

Some numerical aspects of nonlinear estimation¹

B.D. McCullough

Department of Decision Sciences, Drexel University, Philadelphia, PA 19104, USA
E-mail: bmccullo@fcc.gov

Charles G. Renfro

601 W. 113th Street, #12G, New York, NY 10025, USA
E-mail: cgrenfro@modler.com

Given a nonlinear estimation problem, it is well-known that one algorithm might produce a solution where another algorithm might fail. What is less well-known is that when both algorithms do produce solutions, one of the solutions might be more accurate than the other. In particular, one might give several accurate digits, while the other gives no accurate digits. Possible sources of the phenomenon are presented, including step-length, stopping rule, convergence criterion and method of derivative calculation. Additionally, practical advice for using nonlinear solvers is offered.

1. Introduction

As a rule, econometric techniques are introduced and justified on theoretical bases. For instance, we know that under certain conditions, Ordinary Least Squares provides a Best Linear Unbiased Estimator. Under other conditions, other techniques can be justified (usually asymptotically). However, historically, it is unusual for authors to be concerned about the computational aspects necessary to implement such techniques, even though these computational aspects may be crucial to obtaining reliable results [26]. As an elementary example, the solution to the linear least squares problem frequently is presented as $\hat{\beta} = (X'X)^{-1}X'y$, but it is rarely noted that $\hat{\beta}$ should not be computed by inverting $(X'X)$ and post-multiplying by $X'y$. Yet, this important computational fact is rarely mentioned in any econometrics text. Consequently many economists, when using matrix-oriented programming languages, mistakenly compute the least squares coefficients in just this fashion.

Nonetheless, it is not true that the existence of alternative algorithms has been ignored. It is common for econometrics texts (e.g., [9,17]) to provide good theoretical discussions of the various nonlinear estimation methods, such as Gauss-Newton or Newton-Raphson. The problem is that these texts rarely treat the computational details of these algorithms. It is also common for these texts to note that for any

¹Discussions with J.G. MacKinnon inspired us to write this paper. For specific comments, thanks are due to G. Fiorentini, M. Nerlove, P. Coomes, C. Rose, M. Smith, S. Schneider, and V. Wiggins. The views expressed herein are those of the authors and do not necessarily reflect those of their employers.

given problem, one algorithm might fail to produce a solution while another might produce a solution. However, it is decidedly uncommon for a text to mention that when both algorithms produce a solution to the same problem, one of the “solutions” may be better than the other.

The need to consider the properties of alternative algorithms and the ways in which particular ones fail became apparent during the development of our recent article on GARCH procedures [25]. That paper assumes some knowledge of computational techniques which, as just indicated, has been ignored in the literature. The present paper attempts to remedy this gap. In particular, it aims to explain why nonlinear procedures can produce inaccurate “solutions,” a prevalent phenomenon as documented in McCullough [22,23], McCullough and Wilson [27], Vinod [36], and Altman and McDonald [2]. Notably, we discuss numerical sources of inaccuracy in nonlinear estimation, and the various ways a procedure can fail. Of necessity our treatment is selective. But within this restriction, our hope is that the novice will gain some appreciation for the intricacies of nonlinear estimation.

The focus of this paper will be on the maximization of a likelihood function (though much of it obviously applies to the special case of nonlinear least squares). Moreover, inasmuch as the present paper was originally suggested by the need for a supporting paper to McCullough and Renfro [25], it can be read in conjunction with that paper, as applying to the particular case considered there. However, it can also be read separately.

