Journal of Economic and Social Measurement 30 (2005) 9–38
IOS Press

The sensitivity of econometric results to alternative implementations of least squares

Houston H. Stokes
Department of Economics, University of Illinois at Chicago, 601 South Morgan Street (M/C 144), Chicago, IL 60607, USA
E-mail: hhstokes@uic.edu

This paper is concerned with a detailed study of the accuracy tradeoffs of differences in data precision and alternative approaches to the estimation of OLS models. The implications of the analysis of a variety of problems, most of which have known answers, extend far beyond OLS modeling and directly impact any empirical analysis when the matrices are at all ill conditioned or "stiff." While the focus here is on linear modeling, the findings are equally, if not more, important to nonlinear modeling. Independent of the effect of the algorithm used, the precision in which the data was initially read was found to have a major impact on accuracy, even when the data was subsequently moved to a higher precision. This finding, illustrated best with the extremely multicollinear Filippelli data set, suggests that if a data base standard is agreed upon, the precision of the data saved will be of critical importance. By the use of variable precision arithmetic software, an extended benchmark was developed for the Filippelli data and the results compared to the real*8 and real*16 QR results. Much of the software developed for this paper has been put in the public domain to be used by other researchers.

1. Introductory remarks

1.1. Introductory remarks

In the last 40 years changes in operating systems, computer hardware, compiler technology and the needs of research in applied econometrics have all influenced econometric software development and the environment of statistical computing. However, despite a number of articles by McCullough and Vinod [14,15] and Renfro [22,23] and others, many economists are not aware of the impact on the accuracy of the calculation of using the alternate solution methods and data precisions that have been implemented in various statistical packages. Furthermore, less thought has usually been given to the impact on the accuracy of the final calculation that can be traced to the initial precision of the data saved in memory before it was moved to a higher precision for the calculation.[1] Where the moment matrix does not have a high degree of multicollinearity, the selection of the appropriate method of analysis may

[1]For example many software systems allow real*4 data storage but move the data to real*8 to make a calculation. In many cases the resulting accuracy is not the same as what would be obtained with a direct read into real*8. A simple example involving 2.00/4.11 will illustrate the problems of precision.

be less critical, provided double precision calculations are made and the underlying linear algebra software used is of high quality. However, due to the popularity of polynomial regression models and the present widespread use of long lag VAR and error correction models, rank problems often occur as the moment matrix becomes increasingly "stiff." While the focus of this paper is on the estimation of OLS models, many of the findings on how to increase accuracy can be used in nonlinear modeling where rank issues can potentially be more serious. Using four data sets of varying difficulty, three of which have "certified" answers from StRD [27], the effect of the estimation method on the number of correct digits is studied. In addition, various means by which accuracy of a given method can be increased are discussed and illustrated. A number of the alternatives considered involve modifications to BLAS [10] software that should be used throughout a modern software system. The consequence of these improvements will be reflected in improved accuracy of many other calculations throughout the software system. The routines developed in this paper have been released for general use and are described in some detail. Finally, the relationship between real*16 (64 bit) estimation and the precision of the underlying data storage is illustrated.

While data read into double precision (real*8) can be converted to real*16 to obtain greater accuracy, the results reported in the paper document the gain in accuracy if the data is directly read into real*16.[2] This finding, illustrated best with the extremely multicollinear Filippelli polynomial regression data set, suggests that if a data base standard is agreed upon, the precision of the data saved will be of critical

str=>vpa .4866180048661800486618004866180048661800486618004866180048661800486618004866180048661800486618004866180048661800486618004866M+0
r*8=>vpa .4866180048661800108045499266467 7M+0
r*8=>r*16 .4866180048661800108045499266467 7E+00
str=>r*16 .4866180048661800486618004866180 0E+00
r*8=>r*8 .4866180048661800000000000000000 0E+00
r*4=>r*4 .4866179820000000000000000000000 0E+00

The line str>vpa lists the exact answer obtained when the data (2.0 and 4.11) are read from a string into a variable precision arithmetic (VPA) routine while the line r*8=>vpa shows what happens to accuracy when the data are first read into real*8 or double precision, then moved to a vpa datatype. The line r*8=>r*16 shows what occurs when the data are first read into real*8, then converted to real*16 before making the calculation. In this case the results are the same as what is obtained with r*8=>vpa but are inferior to the line str=> r*16 where the data are read directly into real*16. The lines r*8=> r*8 and r*4=>r*4 show what can be expected using the usual double precision and single precision math, respectively. The importance of this simple example is it shows the effect of data storage precision and data calculation precision in a very simple problem where each can be isolated. When there are many calculations needed to solve a problem (to invert a 100 by 100 matrix by elimination involves a third of a million operations), round off error can mount, especially when numbers differ in size. Strang [34, p. 32] notes "if floating-point numbers are added, and their exponents c differ say by two, then the last two digits in the smaller number will be more or less lost . . ."

[2] Real*4 or single precision on IEEE machines has a range of $1.18*10^{-38}$ to $3.40*10^{38}$. This gives a precision of 7–8 digits at best. Real*8 or double precision has a range of $2.23*10^{-308}$ to $1.79*10^{308}$ and at best gives a precision of 15–16 digits. Real*16 has a range of 10^{-4931} to 10^{4932} and gives up to 32 digits of precision. VPA or variable precision arithmetic allows variable precision calculations.

importance.[3] By the use of variable precision arithmetic software, an extended benchmark was developed for the Filippelli data and the results were compared to the real*16 results to fully benchmark the gains of real*16 calculation. Testing of real*16 implementations is of increased importance due to the coming availability of 64 bit machines, which lower the cost of real*16/complex*32 calculation. While most modern Fortran compilers have supported real*16 and complex*32 data types, using software emulation, the availability of hardware implementations of these data types will make their growing availability in software systems in the future more likely. Since a number of software systems still save data and make calculations in real*4, the Pontius data set, which is of intermediate difficulty, is estimated using a variety of methods for real*8 data and real*4 data. The interesting result is while methods of analysis that involved formation of $(X'X)$ failed the condition check for both real*8 and real*4 data, if these checks were ignored, the results were surprisingly good. The QR method, however, gave superior performance in terms of accuracy. We next turn to some of the statistical issues before moving to a discussion of the examples.

1.2. Statistical background

Assuming $X$is a matrix of N observations on K right-hand side variables and y is a N element vector of values of the left-hand side variable, econometric textbooks tell us that the estimated solution vector to the ordinary least squares problem is $\hat{\beta} = (X'X)^{-1}X'y$ with coefficient standard errors as the square root of the diagonal elements of $\hat{\sigma}^2(X'X)^{-1}$ where $\sigma^2 = (y - X\hat{\beta})^2/(N - K)$. Usually, there is little emphasis on how best to solve $(X'X)^{-1}$, or whether in fact to calculate it at all. The condition of a matrix $X, C(X)$, defined as the ratio of the largest to the smallest singular value (to be defined below), can be used to help in this decision. Values of $C(X)$ obtained near 1 indicate the inverse of $X'X$ can be accurately formed and the matrix is deemed to be well conditioned. However, if $C(X)$ increases, there are increased difficulties in obtaining the inverse accurately. It can be proved that $C(X'X) = [C(X)]^2$. Since the formation of $X'X$ squares the condition, in cases where there is multicollinearity and the condition was already large, methods of solving for $\hat{\beta}$ that do not require the formation of $X'X$ such as the QR (defined later) and the SVD (defined later) may very well be the method of choice. The goal of this paper is to illustrate the gains of such methods as well as to discuss various ways to increase accuracy. Then impact of data precision will also be discussed.

If the inverse is desired, many practicing economists give little thought of the choice of an inversion approach. Assuming a full rank system, since $X'X$ is positive

[3]The question is whether the data base produces a real*4 data value, a real*8 data value, or a character representation of the exact digits of the basic data, which could then be read into the investigators precision of choice.

definite, the Cholesky factorization has been found to be substantially faster by a factor of at least 2 over a general matrix solution technique such as the LU factorization, although this is usually not discussed.[4] In discussing alternatives to the usual formula, Greene [7, p. 175] [5] notes, "the loss of accuracy in least squares computations occurs not in inverting $X'X$ but in accumulating it." Greene goes on to note "the singular value decomposition and QR decomposition . . . are generally the preferred approach to the computation of least squares."[6] A major focus of the present paper is to study empirically both "accumulation/precision issues" and "method of calculation issues," with the objective of showing what can be expected in terms of accuracy using various approaches on a variety of linear test problems. The examples have been selected to stress the software. All but one has been taken from the StRD data sets, which have been designed to exhibit data stiffness.[7] While the issue of accuracy was first brought to the attention of the profession by the work of Longley [11], in recent years a number of important papers by McCullough and Vinod [14,15], McCullough and Renfro [13], McCullough [16–18] and Renfro [23,24] have demonstrated that this is a subject that continues to be relevant today. This paper is a contribution to this ongoing discussion.

1.3. Overview of the paper

After first briefly discussing the alternative approaches to estimation, such as LU, Cholesky, QR and SVD, and why they might be used, this paper will outline a number of modifications to the BLAS library of utility programs that will increase real*8, real*16 and real*4 accuracy of the various approaches.[8] Issues of data storage,

[4] An exception is Judd [9, p. 60] who notes, "The advantages of the Cholesky decomposition is that it involves only $n^3/6$ multiplications and n square roots, which is about half the cost of Gaussian elimination for large n. It is also more stable than LU decomposition, particularly since there is no need for pivots." Once the Cholesky R is found, it is possible to directly solve the system of equations without explicitly obtaining $(X'X)^{-1}$. In a related paper on cointegration methods, Doornik and O'Brien [6] recommend a number of numerically stable methods that include the QR and the SVD, with the former being fastest and the latter being more suitable in reduced rank situations.

