of cancellation error, overflow, and underflow must necessarily be accurate.

Consider the following six lines of code:

```
for i = 1:60
    x = x**0.5
end
for i = 1:60
    x = x**2
end
```

where $x{**}0.5 \equiv \sqrt{x}$ and $x{**}2 \equiv x^2$. Note that this algorithm is free from cancellation error, underflow, and overflow. Calculated without error, this algorithm will return x for any non-negative x that is entered, i.e., this algorithm represents the function $f(x) = x, x \geq 0$. However, as seen in the previous subsection, computers necessarily calculate with error, and so there is reason not to be surprised if the computed function, $\hat{f}(x)$, differs from the theoretical function $f(x)$. On Brand X programming software, this algorithm actually computes not $f(x)$ but

$$\hat{f}_1(x) = \begin{cases} 0, 0 \leq x < 1, \\ 1, x \geq 1. \end{cases} \quad (5)$$

On Brand Y programming software a different answer is obtained:

$$\hat{f}_2(x) = \begin{cases} 0, x = 0, \\ 1, x > 0. \end{cases} \quad (6)$$

Observe that $\hat{f}_1$ and $\hat{f}_2$ are completely different functions. Inputting the numbers

[8] IEEE-754 was developed by hardware specialists; programming language and compiler specialists were not involved. Some aspects of IEEE-754 therefore are particularly difficult for compilers and languages to support, but much progress is being made.

[9] See, for example, George Forsythe's (1970) aptly titled article, "Pitfalls in Computation, or Why a Math Book Isn't Enough," for an elementary discussion of why ideas which work in mathematical theory often fail in computational practice. A related article is L. Fox's (1971) "How To Get Meaningless Answers in Scientific Computation (and What To Do About It)."

0.0, 0.1, 0.2, . . . , 1.0, $\hat{f}_1$ returns 10 zeroes followed by unity, while $\hat{f}_2$ returns a zero followed by 10 units. One might be tempted to think that, surely, one or both answers indicate "bad" software; in fact such a conclusion would be unwarranted. There is nothing wrong with the software. The problem is simply that the algorithm exhausts the computer's precision and range. A user should always have some idea of the software's precision and range, and whether his combination of algorithm and data will exhaust these limits.

As a trivial example, a user should not attempt to manipulate ten-digit numbers on a program that uses single precision storage.[10] At the other extreme, a researcher with a hundred thousand observations might choose not to run a regression on a PC. With so many observations, the potential for disastrous cumulated rounding error is an evident concern. Least squares algorithms that are generally robust tend to be memory-intensive, and a PC might not have enough memory to solve a large system with such an algorithm. A less-robust least squares algorithm with less-demanding memory requirements might be able to produce a solution, but the solution might well be so contaminated with rounding error as to be completely unreliable.[11]

Misconception Number Two: Increasing the precision at which a computation is performed increases the accuracy of the answer.

Equation (1) shows that the bound on the error is proportional to machine epsilon. When the error bound is attained, if the same problem is solved in single precision and then again in double precision, the double precision error bound will be smaller than the single precision error bound by a factor of $\varepsilon_D/\varepsilon_S = 2^{-53}/2^{-24} \approx 10^{-9}$, i.e., the double precision answer will have approximately nine more decimal digits correct than the single precision answer. However, actually attaining the error bound is a low probability event, so there is no guarantee that a result computed in t digits of precision will be more accurate than a result computed in s digits of precision for $t > s$ (Higham 1996, §1.13).[12] The "convenient fiction" that increasing the precision necessarily increases the accuracy is just that; nor is it a pathological case that would never happen in practice.

As an example that this is more than a theoretical curiosum, Longley's results with an IBM 360 were more accurate in single precision than in double precision (see Table 10 of Longley 1967, p. 837). Had Longley not worked out the correct answer by hand, many persons naturally would assume that the double precision estimates were more accurate than the single precision estimates. When a user says that he encountered roundoff error in single precision and then switched to double precision to get a better answer, a better interpretation of what he really means is provided by William Press et al. (1992, p. 882), "For this particular algorithm, and my particular data, double precision seemed able to restore my erroneous belief in the 'convenient fiction'." All this is not

[10] Some packages have "single precision storage" with "double precision calculation." The use of "single precision storage" can have adverse effects on accuracy. See McCullough (1999a).

[11] This illustrates one of the many dilemmas confronting developers: how to balance the choice of algorithm against the need to handle large datasets. Further discussion of the developer's viewpoint can be found in Charles Renfro (1997).

[12] J. H. Wilkinson (1963, pp. 25–26) describes the circumstances under which the bound is attained for floating-point multiplication. Not only must each individual error attain its maximum, but the distribution of the multiplicands must follow a special law. Taken together, these imply that attaining the bound is a low probability event. See also William Kennedy and James Gentle (1980, pp. 32–33).

to suggest that double precision should be abandoned in favor of single precision; far from it. The point is simply that, contrary to what intuition based on pencil-and-paper experience might tell us, double precision is not always more accurate than single precision.

> *Misconception Number Three:* The final computed answer from an algorithm cannot be more accurate than any of the intermediate quantities; that is, errors cannot cancel.

Earlier we suggested that rounding errors do not cancel. However, that is a general principle, and a clever programmer can force exception to the rule. We now demonstrate this notion. Consider $f(x)=(e^x-1)/x$. The seemingly natural way to program this function is Algorithm 1:

```
if x = 0
   f = 1
else
   f = (e**x - 1)/x
end
```

but an alternative is Algorithm 2:

```
y = e**x
if y = 1
   f = 1
else
   f = (y-1) / ln(y)
end
```

If $x = 9 \times 10^{-8}$ and final precision $u \approx 6 \times 10^{-8}$, then a precisely calculated solution is $f(x) = 1.00000005$. Algorithm 1, the natural way to program the function, yields 1.32454766 while Algorithm 2 yields 1.00000006. The reason the natural method fails miserably is because the intermediate step of the calculation is imprecise: for $x \approx 0$ the numerator is swamped by rounding and cancellation error. Again let a circumflex (^) denote a computed quantity. In Algorithm 2 it is true that the intermediate quantities $\hat{y} - 1$ and $\ln \hat{y}$ do not approximate $y - 1$ and $\ln y$ for $y \approx 1$. Nonetheless, $(\hat{y} - 1)/\ln \hat{y}$ is an extremely good approximation to $(y - 1)/\ln y$ in that range, because the latter varies slowly and in fact has a removable singularity at the point $y = 1$. This demonstrates that the "natural" way to program an equation may not be the computationally accurate way. Users and developers need to be cognizant of this important point. As an example of this phenomenon, not infrequently a nonlinear solver will fail for one parameterization of an equation, but will produce a solution for another equivalent parameterization.

Again it is seen that computer arithmetic, when performed properly, may not be at all like paper-and-pencil arithmetic. Neither are mistakes computers make like the ones humans make. With paper and pencil, the order in which several numbers are added makes no difference to the final sum; not so with computers. In our arithmetic, if $x - y = 0$ then $x = y$, but not in computer algebra unless the computer supports gradual underflow. Two formulae that are equivalent in our algebra are not necessarily equivalent when programmed into a computer. Such differences as these mean that our everyday notions of arithmetic errors do not coincide with the arithmetic errors made by computers. Even when the arithmetic is good and the bounds on the truncations and rounding errors are tiny, caution still must be observed, for "small" differences can matter appreciably.

2.4 *"Small" Differences Matter*

Recall the dictum from the days of pencil-and-paper computation: if you want one decimal of accuracy, carry your calculations out to two decimals, i.e., if you want n digits of accuracy carry out the calculations to $n + 1$ digits. An example that illustrates the falsity of this dictum is due to J. Vandergraft (1983), in which carrying

calculations out to four digits produces zero digits of accuracy. Consider the regression $Y = \alpha + \beta X + \epsilon$ where

X: 10.00, 10.10, 10.20, 10.30, 10.40, 10.50.

Y: 1.000, 1.200, 1.250, 1.267, 1.268, 1.276.

The normal equations are

$$6\hat{\alpha} + 61.5\hat{\beta} = 7.261$$
$$61.5\hat{\alpha} + 630.5\hat{\beta} = 74.5053$$

which yields a regression line of $\hat{Y} = -3.478 + 0.457X$. Suppose that the normal equations[13] had been computed only to four significant digits as

$$6\hat{\alpha} + 61.5\hat{\beta} = 7.261$$
$$61.5\hat{\alpha} + 630.6\hat{\beta} = 74.50.$$

The resulting regression line would have been $\hat{Y} = -2.651 + 0.377X$. Changing the fourth significant digit of the normal equations changed the first significant digit of a coefficient, resulting in no digits of accuracy for either the intercept or the slope. The two regression lines fit quite differently when cast against a scatterplot of X and Y. This example further dispels the errant notion that "arithmetic more precise than the data it operates on is needless." To the contrary, as Longley made clear, carrying calculations out to single or double precision is no guarantee of even a single digit of accuracy, and the choice of the proper algorithm matters, too.