2. Fundamental concepts

Generally, considering a (conditional) likelihood function, $L(\cdot)$, its maximization can be algorithmically implemented in many ways. We necessarily draw a distinction between the concept of an algorithm and its implementation. The algorithm is a set of rules, and this is what usually is presented in econometrics texts. How these rules are coded in a programming language is the implementation, and it is this with which we are concerned. It will soon become apparent that two different implementations of the same algorithm do not necessarily yield the same answer. As already noted, standard textbook discussions of nonlinear methods can be found in Davidson and MacKinnon [9] and Greene [17]. Attention to the numerical details thereof can be found in Bard [3] and Quandt [32]. The classic reference on statistical computing is Kennedy and Gentle [20], and Thisted also is quite good, with more advanced topics treated by Lange. The classic references on optimization (especially unconstrained) are Gill, Murray and Wright [16], which is quite accessible, and Dennis and Schnabel [10], which is only slightly more technical. Useful expositions on optimization can also be found in texts on nonlinear programming, e.g., Luenberger [21] and Bazarra et al. [5]. Bard (pp. 83–88) presents a good presentation of the basic concepts of unconstrained optimization that underlie the discussion here, and Appendix A5 of Hendry, at 30 pages in length, is short and instructive. For the reader interested in the numerical

aspects of computation, we highly recommend Judd. Though not written explicitly for econometricians, it provides a full treatment of the fundamentals underlying any econometric discussion of numerical methods. Moreover, it covers many advanced methods of direct relevance to econometrics, e.g., quadrature (numerical integration of density functions).

Maximization of a nonlinear (conditional) likelihood function, $L(\theta)$, usually involves iteration. Let $\theta = [\theta_1 \theta_2 \dots \theta_p]'$ be the argument of the likelihood function and let θ^0 be a vector of starting values.¹ The iterative process

$$\hat{\theta}^{k+1} = \hat{\theta}^k - \lambda_k d_k \quad k = 0, 1, \dots \quad (1)$$

can be shown to be a member of a very broad class of algorithms, where d_k is the direction moved and the step length, λ_k , is the distance moved on the k -th iteration [32, p. 717]. The direction, d_k , generally varies with the algorithm, and needs to be considered in that context.

As a general rule, until terminated, the iteration process involves successively changing θ by an amount and a direction. The amount of change, λ_k , is also called the “step size”. In a multivariate context, the direction of change, d_k , is represented by a vector, possibly representing the gradient and/or Hessian of the objective function, which vector often is chosen so as to conform with a particular negative definite matrix. Various methods differ in the ways that λ_k and d_k are computed.

The termination of an iterative sequence is a critical issue that is often misconceived. The change from the k -th to the $(k + 1)$ -th iteration is evaluated at each pass using what is known as a “stopping rule”. This test, of course, determines whether the change is sufficiently small that further iterations are not necessary. If this test is successful and iteration is halted, then under ideal circumstances the program should have found a stationary point and upon further (successful) testing of the convergence criteria, should report that it has found a maximum. In McCullough and Renfro [25, § 4], the FCP program was correctly described as possessing both a stopping rule and a test for successful convergence. Frequently, discussions of nonlinear estimation conflate the stopping rule and convergence criterion, treating them as one and the same.

Note that the test for convergence is applied conditional upon the stopping rule test. If this secondary testing is unsuccessful, so that iterations have terminated but convergence has not not been established, then the program may report that it has “stalled” – stopped at a point that is not a maximum. As discussed further below, some packages do a poor job of conducting this examination, and consequently have

¹A discussion of starting values does not fall within the scope of this paper. However, it is difficult to underestimate the importance of good starting values (Press et al. [31, p. 341]). Also, to guard against the local/global maximum problem, it is important to employ a variety of starting values. Dufour, Gaudry and Liem [13] give an example where a Cochrane-Orcutt correction has multiple extrema for the parameter ρ . Newbold, Agiakloglou and Miller [30, § 4] present an ARIMA(2,1,2) model with multiple optima.

a marked tendency to report having found a maximum when, in fact, they have not. Stalling can also occur when the program reaches a point from which either an acceptable step or direction cannot be determined. In this case, the program may report, “unable to improve current iterate – iterations terminated,” or some similar message.

3. Step length

With respect to determining a good step-length, the requirements are simple: it must not be too large and it must not be too small (of course, the step must also be in a good direction). If too large, the procedure can easily miss a stationary point by stepping over it. If too small, the procedure might never get to a stationary point: imagine trying to walk up a hill by taking successively smaller steps; your path might well converge to a point that is only part-way up the hill. Now, assume a good direction, \tilde{d} , has been found for the current iteration (\tilde{d} may well change from one step to the next). Then one common method of determining the step-length is to find λ^* that minimizes

$$L(\theta_k - \lambda^* \tilde{d}) \quad (2)$$

and then set $\lambda_k \equiv \lambda^*$. There are many other ways to solve this problem, many of them based on the concept of a “line search” – admissible values for λ are restricted to some interval, and this interval is searched for the value that minimizes (2). See Dennis for a discussion of step-length determination.