[5] In 2000 this was in the chapter on computation (3). In the 5^{th} edition in 2005, much of this material was moved to page 833 of Appendix A.

[6] Press [21, pp. 513–518] cautions the reader to use the QR method or the SVD method, except when the problem is easy. Press comments on the speed loss of the SVD but notes, "Its great advantage, that it (theoretically) cannot fail, more than makes up for its speed disadvantage." Results reported later suggest that not all SVD routines are created equal and use of the SVD can result in substantial accuracy loss in some cases.

[7] StRD [27] documentation refers to the work of Simon and Lesage [28], who note that as the number of constant leading digits increases, it becomes increasingly more difficult to make accurate computations.

[8] The changes to BLAS [10], LINPACK [5] and EISPACK [29] discussed in this paper and implemented in B34S (Stokes [31,32]) are being made available for other researchers to be freely used in any software systems, provided that attribution is given. The FTP library for this material is available under the research page of www.uic.edu/∼hhstokes. The file sourc3.f contains changes and additions made to LINPACK, BLAS and EISPACK. Other public domain code, such as FFTPACK, which was not modified, is also

whether to save data in real*8, real*4 or real*16, which have had little discussion in the literature, will be explored, using known examples.[9] Finally, variable precision arithmetic is used to extend a famous benchmark to validate the accuracy of the reported real*16 calculations.

2. Brief notes on various approaches to making and improving an OLS Model

2.1. Problems in solving an OLS model using the usual formulas

Since $(X'X)$ is a positive definite matrix , if the system is full rank, rather than using a LU factorization to calculate the inverse, one way to proceed is to perform Cholesky decomposition and express $(X'X) = R'R$ where R' is lower triangular. Rather than forming $(X'X)$ and losing accuracy in the process, the QR approach expresses $Q'X = \begin{bmatrix} R \\ 0 \end{bmatrix}$ where Q is N by N and orthogonal $(Q'Q = I)$. The QR approach to obtaining R is substantially more accurate, particularly in a number of difficult problems shown later. Once the QR factorization is performed, $\hat{\beta} = R^{-1}Q'y$. If $(X'X)^{-1}$ is needed to obtain the SE, the more accurate R obtained from the QR factorization of X can be used in place of the R obtained from a Cholesky factorization of $X'X$ to obtain $(X'X)^{-1}$. The SVD approach factors $X = U\Theta V'$ where U is N by K, V is K by K and U and V are orthogonal. Θ is a diagonal matrix with the singular values along the diagonal.[10] It is easy to

present in this file. Fortran code in this library can be freely placed in any software code, provided that attribution is made to the source. The nature of these changes will be discussed later in this paper. The file sourc2.f contains the LAPACK [1] library used and is basically unchanged over what is available from netlib. Comments on the routines in sourc3.f are welcome and should be addressed to hhstokes@uic.edu. In addition to these libraries, all jobs (that include data sets) and output that are used in this paper are shown.

[9]Modern software systems, such as SAS [2] and RATS [4], save data in real*8 or double precision and thus make the decision on data storage for the user. Up until recently this was the case for Matlab [12]. However, with the recent release (version 7.0 R14) this is no longer true. While the default data storage precision is real*8, real*4 is also supported. The decision on precision for data storage becomes more critical, since uninformed users can make a wrong choice. Results in this paper to be shown latter suggest that substantial gains in accuracy can be made if data is saved internally in real*16, not the default real*8. This finding suggests that real*4 storage of data may be more dangerous than previously thought. Renfro [26] in 1980 and more recently in a major paper in 1995 [25] has discussed data base system standards. In Fig. 1 of Renfro [25] it shows in single precision 6–7 digits are saved while in double precision 12–14 digits are saved. The results reported later in this paper suggest the former standard may be less than optimum, especially for data in log form.

[10]The singular values $s(i)$ of X are the square root of the eigenvalues of $X'X$'. Thus$[s(1)/s(k)]^2$ is the condition of the matrix $X'X$, since the singular values are ordered from large to small. The singular values of $X'X$ are in fact the eigenvalues sorted from largest to smallest. If the condition of $X'X$ is 10^d then the elements of $(X'X)^{-1}$ "can usually be expected to have fewer significant figures of accuracy that the elements of" $X'X$ LINPACK [5, p. 1.1]

Table 1
Partial list of improvements to BLAS

subroutine real16add(i)	Controls math accuracy
Scale a vector	
subroutine dscal(n,da,dx,incx)	Linpack
subroutine dqdscal(n,da,dx,incx)	Math done in real*16
subroutine cqscal(n,za,zx,incx)	complex*32 za complex*32
subroutine cqqscal(n,da,zx,incx)	complex*32 da real*16
subroutine qscal(n,da,dx,incx)	real*16
subroutine zscal(n,za,zx,incx)	Linpack
subroutine zdscal(n,da,zx,incx)	Linpack
Swap vectors for various precisions	
subroutine dswap(n,dx,incx,dy,incy)	Linpack
subroutine zswap(n,zx,incx,zy,incy)	complex*16
subroutine qswap(n,dx,incx,dy,incy)	real*16
subroutine cqswap(n,dx,incx,dy,incy)	complex*32
Dot product	
integer function idot(n,dx,incx,dy,incy)	
real*8 function ddot(n,dx,incx,dy,incy)	Linpack
real*8 function ddot_16(n,dx,incx,dy,incy)	High accuracy using IMSL
real*8 function ddot_2(n,dx,incx,dy,incy)	ACC2
real*16 function qdot(n,dx,incx,dy,incy)	Real*16
Accuracy improvements to real*8 calculations	
real*16 function qdble(x)	real*8 to real*16
real*16 function qqdmult(a,b)	real*16 mult of real*8
real*8 function dqdmult(a,b)	real*16 mult of real*8
real*16 function qqdadd(a,b)	real*16 add of real*8
real*8 function dqdadd(a,b)	real*16 add of real*8
real*16 function qqdsub(a,b)	real*16 sub of real*8
real*8 function dqdsub(a,b)	real*16 sub of real*8
real*16 function qqddiv(a,b)	real*16 div of real*8
real*8 function dqddiv(a,b)	real*16 div of real*8
real*16 function qqdpow(a,b)	real*16 ** of real*8
real*8 function dqdpow(a,b)	real*16 ** of real*8
Sum absolute values	
real*8 function dasum(n,dx,incx)	Linpack
real*8 function dqdasum(n,dx,incx)	real16add path
real*8 function dasum_2(n,dx,incx)	acc2 path
real*8 function dzasum(n,zx,incx)	Linpack
real*16 function qcqasum(n,zx,incx)	complex*32 version
real*16 function qasum(n,dx,incx)	real*16 version
Sum of a vector	
real*8 function dsum(n,dx,incx)	Linpack
real*8 function dqdsum(n,dx,incx)	real16add path
real*8 function dsum_2(n,dx,incx)	acc2
real*16 function qsum(n,dx,incx)	real*16 version
real*16 function qsum_2(n,dx,incx)	high accuracy real*16
complex*32 function cqsum(n,dx,incx)	complex*32 version

Table 1, continued

`integer function isum(n,dx,incx)`	`integer*4`
`double complex function zsum(n,dx,incx)`	`complex*16`
`Constant times a vector plus a vector`	
`subroutine daxpy(n,da,dx,incx,dy,incy)`	`Linpack`
`subroutine dqdaxpy(n,da,dx,incx,dy,incy)`	`Real16add path`
`subroutine qaxpy(n,da,dx,incx,dy,incy)`	`Real*16 version`
`subroutine cqaxpy(n,za,zx,incx,zy,incy)`	`Complex*32 version`

show that $\hat{\beta} = (V')^{-1}\Theta^{-1}U'y = V\Theta^{-1}U'y$. Since calculation of $(X'X)^{-1}$ is needed to obtain the SE of a OLS model, this can be obtained very quickly since it can be shown that $(X'X)^{-1} = V\Theta^{-2}V'$ since $(X'X) = V\Theta^{2}V'$ and $V' \equiv V^{-1}$. A problem arises in cases when the diagonal elements of Θ get very small due to multicollinearity and thus become very large when forming Θ^{-2}, causing numerical problems.

2.2. Simple modifications to BLAS to improve accuracy

In an important and widely cited paper, McCullough and Vinod [14] argued that the standard formula would not calculate the variance accurately in a number of test cases[11], while their "corrected" formula would. While this was a useful and important exercise to raise the consciousness of software developers on the need for numerical accuracy, the fact that the variance formula in many software systems was subsequently improved, and the software subsequently passed the benchmark test, gave no assurance that the rest of the program was sufficiently accurate. Furthermore, there was no really good way to test the sensitivity of results to accuracy without extensive and often impractical code modifications. The BLAS routines, released in 1979 by Lawson and others [10], provided a fast and modular way to perform basic calculations. The BLAS was extensively used in LINPACK [5] and subsequently extended to BLAS levels 2 and 3 and used in LAPACK [1]. If these routines are used throughout a software system, then changes in accuracy of basic calculations, such as dot product ($a = \sum_{i=1}^{n} x_i y_i$), summation $a = \sum_{i=1}^{n} x_i$ and elementary vector operations ($y = ax + y$), can be changed in one place and their effect on the resulting accuracy of calculations measured in many other places. The B34S (Stokes [31,32]) made these changes and the code for these improvements has been released as part of this article.[12] A partial list of the key real*8 BLAS routines is given in Table 1.

[11] While the text book formula for the variance is $\sum (x - \overline{x})^2/(n-1)$, a formula less likely to have rounding error is $\frac{1}{(n-1}\{\sum_{i=1}^{n}(x_i - \overline{x})^2 - \frac{1}{n}[\sum_{i=1}^{n}(x_i - \overline{x})]^2\}$.