To understand how results can be adversely affected, realize that roundoff error and truncation error, if combined in an unstable method (e.g., repeated division of a large number by a small number) can interact until the combined error overwhelms the result. Consider the system of equations $Xb = y$ where the matrix X and vector y are known and the vector b is to be calculated. From basic econometrics we know that $b = (X'X)^{-1}X'y$. If X is changed a little and b changes a little then the data are *well-conditioned*; if b changes a lot then the data are *ill-conditioned*. In traditional least-squares, if the data matrix X consists of collinear data, then X might be an ill-conditioned matrix and solution could be problematic. A method of obtaining b that works for well-conditioned data might not work at all for ill-conditioned data. The notion of condition is not limited just to data; entire classes of problems can be said to be ill-conditioned, such as Fredholm equations of the first kind.

As another demonstration that small differences matter, Beaton, Rubin, and Barone (1976) perturbed the independent variables of Longley's data by adding a random number uniformly distributed on [-0.5, 0.499] to the last published digit. For example, 1947 GNP deflator is reported as 83, but it might well have been anywhere between 82.5 and 83.499. Then a regression was run. This procedure was repeated 1000 times. Since the perturbation of the independent variables is within rounding tolerance of the data, one might expect little change in the estimated coefficients. However, the Longley data are ill-conditioned and so a small change in X can produce a large change in b. In the 1000 regressions, the coefficient on the GNP deflator assumed values from –232.3 to 237.0. The ill-conditioned nature of the Longley data is apparent in Table 3, which presents Longley's coefficients and statistics for the coefficients from 1000 perturbed regressions. One might think that the means of the coefficients from these 1000 perturbed regressions would be near Longley's unperturbed solution, but such is not the case. See Vinod (1982) for a discussion of the numerical and statistical issues involved. As much as it highlights the need for accurate data, this experiment

[13] Of course, no one should use the normal equations to solve for regression coefficients. While the normal equations work algebraically, computationally they are a disaster.

TABLE 3
LONGLEY RESULTS AND SUMMARY STATISTICS ON 1000 PERTURBED REGRESSIONS

coefficient	β_0	β_1	β_2	β_3	β_4	β_5	β_6
Longley	−3482258.63	+15.06	−0.04	−2.02	−1.03	−0.05	+1829.2
Beaton, et al.							
mean	−1152648.37	−26.44	+0.03	−0.96	−0.72	−0.28	+637.1
minimum	−3483280.69	−232.3	−0.09	−2.42	−1.33	−0.94	−1706.9
maximum	+3452563.48	+237.0	+0.20	+1.77	+0.36	+0.48	+1800.9

Source: Beaton, Rubin, and Barone (1976).

also underscores the idea that small differences can matter appreciably and the need for accurate algorithms.

2.5 *Algorithms Matter*

When solving $y = Xb$ we know that $b = (X'X)^{-1}X'y$. This is what the least squares result looks like, but is not how it should be computed. The person unfamiliar with numerical methods should be forgiven for concluding that the way to calculate the least-squares coefficient vector b is to calculate $(X'X)^{-1}$ and then post-multiply it by $X'y$ (multiplying $(X'X)^{-1}$ by an estimate of the error variance yields $\text{cov}(b)$). However, direct solution of the normal equations is very susceptible to roundoff error and hence extremely undesirable. Vinod (1997, p. 218) criticizes the "estimating function" literature for ignoring the numerical instability associated with direct solution of the normal equations. Numerically, a better way is first to calculate b without inverting $X'X$, and then use b to calculate $(X'X)^{-1}$ and $\text{cov}(b)$. Wampler (1980) discusses various methods.

One such method of finding b without inverting $X'X$ is the LU decomposition (Press et al. 1994, §2.10), which will work on a well-conditioned matrix, but is prone to failure if the data are collinear. For a near-singular matrix, the *singular value decomposition* (SVD, Press et al. 1994, §2.6) will work, though it is slower (requires more operations) than the LU decomposition. In regression analysis, the SVD is the method of choice (Sven Hammarling 1985; Press et al. 1992). Vinod and Ullah (1981, p. 5) is one of the few econometrics books that recommends the SVD estimate of the regression problem. The SVD should be used until one has proved that a faster algorithm will be adequate or necessary.[14] If it cannot be shown that the data will always be well-conditioned, they must be presumed to be ill-conditioned. To assume otherwise lulls the user into a false sense of trust and then, at some random time, he is unknowingly betrayed. Surprisingly, few packages mention what algorithm is used to calculate b and $\text{cov}(b)$, and clearly a poor choice of matrix inversion method can lead to disaster, as was made clear by Longley. As a general rule in computing, the method of solution greatly affects the accuracy of the solution, and properly implementing a well-chosen algorithm is important. While describing how different methods of calculating b and $\text{cov}(b)$ can lead to different answers is beyond the scope of this paper, the basic ideas can be illuminated by demonstrating how

[14] We have the luxury of concerning ourselves solely with accuracy. As a practical matter, it should be noted that the SVD is both memory- and time-intensive. A developer cannot be faulted for implementing the QR decomposition in its stead. We admit to preferring the QR to the SVD when bootstrapping.

different methods of calculating the variance of a series can lead to different answers.

Robert Ling (1974) considered five methods for calculating the variance, of which we present three. The least precise he showed to be the "calculator" formula (since it requires fewer keystrokes, it often is suggested as an alternative formula for calculation):

$$V_1 = \frac{\Sigma x^2 - (1/n)(\Sigma x)^2}{n-1} \tag{7}$$

while the standard formula

$$V_2 = \frac{\Sigma(x-\bar{x})^2}{n-1} \tag{8}$$

is more precise because V_1 squares the observations themselves rather than their deviations from the mean, and in doing so loses more of the smaller bits than V_2. Specifically, V_1 is much more prone to cancellation error than V_2. Yet a third, less familiar, formula is the "corrected two-pass method":

$$V_3 = \frac{1}{n-1}\left\{\sum_{i=1}^{n}(x_i-\bar{x})^2 - \frac{1}{n}\left[\sum_{i=1}^{n}(x_i-\bar{x})\right]^2\right\} \tag{9}$$

which is designed to account for rounding error. Algebraically, the right-most term equals zero, but computationally it will not. Again, pencil-and-paper arithmetic is not like computer arithmetic. If calculation of the mean is exact (with no rounding error) then the second term on the right-hand side equals zero and V_3 collapses to V_2. In the presence of rounding error, V_3 is likely to be more accurate than V_2.

Sometimes a developer claims to use one formula when he actually uses another, or does not specify a formula at all. How might the quality of the variance-calculating algorithm be assessed? Calculate the sample variance of three observations: 90000001, 90000002, and 90000003 (obviously the correct answer is 1.0). L. Wilkinson and Gerard Dallal (1977) used this test on a variety of mainframe packages and found that all but one failed to produce the correct answer. Many econometric packages fail this test. A disingenuous counter-argument is that the data can always be rescaled to take better advantage of the floating point representation. The appropriate rejoinder is: "How do you know that you need to rescale in the first place?" Our practical advice is: Test the software.

This, though, is the usual state of affairs in using econometric software: performing usual operations on an econometric package usually will give no indication that anything is wrong unless the package produces a glaring error such as a negative variance or $R^2 > 1$. For example, when computing correlation matrices, users of package "X2" or "X3," absent prior benchmarking, would have no inkling that their program was faulty; a user of package "X4," since the correlations were greater than unity, might suspect something. Interestingly, each of the four packages passes the Longley Benchmark—that a program does one thing correctly is no assurance that it does another thing correctly.

The above discussion may convey the impression that computer arithmetic is hopelessly imprecise, but nothing could be farther from the truth. Indeed, rules for computer arithmetic, if rigorously implemented, can lead to theorems and proofs of accuracy. The problem is not that the computer is inaccurate; it is how the inaccuracy is handled. If the inaccuracy is handled properly, then theorems can be proved which define an arithmetic for floating point computations and so assure the final accuracy of an answer. See David Goldberg (1991) for an extended layman's discussion and

examples of such theorems. Additional useful references for computer arithmetic are Kennedy and Gentle (1980, ch. 3) and Ronald Thisted (1988, ch. 2). More generally, for the economist interested in matters computational, we highly recommend Kenneth Judd's (1998) recent textbook *Numerical Methods in Economics*.

3. *General Benchmarks*

Given the history of benchmarking statistical software, simply trusting the software represents the triumph of hope over experience and is an invitation to disaster. This, then, is the entire purpose of benchmarking: with given inputs and correct outputs in hand, the program is supplied with the input and its output is checked against the correct answer. Exact accuracy is not demanded, but the program's answer should be close enough to the correct answer. This develops a knowledge of where the program errs and whether the error is likely to affect results. First examine the program's general procedures. After determining that they are reliable, then examine specific econometric procedures based on the general procedures. For example, first determine that the linear least squares routine "works." Then check that a specialized procedure based on linear least squares, e.g., calculation of autocorrelation coefficients, is implemented properly.