4. Stopping rules and convergence criteria

Let $g(\theta)$ be the gradient of the likelihood function. In principle, iteration should continue until $g(\theta) = 0$ for some $\theta = \hat{\theta}$, which is, of course, the requirement for a stationary point, because $g(\theta)$ is the first derivative. Then $L(\hat{\theta})$ can be investigated to determine whether $\hat{\theta}$ is a maximum. However, given the nature of finite precision computation, it is a far too stringent test to impose the operative requirement that $g(\theta) = 0$ exactly. Bear in mind that decimal numbers are represented in the computer by finite-precision binary numbers, which are essentially approximations. Consequently, it is a rare event that a computation yields the number zero exactly. Thus it is appropriate to evaluate the results of computations on the basis not that they evaluate to exactly zero, but rather approximately zero, i.e., $-\epsilon < g(\theta) < \epsilon$ for some small value, ϵ , called the “tolerance.” Clearly this condition admits not only stationary points, but nonstationary points for which the gradient is near but not equal to zero. Thus, even locating a stationary point is fraught with the potential for numerical error.

It can be shown [3, p. 86–88] that, in principle, once a stationary point is found, successive values of θ^k should be approximately the same (which is the definition of “stationary”). Therefore, the change from one iteration to the next should be “essentially” zero; that is, it should be smaller than epsilon upon testing using another stopping rule $|\theta^{k+1} - \theta^k| < \epsilon$.² Suppose, though, that the step-length rule happened to produce an exceptionally small λ_k so that θ^{k+1} was close to θ^k . Then yet another rule would be necessary to determine whether θ^{k+1} really was a stationary point, and still the convergence criterion must be applied to determine whether the stationary point is an extremum. This demonstrates why stopping rules and convergence criteria should not be conflated. In fact, there are many possible stopping rules. Some common ones are

1. $k > K$ where K is an upper bound on the number of iterations; “maximum iterations reached”
2. $|L(\theta^{k+1}) - L(\theta^k)| < \epsilon$; “function convergence”
3. $\max_i [|\theta_i^{k+1} - \theta_i^k|] < \epsilon \quad i = 1, 2, \dots, p$; “parameter convergence”
4. $\max_i [(\theta_i^{k+1} - \theta_i^k)^2] < \epsilon \quad i = 1, 2, \dots, p$; “squared parameter convergence”
5. $\|g(\theta)\| < \epsilon$; “gradient convergence”

The choice of stopping rule can affect the robustness of the procedure; consequently the argument can be made that the user should have the option of selecting alternative stopping criteria. See also Belsley [6], Dennis, Gay and Welsch [11], and Gay [15] for further discussion of stopping rules.

These stopping rules can be used singly or in combination, frequently the latter because any one does not necessarily imply any other. When convergence is achieved, set $\hat{\theta} = \hat{\theta}^{k+1}$; the procedure has converged in $k + 1$ iterations.³ Ideally, $\hat{\theta}$ corresponds to a stationary point of the objective function. It may or may not be a local extremum, and so $L(\hat{\theta})$ must be examined carefully to determine whether or not it is a maximum, in order to avoid reporting a false maximum. Examination of the Hessian can be very useful in this regard.

Many algorithms do not make use of second derivative information, and others employ only crude approximations to the Hessian. For such procedures, the second derivative cannot be examined to determine whether or not second-order criteria for a maximum have been satisfied. Thus, $|L(\theta^{k+1}) - L(\theta^k)| < \epsilon$ might be the stopping rule and $|g(\theta)| < \epsilon$ might be the convergence criterion. It is not uncommon for a package to have a single criterion that serves as both stopping rule and convergence criterion, e.g., $|L(\theta^{k+1}) - L(\theta^k)| < \epsilon$. Such a criterion may only locate a flat region of the likelihood, will not even check that $|g(\theta)|$ is close to zero, and certainly does

²This is yet another reason that a step length should not be “small.” If the step length could be arbitrarily small, then θ^{k+1} would necessarily be close to θ^k , and the rule could be satisfied far from the maximum.