[12] The changes to BLAS for inner products were to DDOT/ZDOTU/ZDOTC/QDOT/CQDOTU and CQDOTC. For summation DSUM/ZSUM/QSUM/CQSUM were changed. The absolute sum routines DASUM/QASUM were also changed as were the scale routines DSCAL/ZSCAL/QSCAL/CQSCAL and the transformation routines DAXPY/ZAXPY/QAXPY/CQAXPY. It is to be noted that the Qxxxx and CQxxxx routines were developed for B34S and are used to increase accuracy of real*16 and complex*32 data types. These are not the BLAS names.

Real*4 routines are not listed to save space.

As an example of what is involved in enhancing a BLAS routine, consider DDOT, which was modified as

```
      real*8 function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     This version has been simplified to not use unrolled loops.
c     It is intended to show the simple changes needed to implement
c     different accuracy paths. The unrolled loop version should be
c     used in production code.
c
c     Note: => dqddot is an IMSL routine!
c           => ddot_16 uses HHS real*16 mult
c
      implicit real*8(a-h,o-z)
      double precision dx(*),dy(*),dtemp,ddot_16,dqddot
      integer i,incx,incy,ix,iy,m,mp1,n
      logical ison
      common/real16/ison(3)
      save/real16/
c
      dtemp = 0.0d0
      ddot  = 0.0d0
      if(n.le.0)return
c
      if(ison(1))then
      if(ison(2))     ddot=dqddot(n,dtemp,dx,incx,dy,incy)
      if(.not.ison(2))ddot=ddot_16(n,dx,incx,dy,incy)
      return
      endif
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
c
      do i = 1,n
      dtemp = dtemp + (dx(ix)*dy(iy))
      ix = ix + incx
      iy = iy + incy
      enddo
c
      ddot = dtemp
      return
      end
```

to allow a branch to dqddot if ison(1) and ison(2) were .true. and ddot_16 if ison(1) was .true. and ison(2) was .false. . The routine ddot_16 does real*16 math internally and has the same accuracy as dqddot. A version without unrolled loops is

```
      real*8 function ddot_16(n,dx,incx,dy,incy)
      implicit real*8(a-h,o-z)
c
c     forms the dot product of two real*8 vectors.
c     uses real*16 math
c
      real*8 dx(*),dy(*),dbleq
      real*16 qqdmult,dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      dtemp   = 0.0q0
      ddot_16 = 0.0d0
      if(n.le.0)return
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
c
      do i = 1,n
      dtemp = dtemp + qqdmult(dx(ix),dy(iy))
      ix = ix + incx
      iy = iy + incy
      enddo
c
      ddot_16 = dbleq(dtemp)
      return
      end
```

where `qqdmult` multiplies two real*8 numbers in real*16 and saves the result in real*16.

```
      real*16 function qqdmult(a,b)
      implicit real*16(a-h,o-z)
c
c     multiplies two real*8 numbers in real*16
c     saves in real*16
c     built 26 May 2003 by Houston H. Stokes
c
      real*8 a,b
      external qdble
c
      qqdmult=qdble(a)*qdble(b)
      return
      end
```

As will be shown later `ddot_16` allows the inner product summation and multiplication to be done in real*16 and returns the answer in real*8, which improves accuracy every place that `DDOT` is called. The routine `dqddot` is an IMSL [20] routine that does the same thing as `ddot_16` except faster since it uses the IMSL routines `DQINI, DQADD, DQMUL` and `DQSTO,` which operate using real*8 math but give real*16 accuracy. When 64 bit hardware and compilers are available, the reverse will most

likely be true. The multiplication calculation in `ddot_16` could be replaced with inline code if `qqdmult(dx(ix),dy(iy))` was replaced by `qdble(dx(ix))*qdble(dy(iy))`. The reason this was not done was to experiment with and isolate real*8 to real*16 conversion in one routine, `qdble` which is listed next:

```
      real*16 function qdble(x)
c
      function to convert real*8 to real*16
c
      real*8 x
      real*16 y
      call r8tor16(x,y)
      qdble=y
      return
      end
      subroutine r8tor16(x,y)
      real*8 x
      real*16 y
c
c     real*8 to real*16 conversion since no fortran
c     function.
c
      y=x
c
      return
      end
```

Real*16 math accuracy can be enhanced by use of routines `QVXADD, QVXMUL` and `QVXSTO`, which were developed based on logic from no longer supported IMSL code from the 1980's. The routine `LGCOPY` is similar to the BLAS routine `DCOPY,` except that logical*1 data is copied to zero out portions of the real*16 number to perform the split.

```
      subroutine qvxadd(a,acc)
c
c    purpose        - extended precision add - better than real*16
c
c    usage          - call qvxadd (a,acc)
c
c    arguments  a   - real*16 number to be added to the
c                     accumulator. (input)
c               acc - accumulator. (input and output)
c                     acc is a real*16 vector of length
c                     2. on output, acc contains the sum of
                      input acc and a.
c
      real*16 a,acc(2),x,y,z,zz
c
      x = acc(1)
      y = a
      if (qabs(acc(1)).ge.qabs(a)) go to 1
```

```
      x = a
      y = acc(1)
c                          compute z+zz = acc(1)+a exactly
    1 z = x+y
      zz = (x-z)+y
c                          compute zz+acc(2) using real*16 math
      zz = zz+acc(2)
c                          compute acc(1)+acc(2) = z+zz exactly
      acc(1) = z+zz
      acc(2) = (z-acc(1))+zz
      return
      end
      subroutine qvxmul(a,b,acc)
c
c     purpose                 - real*16 extended precision (better than
c                               real*16 multiply)
c
c     usage                   - call qvxmul (a,b,acc)
c
c     arguments      a        - input real*16 number
c                    b        - input real*16 number
c                    acc      - accumulator. (input and output)
c                               acc is a real*16 vector of length
c                               2. on output, acc contains the sum of
c                               input acc and a*b.
c
c     logic changed 1 October 2004 by Houston H. Stokes for real*16
c
      real*16           a,b,acc(2),x,ha,ta,hb,tb
      logical           lx(16)
      equivalence       (x,lx(1))
c
c     logic split  a = ha+ta
c                  b = hb+tb
c                  compute ha*hb,ha*tb,ta*hb, and ta*tb
c                  and call qvxadd to accumulate the sum
c
      x=a
      ha=0.0q+00
      call lgcopy(8,.true.,0,lx,1)
      ha=x
      ta=a-ha
      x=b
      hb=0.0q+00
      call lgcopy(8,.true.,0,lx,1)
      hb=x
      tb=b-hb
c
      x = ta*tb
      call qvxadd(x,acc)
      x = ha*tb
      call qvxadd(x,acc)
      x = ta*hb
```

```
      call qvxadd(x,acc)
      x = ha*hb
      call qvxadd(x,acc)
      return
      end
      subroutine qvxsto(acc,d)
c
c     routine built by Houston H. Stokes
c     purpose           - real*16 store. (Better than real*16)
c
c     usage             - call qvxsto(acc,d)
c
c     arguments   acc   - accumulator. (input)
c                         acc is a real*16 vector of length
c                         2. acc is assumed to be the result of
c                         calling qvxadd or qvxmul to perform extended
c                         precision operations.
c                   d   - real*16 scalar. (output)
c                         on output, d contains a double precision
c                         approximation to the value of the extended
c                         precision accumulator.
c
      real*16 acc(2),d
c     first executable statement
      d = acc(1)+acc(2)
      return
      end
```

While enhancements to BLAS routines to provide additional accuracy are a relatively easy way to increase the accuracy of currently running code developed using real*8 and complex*16, the development of real*16 and complex*32 versions of key routines in EISPACK [29] and LINPACK [5] will be shown to provide further, and in many cases needed, accuracy for difficult problems.

2.3. Effect of data storage in memory and data reading

While most modern software systems save data in real*8, there are exceptions thatsave data in real*4 by default.[13] A related problem is data base systems that save transformed data in real*4 and thus negatively impact being able to perform accurate calculation of some statistical procedures.[14] With 64 bit hardware computing on the horizon and 64 bit software-based computing possible, it is important to seethe effect

[13] McCullough [17, p. 152] discussed problems of real*4 data storage. He notes, "... users should also be aware that single-precision storage can have an adverse effect on accuracy, even when the input data are single-precision." SCA is an example of a software system that saves data in real*4 by default, although real*8 storage is also possible. The effect of real*4 data storage on accuracy is studied later with the Pontius data set.

[14] Since the B34S allows data to be transformed to real*4 and then recopied back to real*8, it is possible to simulate the effect of real*4 databanks.

of data storage on accuracy.[15] While the usual approach is just to boost accuracy for a calculation, evidence shown later in this paper suggests that much is lost by this approach. For difficult problems, more accuracy can be obtained if data are read directly into a real*16 variable. This will be shown later using the StRD [27] Filippelli data set. While the Filippelli OLS data set, if read into a real*8 variable, will not solve with a Cholesky factorization, if the data are loaded into a real*16 variable, more accuracy than the QR on real*8 data is obtained with a Cholesky factorization.[16] However, if the same data were directly loaded in real*16, then many more accurate digits are obtained. This finding, which will be discussed in some detail later, suggests a number of important things. First, a data copy into real*16 from real*8 can help but is no substitute for a direct read into real*16. While it could be said that the Filippelli data set is known to cause problems and that the findings are not relevant for most problems, it remains interesting to see what happens when one can see a difference.