Following Sawitzki (1994), we recommend the use of the Wilkinson Tests for entry-level purposes. Beyond that, in Section One we mentioned several benchmarks for least squares procedures, such as univariate summary statistics, analysis of variance, and linear regression. An obvious gap in the literature has been benchmarks for nonlinear least squares procedures. This deficiency was remedied by the Information Technology Laboratory of the Statistical and Engineering Division at the National Institute for Standards and Technology (NIST), which recently released its "Statistical Reference Datasets" (StRD), a collection of benchmarks for statistical software.[15] While NIST has plans to expand the number of benchmarks, at this writing it contains 58 benchmarks in four suites: univariate summary statistics (9 benchmarks), analysis of variance (11), linear regression (11), and nonlinear regression (27).

To circumvent rounding error problems, NIST used multiple precision calculations, carrying 500 digits for linear procedures and using quadruple precision for nonlinear procedures. Complete computational details and a discussion of test problem selection are in Janet Rogers et al. (1998). The results, rounded to fifteen digits for linear procedures and eleven digits for nonlinear procedures, are referred to as "certified values." Certified values are provided for a number of statistics for each suite. For example, for univariate summary statistics the mean, standard deviation, and first-order autocorrelation coefficient are given to fifteen places. While the output from so many tests is voluminous, McCullough (1998) described how to condense the output, and applied the benchmarks to three popular statistical packages (McCullough 1999). While the Longley lesson has been learned—all packages did well on linear regression benchmarks—gross errors were uncovered in analyses of variance routines (including negative sums of squares), and some programs produced solutions to nonlinear problems that had zero digits of accuracy. One of the

[15] On the web at http://www.nist.gov/itl/div898/strd

statistical packages solved all 27 of the StRD nonlinear benchmarks, and another solved 26 of 27, returning one incorrect answer. Of course, there are two ways a nonlinear procedure can fail, as described by W. Murray (1972, p. 107): "The first is miserable failure which is discovered when an exasperated computer finally prints out a message of defeat. The second is disastrous failure when the computer and trusting user mistakenly think they have found the answer."

McCullough (1999a) also applied the StRD to econometric software. Again, the Longley lesson has been learned, but the problems with nonlinear estimation appear to be more severe with econometric software than with statistical software. At one extreme, one econometric package correctly solved 26 of the 27 nonlinear estimation problems, once producing a miserable solution (of course, no solution is better than an incorrect solution). At the other extreme, a popular econometric package correctly solved nine, failed miserably four times, and for the remaining fourteen produced disastrous solutions. Absent a benchmark, a user would have no idea that this package produced completely inaccurate "solutions" for over 50 percent of the test problems. A third package produced eight disastrous solutions. Thus, the results of any study involving nonlinear estimation that used the latter two packages must be called into question. It is not too far off the mark to suggest, at least for econometric packages, that nonlinear estimation is today where linear estimation was thirty years ago. As an aside, we strongly caution any economist who uses a spreadsheet package for econometric estimation to benchmark the package first (McCullough and Wilson 1999).

Also related to nonlinear estimation, we note that some packages report no details of their nonlinear estimation (not even the method), and others are vague about important details such as whether numerical or analytic derivatives are used.[16] Still others make no mention of how the standard errors are computed, (e.g., via the gradient or inverse of the Hessian) despite the fact that this can affect inference. Some packages conflate the minimization and nonlinear least squares routines, offering only a single procedure. The basis for this is that any minimization problem can be written as a maximization problem merely by appending a minus sign to the objective function. Typically, though, separate routines are used for minimization of a sum of squares and maximization of likelihood functions. The primary reason is that the objective function of a least squares problem has a special form, and more efficient, specialized algorithms have been devised for the special form. Some packages offer only one solver, though neither modified Gauss-Newton nor Levenberg-Marquardt is sufficiently dependable to serve as the sole general nonlinear least squares method in supporting software systems (Kennedy and Gentle 1980, p. 484).

All these problems are compounded by the fact that virtually all journals do not require authors to reveal computational details, not even the software used. Thus, even if we know that some software package is defective, we have no idea which published results are based on defective software and might

[16] Generally speaking, analytic derivatives are more accurate than numerical derivatives. Numerical derivatives crafted for a particular problem can be as accurate as analytic derivatives, but in a general purpose nonlinear solver, analytic derivatives are preferred (Jonathan Bard 1974, p. 117; J. E. Dennis and Robert Schnabel 1996, p. 106). See Janet Donaldson and Schnabel (1987) for Monte Carlo evidence.

well be wrong. More to the point, nor do we have any idea which results might be right. Even in rare instances when a software package is identified in an article, and the package is later discovered to be defective in a way which affects the article's results, updating the results with a reliable software package is problematic. The reason is that virtually no journals require authors to archive either their data or their code, and this constitutes an almost insurmountable barrier to replication in the economic science. Scientific content is not dependent merely on writing up a summary of results. Just as important is showing the precise method by which the results were obtained and making this method available for public scrutiny. To our knowledge, only the journal *Macroeconomic Dynamics* (*MD*) requires both data and code, while the *Journal of Applied Econometrics* (*JAE*) requires data and encourages code, and the *Journal of Business and Economic Statistics* (*JBES*) and *The Economic Journal* require data; all four journals have archives which can be accessed via the worldwide web. In the context of replicability and the advancement of science, the advantage of requiring code in addition to data is obvious. While it may be trivial to use the archived code to replicate the results in a published article, only if the code is available for inspection will other researchers have the opportunity to find errors in the code. Just as commercial software needs to be checked, so does the code which underlies published results.

The StRD, while the foremost collection of benchmarks for econometric purposes, is not the only one. We note also the existence of suites of benchmarks for matrices (including inversion, multiplication, eigenvalue decomposition, etc.). Given the prevalence of specialized covariance matrices whose calculation is not automated, and also the increasing use of eigenvalue analysis for dynamical systems, these could profitably be applied to testing the matrix-handling facilities of econometric packages. They also would be suitable for testing matrix-based languages such as GAUSS and Ox. The "Matrix Market" of R. Boisvert et al. (1997) has over 500 matrices, each with its own web page, though many of them are quite specialized and rarely encountered in economics. A more managable number, about 70, with more relevance to economics is Higham's (1991, 1995) "TESTMAT" collection. We are unaware of any econometric software review that assesses the accuracy of matrix calculations. We have applied some of Higham's test matrices to some econometric packages and found errors in inversion and eigenvalue routines, which suggests the need for such assessments.

Checking the basic estimation machinery is only the beginning. That the linear or nonlinear solver "works" is no assurance that it has been properly implemented in specialized procedures such as FIML and GARCH. Such specialized procedures need benchmarks, of which there is a marked dearth.

4. *The Need for Specific Benchmarks*

Discrepancies abound between different software packages, and many of us are familiar with getting two answers to the same problem using two different packages. Here we do not refer to minor differences due to rounding error, hardware configuration, or operating system, but major differences due to some unknown reason, such as the discrepancy between FIML results presented in our Table 1. All three sets of answers cannot be right; and in fact they are not: the correct answer is by Calzolari and Panattoni (1988), who

provided a benchmark for FIML, including the asymptotic covariance matrix thereof computed by various methods. See Julian Silk (1996) for a discussion.

An unfortunate consequence of two different packages providing two different answers to the same problem is a lack of replicability in economic research. In their important article, William Dewald, Jerry Thursby and Richard Anderson (1986) showed that replicating economic research is a nearly impossible task. Part of the replicability problem is that, in addition to the data and the code used to run a commercial package, an aspiring replicator also needs the same commercial package as the original author. Yet, a researcher should be able to expect that FIML estimates of Klein's Model I are not dependent on the package used. Clearly, benchmarks, replication, and software reliability are fundamentally related.

The advantage of writing software to meet existing benchmarks is obvious. Yet even when benchmarks exist, few econometric software developers provide users with benchmarks results, or even provide sufficient information on the algorithm that a user can ascertain whether or not the procedure is likely to produce correct results. For example, many packages have "autocorrelation" and "partial autocorrelation" procedures which, given an input series, produce the lagged autocorrelations and lagged partial autocorrelations. These are critical for choosing the number of AR and MA lags for Box-Jenkins modelling. Yet rarely is the user told how these results are calculated, and this is important information, since some poor algorithms are in widespread use. An example follows.