³The information in d_k usually provides the standard errors for the coefficients. Note that at this point, the subscript on d is k , so if d is used to compute standard errors, one more iteration must be made to update d to $(k + 1)$.

not ensure that the Hessian is negative definite at the alleged maximum. Because Newton-Raphson requires explicit calculation of the Hessian, we prefer packages that offer Newton-Raphson as an algorithm. For procedures that do not make explicit use of the Hessian, it is advisable to “polish” such a solution by submitting it to a few iterations of Newton-Raphson, if only so that second-order conditions can be verified (an additional benefit of “polishing” – and an example thereof – will be given in the penultimate section).

The ability to display the final gradient is a particularly useful feature. Suppose that the convergence criterion and stopping criterion are conflated in the form: $|L(\theta^{k+1}) - L(\theta^k)| < \epsilon$, and the procedure reports that it has found a maximum according to this criterion. If the gradient is immediately available, the user can check to see whether each element is approximately equal to zero. If not, then a false maximum has been reported.

Some of these criteria may be dependent upon the scaling of the variables. For example, whether some $|\theta_i^k - \theta_i^{k+1}| < \epsilon$ may depend on whether the corresponding variable, x_i , is measured in dollars or billions of dollars. Yet, since the variables themselves already should be scaled to similar magnitudes, ensuring that convergence criteria are expressed in relative terms often makes sense. Such a criterion which merits mention is the “relative offset convergence criterion” discussed in Bates and Watts [4, ch. 2]. As the formula is based on a QR decomposition, we do not present it here. We also mention that $g'H^{-1}g < \epsilon$ is another reliable criterion that makes use of both gradient and Hessian information and, by design, is scale invariant.

5. Derivatives

To see the difference between numerical and analytic derivatives, consider the forward-difference approximation to the first derivative to a univariate function, $f'_F(x) = (f(x+h) - f(x))/h + R(h)$ where $R(h)$ is the remainder. Note that $R(h)$ might not be negligible if the function is highly nonlinear. Exact analytic derivatives are not contaminated by this approximation error, $R(h)$. This approximation error can be decreased at the expense of computational time by using a more sophisticated form of numerical derivative (also referred to as a “finite-difference approximation”), such as the central difference: $f'_C(x) = (f(x+h) - f(x-h))/2h + R(h)$. Of two otherwise identical programs, one that uses forward differences and another that uses central differences, the latter can be expected to be more accurate. As a simple example, consider $f(x) = x^3$, for which the analytic derivative is $f'(x) = 3x^2$. It is elementary to show that $f'_F = 3x^2 + 3xh + h^2$ and $f'_C = 3x^2 + h^2$. Both forward and central differences are “two-point” approximations to the slope. Four- and eight-point approximations also can be used, which achieve concomitantly more accurate approximations. However, accurate computation of numerical derivatives becomes more problematic as the number of variables increases: it takes at least p function evaluations (evaluations of the objective function, in the present case,

$L(\theta)$) to obtain the finite difference approximation for a function with p variables. In some cases, an algorithm based on the most accurate numerical first derivatives can stall in a region where the norm of the gradient, $\|g(\theta)\|$, is small but non-zero, whereas a similar algorithm based on analytic derivatives might encounter no difficulty. Since convergence tests (as opposed to stopping rules) are based on derivatives, *cet. par.*, inaccurate computation of derivatives can lead to incorrect convergence results. Sometimes the procedure will fail to recognize a maximum it has encountered but, more commonly, will label as a maximum a point that is not even stationary. Obviously, to the extent that the direction, d_k , is based on derivative information, inaccurate derivatives can lead to a bad direction and an attendant failure to locate the maximum.