2.4. Accuracy issues involving the variance

To test if the McCullough-Vinod [14] formula was actually needed to calculate the variance, the StRD Numerical-Accuracy-4 data set, which provides the "stiffest" variance calculation, was selected. Here the sample mean, standard deviation and autocorrelation are known exactly and are 10000000.2, 0.1 and -0.999, respectively. To measure the number of significant digits in the answer, McCullough [16] suggests using the log relative error, defined as $LRE = -\log_{10}(|x - c|/|c|$ for $c \neq 0$ and $-\log_{10}(|x|)$ otherwise, where $x =$ the answer obtained and $c =$ the "correct" or target answer. Using the standard B34S[17] matrix command formulas for the mean, standard deviation and auto correlation, the answers obtained for the above problem were, respectively, 10000000.20000000, 0.1000000005587935 and -0.9989999999813732. These give LRE values of 15.0, 8.25 and 10.73 respectively.[18] The test data set consists of 1001 observations of three data values (10000000.2, 10000000.1, 10000000.3). A number of tests were run, using first the default B34S variance

[15] Fortran 77 has had the data types REAL*16 and COMPLEX*32 for many years. The implementation of arithmetic using these data precisions was software-based and slow. With 64 bit machines available, these calculations will substantially speed up in the future as hardware solutions will be implemented.

[16] The QR using real*16 data is still more accurate, although for data converted from real*8 to real*16, both the Cholesky and the QR will have the same degree of accuracy. The Filippelli problem, when estimated with RATS [4] version 6.03, gives no indication that there is a problem except producing some 0.0 coefficients. If the same problem is run with the SAS ORTHOREG procedure, no correct coefficients are produced and β_9 is set to missing. Table 10, discussed in Section 4, documents a real*16 Filippelli benchmark developed with variable precision arithmetic.

[17] All calculations have been done using the B34S MATRIX command unless otherwise stated.

[18] The error of the autocorrelation was -0.1862676679564856E-10, which is clearly acceptable for most work. The same can be said for the standard deviation calculation. As will be shown later, the LRE value can be increased to 26.31 if data are read into real*16 and real*16 math is used. McCullough [18, p. 17] obtained LRE values of 15, 8.3 and 15 for real*8 data.

Table 2
The effect of formulas and data precision on standard deviation calculation

```
Experiment # 1 No Accuracy improvements for Case B. Real*8 Data
Case A 0.1000000005587935                    LRE 8.25
Case B 0.1000000005588410                    LRE 8.25

Experiment # 2 Accuracy improvements for Case B. Real*8 Data
Case B 0.1000000005587935                    LRE 8.25

Experiment # 3 Real*8 to Real*16 Data.  No accuracy improvements
Case A 0.10000000055879354477361958556452   LRE 8.25
Case B 0.10000000055879354477361958556452   LRE 8.25

Experiment # 4 Data read directly into Real*16
Case A 0.10000000000000000000000000048468   LRE 26.31
Case B 0.10000000000000000000000000048468   LRE 26.31

Experiment # 5 Real 8 to Real*16 Data.  Real32 accuracy on
++++++++++ converted real*16 +++++++++++++++
Case A 0.10000000055879354477361958556452   LRE 8.25
Case B 0.10000000055879354477361958556452   LRE 8.25 ++++++++++

Experiment # 6 Data read directly into real*16.  Real32 on.
Case A 0.10000000000000000000000000048468   LRE 26.31
Case B 0.10000000000000000000000000048468   LRE 26.31
```

Case A uses the high accuracy formula with high accuracy enhancements.
Case B uses the standard formula as listed in footnote 9.
LRE is the McCullough [16] accuracy measure. The exact answer is .1.

routine that consists of both the McCullough-Vinod formula and high accuracy summation and product enhancements and, second, the traditional formula (programmed with the B34S matrix command language), with and without accuracy enhancements. Both approaches were run on real*8 data, real*8 data copied to real*16 and data read directly into real*16. The results are listed in Table 2.

In experiment # 1 for real*8 data the LRE was the same for both approaches (8.25) but the answers were slightly different. In experiment # 2 BLAS accuracy enhancements are turned on and both approaches obtain the same answer. This suggests that the real*8 accuracy enhancements obviate the need for the "correct" variance formula discussed in footnote 11.

Experiment # 3 copies the data to real*16 from real*8, makes all calculations in real*16 and observes no accuracy gain. Experiment # 4, however, reads the data directly into real*16 before making calculations in real*16 and obtains a LRE of 26.31, a substantial gain. The importance of this is that the data storage precision makes a difference. If calculations are desired in real*16, for difficult problems it is imperative that accuracy not be lost in the initial data read, where truncation errors occur due to inexact binary representation, especially for "stiff" data. While it is known that this problem may occur, the above example illustrates the nature of the loss. Reading into real*16 directly makes a major difference. Experiments # 5 and # 6 test if accuracy enhancements to real*16 calculations, using `QVXADD, QVXMUL` and `QVXSTO`, will make a difference. The finding is that for this problem it does not make a difference. Variable precision arithmetic calculation, not shown in the table, using 60 digits of accuracy produces the exact answer.

2.5. Speed issues for the SVD

Press et al. [21] advise use of the QR or SVD but remarks that there are speed disadvantages of the SVD. While results presented in the next section suggest that there can be some accuracy losses using the SVD, this section will address the speed issues of two SVD candidates using two types of CPU. In test # 1, reported on the top of Table 3, the results suggest that up to a matrix of order 400, LINPACK is faster, while for larger systems LAPACK runs more quickly. These tests were run on a Dell Latitude running a 1.1 Gh processor and Windows 2000. Since most problems involving OLS are for matrices of order less than 350, the choice appeared to be to use LINPACK, based on speed of calculation for these matrices and LAPACK for systems bigger than order 350. However, when the above tests are run on a Dell 650 workstation running a 3.05 Xeon chip and Windows XP, there appears to be no speed gain if LAPACK is used for large systems. Due to the chip design, there were measured performance gains for both routines.[19] For example, when we compare 600-order systems to 200-order systems on the Dell Latitude, the time increased 84.6 fold (27.96/0.3305) for LINPACK and 52.3 (20.43/0.3906) for LAPACK. For tests using the Xeon chip, these numbers were 38.1 (4.766/0.1250) and 37.7 (6.484/0.1719), respectively. For LAPACK the optimum workspace was set by the routine on the first call. For all tests the singular values Θ, the first K rows of U and the full V was calculated. The square matrices used were built using the IMSL [20] random normal generator. In the next section the two SVD code choices are tested for accuracy differences.

3. Results for OLS models

3.1. Filippelli polynomial data set

To measure the effects of data precision and calculation method on accuracy requires a number of different test data sets. The first problem attempted was the StRD [27] Filippelli data set, which contains 82 observations on a polynomial model of the form $y = \beta_0 + \sum_{i=1}^{10} \beta_i x^i + e$ where x ranges from -3.13200249 to -8.781464495 and x^{10} ranges from 90,828.258 to 2,726,901,792.451598. Answers to 15 digits are supplied by StRD.[20] Table 4 reports 15 experiments involving various

[19] To test if this finding was due to Windows 2000 vs XP, the above exercise was run on a Dell Xeon workstation running Red Hat Linux and having 2.5 Gh chips. Results comparable to the Xeon results reported in Table 3 were obtained. This suggests it is multi-thread chip design that is causing the difference. All tests were run using Lahey version 7. Fortran compilers. The developers of LAPACK [1] have noticed similar CPU design-sensitive effects. Their results, done in 1992, did not include modern Intel chips.

[20] StRD documentation reports that while 15 digits are given, due to truncation, the answers are certified up to the last digit. To summarize accuracy, the average LRE value is given for each model. Moler [19,

ways to estimate the model. The LINPACK Cholesky routines and general matrix routines detect rank problems and will not solve the problem if the data are not converted to real*16. The QR approach obtains an average LRE of 7.306, 7.415 and 8.368 on the coefficients, SE and residual sum of squares. The exact numbers obtained are listed in Table 5. If the accuracy improvements for the BLAS routines suggested in section 2.2 are enabled, these LRE numbers jump to 8.118, 8.098 and 9.803, respectively. Note that both accuracy improvements result in the same gain. Experiments # 4 and # 5 first copy the data that have been first read into real*8 into a real*16 variable and attempt estimation with a Cholesky and a QR approach. The LRE's are the same for both approaches (7.925, 8.708, 8.167). This experiment shows the effect of calculation precision and at first would lead one to believe that there is little gain obtained using real*16 calculation except for the fact that the Cholesky condition is not seen as 0.0. However, this interpretation would be premature without checking for data base precision effects (i. e., at what precision was the data initially read), which we do below.

Experiments 6–12 test various combinations of calculation precision and routine selection. In Experiment # 6 we use the LINPACLK SVD routines on real*8 data. The results are poor (LRE numbers of 2.195, 2.132 and 4.039). [21] When the accuracy improvements are enabled, (experiment 7 and 8), there is a slight loss of accuracy on the coefficients to 1.901 but a slight gain on the SE to 2.431. However, when the real*8 data are copied to real*16 in experiment 9, the SVD LRE numbers jump to 7.924, 8.708 and 8.167, respectively, which are similar to what was found in experiments 4 and 5 and show clearly the effect of calculation precision conditional on data reading into real*8 before the data are moved to real*16. These results are similar to those in the real*16 Cholesky experiment 4 and the real*16 QR experiment 5.

Experiments 10–12 study the effect of using LAPACK's SVD routine in place of LINPACK. For experiment 10, the coefficient LRE jumps to 7.490, which is quite good and in fact beats the QR LRE reported for experiment 1. This value is far better than the LINPACK LRE of 2.195. [22] However, the LRE of the SE is poor with

Chapter 5] discussed this data set in problem 5.10 noting that this problem is "controversial" and that there are "several opinions about whether or not this is a reasonable problem." There is no disagreement over the fact that this is a hard problem and thus makes an excellent candidate for stressing a solution method or software implementation. McCullough [17] reports no solution for SAS or SPSS but LRE values of 7.1, 7.0 and 7.8 for SPLUS for coefficients, SE and r, respectively. These LRE values are in line with what one would expect with a QR solution using real*8 data. Somewhat better results are reported for B34S in Table 4.