Let $x0$ be a series and let $x1$ and $x2$ be its first and second lags. Let ρ_1 be the first-order autocorrelation coefficient (the simple correlation between $x0$ and $x1$) and let π_2 be the second-order partial autocorrelation coefficient (the correlation between $x0$ and $x2$ after the effect of $x1$ on $x0$ is removed). By definition the first-order partial autocorrelation coefficient is equal to the first-order autocorrelation coefficient, i.e., $\pi_1 \equiv \rho_1$. There are many ways to calculate the autocorrelation and partial autocorrelation coefficients, and some are better than others. With respect to calculation of partial autocorrelation coefficients, George Box and Gwilym Jenkins (1976, p. 65) note that the regression method is more accurate than methods based on the Yule-Walker equations (YWE). M. Priestley (1981, pp. 350–52) lists four methods in decreasing order of accuracy: exact maximum likelihood, conditional maximum likelihood (CML), approximate least squares, and the YWE. In general, the YWE are to be eschewed for economic data (Dag Tjostheim and Jostein Paulsen 1983), yet they are frequently employed in econometric software and often the only method mentioned in time series and econometrics texts. It is interesting to compare the YWE to CML.

Define a series $x0 = \{1, 2, \ldots 10\}$. It is linear without errors, so $\rho_1 \equiv 1$. Once the effect of $x1$ is removed from $x0$, there is nothing left to explain, so $\pi_2 \equiv 0$. These intuitive results can be verified by calculation from first principles, e.g., elementary formulae for correlation and partial correlation (Alan Stuart and J. Ord, 1991, p. 1012). A standard implementation of the YWE yields $\hat{\rho}_1 = 0.70$ and $\hat{\pi}_2 = -0.153$ while CML yields $\hat{\rho}_1 = 1.0$ and $\hat{\pi}_2 = 0$. The effect of inaccurate estimation of partial autocorrelation coefficients on ARMA model identification needs no elaboration. Accurate computation of partial autocorrelation coefficients is discussed in McCullough (1998a).

As another example of the need for benchmarking specialized procedures, consider the staggering number of published papers using GARCH. It is well-known that different packages produce different answers to the same problem. While until recently there was no benchmark for GARCH, this problem was all but solved by Gabrielle Fiorentini, Calzolari and Panattoni (1996, hereafter FCP). They provided complete and usable closed-form expressions for the gradient and Hessian of a univariate GARCH model (recall that analytic derivatives are more accurate than the numerical derivatives commonly used in GARCH procedures). They also provided FORTRAN code for estimating such a model. Using this code on a well-known GARCH dataset constitutes a benchmark.

The *JBES* archive has the $T = 1974$ observations on the daily percentage nominal returns for the Deutschemark/British pound exchange rate from Tim Bollerslev and Eric Ghysels (1996, hereafter BG). The *JAE* archive has the FCP FORTRAN program. McCullough and Renfro (1999) used these data and code to produce the FCP GARCH benchmark and applied it to seven packages, "Y1" through "Y7." The model is $y_t = \mu + \epsilon_\tau$, where $\epsilon_\tau | \Psi_{t-1} \sim N(0, h_t)$ and $h_t = \alpha_0 + \alpha_1 \epsilon_{\tau-1}^2 + \beta_1 h_{t-1}$. (BG estimated a different parameterization of this model; their results are consistent with benchmark.) Since this model must be estimated by nonlinear methods, starting values for the coefficients and a method for initializing the series h_t and ϵ_τ^2 at time $t = 0$ must be specified. FCP, BG and many others have used the initialization $h_0 = \epsilon_0^2 = SSR/T$ (SSR is the sum of squared residuals).

Surprisingly, not all packages could even begin to estimate this simple GARCH model which has been much-used in applied work. Package Y2 allows the user to control neither the starting values nor the initialization of h_0 and ϵ_0^2. Packages Y1, Y4, and Y6 allow the user to specify the starting values, but offer no control over the initialization. Packages Y3 and Y5, which use analytic rather numerical first derivatives, achieved two or three digits of accuracy for the coefficients (the accuracy of the standard error estimates is another matter altogether). Whether this degree of accuracy is suitable as input for an option pricing model is unknown. Only one package, Y7, hit the benchmark to several digits of accuracy, and did so for both coefficients and standard errors. This package uses analytic first and second derivatives.

In all but two cases, it was necessary to contact the developer to determine at least one and sometimes all of the following: the precise method of initializing h_0 and ϵ_0^2; the type of derivatives used; and the method of calculating standard errors (there are at least five ways). That these matters are not explicitly addressed in the documentation is a serious omission. That four of seven GARCH procedures cannot accommodate a simple and popular GARCH model suggests that some vendors take an idiosyncratic approach to programming. For these four packages, the user has very little choice as to the conditional likelihood to be maximized. Such approaches to documentation and programming are an impediment to good research.

Some procedures are easy to benchmark, as (partial) autocorrelation coefficients and two-stage least squares. These, however, are the exception. Generally, devising a benchmark is an arduous process, as the work of Calzolari et al. and Fiorentini et al. attests. J. Dongarra and G. Stewart (1984) noted, while describing their pioneering efforts to test only linear algebra

routines, "In some cases, the test programs were harder to design than the programs they tested." The general state of affairs is that there are no benchmarks for most specialized econometric procedures. Perusing the list of commands for any econometric software package yields many procedures for which we were unable to find a benchmark and for which we found discrepancies between packages: linear estimation with AR(1) errors, estimation of an ARMA model, Kalman filtering, limited dependent variables models, SUR estimation, three-stage least squares, and so on. Absent benchmarks, we cannot be sure that an econometric package gives us reliable answers. Estimation, however, is not the only part of a package whose reliability needs to be verified.

5. *Testing the Random Number Generator*

Only a few years ago, the use of random numbers in economics was the arcane province of a few specialists. With the recent explosion in computing power and concomitant theoretical advances, this is no longer the case. Simulation (Christian Gourierioux and Alain Montfort 1996), bootstrapping (Vinod 1993), and Bayesian econometrics are but three econometric methods which make extensive use of the random number generator (RNG). The recent text by Gentle (1998a) is a useful reference. Good introductions to RNGs are Stephen Park and Keith Miller (1988) and Press et al. (1992, ch. 7), both of which stress that an RNG should not be treated by the user as a "black box"—the user who chooses to remain uninformed of the properties of his RNG does so at his own peril. Yet, we examined several econometric packages and found that most gave no information whatsoever about the RNG employed, not even a citation to the article in which the RNG was published. Econometric software developers typically do present the RNG as a black box.

As is well known, the random numbers produced by a computer are not random, but *pseudo-random*, i.e., they are produced deterministically, but (if the RNG works well) appear to be random. One popular RNG is the linear congruential generator (LCG), which for a large integer m produces a sequence of numbers $I_1, I_2, I_3, \ldots$ between 0 and $m - 1$ via a recursion

$$I_{j+1} = aI_j + c \quad (mod m) \tag{10}$$

based on an initial "seed" I_0, which often is supplied by the user. Eventually, the sequence repeats itself with a *period* no greater than m, and exactly equal to m if the parameters a, c, and m are carefully chosen. Typically, output is restricted to the interval (0,1) by returning I_{j+1}/m. This sequence should be uniformly distributed. For the LCG, though not for all RNGs, the same sequence will be produced with the same seed. This *reproducibility* is a desirable feature for debugging purposes. When debugging a program, it is necessary to examine the input which caused the bug, and if the RNG is not reproducible, the input no longer is available.

Ripley (1990) lists the desirable characteristics of an RNG. In short, an RNG should:

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

We already addressed the first point; we address the remaining three in turn.

Poor choices for the parameters of any RNG can drastically impair its quality, causing the period to be unnecessarily short or even producing non-uniform random numbers. For example, the infamous RANDU (IBM 1968, p. 77) generator distributed with the IBM 360 mainframe computer had such poor choices for the parameters that Donald Knuth (1997, p. 188) called it "really horrible." RANDU failed even simple tests of randomness. Complicating matters for users, RANDU was widely imitated, and even recommended in textbooks long after its faults were well-known (Park and Miller 1988, p. 1198). As another example, Sawitzki (1985) describes the RNG for IBM PC BASIC as being not only decidedly non-uniform, but having a period of only 2^{16}. "Short" is a relative term, and periods which were of acceptable length only a few years ago might now be unacceptable, given the recent surge in the demand for random numbers. Knuth (1997, p. 195) suggests that the period should be at least one thousand times larger than the number of values used, i.e., if p is the period and n is the number of calls to the RNG, then $p > 1000n$. The reason is that the discrepancy between an RNG's output over its entire period and true randomness can be large, especially for linear-type generators, so at most a fraction of the RNG's output should be used. Therefore, a $p \approx 2^{31}$ generator should be used for no more than 2.1 million calls. Yet a modest double bootstrap (see McCullough and Vinod 1998) with 1999 first stage and 250 second stage resamples requires that the residual vector be resampled half a million times, so that such an RNG can support a sample size of no more than four observations.