In the use of either forward or central differences, there is an optimal choice of h . If h is chosen too large, then truncation error dominates the result, while if h is chosen too small, then roundoff error and cancellation error dominate the result. See Press et al. [31, §5.7], for further elementary discussion. However, determining the optimal choice of h for a numerical first derivative usually entails knowledge of the second derivative, producing a Catch-22: the second derivative cannot be calculated before the first derivative. Therefore, if two otherwise identical programs both use forward differences, but differ in how h is calculated, there might be some difference in accuracy [10, §5.4]. Gill, Murray and Wright [16, p. 285] have noted, “[T]he user should be aware of the increased complexity and decreased reliability that result when exact first derivatives are not available to an algorithm.” Thus, if a package uses numerical derivatives as default but accepts user-supplied analytic derivatives, the user is well-advised to take the time and trouble to supply the analytic gradient.

For a slightly different perspective on the problem with numerical derivatives, consider the scalar function $f(x)$, and suppose that it is minimized by x_0 . For a point close to x_0 and for f quadratic in a region about x_0 , a Taylor series expansion yields $f(x) = f(x_0) + f'(x_0)(x - x_0)^2/2$. The point x will be numerically distinct from x_0 if $(x - x_0)^2$ is greater than machine precision.⁴ Consequently, if machine precision is on the order of 1E-16 (as is common on PCs), it is often not meaningful to set the tolerance below 1E-8 when numerical derivatives are involved, at least when beginning the search for a maximum. If an approximate maximum has been located, then the tolerance can be tightened up a bit. By contrast, analytic derivatives are not bound by this numerical restriction. Further discussion of numerical derivatives can be found in Gill, Murray, and Wright [16] and Dennis and Schnabel [10].

Analytic derivatives are not always preferred to numerical derivatives. A well-implemented algorithm with numerical derivatives can perform more reliably than a poorly-implemented algorithm with analytic derivatives. Also, for some problems analytic derivatives are too difficult to program (e.g., complicated GARCH-type models) or simply cannot be computed (e.g., consider the accommodation of missing values).

⁴Let $z = 1.0 + \epsilon_M$. The smallest (in magnitude) number ϵ_M such that a test of the condition $z = 1.0$ returns false is called “machine precision.” See any computing text for further discussion.

6. Algorithm choice

As Quandt [32] has noted, algorithms have a variety of characteristics, not all of which can necessarily be embodied in a specific implementation. Thus the choice of method of maximization necessarily involves tradeoffs, for there is no best algorithm and, in fact, the characteristics of any one algorithm may vary from problem to problem, being less amenable to one class of problems and more amenable to another. This is because different algorithms make different assumptions about the problem structure. For example, nonlinear least squares (NLS) problems have a special structure, and methods designed specifically for NLS are likely to be better at solving NLS problems than general unconstrained maximization routines. Nevertheless, a good implementation of a general unconstrained routine can produce better results on NLS problems than a poor implementation of a specialized NLS routine. Likewise, some algorithms are more sensitive to the control parameters (starting values, step-length, etc.) than others. For example, Newton-Raphson is more dependent on the choice of starting values than quasi-Newton methods.

Two important considerations are: (1) the robustness of an algorithm, that is, its ability to find a potential “solution” and (2) the cost of execution. With today’s powerful desktop computers, for many problems cost is essentially a matter of time, but in the past it was defined in terms of computer resource use, especially on mainframe computers. However, as computers become more powerful, they are given more difficult tasks, and so efficiency still remains a great concern.

The robustness of an algorithm can be a subtle issue, depending not only upon the problem at hand, stopping rules, and step length, as noted, but also on the amount of derivative information (gradient, or gradient and Hessian), and the types of derivatives (numerical or analytic). Fraley [14] gives a lucid discussion of these issues in the context of assessing algorithms for nonlinear least squares.