[21] The author has used the LINPACK code since 1979. These results were not expected and seem to be related to the extreme values in the X matrix in the Filippelli data. When real*16 is used, accuracy of the LINPACK SVD routine improves.

[22] McCullough [18] Used LAPACK QR and SVD routines to estimate the coefficients of the Filippelli data finding that "QR generally returns more accurate digits than SVD." The LRE values found were 7.4 and 6.3 respectively. For S-PLUS he found 8.4 and 5.8, respectively, where the underlying routines were not known.

Table 3
Speed differences of the SVD calculation by CPU type and matrix size

Obs	Order	Linpack	Lapack	Ratio
Test 1 Relative Speed of Linpack/LAPACK SVD on Dell Latitude 1.1 Gh				
1	150.0	0.1302	0.1302	1.000
2	200.0	0.3305	0.3906	0.8462
3	250.0	0.8112	1.001	0.8100
4	300.0	1.793	2.063	0.8689
5	350.0	3.405	3.555	0.9577
6	400.0	5.798	5.778	1.003
7	450.0	8.863	8.132	1.090
8	500.0	12.60	12.30	1.024
9	550.0	21.10	15.49	1.362
10	600.0	27.96	20.43	1.369
Test 2 Relative Speed of Linpack/LAPACK SVD on DELL 650 3.05 Gh Xeon				
1	150.0	0.4688E-01	0.6250E-01	0.7500
2	200.0	0.1250	0.1719	0.7273
3	250.0	0.2344	0.3750	0.6250
4	300.0	0.5312	0.7344	0.7234
5	350.0	0.8906	1.203	0.7403
6	400.0	1.359	1.906	0.7131
7	450.0	1.969	2.594	0.7590
8	500.0	2.703	3.641	0.7425
9	550.0	3.656	4.781	0.7647
10	600.0	4.766	6.484	0.7349

The LINPACK SVD routine used was DSVDC, while for LAPACK DGESVD was used.

a LRE of 1.910, which is less than that found with the LINPACK code of 2.132. The LRE of $e'e$ of 1.606 is also less than the LINPACK LRE of 3.258. Since the SE requires knowledge of $(X'X)^{-1}$, calculated as $(X'X)^{-1} = V\Theta^{-2}V'$, extreme values along the diagonal of Θ may be causing errors when forming Θ^{-2}. However, this possibility does not explain the poor performance of the residual sum of squares LRE of 1.606.[23] The reason may be related to the fact that the data set has such high x^{10} values that minor coefficient differences will result in substantial changes in the relative residual sum of squares.

Experiments 13–15 first load the data in real*16 and proceed to the same routines as used for experiments 4–6. Here we see LRE numbers of 14.68 14.99 and 15.00 for the Cholesky experiment and 14.79, 14.96 15.00 for the QR experiment which is the same as SVD (LINPACK). These are close to perfect answers. Table 5 lists the coefficients obtained for experiment 1, which used real*8 data while Table 6 lists the exact coefficients obtained for the QR using data read directly into real*16. Experiments 13–15 show gain from reading the Filippelli data set in real*16. Since all there experiments produced similar LRE values, it suggests that if the data are read with enough precision, the results are less sensitive to the estimation method.

[23] The sum of squares was tested against the published value of 0.795851382172941E-03. The LAPACK SVD routine obtained 0.8155689538070673E-03.

Table 4
LRE for various approaches to an OLS model of the Filippelli data

Experiment	Type	Coef	SE	RSS_LE
Various options of real*8 data				
1	QR	7.306	7.415	8.368
2	ACC_1	8.118	8.098	9.803
3	ACC_2	8.118	8.098	9.803
4	R16_CHOL	7.924	8.708	8.167
5	R16_QR	7.924	8.708	8.167
6	SVD	2.195	2.132	4.039
7	SVD_ACC1	1.901	2.431	3.258
8	SVD_ACC2	1.901	2.431	3.258
9	SVD_R16	7.924	8.708	8.167
10	SVD_LAPK	7.490	1.910	1.606
11	SVD2ACC1	7.490	1.910	1.606
12	SVD2ACC2	7.490	1.910	1.606
Various Options using Data read directly in real*16				
13	R16_CHOL	14.68	14.99	15.00
14	R16_QR	14.79	14.96	15.00
15	R16_SVD	14.79	14.96	15.00

Experiments 4, 5 and 9 involve reading data first into real*8 and then converting the data to real*16. Experiments 1–3, 6–8 and 10–12 involve real*8 data. Experiments 13–15 use data read directly into real*16. See Section 2.1 for a detailed discussion of the methods used, the data and the software and settings involved. The coefficients obtained for experiment # 1 and 14 are listed in Tables 5 and 6.

This finding has important implications for data base design and is similar to what was found with the variance calculations in Section 2. The next task is to study less extreme (stiff) data sets and observe the results.

3.2. Analysis of the residual sum of squares of the Gas Furnace data

The Box-Jenkins [3] Gas Furnace data have been widely studied and modeled and are close in difficulty to what are found in many applied models in time series. While "correct" agreed upon answers are not available, it is possible to study the effect on the residual sum of squares using 11 approaches reported in Table 7.[24] Since OLS minimizes the sum of squared errors, a "better" answer is one with a smaller $e'e$. Using this criteria, the LINPACK general matrix solver DGECO, Experiment 3, is "best" followed closely by the LAPACK general matrix solver, Experiment 4, and the LINPACK SVD routine, Experiment 10. Experiments 5 and 6 use the LAPACK general matrix solver that allows refinement and, in the case of Experiment 6 refinement and equilibration. These approaches did not do as well in determining a minimum $e'e$ and were substantially more expensive in terms of computer time. Of interest is why Experiment 1 and Experiment 8 did not produce the same answer

[24] Since this data set does not have the rank problems found with the Filippelli data, it is possible to attempt a number of alternative procedures. Not all these procedures should be used.

Table 5
Coefficients and SE estimated using QR on Real*8 Filippelli data

	Test value	Value obtained	LRE
Coef 1	−2772.179591933420	−2772.179723094652	7.33
Coef 2	−2316.371081608930	−2316.371192269638	7.32
Coef 3	−1127.973940983720	−1127.973995395338	7.32
Coef 4	−354.4782337033490	−354.4782509735776	7.31
Coef 5	−75.12420173937571	−75.12420543777237	7.31
Coef 6	−10.87531803553430	−10.87531857690271	7.30
Coef 7	−1.062214985889470	−1.062215039398714	7.30
Coef 8	−0.6701911545934081E-01	−0.6701911887876555E-01	7.29
Coef 9	−0.2467810782754790E-02	−0.2467810910390330E-02	7.29
Coef 10	−0.4029625250804040E-04	−0.4029625462234867E-04	7.28
Coef 11	−1467.489614229800	−1467.489683023960	7.33
Mean LRE	7.306448565286121		
Variance LRE	2.587670394878226E-04		
Minimum LRE	7.280096349919187		
Maximum LRE	7.329023461850447		
SE 1	559.7798654749500	559.7798867059487	7.42
SE 2	466.4775721277960	466.4775900975754	7.41
SE 3	227.2042744777510	227.2042833290517	7.41
SE 4	71.64786608759270	71.64786889794284	7.41
SE 5	15.28971787474000	15.28971847592676	7.41
SE 6	2.236911598160330	2.236911685945726	7.41
SE 7	0.2216243219342270	0.2216243305780890	7.41
SE 8	0.1423637631547240E-01	0.1423637686503493E-01	7.41
SE 9	0.5356174088898210E-03	0.5356174292732132E-03	7.42
SE 10	0.8966328373738681E-05	0.8966328708850490E-05	7.43
SE 11	298.0845309955370	298.0845420801842	7.43
Mean LRE	7.414701487211084		
Variance LRE	7.386168559949404E-05		
Minimum LRE	7.405390067654106		
Maximum LRE	7.429617565744895		
Residual sum of squares:			
RSS	0.7958513821729410E-03	0.7958513787598208E-03	8.37

Test values are reported on the left-hand side. LRE = log relative error. The coefficients report experiment # 1 from Table 4. The same LINPACK QR routine was modified by Stokes [31,32] to run for real*16 data. Results for this experiment are shown in Table 6.

since they both used the LINPACK Cholesky routines. The answer relates to the way the coefficients are calculated. In the former case the Cholesky R is used to obtain the coefficients without explicitly forming $(X'X)^{-1}$ using the LINPACK routine DPOSL, while in the latter case $(X'X)^{-1}$ is formed from R using DPODI. In general, the answers are very close for this exercise.