The situation is even more stark for more computationally intensive applications such as Bayesian inference with numerical integration, Monte Carlo studies, and calculation of non-standard test statistics. John Geweke and Michael Keane (1997) used one billion random numbers to conduct Bayesian inference via Gibbs sampling. The Monte Carlo of the double bootstrap by David Letson and McCullough (1998) required 45 billion calls to the RNG. MacKinnon (1996) used more than 100 billion to tabulate the distribution functions of unit root and cointegration statistics. With such considerations in mind, Geweke (1996) showed how to set up an RNG with $p = 2^{100}$. One expert on random numbers has written (Pierre L'Ecuyer 1992, p. 306) "No generator should be used for any serious purpose if its period (or, at least, a lower bound on it) is unknown." A researcher needs to know about the RNG in his econometric package. Simply having a citation for the RNG is insufficient, because faulty RNGs are still proposed in journal articles (L'Ecuyer 1994), and some RNGs with bad properties are in widespread use.

Not only should an RNG have a long period, it should also pass statistical tests for randomness, because correlated output can wreak havoc. For example, first-order autocorrelation of the random numbers might not be a problem if Monte Carlo methods are used to evaluate a one-dimensional integral, but almost certainly would be disastrous for evaluating a two-dimensional integral. The idea behind testing is to see whether the RNG's output is significantly different from the behavior of a truly independently and identically distributed sequence of uniform random variables. Strictly speaking, the null hypothesis that the output from an RNG is uniform, i.i.d. is false, and no single best test exists. Yet, since what is wanted from an RNG is the *appearance* of randomness, ceteris paribus, we

recommend an RNG that passes a given test for uniformity to one that fails it. Moreover, one test is insufficient. Since there are many possible departures from randomness, many tests should be applied.

The following simple example describes testing a sequence of numbers for randomness, which often involves *two-level testing*. Divide the unit interval into 30 equal bins. Make three hundred calls to the RNG and place each in its appropriate bin. If the RNG does provide truly uniform numbers, there should be 10 numbers in each bin, plus or minus random deviations. This can easily be tested by using the χ^2 comparison of the actual and expected number in each bin. Repeat this test 10 times, yielding 10 χ^2 statistics which can then be compared to a theoretical χ^2 distribution with 29 degrees of freedom using the Kolmogorov-Smirnoff (K-S) test. This will test whether the RNG produces numbers that are approximately uniformly distributed. Of course, the number of bins can be increased, as well as the number of calls to the RNG, to provide finer discrimination. The reason for running 10 χ^2 tests and then using the K-S test (two levels) instead of conducting one large χ^2 on 3000 random numbers (one level) is that this increases the power of the test (L'Ecuyer 1994, sec. 4.5.1), which is desirable since the null hypothesis is false.

The above procedure tests how well the RNG can fill the real line, but can it fill a square? To test this, make a square with 30 equal intervals on each side, for a total of 900 bins. Draw 9000 pairs of random numbers, which should put 10 in each bin. Uniformity can be tested by comparing actual and expected numbers for each bin with a $\chi^2(899)$ distribution. Then, the test can be repeated 10 times and those 10 χ^2 statistics subjected to a K-S test. Many simple RNGs will pass the extension to three dimensions; RANDU will not. While RANDU's output appeared random in one and two dimensions, its correlation showed up strongly in three dimensions, i.e., RANDU's output was not uncorrelated in a moderate number of dimensions. Extensions to still higher dimensions are easily generalized. R. Coveyou and R. MacPherson (1967) first observed and George Marsaglia (1968) explicitly discussed that LCGs have a lattice structure. That is, in 3–space, for example, the points (z_1, z_2, z_3), (z_2, z_3, z_4) and (z_3, z_4, z_5) all fall on a finite number of planes. Therefore, the number of planes should be large and they should be close together. Hence, it is important to test for correlation in higher dimensions, though for practical reasons the number of dimensions is limited to about eight.

Batteries of such tests are offered by Michael Stephens (1986) and She Tezuka (1995), though the first standard battery of tests of which we are aware was given in the 1981 edition of Knuth (1997), with FORTRAN and C implementations by E. Dudewicz and T. Ralley (1981) and Jerry Dwyer and K. Williams (1996), respectively. Marsaglia (1985) noted that Knuth's tests were not very stringent. This is especially true today, as the scale of computer simulations has increased concomitantly with computing power. In particular, some RNGs that pass the Knuth tests frequently produce significant biases when used in large-scale simulations. To remedy this deficiency, Marsaglia (1996) produced "Diehard: A Battery of Tests of Randomness," also known as "Marsaglia's Diehard Tests," for which automated executable files for DOS and LINUX are available, as well as source files in C. The executable operates on a file of about three million

random numbers created by an RNG. While three million random numbers will accommodate only the smallest and simplest of Monte Carlo studies, as a first assessment of an RNG, DIEHARD has ease of use to recommend it.[17] For testing more than three million, the source files must be modified and recompiled. Names and descriptions of the tests are produced as part of the output. Marsaglia is planning a revision of DIEHARD. L'Ecuyer's TESTU01 program is in the testing stage, and is scheduled to be released in the next year or so.

The developer of an RNG often will subject the RNG to several tests and note this in the article in which the RNG is published. However, implementation of an RNG can be difficult, so it is imperative that the developer's implementation be tested. Our own informal application of DIEHARD to a few econometric packages found that while some passed, some did not. Still, even those that passed would be unsuitable for large-scale applications. Importantly, not one package mentioned the period of its RNG.

6. *Statistical Distributions*

Some persons might think that computer programs are more accurate than statistical tables, since the statistical tables only report a few decimals and computer programs report several, but such is not necessarily the case. Imagine an economist conducting a Chow test with 3 restrictions on 126 observations. Suppose he calculated a test statistic of 4.0. He might well have an interest in the F(3,120) distribution. If he uses Package "X4" to calculate the 1 percent critical value, he obtains 4.12. Thus, he does not reject the null—until a referee points out that a standard statistical table gives a critical value of 3.95, and so the null is rejected along with the article. As another example, the documentation for Package "X5" says that its Student's-t function returns "the probability that a t-statistic with d degrees of freedom exceeds X. Letting $X = 1.345$ and choosing $d = 14$, a statistical table shows that the upper tail is 0.10. Yet Package "X5" returns 0.2000—the value for a two-tail test. Erroneous inference awaits the user who trusts such documentation. Leo Knüsel (1995) has documented the inaccuracy of statistical distributions in GAUSS v3.2.6, and also that these same inaccuracies were not corrected in release v3.2.13 (Knüsel 1996). Even more serious inaccuracies were revealed in the statistical distributions of Excel97 (Knüsel 1998). McCullough (1999b) used Knüsel's (1989) ELV program to document similar inaccuracies in econometric packages.

Statistical distributions have two fundamental applications: calculating p-values and calculating critical values for some level of significance. Let $F(x)$ be the cumulative distribution function (cdf) of the random variable X whose probability density function is $f(x)$. The first step in calculating a (one-sided) p-value for a calculated statistic, $\tilde{x}$, is to determine $\tilde{p} = P(X \leq \tilde{x}) = F(\tilde{x})$. Usually, there is no closed-form expression available, and so the problem becomes one of approximating an integral

$$F(\tilde{x}) = \int_{-\infty}^{\tilde{x}} f(t)dt \tag{11}$$

The problem of finding a critical value amounts to approximating the inverse of the above integral, i.e., finding x_c such that

$$x_c = F^{-1}(1 - p) \tag{12}$$

for some specified value of p.

[17] The documentation for DIEHARD is sketchy. Persons wishing to use DIEHARD should consult McCullough (1998, 1999) for implementation details.

It is well-known that the cdf of the standard normal distribution must be evaluated numerically. Numerical evaluation, or integral approximation, is rife with technical difficulties. Improper integrals (with limits of $-\infty$ or $+\infty$) must be dealt with, and integrals with vertical asymptotes and other such stumbling blocks must be overcome. General expositions on these details are available in Kennedy and Gentle (1980, ch. 5) and Thisted (1988, ch. 5), with discussions of specific distributions in Norman Johnson, Samuel Kotz, and N. Balakrishnan (1994, 1995). Algorithmic error is especially critical here. Barry Brown and Lawrence Levy (1994) examined several algorithms for the incomplete beta distribution (which is the basis of the F-distribution, among others) and found only one of them to be reliable. B. Bablok (1988) uncovered several errors in commonly used formulae for non-central statistics. As an example, a procedure for the non-central F based on Eq. 26.6.18 of Milton Abramowitz and Irene Stegun (1972) will return incorrect results.

It is sometimes suggested that statistical distributions need be accurate only to two or three digits. There are two objections to this. First, a package that at best gets two or three digits is more likely to provide zero digits than a package that at best gets several digits. Second, there are many applications for which two or three digits are insufficient. Among these are Edgeworth expansions, Monte Carlo Markov chains, size and power calculations, quantile-quantile plots, censored and truncated regressions, and many more. These methods sometimes require accurate evaluation of probabilities as small as 1.E-12 and smaller. Knüsel (1995) suggested that the minimum requirement for statistical distributions is that they should be accurate to all displayed digits, and observed that this has three implications:

- If a probability is smaller than 0.5E-4 and the program prints out 0.0000, then the program is correct.
- If a probability is zero and the program prints 0.76543E-11, then the program is incorrect.
- If a probability is 3.456E-10 and the program prints 3.401E-10, this is incorrect, for the result is not correct as printed.