For the maximization of a function $F(\theta)$, a common conceptual starting point is the specification of a second-order Taylor approximation at a point θ_k :

$$F(\theta) = F(\theta_k) + (\theta - \theta_k)'g_k + \frac{1}{2}(\theta - \theta_k)'H_k(\theta - \theta_k) + R(\theta - \theta_k) \quad (3)$$

where g_k is the vector of first derivatives (the gradient) evaluated at the point θ_k and H_k is matrix of second derivatives (the Hessian) evaluated at the point θ_k , which reflects the concavity property of the function to be maximized.⁵ Then from (1)

⁵It is commonly asserted that if F has continuous n -th order derivatives then the behavior of F can be locally approximated by an n -th order Taylor series. This is incorrect. The standard counter-example is: $F(x) = \exp[-1/x^2] \forall x \neq 0, F(0) = 0$. This function is infinitely differentiable everywhere. The correct assumption is that F is *analytic*. See Aleksandrov et al. [1, p. 127], Courant [7, p. 549], or Kaplan [19, p. 357].

choosing $d_k = H_k^{-1}g(\theta_k)$ and setting $\lambda^k \equiv 1$ defines the Newton method, also called “Newton-Raphson”⁶

$$\theta^{k+1} = \theta^k - H_k^{-1}g_k \tag{4}$$

The Newton method critically depends for its properties on the starting value θ_0 , being “close” to the value that maximizes the likelihood. The reason is that in order for H_k^{-1} to be a part of a “good” descent direction, H_k^{-1} must be negative definite. Though H_k is negative definite when θ_k is close to θ , when θ_k is not close to θ , H_k might not be negative definite and the procedure breaks down.

A variety of schemes for solving this problem of negative definiteness center around replacing H_k in (4) with some Q_k , where Q_k is constructed so as to be negative definite whatever the value of θ_k . A decided advantage of this approach is that the convergence properties no longer depend so critically on the starting values, as compared to Newton-Raphson. One particular class of such schemes, the quasi-Newton (variable metric) method, constructs Q_k using F and g . In quasi-Newton routines, an approximation to the Hessian is built over several iterations with the hope that, by the time iterations converge, $Q_k \approx H_k$. On each iteration, Q_k is updated by $Q_{k+1} = Q_k + U_k$ where U_k is the updating matrix. One rule for constructing U_k leads to the DFP algorithm, while another leads to the BFGS algorithm. Since each iteration yields information only about one direction, Q_{k+1} can be expected to differ from Q_k by a low-rank matrix. When U_k has rank one, this is called a “rank one update.” Naturally, there exist methods based on rank two updates, such as BFGS and DFP. Because these methods build up curvature information slowly, if the procedure terminates in only a few iterations, Q_k is likely to be a poor approximation to H_k ; the coefficient estimates may be good, but the standard error estimates will not. Even if several iterations occur, standard errors based on Q_k will be inaccurate to the extent that $Q_k \neq H_k$. An additional difficulty with the quasi-Newton approach and other methods that use only gradient information is that they cannot detect and move away from a saddle-point as well as Newton-Raphson can.

One common approach to securing the best of both worlds is to use a “hybrid algorithm.” An algorithm that is not so sensitive to starting values, such as a quasi-Newton method, is used to move the iterate “close enough” to the solution, and the problem is handed off to a method with superior convergence properties, such as Newton-Raphson. When both algorithms are of the gradient variety, this is called a “mixed gradient algorithm” [8].

Generally, choice of algorithm and its implementation, starting values, stopping rule, and the method of computing derivatives all interact to determine whether the program can compute the optimum and, if so, how accurately. The reasons a procedure can fail can be simply enumerated.

⁶If $\lambda_k \neq 1$ in (10), then the algorithm often is referred to as a “modified Newton method.”

7. Ways a procedure can fail

The preceding discussion makes clear several of the reasons that, when two packages solve the same problem, the solutions might not be equally accurate. These same reasons can also shed light on the ways a package can fail to produce a solution [28].

There are a variety of ways a procedure can fail to obtain satisfactory parameter estimates, but these can be roughly categorized under four headings:

1. failure to converge (the iterations do not stop until the upper bound is hit);
2. failure to improve the current iterate (the procedure reaches a point from which it cannot find another step to take);
3. failure to recognize a maximum (the procedure stops at a maximum, but does not recognize that the point is a maximum); and
4. “false maximum” (the procedure stops at a point that is not a maximum but indicates that a solution has been found).