3.3. Pontius and Eberhardt data

The StRD Pontius data are classified as of a lower level of difficulty, although more challenging than the gas furnace data studied in the prior section. The Pontius

Table 6
Coefficients estimated with QR using Real*16 Filippelli data

		LRE
Coefficients Using QR on Data Loaded into Real*16		
1.	−2772.17959193342392802844755357 21	14.85
2.	−2316.37108160893075882196791409 78	15.00
3.	−1127.97394098371569857167001419 98	14.42
4.	−354.47823370334877161073848496470	15.00
5.	−75.12420173937571389052207552268 4	15.00
6.	−10.87531803553425108528108117714 5	14.35
7.	−1.06221498588946766459661122023 56	14.66
8.	−0.67019115459340837592673412281191E-01	15.00
9.	−0.24678107827547865084085445245647E-02	15.00
10.	−0.40296252508040367129713154870917E-04	15.00
11.	−1467.48961422979588228784851359 61	14.55
Mean LRE	14.78849032026654398083538227609 1684	
Variance LRE	6.35696189088290126357127829540993 25E-0002	
Minimum LRE	14.34700240396972432281375901621 1991	
Maximum LRE	15.00000000000000000000000000000000	
SE Using QR on DATA Loaded into Real*16		
1.	559.77986547494987457477254797527	15.00
2.	466.47757212779645269310982974610	15.00
3.	227.20427447775131062939817526228	14.86
4.	71.64786608759273726166572085071 8	15.00
5.	15.28971787474000650307567897859 2	15.00
6.	2.23691159816033275551862340397 71	14.91
7.	0.22162432193422740206612983379340	14.74
8.	0.14236376315472394891823309147959E-01	15.00
9.	0.53561740888982093625865193118466E-03	15.00
10.	0.89663283737386822210041526987951E-05	15.00
11.	298.08453099553698520055234224439	15.00
Mean LRE	14.95590357667528354544498664221 3045	
Variance LRE	7.21747790968588647686082879348146 69E-0003	
Minimum LRE	14.74131993032301177290697604300 0329	
Maximum LRE	15.00000000000000000000000000000000	
Residual sum of squares		
0.79585138217294058848463068814293E-03		15.00

LRE = log relative error. This is experiment # 14 from Table 4. LINPACK QR routine modified by Stokes [31,32] to run with real*16 data. For this experiment the data was read directly into real*16.

data consists of 40 observations of a model of the form $y = \beta_0 + \beta_1 x + \beta_2 x^2$ for a model which is almost a perfect fit. The eigenvalues of $(X'X)$, as calculated by the EISPACK routine `RG`, were `0.8109E+13, 0.7317E+27, 3.613,` giving a condition estimate that tripped the condition tolerance in the LINPACK LU and Cholesky routines for both real*8 and real*4 data. Calculations were "forced" by ignoring this check.[25] Results are reported for a number of experiments in Table 8 that

[25]The same data was estimated in Windows RATS [4] version 6.0. While the reported coefficients agreed with the benchmark for 11, 11 and 14 digits, respectively, RATS unexpectedly produced a SE of

Table 7
Residual sum of squares on a VAR model of order 6 – Gas Furnace data

Residual Sum of Squares for various methods		
1.	OLSQ using Linpack Chosleky -- solving from R	16.13858295915815
2.	OLSQ using LINPACK QR	16.13858295915821
3.	OLSQ using LINPACK DGECO	16.13858295915803
4.	OLSQ using LAPACK DGETRE-DGECON-DGETRI	16.13858295915806
5.	OLSQ using LAPACK DGESVX	16.13858295935751
6.	OLSQ using LAPACK DGESVX with equilibration	16.13858295963500
7.	OLSQ using LAPACK DPOTRF-DPOCON-DPTTRI	16.13858295915812
8.	OLSQ using LINPACK DPOCO-DPODI	16.13858295915811
9.	OLSQ using LINPACK DSICO-DSIDI	16.13858295915814
10.	OLSQ using SVD Linpack	16.13858295915808
11.	OLSQ using SVD Lapack	16.13858295915810

Model estimated was gasout=f(gasout{1 to 6}, gasin{1 to 6}). Data from Box-Jenkins [3]. Data studied in Stokes [32].
Experiment 1 solves for β using Cholesky R directly.
Experiments 3–9 form $(X'X)^{-1}$.

vary precision, method of calculation and degree of Fortran optimization for real*4 data. The base method was the QR for real*8 data which gives a LRE $= 13.54$ for β. When accuracy was enabled the LRE for the SE and $e'e$ increased slightly from 12.39 to 12.51 and 12.09 to 12.21 respectively in experiments 1 and 2. The LINPACK SVD produced a LRE of 13.92, 13.92 and 13.53 for the coefficient, the SE and $e'e$, respectively, while for LAPACK these were 13.48, 12.74 and 12.93, respectively, in Experiments 3 and 4. Here, using accuracy as a criteria, LINPACK edged LAPACK. Since in the Filippelli data set the reverse was found, there appears to be no "best" SVD routine for all cases. In addition to accuracy, there are other aspects of the selection process that include relative speed of execution (tested in Table 3 and found to be a function of the size of the problem and computer chip) and memory requirements that are not tested here since they are published. [26]

Experiments 5–8 show forced LINPACK LU and Cholesky models for real*8 data. In Experiments 7–8, added accuracy in the accumulators was enabled. Slight accuracy gains were observed, especially in the RSS calculation where the LRE jumped from 12.77 & 12.73 to 13.23 and 13.39, respectively. What is interesting is that in this case, even though the condition of $(X'X)$ was large, the LU and Cholesky approaches were able to get reasonable answers. The LINPACK condition check appears to be conservative since in the usual case the software would not attempt the solution of this problem.

0.0 and a t of 0.0 for the β_2 term. The "certified" coefficients and standard errors are:
β_0 =0.673565789473684E-03 0.107938612033077E-03
β_1 =0.732059160401003E-06 0.157817399981659E-09
β_2 =-0.316081871345029E-14 0.486652849992036E-16
which produce a t for β_2 of -64.95, not zero.

[26] For LAPACK the memory was set to the suggested amount from the first call to the routine. Experimentation with alternative LAPACK memory, possible with the B34S system implementation of LAPACK, was not attempted for his paper.

Table 8
LRE for various estimates of coef, SE and RSS of Pontius data

#	Method	Coef	SE	RSS
Real*8 Data				
1.	QR	13.54	12.39	12.09
2.	QR_AC	13.52	12.51	12.21
3.	SVD-LINPACK	13.92	13.92	13.53
4.	SVD_LAPACK	13.48	12.74	12.93
5.	LU-Forced	12.61	13.02	12.77
6.	Chol-Forced	12.11	13.00	12.73
7.	LU-Forced_AC	12.77	13.61	13.23
8.	Chol-Forced_AC	12.17	13.63	13.39
Real*4 Data Optimization = 1				
9.	QR	5.36	6.01	5.65
10.	QR_AC	5.36	4.37	4.06
11.	LU-Forced	3.93	5.27	5.36
12.	Chol-Forced	3.97	3.36	3.06
13.	LU-Forced_AC	3.95	5.30	4.78
14.	Chol-Forced_AC	4.01	3.32	3.02
Real*4 Data Optimization = 0				
9.	QR	5.36	4.21	3.91
10.	QR_AC	5.36	4.37	4.06
11.	LU_Forced	4.31	4.80	4.45
12.	Chol_Forced	4.48	4.51	4.26
13.	LU_Forced_AC	3.95	5.30	4.78
14.	Chol_Forced_AC	4.16	3.79	3.48

All data were initially read in real*8. For real*4 results data were then converted to real*4. Forced means that the LINPACK condition check has been bypassed for testing purposes. All reported LRE values are for the means. All real*4 tests have been done with LINPACK routines. Real*4 accumulators have not been enabled in cases where _AC is not added to the method name.

Experiments 9–14 concern real*4 data.[27] Again, the QR was found to be most accurate, with scores of 5.36, 6.01 and 5.65 for the coefficients, SE's and RSS, respectively. These runs were made with code compiled by Lahey Fortran version 7.10 running opt = 1. When accuracy enhancement was enabled, the LRE for the SE fell from 6.01 to 4.37. This difference was traced to the fact that the BLAS routine SDOT is optimized to hold data in registers while the higher accuracy routine SDSDOT did not optimize to the same extent. This is shown when the same calculation was done with opt=0. the QR SE accuracy was 4.21 and 4.37 for non-accuracy and accuracy-enabled code respectively. Higher accuracy was observed for opt=1 LU-forced of 5.27 vs 4.80 for opt=0 calculations. Why the forced Cholesky experiment seems to run more accurately at opt=0 than opt=1 (see Experiment 12) is not clear.

[27] Data was first read in real*8. Then the B34S routine RND( ) first checked for maximum and minimum allowable real*4 size, using the Fortran functions HUGH( ) and TINY( ). Next, the real*8 data was written to a buffer, using g25.16, and re-read into real*4, using the format g25.16. This approach gives a close approximation to having read the data directly into real*4. Use of the Fortran function sngl( ) can be dangerous in that, among other things, range checking is not performed.

What seems to be the case is that the level of optimization and its resulting changes in registers seems to make a detectable difference only with real*4 precision data. A strong case can be made not to use this precision for this problem. When real*8 calculations are used, these knife edge type differences are not observed.

The Eberhardt data consist of 11 observations of a one input model $y = \beta_1 x$. The level of difficulty is rated as average. Results are shown in Table 9. Here the Cholesky, the LINPACK SVD and the LAPACK SVD all produce 100% identical LRE values of 14.72, 15.00 and 14.91 respectively. For the QR the Coefficient LRE was 14.72 while the SE and residual LRE's were marginally less at 14.40 and 14.05. Here again the methods being considered run very close together.

The above results suggest that in certain problems that have a high degree of multicollinearity, the results are sensitive to the level of precision of the calculation as well as the method of the calculation. A challenging example was the Filippelli polynomial data set which was discussed earlier. However, the discussion was not complete because the real*16 QR results were only compared to the 15-digit "official" benchmark, and not a benchmark with more digits. Since real*16 will give more than 15 digits of accuracy, an important final task for the next section is to extend the Filippelli benchmark, using variable precision arithmetic to benchmark the accuracy of the real*16 results obtained.