This last point is easiest to see when it is remembered that *relative error*, and not *absolute error*, is the relevant criterion. While 3.401E-10 − 3.456E-10 = -0.55E-10 is a small number, the relative error is (|3.401E-10 − 3.456E-10|)/3.456E-10 ≈ 19%, which is not small.

In order to assess accuracy, accurate results must be obtainable. Here, Knüsel's (1989) ELV, available as a DOS executable, and Brown's (1997) DCDFLIB, available in FORTRAN77 and C tar files, can be of use. A basic sequence of percentiles (BSP) {0.0001, 0.001, 0.01, 0.1, 0.2, . . . , 0.9, 0.99, 0.999, 0.9999 can be profitably employed. Probabilities outside this range are referred to as the "extreme tails." Use ELV or DCDFLIB to generate critical values for the BSP. Feed these critical values into the econometric package to see whether the correct tail values are returned. If they are, then the distribution seems accurate and the extent of its accuracy can be assessed by using ELV or DCDFLIB to answer the following questions:

- How far can the degrees of freedom be extended before the program breaks?
- Are the results still sensible for extreme parameters or is nonsense output, such as negative probabilities, produced?

- Does the program correctly calculate very small probabilities? (This amounts to checking the "extreme tails" to determine the point at which the package no longer produces accurate answers.)
- Does the program give very small results such as 0.76345E-37 that are completely wrong?

To test inverse functions, feed the BSP into the inverse function and see whether the exact critical values are returned. A common problem which can be observed is that one tail is more accurate than the other. This is because many packages do not compute upper and lower tails separately, but compute only one tail and find the other by complementation. When a probability is near zero or unity, this can lead to cancellation error if upper and lower tails are not computed separately. Consider determining $p = P(X > 265)$ where X is chi-square with 100 degrees of freedom. Many packages, "Package X3" included, will compute only $P(X \leq 265)$, so p must be determined by complementation, $p = 1 - P(X \leq 265)$ which produces $p = 0.11102E{-}15$ rather than the correct p = 7.2119E-17. The reason is that $P(X \leq 265)$ is very close to unity, and so the subtraction necessary to obtain p is contaminated by cancellation error to the point that the result has no accurate digits. Packages should compute upper and lower tails separately.

7. *Conclusions and Recommendations*

We have presented several cases of serious numerical discrepancies between econometric packages, including FIML, GARCH, (partial) autocorrelation, and Cochrane-Orcutt corrections. We have suggested that inadequate nonlinear solvers are not uncommon. There can be no doubt that many more such discrepancies exist, and they can be attributed in part to a lack of benchmarks. We have also shown that random number generators and statistical distributions can suffer from numerical problems. These problems can be solved only by a concerted effort on the part of users, developers, and the economics profession as a whole.

Users should recognize that accuracy is at least as important as either speed or user-friendliness. This does not mean each user must immediately visit the StRD website and commence benchmarking his package. At the very least it does imply that users should be cognizant of the fact that writing accurate software is more demanding than writing either fast or friendly software. Developers need to benchmark their software. This does not mean developers should suspend implementation of new procedures and devote all their resources to developing needed benchmarks. At the very least, it means that developers should make use of existing benchmarks, such as StRD, TESTMAT, and others which will be developed in the future. When a benchmark exists for a procedure, as in the case of FIML, the developer should note in the manual that the procedure achieves the benchmark, or else explain why it does not. Moreover, developers need to document their procedures better. As Lovell and Selover (1994) discovered in their consideration of AR(1) procedures, there are many admissible implementations of AR(1) procedures, and developers often do not describe precisely which implementation they use. Documentation for econometric software can only be improved by paying proper attention to numerical and algorithmic matters. RNGs need to be documented, including the type of RNG, its

period, and tests the developer's implementation of the RNG has passed. Statistical distributions also need to be better documented, and to have their accuracy verified.

Accurate econometric software is not just the responsibility of the developers, it requires active participation by the profession. Obviously, creating benchmarks is beyond the capability of any one developer, or even the small collection of developers. So, too, is the matter of deciding which procedures are most in need of being benchmarked. A strong suggestion that a method needs to be benchmarked is that two packages give two different answers to the same problem.[18] At present, two different packages are not often used to solve the same problem, but the widespread use of archives by journals would change this. Therefore, journal editors should require that authors identify their software (including version number) and make their code and their data widely available via archives. In addition to uncovering discrepancies between packages, this will provide developers and users more of an incentive to rely upon benchmarked procedures, and thus attenuate the problem of two packages providing two different answers to the same problem.

Why journal readers do not demand archives is the same issue as why software users do not demand evidence of accuracy. Yet, just as results from a software package that passes benchmark tests are preferable to results from a software package that fails benchmark tests, so, too, studies from a journal whose results are capable of being verified are preferable to studies from a journal whose results are not verifiable. Regrettably, neither benchmarking by vendors nor archiving by journals is yet a common practice.

Two related events are worth noting. First, following the publication of Dewald, Thursby, and Anderson (1986), the *Journal of Money, Credit and Banking* began requesting data from its authors, and the NSF established an archive for the storage and distribution of authors' data at the University of Michigan's Interuniversity Consortium for Political and Social Research. The NSF economics program invited journal editors to request that authors place their data in this archive; 22 editors declined this invitation (Anderson and Dewald 1994). Second, in 1993, the *JMCB* discontinued its practice of requesting data from authors. However, both events occurred before the advent of the worldwide web.

While some journals have "policies" that authors *should* make available their data and code, there is no penalty attached to an author's refusal to comply, and these policies are honored more often in the breach. The results of Dewald, Thursby, and Anderson (1986), recently revisited by Anderson and Dewald (1994), showed that most authors could not or would not honor a request for data and code. The disincentives for authors to comply with such requests are discussed in Susan Feigenbaum and David Levy (1993, 1994). An archive completely eliminates the problem of obtaining data and code from authors. The *MD* archive, in fact, requires that the code supplied be able to reproduce the reported results. This latter requirement is necessary to ensure the replicability of research, but it is also necessary for the problem of determining whether two packages really do produce different answers to the same problem. A reported discrepancy between two packages might be due to a coding

[18] This does necessarily mean that either package is incorrect. It may mean only that the documentation is poor and the two packages are really doing two different things.

error, but this cannot be determined if the code is not available.

That the methods by which results are obtained be open to the scrutiny of other researchers is a higher standard of quality than is usually found in economics, but it is a common standard in other sciences. Economic research would generally benefit if this standard was more widely adopted, and so would software.[19] We observe that some authors maintain this higher standard by making use of personal archives when they publish in journals that have no archive. To give but three examples, the articles by Bronwyn Hall (1993), James Hamilton (1997), and Mark Watson (1993) all have data and code archived at the authors' homepages.

Until journal archives are commonplace, many things can be done. Researchers should ensure that their software is up to the task of producing replicable research. Referees can ask authors to provide computational details, with reference to making sure that results are reproducible. Software review editors can do at least two things. When a developer provides no evidence that his software meets existing benchmarks, the editor can suggest that the reviewer apply known benchmarks. Second, software review editors can encourage reviewers to propose new benchmarks. The more quantitative journals can devote space to publication of more sophisticated benchmarks or to discussion of software; for example, the *Journal of Economic and Social Measurement* has a special issue on econometric software forthcoming.

Reliable econometric software is the joint responsibility of the users, developers, and the profession. We hope the day is not too far off when the advertisements for econometric software read, "Obtains the *correct* solution in x seconds," rather than the current, "Obtains *a* solution in y seconds," and that users will appreciate this distinction, even though x may be greater than y.

[19] The journals themselves might also benefit. Tauchen (1993) has argued that readers of journals should be interested in data and code, and that when they are made available a journal's prestige and circulation should increase.

References

Abramowitz, Milton and Irene A. Stegun. 1972. *Handbook of Mathematical Functions*. 9th printing. NY: Dover.

Anderson, Richard G. and William G. Dewald. 1994. "Scientific Standards in Applied Economics a Decade After the *Journal of Money, Credit and Banking* Project," *Fed. Res. Bank St. Louis Rev.*, 76, pp. 79–83.

Bablok, B. 1988. *Numerische Berechnung nichtzentraler Statist.her verteilungen*. Diploma thesis, Dept. Statistics, U. Munich.

Bankhofer, Udo and Andreas Hilbert. 1997. "Statistical Software for Windows: A Market Survey," *Statist. Pap.*, 38, pp. 393–407.

Bard, Jonathan. 1974. *Nonlinear Parameter Estimation*. NY: Academic Press.

Beaton, Albert E.; Donald B. Rubin, and John L. Barone. 1976. "The Acceptability of Regression Solutions: Another Look at Computational Accuracy," *J. Amer. Statist. Assoc.*, 71:353, pp. 158–68.