In the first three cases the user has some idea that something is amiss, and so can take appropriate action. Diagnostic output can be examined to see whether the iterates are converging appropriately.⁷ The output can also suggest the source of the problem. Perhaps all the parameters have converged save one, which is either diverging or asymptotically approaching some limit. This suggests that the lone parameter is either not identified in an econometric sense, or that the model needs to be reparameterized with this coefficient in mind. In the first three cases, the usual “fixes” can be applied: change the starting value, vary the options, even try another algorithm, rescale the data, or reparameterize the problem. The fourth case is a catastrophic failure, for the user has no idea that anything is wrong. We discuss each case in turn.

In failure to converge, the procedure hits the upper bound on the number of iterations, the user resets the upper bound, and again the procedure hits the bound. There are two primary reasons that a procedure may fail to converge. First, the iterates may diverge monotonically or in an oscillatory fashion. A strong hint that this is occurring comes when the user must increase the maximum number of iterations. It can be confirmed by examining the function values and other diagnostics. For this reason it is important that the trace (output at each iteration) is available to the user. We are suspicious of packages that do not allow the user to access the trace, as it suggests to us that the programmer does not really understand nonlinear estimation. Second, the iterates may cycle, with (nearly) the same sequence of iterates recurring. Again, the user has to reset the maximum number of iterations, and examination of function values and diagnostics reveals some sort of cyclical behavior. There is yet a third reason: the algorithm may be exceedingly slow for this particular problem,

⁷If the algorithm is quadratically convergent, then the final iterations should exhibit this behavior [16, §8.3].

making very little progress on each iteration, perhaps requiring several hundred iterations. This is especially true of secant methods.

That a step length and a step direction have been found does not mean that a step is taken. This is the problem underlying “failure to improve current iteration”. The step must be “acceptable” before it is taken. Generally, this means that for a candidate θ^{k+1} , it must be true that $L(\theta^{k+1}) > L(\theta^k)$. If such a step cannot be found, then the program will issue a “failure to improve the current iterate” message and terminate. One common reason for such a message is that the convergence tolerance has been set too small for the accuracy to which the function and gradient are computed. This can be especially true if the likelihood is flat in the region of the maximum. Alternatively, the function or one of its derivatives might be discontinuous near the point at which the algorithm stalls. This discontinuity might be real, or it might be a numerical artifact of finite precision (see [16, p. 327]). Yet another possibility is that the derivatives might be inaccurate, especially if they are user-supplied and incorrectly coded or calculated, although numerical derivatives can also be sufficiently inaccurate to cause this problem.

Formally, failure to recognize a maximum is a special case of failure to improve. If the program stops at a maximum but fails to recognize it as such, it is likely to produce the error message “failure to improve,” indicating the inability to determine a next step that increases the value of the given objective function, yet also the inability to determine the satisfaction of the second order conditions. A simple test that often works in practice is to increase the convergence tolerance. If the procedure converges to the same point but this time labels it a maximum, it probably is a maximum.

“False maximum” occurs when the procedure stops at a point that is not stationary, and this point somehow satisfies the program requirements for the point to be labelled a maximum. This problem often arises due to poor implementation of the termination criteria [16, p. 306]. One method of guarding against a false maximum is to compute the gradient evaluated at the maximum. Each component of the gradient should be numerically close to zero, if the data have been properly scaled so that all variables are of the same order. (If the data have not been so scaled, then each component will be close to zero, but only on the order of the covariate.⁸) Additionally, careful examination of the diagnostics might indicate that the point is not a maximum, but even then there is no “fix” which the user can apply using that same software package. For this reason a package should err on the side of caution: better to label a maximum as “no solution found” than to label a non-solution as a solution. As revealed by reliability assessments [22–24,27], some packages have a marked tendency to report false maxima, and users contemplating nonlinear estimation should avoid the use of such packages.

⁸A useful way to see this is to solve one of the NIST StRD nonlinear problems and evaluate the gradient. Then rescale the data, solve again, and re-examine the gradient.