4. Variable precision results

The variable precision library developed by Smith [30] was implemented in the B34S to extend the Filippelli benchmark and thus fully test the true accuracy of the reported real*16 results. The LINPACK LU inversion routines DGECO, DGEFA and DGEDI were rewritten to allow variable precision calculations. What was formerly a real*8 variable became a 328 element real*8 vector. Simple statements, such as `A=A+B*C`, had to be individually coded, using a customized pointer routine, `IVPAADD( )` that would address the correct element to pass to a lower level routine to make the calculation. A simple example shows how this is done:

```
c
c     if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
c
      if(vpa_logic(kindr,
     * z(ivpaadd(kindr,k,1,k,1)),'ne',     vpa_work(i_zero)) )then
      call vpa_mul(kindr,vpa_work(i_mone), z(ivpaadd(kindr,k,1,k,1)),
     *                    vpa_work(iwork(4)))
      call vpa_func_2(kindr,'sign', vpa_work(i_ek),
     *                              vpa_work(iwork(4)),
     *                              vpa_work(iwork(5)) )
      call vpa_equal(kindr,vpa_work(iwork(5)), vpa_work(i_ek))
      endif
```

Table 9
LRE for QR, cholesky, SVD LINPACK and LAPACK for Eberhardt data

Method	COEF	SE	RSS
QR	14.72	14.40	14.05
Chol	14.72	15.00	14.91
SVD-LINPACK	14.72	15.00	14.91
SVD-LAPACK	14.72	15.00	14.91

All data read in real*8.

vpa_work( ) is a 328 by 20 work array. The line z(ivpadd(kindr,k,1,k,1) addresses the k^{th} element of Z, which is 328 by k, and compares it to a constant = 0.0 saved in vpa_work(i_zero). If these two variables are not equal then the three calls are executed to solve ek = dsign(ek,-z(k)). The first call forms –z(k) and places it in VPA_work(iwork(4)). The variable vpa_work(i_mone) contains –1.0. Next, the SIGN function is called and the result placed in VPA_work(iwork(5)). Finally a copy is performed. This simple example shows what is involved to "convert" a real*8 program to do VPA math.

The results can be spectacular.[28]

Table 10 shows the Filippelli Data set benchmark, an extended printout of the QR real*16 results and the expanded Filippelli benchmark calculated with VPA data to 40 digits. A ruler listed at the top table is designed to assist the reader in determining at which digit there is a difference. Consider coefficient # 1. The VPA beta agrees with the real*16 QR beta up to the 28th digit, which is *far* beyond the 15^{th} digit, which was all that was listed for the "benchmark" which is shown again in Table 10. The VPA experiment documents that the real*16 calculation is in fact *substantially* more accurate than the best real*8 QR, which produced, on average 7 digits, as reported in Table 5. Recall that the "converted" real*16 results (data converted from real*8 to real*16), reported in Table 4 Experiment 5, had only a marginally better LRE of 7.924 than the real*8 QR results that found the LRE was 7.31. Although in Tables 4 and 6 it was reported that the "true" real*16 QR results (data loaded directly into real*16), had a LRE value was 14.79, once we had the VPA benchmark for 40 digits, it was apparent that the LRE was substantially larger. Even calculations of the 10^{th} and 11^{th} coefficients, when compared with the VPA data, produced 27 digits of accuracy. It should be remembered that these impressive results for real*16 are due to *both* the accuracy of the calculation and the fact that the data was directly read into real*16, not converted from real*8 to real*16. As we have shown, the data base

[28] The job vpainv, in paper_86.mac which is distributed which B34S, illustrates the gains in accuracy for alternative precision settings. Assuming a matrix X, X*inv(X) produces off diagonal elements in the order of |.1e-1728|, which is far superior to what can be obtained with real*4, real*8 or real*16 results which are also shown in the test problem. The B34S VPA implementation allows these high-accuracy calculations to be mixed with lower precision commands, using real*4, real*8 and real*16, since data can be moved from one precision to another. This allows experimentation concerning how sensitive the results are to accuracy settings.

Table 10
VPA alternative estimates of Filippelli data set

		10 20 30 40 50 12345678901234567890123456789012345678901234567890
VPA BETA	1	−.27721795919334239280284475566495960444434M+4
Real*16 QR	beta	−0.27721795919334239280284475535721085000000E+04
Answer for	coef	−0.2772179591933420E+04
VPA SE	1	.55977986547494987457477255080216514897 27M+3
Real*16 QR	SE	0.55977986547494987457477254797527487000000E+03
Answer for	SE	0.559779865474950 0E+03
VPA BETA	2	−.23163710816089307588219679165010449361 38M+4
Real*16 QR	beta	−0.23163710816089307588219679140978202000000E+04
Answer for	coef	−0.2316371081608930E+04
VPA SE	2	.46647757212779645269310983204844711248 38M+3
Real*16 QR	SE	0.46647757212779645269310982974610051000000E+03
Answer for	SE	0.4664775721277960E+03
VPA BETA	3	−.11279739409837156985716700152662497314 14M+4
Real*16 QR	beta	−0.11279739409837156985716700141998261000000E+04
Answer for	coef	−0.1127973940983720E+04
VPA SE	3	.22720427447775131062939817635102447383 52M+3
Real*16 QR	SE	0.22720427447775131062939817526228267000000E+03
Answer for	SE	0.2272042744777510E+03
VPA BETA	4	−.35447823370334877161073848525952818752 94M+3
Real*16 QR	beta	−0.35447823370334877161073848496469669000000E+03
Answer for	coef	−0.3544782337033490E+03
VPA SE	4	.71647866087592737261665721181584437353 26M+2
Real*16 QR	SE	0.71647866087592737261665720850717801000000E+02
Answer for	SE	0.7164786608759270E+02
VPA BETA	5	−.75124201739375713890522075574811872228 74M+2
Real*16 QR	beta	−0.75124201739375713890522075522683654000000E+02
Answer for	coef	−0.7512420173937570E+02
VPA SE	5	.15289717874740006503075679046071407820 62M+2
Real*16 QR	SE	0.15289717874740006503075678978592207000000E+02
Answer for	SE	0.1528971787474000E+02
VPA BETA	6	−.10875318035534251085281081182900835317 22M+2
Real*16 QR	beta	−0.10875318035534251085281081177144926000000E+02
Answer for	coef	−0.1087531803553430E+02
VPA SE	6	.22369115981603327555186234133238507450 16M+1
Real*16 QR	SE	0.22369115981603327555186234039770805000000E+01
Answer for	SE	0.2236911598160330E+01
VPA BETA	7	−.10622149858894676645966112205915973639 44M+1
Real*16 QR	beta	−0.10622149858894676645966112202355966000000E+01
Answer for	coef	−0.1062214985889470E+01
VPA SE	7	.22162432193422740206612983466088979396 87M+0
Real*16 QR	SE	0.22162432193422740206612983379340330000000E+00
Answer for	SE	0.2216243219342270E+00
VPA BETA	8	−.67019115459340837592673412288488449769 73M-1
Real*16 QR	beta	−0.67019115459340837592673412281191362000000E-01
Answer for	coef	−0.6701911545934080E-01

Table 10, continued

VPA SE	8	.142363763154723948918233091995327885249 8M-1
Real*16 QR	SE	0.142363763154723948918233091479593620000 0E-01
Answer for	SE	0.142363763154724 0E-01
VPA BETA	9	−.246781078275478650840854452418918855583 9M-2
Real*16 QR	beta	−0.246781078275478650840854452456467050000 0E-02
Answer for	coef	−0.24678107827547 90E-02
VPA SE	9	.535617408889820936258651932955578380227 9M-3
Real*16 QR	SE	0.535617408889820936258651931184658390000 0E-03
Answer for	SE	0.53561740888982 10E-03
VPA BETA	10	−.402962525080403671297131548527642644582 1M-4
Real*16 QR	beta	−0.402962525080403671297131548709169580000 0E-04
Answer for	coef	−0.40296252508040 40E-04
VPA SE	10	.896632837373868222100415272541027204780 8M-5
Real*16 QR	SE	0.896632837373868222100415269879510220000 0E-05
Answer for	SE	0.89663283737386 80E-05
VPA BETA	11	−.146748961422979588228784851530728712754 6M+4
Real*16 QR	beta	−0.146748961422979588228784851359607080000 0E+04
Answer for	coef	−0.14674896142298 00E+04
VPA SE	11	.298084530995536985200552343775516695431 3M+3
Real*16 QR	SE	0.298084530995536985200552342244390360000 0E+03
Answer for	SE	0.29808453099553 70E+03

precision makes a real difference in addition to the precision of the calculation. The important implication is that the inherent precision of the calculation method will be *no help* and in fact may give misleadingly "accurate" results unless the data is read with sufficient precision.[29]

Some of the key lesions of this paper are listed in Table 11. The main finding is the accuracy tradeoff between the precision of the data and the calculation method used. In all cases, it is important to check for rank problems before proceeding with a calculation. The less the precision of the data the more appropriate it is to consider higher accuracy solution methods such as the QR and the SVD approach. [30]

[29] In order to 100% isolate the VPA results from data reading issues, the loading of data into the VPA array proceeded as follows. The real*16 data was printed to a character*1 array using e50.32. Next, the VPA string input routine was used to convert this character*1 array into a VPA variable. This way both real*16 and the VPA results were using the same data. Experiments were also conducted by reading the data in character form directly into the VPA routines. For this problem both methods of data input into VPA made no difference since there were relative few digits. In results not reported but available in `paper_86.mac`, the Filippelli problem was "extended" by adding $x^{11}, \cdots, x^{20}$ to the right hand side to make the problem more difficult (stiff). Both the VPA and the native real*16 experiments were run and both successfully solved the problem, suggesting "reserve" capability to handle a stiff problem.

[30] While the main trust of the paper has been to show the effect of various factors on the number of "correct" digits of a calculation, in applied econometric work an important consideration is how many digits to report. If the government data is known only to k digits, many researchers argue that only k digits of accuracy should be reported. In many situations, this is appropriate although such a practice makes it difficult to access the underlying accuracy of the calculation routines used in the software system. Clearly if variables such as $\hat{y}$ or $\hat{e}$ are to be calculated, all estimated digits should be used to insure $\sum e = 0$ etc.