Bernhard, G.; M. Herbold, and W. Meyers. 1988. "Investigation on the Reliability of Some Elementary Nonparametric Methods in Statistical Analysis Systems," *Statist. Software Newsletter*, 14, pp. 19–26.

Berndt, Ernst. 1990. *The Practice of Econometrics*. Reading, MA: Addison-Wesley.

Boisvert, R.; R. Pozo, K. Remington, R. Barrett, and J. Dongarra. 1997. "Matrix Market: A Web Resource for Test Matrix Collections," in *The Quality of Numerical Software: Assessment and Enhancement*. Ronald Boisvert, ed. London: Chapman and Hall, pp. 125–37. (http://math.nist.gov/MatrixMarket)

Bollserlev, Tim and Eric Ghysels. 1996. "Periodic Autoregressive Conditional Heteroscedasticity," *J. Bus. Econ. Statist.*, 14:2, pp. 139–51.

Box, George E. P. and Gwilym Jenkins. 1976. *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

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

Brown, Barry W. and Lawrence B. Levy. 1994. "Certification of Algorithm 708: Significant Digit Computation of the Incomplete Beta," *ACM Transactions on Math. Software*, 20:3, pp. 393–97.

Campbell, John Y. and N. Gregory Mankiw. 1987. "Are Output Fluctuations Transitory?" *Quart. J. Econ.*, 102, pp. 857–80.

Calzolari, Giorgio and Lorenzo Panattoni. 1988. "Alternative Estimators of FIML Covariance Matrix: A Monte Carlo Study," *Econometrica*, 56:3, pp. 701–14.

Chaitin-Chatelin, Francoise and Valérie Frayssé. 1996. *Lectures on Finite Precision Computations*. Philadelphia: SIAM.

Coveyou, R. and R. MacPherson. 1967. "Fourier Analysis of Uniform Random Number Generators," *J. ACM*, 14:1, pp. 100–19.

Dahlquist, Germund and Ake Björck. 1974. *Numerical Methods*. New York: Prentice-Hall.

Davidson, Russell and James G. MacKinnon. 1993. *Estimation and Inference in Econometrics*. New York: Oxford.

Demmel, James. 1981. "Effects of Underflow on Solving Linear Systems," manuscript, Computer Science Div., U. C. Berkeley.

———. 1984. "Underflow and the Reliability of Numerical Software," *SIAM J. Sci. Statist. Computing*, 5:4, pp. 887–919.

Dennis, J. E. and Robert B. Schnabel. 1996. *Numerical Methods for Unconstrained Optimization*. Philadelphia: SIAM Press.

Dewald, William G.; Jerry G. Thursby, and Richard G. Anderson. 1986. "Replication in Empirical Economics: The *Journal of Money, Credit and Banking* Project," *Amer. Econ. Rev.*, 76:4, pp. 587–603.

Dongarra, J. J. and G. W. Stewart. 1984. "LINPACK—Package for Solving Linear Systems," in *Sources and Development of Mathematical Software*. W. Cowell, ed. NY: Prentice-Hall, pp. 20–48.

Dudewicz, E. and T. Ralley. 1981. *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code*. Columbus, OH: American Sciences Press.

Dwyer, Jerry and K. B. Williams. 1996. "Testing Random Number Generators," *C/C++ Users J.*, June, pp. 39–48.

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

Elliott, Alan C.; Joan S. Reisch, and Nancy P. Campbell. 1989. "Benchmark Data Sets for Evaluating Microcomputer Statistical Programs," *Collegiate Microcomputer*, 7:4, pp. 289–99.

Feigenbaum, Susan and David M. Levy. 1993. "The Market for (Ir)Reproducible Econometrics," *Soc. Epistemology*, 7:3, pp. 243–44.

———. 1994. "The Self-Enforcement Mechanism in Science," manuscript, presented at the AEA meetings, Boston.

Fiorentini, Gabriele; Giorgio Calzolari, and Lorenzo Panattoni. 1996. "Analytic Derivatives and the Computation of GARCH Estimates," *J. Appl. Econometrics*, 11:4, pp. 399–417.

Ford, B. and J. Rice. 1994. "Rationale for the IFIP Working Conference: The Quality of Numerical Software: Assessment and Enhancement," IFIP–WG2.5 doc., Raleigh.

Forsythe, George E. 1970. "Pitfalls in Computation, or Why a Math Book Isn't Enough," *Amer. Math. Monthly*, 77, pp. 931–56.

Fox, L. 1971. "How To Get Meaningless Answers in Scientific Computation (and What To Do about It)," *IMA Bul.*, 7:10, pp. 296–302.

Francis, Ivor. 1981. *Statistical Software: A Comparative Review*. NY: North-Holland.

———. 1983. "A Survey of Statistical Software," *Computational Statist.*, 1:1, pp. 17–27.

Francis, Ivor; Richard M. Heiberger, and Paul F. Velleman. 1975. "Criteria and Considerations in the Evaluation of Statistical Software," *Amer. Statistician*, 29:1, pp. 52–56.

Gentle, James E. 1998. *Numerical Linear Algebra with Applications in Statistics*. NY: Springer.

———. 1998a. *Random Number Generation and Monte Carlo Methods*. NY: Springer.

Geweke, John. 1996. "Monte Carlo Simulation and Numerical Integration," in *Handbook of Computational Economics, vol. 1*. Hans M. Amman, David A. Kendrick, and John Rust, eds. Amsterdam: North-Holland, pp. 731–800.

Geweke, John and Michael Keane. 1997. "An Empirical Analysis of Income Dynamics Among Men in the PSID," Staff Rep. 233, Fed. Res. Bank Minneapolis.

Goldberg, David. 1991. "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys*, 23:1, pp. 5–48.

Goldberg, I. B. 1967. "27 Bits Are Not Enough for 8–Digit Accuracy," *Communications of the ACM*, 10:2, pp. 105–106.

Gourieroux, Christian and Alain Montfort. 1996. *Simulation Based Methods in Econometrics*. NY: Oxford U. Press.

Greene, William H. 1997. *Econometric Analysis*, 3e. NY: MacMillan.

Hall, Bronwyn. 1993. "The Stock Market's Valuation of Research and Development Investment During the 1980s," *Amer. Econ. Rev.*, 83:2, pp. 259–64.

Hamilton, James D. 1997. "Measuring the Liquidity Effect," *Amer. Econ. Rev.*, 87:1, pp. 80–97.

Hammarling, Sven. 1985. "The Singular Value Decomposition in Multivariate Statist.," *ACM Special Interest Group on Numerical Math.*, 20, pp. 2–25.

Higham, Nicholas J. 1991. "Algorithm 694: A Collection of Test Matrices in MATLAB," *ACM Transactions on Math. Software*, 17:3, pp. 289–305.

———. 1995. "The Test Matrix Toolbox for MATLAB v3.0," Numerical Analysis Rep. 276, U. Manchester, England. (http://www.ma.man.ac.uk/~higham/testmat.html).

———. 1996. *Accuracy and Stability of Numerical Algorithms*. Philadelphia: SIAM.

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

IEEE. 1985. *Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754–1985*. NY: Inst. Electrical and Electronics Engineers. Reprinted in *SIGPLAN Notices*, 1987, 22, pp. 9–25.

Johnson, Norman L.; Samuel Kotz, and N. Balakrishnan. 1994. *Continuous Univariate Distributions, Vol. 1, 2e*. NY: Wiley Interscience.

———. 1995. *Continuous Univariate Distributions, vol. 2, 2e*. NY: Wiley Interscience.

Judd, Kenneth L. 1998. *Numerical Methods in Economics*. Cambridge, MA: MIT Press.

Kahan, William. 1997. "The Baleful Effect of Computer Languages and Benchmarks upon Applied Mathematics, Physics and Chemistry," John von Neumann Lecture, 45th Annual Meeting of SIAM.

Kennedy, William J. and James E. Gentle. 1980. *Statistical Computing*. NY: Marcel-Dekker.

Knüsel, Leo. 1989. *Computergestützte Berechnung Statistischer Verteilungen*. Oldenburg, München-Wien. (English versions at www.stat.uni-muenchen.de/~knuesel/elv).

———. 1995. "On the Accuracy of the Statistical Distributions in GAUSS," *Computational Statist. Data Analysis*, 20:6, pp. 699–702.

———. 1996. "Telegrams," *Computational Statist. Data Analysis*. 21:1, p. 116.

———. 1998. "On the Accuracy of Statistical Distributions in Microsoft Excel," *Computational Statist. Data Analysis*, 26:3, pp. 375–77.

Knuth, Donald E. 1997. *The Art of Computer Programming, 3e*. Reading, MA: Addison-Wesley.

Lachenbruch, P. A. 1983. "Statistical Programs for Microcomputers," *Byte*, 8:8, pp. 560–70.

L'Ecuyer, Pierre. 1992. "Testing Random Number Generators," in *Proceedings of the 1992 Winter Simulation Conference*. J. J. Swain, D. Goldsmith, R. C. Crain, and J. R. Wilson, eds. NY: IEEE Press, pp. 305–13.