Table 11
Lessons to be learned from this paper

1. The QR method of solving an OLS regression model can provided more digits of accuracy and in fact may be the only way to successfully solve a "stiff" or multicollinear model.
2. The precision in which data are initially loaded into memory (for example, single precision) impacts accuracy, even in cases when it is later moved to a higher precision (for example double precision) for the calculation. This suggests that data should be read into the precision in which the calculation is made to avoid numeric representation accuracy issues that occur when the precision of the data is increased.
3. In many cases, accuracy gains can be made by boosting the precision of accumulators such as the BLAS routines for sum, absolute sum and dot product. Such routines should be used throughout software systems and will increase the accuracy of the variance and other calculations. It is desirable to be able to switch on and off such accuracy improvements to test the sensitivity of the given problem to these changes.
4. Data base design should take into account the needs of the users who may want to read data into higher-than-usual precision. For data that is not transformed in a data bank, the user should be able to get all reported digits of precision without rounding (due to numeric representation) loss.
5. The new 64-bit computers will make higher-precision calculations more viable and may prove useful for the estimation of problems requiring high precision for their successful solution. Real*16 and complex*32 will not have to be emulated in software by the compilers. These technological changes on the hardware side suggest that software designers may want to offer greater than double precision math in future releases of their products.
6. The lower the precision of the data, the more imperative it is to check for rank problems, use high-quality numeric routines (LAPACK/LINPACK etc.) and utilize inherently higher accuracy solution methods, such as the QR. For many problems, however, if data are read with sufficient accuracy, this may not be needed.
7. If data are not initially read with sufficient precision, high-accuracy methods of calculation, such as the QR, can provide misleadingly "accurate" results that are in fact tainted by numeric representation issues inherent in the initial data read. This initial data "corruption" cannot be "cured" by any subsequent increase in data precision. The more "stiff" the problem, the more this becomes an important consideration.

5. Conclusion

A number of import conclusions emerge from the tests run in this paper. These have been summarized in Table 11. The first and foremost is that accuracy improvements can and should be made to production econometric software to insure that accuracy problems do not unexpectedly occur. The work of McCullough and Vinod [14] argued that the software developers should use an improved formula to calculate the variance. While technically correct, results reported in this paper suggest that if accuracy improvements are made to a number of key BLAS routines, not only will the accuracy of the variance calculation be improved, but, more importantly, depending on how widespread BLAS has been implemented in the software, there will be many other important accuracy improvements. In addition, the user has the ability to switch back and forth to see the effect of accuracy enhancements on the results of specific problems.[31]

[31] By having the accuracy improvements able to be switched on and off, it is possible to replicate stock LINPACK and LAPACK results.

Renfro [24,25] has argued for data base standards. An important decision in implementing a data base is the precision of numbers saved. It has been argued that since we may only know numbers to a small number of digits, then single precision storage is sufficient. The problem with this view is that while we may know only a relatively few digits, if too small a precision is used, these digits get saved in a manner that precludes their use later at higher precision due to accuracy of saving these few digits. If the digits were saved in character form, then saving *only* the number of digits that are known would be technically correct. This "solution" would not work if the data had been transformed to a log for example, since more digits would be needed. Results reported in this paper illustrate that for difficult problems it makes a difference whether real*16 calculations are being made with data read into real*8 and then converted to real*16 versus data that are read directly into real*16. This finding suggests that for data saved in real*4, and analyzed in real*8, the problems may become substantially more acute.

Press [21], McCullough and Vinod [14], Greene [7] and others have argued for the SVD or QR approach to ordinary least squares estimation since there are accuracy gains. Results presented here suggest that the QR is quite accurate as is the SVD for most problems. However, with a stiff OLS problem, such as the Filippelli data set, even with quality software, such as LINPACK and LAPACK, it can make a difference what SVD routine is being used. For less difficult data sets, such as Pontius, Eberhardt and a VAR model on the gas furnace data, the selection of estimation method is less critical, provided that rank tests are made so that multicollinearity can be detected. McCullough and Vinod [15] and Stokes [33] suggest that more than one software system be used for nonlinear estimation. Results presented in this paper suggest that if the condition of $X'X$ will not allow estimation with the space-saving Cholesky approach, the QR or SVD approach should be used. In cases where the SVD method is selected, it is important to try different software systems. While moving the data to a higher precision before making the OLS calculation may give the illusion of assisting in the solution, it will most likely mask the effect of truncation of the data that occurred when it was initially read at the lower precision. A better choice would be to read directly into the higher precision.[32]

Acknowledgements

A number of suggestions received from B. D. McCullough are most appreciated and have helped strengthen the paper. Charles Renfro initially suggested the topic and made a large number of suggestions for important improvements that have been implemented. Some of the ideas of this paper were presented at the American

[32] If the data was coming from SAS or another system that only supports real*8, the user is trapped if a move to real*16 is required.

Economic Association Meetings 9 January 2005 in Philadelphia as part of comments on William Greene's work in this area. Diana A. Stokes provided editorial assistance. Any remaining errors are the responsibility of the author.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorenson, *LAPACK User's Guide*, Siam, Philadelphia, 1992.

[2] A.J. Barr, J.H. Goodnight, J.P. Sall and J.T. Helwig , *A User's Guide to SAS 76*, SAS Institute, Raleigh NC, 1976.

[3] G.E.P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*, rev. ed. San Francisco: Holden Day, 1976.

[4] T. Doan, *RATS User's Manual*, Evanston, Estima, 1992.

[5] J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart, *LINPACK User's Guide*, Siam, Philadelphia, 1979.

[6] J. Doornik and R.J. O'Brien, Numerically stable cointegration analysis, *Computational Statistics and Data Analysis* **41** (2002), 185–193.

[7] W.G. William, *Econometric Analysis*, Prentice Hall, New York, 2000 4^{th} edition and 2005 5^{th}, edition.

[8] IEEE, Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754–1985 NY Institute of Electrical and Electronics Engineers, *1985 reprinted in SIGPLAN Notices* **22** (1987), 9–25.

[9] K. Judd, *Numerical Methods in Economics*, MIT Press, Cambridge, MA, 1998.

[10] C. Lawson, R. Hanson, D. Kincard and F. Frogh, Basic linear algebra subprograms for fortran usage, *ACM Transactuions Math Software* **5**(3) (1979), 308–371.

[11] J. Longley, An appraisal of least squares programs for the electronic computer from the point of view of the user, *Journal of the American Statistical Association* **62**(319) (1967), 819–841.

[12] MATLAB: The Language of Technical Computing, Mathworks, Natick, Mass 2000.

[13] B.D. McCullough and C.G. Renfro, Some numerical aspects of nonlinear estimation, *Journal of Economic and Social Measurement* **26** (2000), 63–77.

[14] B.D. McCullough and H.D. Vinod, The numerical reliability of econometric software, *Journal of Economic Literature* **37** (June, 1999), 633–665.

[15] B.D. McCullough and H.D. Vinod, Verifying the solution from a nonlinear solver: A case study, *American Economic Review* **93**(3) (June, 2003), 873–892.

[16] B.D. McCullough, Econometric software reliability: EViews, LIMDEP, SHAZAM and TSP, *Journal of Applied Econometrics* **14** (1999), 191–202.

[17] B.D. McCullough, Assessing the reliability of statistical software: Part II, *The American Statistician* **53**(2) (May, 1999), 149–159.

[18] B.D. McCullough, Experience with the StRD: Application and Interpretation, *Computing Science and Statistics* **31** (2000), 16–21.

[19] Cleve, Moler, *Numerical Computing with Matlab*, Siam, Philadelphia, 2004.

[20] Visual Numerics, *IMSL Stat/Library and Math/Library*, IMSL, Houston, Texas, 1987.

[21] Press, H. William, B. Flannery, S. Teukolsky and V. William, *Numerical Recipes: The Art of Scientific Computing* (*Fortran Edition*), Cambridge University Press, Cambridge, New York, 1989.

[22] R.E. Quandt, Computational Problems and Methods, *Handbook of Econometrics*, Z. Griliches and M.D. Intrilligator, eds, North Holland, Amsterdam, 1983, pp. 699–764.

[23] C. Renfro, Econometric Software: The First Fifty Years as Perspective, *Journal of Economic and Social Measurement* **29**(1–3) (2004), 9–208.

[24] C. Renfro, Normative considerations in the development of a software package for econometric estimation, *Journal of Economic and Social Measurement* **23** (1997), 277–330.

[25] C. Renfro, Economic data base systems: Further reflections on the state of the art, *Journal of Economic and Social Measurement* **23** (1997), 43–85.

[26] C. Renfro, Economic data base systems: Some reflections on the state of the art, *Review of Public Data Use* **8** (1980), 121–139.
[27] J. Rogers, J. Filliben, L. Gill, W. Guthrie, E. Lagergren and M. Vangel, *StRD: Statistical Reference Data Sets for Assessing the Numerical Accuracy of Statistical Software*, NIST TN#1396, National Institute of Standards and Technology, 1998.
[28] Simon and Lesage, Assessing the accuracy of ANOVA calculations in statistical software, *Computational Statistics and Data Analysis* **8** (1989), 325–332.
[29] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema and C.B. Moler, *Matrix Eigensystem Routines – EISPACK Guide*, (2nd ed.), Springer-Verlag, Berlin, 1976.
[30] D.M. Smith, Algorithm 693, *ACM Transactions on Mathematical Software* **17**(2) (June, 1991), 273–283.
[31] H.H. Stokes, The evolution of economic software design: A developer's view, *Journal of Economic and Social Measurement* **29**(1–3) (2004), 205–260.
[32] H.H. Stokes, *Specifying and Diagnostically Testing Econometric Models*, (2nd ed.), Quorum Press, Westport, Conn, 1997.
[33] H.H. Stokes, On the advantage of using two or more econometric software systems to solve the same problem, *Journal of Economic and Social Measurement* **29**(1–3) (2004), 307–320.
[34] G. Strang, *Linear Algebra and Its Applications*, Academic Press, New York, 1976.