———. 1994. "Uniform Random Number Generation," *Annals of Operations Research*, 53, pp. 77–120.

Lesage, James P. and Stephen D. Simon. 1985. "Numerical Accuracy of Statistical Algorithms for Microcomputers," *Computational Statist. Data Analysis*, 3:1, pp. 47–57.

Letson, David and B. D. McCullough. 1998. "Better Confidence Intervals: The Double Bootstrap with No Pivot," *Amer. J. Agr. Econ.*, 80:3, pp. 552–59.

Ling, Robert F. 1974. "Comparison of Several Algorithms for Computing Sample Means and Variances," *J. Amer. Statist. Assoc.*, 69:348, pp. 859–66.

Longley, James W. 1967. "An Appraisal of Computer Programs for the Electronic Computer from the Point of View of the User," *J. Amer. Statist. Assoc.*, 62:319, pp. 819–41.

Lovell, Michael C. and David D. Selover. 1994. "Econometric Software Accidents," *Econ. J.*, 104, pp. 713–26.

MacKie-Mason, Jeffrey K. 1992. "Econometric Software: A User's View," *J. Econ. Perspectives*, 6:4, pp. 165–88.

MacKinnon, James G. 1996. "Numerical Distribution Functions for Unit Root and Cointegration Tests," *J. Appl. Econometrics*, 11:6, pp. 601–18.

Marsaglia, George. 1968. "Random Numbers Fall Mainly in the Planes," *Proceedings Nat. Academy Sciences USA*, 60, pp. 25–28.

———. 1985. "A Current View of Random Number Generators," *Computer Science and Statistics: 16th Symposium on the Interface*. L. Billard, ed. Amsterdam: North-Holland, pp. 3–10.

———. 1993. "Monkey Tests for Random Number Generators," *Computers and Math. with Applications*, 26:9, pp. 1–10.

———. 1996. *DIEHARD: A Battery of Tests of Randomness*. http://stat.fsu.edu/pub/ diehard.

McCullough, B. D. 1997. "Benchmarking Numerical Accuracy: A Review of RATS v4.2," *J. Appl. Econometrics*, 12:2, pp. 181–90.

———. 1998. "Assessing the Reliability of Statistical Software: Part I," *Amer. Statistician*, 52:4, pp. 358–66.

———. 1998a. "Algorithm Choice for (Partial) Autocorrelation Functions," *J. Econ. Soc. Meas.*, forthcoming.

———. 1999. "Assessing the Reliability of Statistical Software: Part II," *Amer. Statist.*, forthcoming.

———. 1999a. "Econometric Software Reliability: EViews, LIMDEP, TSP and SHAZAM," *J. Appl. Econometrics*, forthcoming.

———. 1999b. "Wilkinson's Tests and Econometric Software," *J. Econ. Soc. Meas.*, forthcoming.

McCullough, B. D. and Charles G. Renfro. 1999. "Benchmarks and Software Standards: A Case Study of GARCH Procedures," *J. Econ. Soc. Meas.*, forthcoming.

McCullough, B. D. and H. D. Vinod. 1998. "Implementing the Double Bootstrap," *Computational Econ.*, 12:1, pp. 79–95.

McCullough, B. D. and Berry Wilson. 1999. "On the Accuracy of Statistical Procedures in EXCEL97," *Computational Statist. Data Analysis*, forthcoming.

McKenzie, Colin. 1998. "A Review of Microfit 4.0," *J. Appl. Econometrics*, 13:1, pp. 77–89.

Murray, W. 1972. "Failure, the Causes and Cures," in *Numerical Methods for Unconstrained Optimization*. W. Murray, ed. NY: Academic Press, pp. 107–22.

Newbold, Paul; Christos Agiakloglou, and John Miller. 1994. "Adventures with ARIMA Software," *Int. J. Forecasting*, 10:4, pp. 573–81.

Park, Stephen K. and Keith W. Miller. 1988. "Random Number Generators: Good Ones Are Hard to Find," *Communications of ACM*, 31:10, pp. 1192–201.

Press, William H.; Saul A. Teukolsky, William T. Vetterling, and Brian R. Flannery. 1994. *Numerical Recipes in Fortran, 2e*. New York: Cambridge U. Press.

Priestley, M. B. 1981. *Spectral Analysis and Time Series*. London: Academic Press.

Renfro, Charles. 1997. "Normative Considerations in the Development of a Software Package for Econometric Estimation," *J. Econ. Soc. Meas.*, 23, pp. 277–330.

Ripley, B. D. 1990. "Thoughts on Pseudorandom Number Generators," *J. Computational Appl. Math.*, 31:1, pp. 153–63.

Rogers, Janet; James Filliben, Lisa Gill, William Guthrie, Eric Lagergren, and Mark Vangel. 1998. "StRD: Statistical Reference Datasets for Testing the Numerical Accuracy of Statistical Software," NIST# 1396, Nat. Inst. Standards and Tech.

Sawitzki, Günther. 1985. "Another Random Number Generator Which Should Be Avoided," *Statist. Software Newsletter*, 11, pp. 81–82.

———. 1994. "Testing Numerical Reliability of Data Analysis Systems," *Computational Statist. Data Analysis*, 18:2, pp. 269–86.

———. 1994a. "Report on the Numerical Reliability of Data Analysis Systems," *Computational Statist. Data Analysis* (SSN), 18:2, pp. 289–301.

Silk, Julian. 1996. "System Estimation: A Comparison of SAS, SHAZAM, and TSP," *J. Appl. Econometrics*, 11:4, pp. 437–50.

Simon, Stephen D. and James P. Lesage. 1988. "Benchmarking Numerical Accuracy of Statistical Algorithms," *Computational Statist. Data Analysis*, 7:2, pp. 197–209.

Stephens, Michael A. 1986. "Tests for the Uniform Distribution," in *Goodness-of-Fit Techniques*. R. D'Agostino and M. Stephens, eds. NY: Marcel-Dekker, pp. 331–66.

Stuart, Alan and J. Keith Ord. 1991. *Kendall's Advanced Theory of Statistics, vol. 2*. NY: Oxford U. Press.

Tauchen, George. 1993. "Remarks on My Term at *JBES*," *J. Bus. Econ. Statist.*, 11:4, pp. 426–31.

Tezuka, She. 1995. *Uniform Random Numbers: Theory and Practice*. Boston: Kluwer.

Thisted, Ronald A. 1988. *Elements of Statistical Computing*. NY: Chapman and Hall.

Tjostheim, Dag and Jostein Paulsen. 1983. "Bias of Some Commonly Used Time Series Estimates," *Biometrika*, 70:2, pp. 389–99.

Vandergraft, J. S. 1983. *Introduction to Numerical Computations*. Orlando, FL: Academic Press.

Veall, Michael R. 1991. "Shazam 6.2: A Review," *J. Appl. Econometrics*, 6:3, pp. 317–20.

Vinod, H. D. 1982. "Enduring Regression Estimator," in *Times Series Analysis: Theory and Practice 4*. O. D. Anderson, ed. Amsterdam: North-Holland, pp. 397–416.

———. 1989. "A Review of Soritec 6.2," *Amer. Statistician*, 43:4, pp. 266–69.

———. 1993. "Bootstrap, Jackknife, and Resampling Methods: Applications in Econometrics," in *Handbook of Statistics: Econometrics, vol. 11*. G. S. Maddala, C. R. Rao, and H. D. Vinod, eds. NY: North-Holland, pp. 629–61.

———. 1997. "Using Godambe-Durbin Estimating Functions in Econometrics," in *Selected Proceedings of the Symposium on Estimating Functions*. I. Basawa, V. P. Godambe, and R. Taylor, eds. IMS Lecture Notes Monograph Series, 32, pp. 215–37.

Vinod, H. D. and A. Ullah, 1981. *Recent Advances in Regression Methods*. NY: Marcel-Dekker.

Vinod, H. D. and L. R. Shenton. 1996. "Exact Moments for Autoregressive and Random Walk Models for a Zero or Stationary Initial Value," *Econometric Theory*, 12:3, pp. 481–99.

Wampler, Roy H. 1980. "Test Procedures and Test Problems for Least Squares Algorithms," *J. Econometrics*, 12:1, pp. 3–22.

Watson, Mark W. 1993. "Measure of Fit for Calibrated Models," *J. Polit. Econ.*, 101:6, pp. 1011–41.

Wilkinson, J. H. 1963. *Rounding Errors in Algebraic Processes*. Englewood Cliffs, NJ: Prentice-Hall.

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

———. 1994. "Practical Guidelines for Testing Statistical Software," in *Computational Statistics*. P. Dirschedl and Rüdiger Ostermann, eds. Berlin: Physica-Verlag, pp. 111–24.

Wilkinson, Leland and Dallal, Gerard E. 1977. "Accuracy of Sample Moments Calculations Among Widely Used Statistical Programs," *Amer. Statistician*, 31:3, pp. 128–31.