8. Some observations

Nonlinear estimation is straightforward only in the simplest cases. In more interesting cases, the user should expect to spend a not inconsiderable amount of time in specifying (and respecifying!) the functional form of the equation, determining good starting values, verifying that a reported “solution” really is a solution, and conducting sensitivity analyses to determine whether the results are robust (e.g., do t -tests on the coefficients change when the method of estimating the covariance matrix changes?).

Consider estimating a nonlinear least squares problem as a maximum likelihood problem. Two possible specifications are:

$$\log L = -\frac{T}{2} \log \sigma^2 - \frac{(\sum_{i=1}^n y_i - \hat{y}_i)^2}{2\sigma^2} \quad (5)$$

$$\log L = -T \log \sigma - \frac{1}{2} \left(\frac{\sum_{i=1}^n y_i - \hat{y}_i}{\sigma} \right)^2 \quad (6)$$

The procedure might fail to converge for (5) and converge for (6) for one problem, and conversely for another. (Of course, an easier approach would be to concentrate out the σ term and estimate the sum of squares directly; then the solver has to search over one less dimension.) Another example comes from actual experience: a problem involving user-supplied analytic derivatives consistently failed to converge, until the derivatives were algebraically simplified.

One particularly nettlesome problem failed to converge despite best efforts to obtain good starting values. Several algorithms were tried, and the solver stopped at different points for each algorithm. The point at which one algorithm stopped was used as a starting value for another algorithm. Eventually a solution was achieved using this method.

One researcher had a GARCH problem which failed to converge on a variety of packages. He finally wrote an Excel procedure to estimate GARCH models and obtained a “solution.” However, Excel has a marked tendency to report that it has found a solution when it has simply ground to a halt, instead of reporting that it is unable to find a solution [27]. Had this researcher taken the time to check whether the “solution” satisfied first-order conditions, he might have realized that his “results” were completely inaccurate.

As mentioned earlier, when a solution is obtained using a method other than Newton-Raphson, it is almost always a good idea to “polish” the solution by submitting it to a few iterations of Newton-Raphson [31, p. 356], so as to further refine the estimates. Even small differences can matter appreciably, as will be seen shortly. Simply decreasing the tolerance and iterating further with the current algorithm does not have the same effect, because Newton-Raphson typically has a higher convergence rate and makes use of more derivative information than other methods, and

Table 1
Two sets of ARCH estimates

coef.	sol2 – solution		sol4 – polished	
	estimate	gradient	estimate	gradient
θ_1	0.677367	0.018194	0.677416	1.3E-5
θ_4	0.305868	-0.0627049	0.304999	-6.9E-7
β_1	-5.07541	0.0341503	-5.07483	4.1E-6
β_2	1.02915	-0.001194	1.03084	3.4E-7
	logL = 243.53372		logL = 243.53376	

so can produce a better solution.⁹ This polishing can be critical when calculation of standard errors is required. As an example, Rose and Smith [34, § 11.2] estimate a four parameter ARCH model, with results presented in the “sol2” columns of Table 1 (this is Rose and Smith’s “sol2”). They then apply a few iterations of Newton-Raphson to this solution; results presented in column “sol4”. The coefficient estimates seem barely to change, only in the third or fourth decimal. The difference in the log likelihood occurs only in the eighth digit, which may lead the untutored to conclude that there is not much difference between the two solutions. This “conclusion” merits comment.

Recall that the magnitude of the coefficients can be affected by scaling, so these changes really should be interpreted not as in the third or fourth decimal, but in the third or fourth significant digit. Assuming that one of the solutions is more correct than the other, the least correct of the pair is accurate only to two or three digits. It is worthwhile to examine the gradient of each “solution”. Note that the gradient of sol2 is not numerically close to zero, so it is not really a solution of the problem, whereas the gradient of sol4 is numerically close to zero. Naturally, the Hessian evaluated at sol2 is different than the Hessian evaluated at sol4. In the present case the difference is slight, but it is not difficult to conjure examples where small differences in the gradient produce large differences in the Hessian. In such a case, the consequences for inference, when the standard error is based on the Hessian, could be serious.

9. Conclusions

To expect a reliable solution from a nonlinear procedure by simply feeding it data and equations is extremely naive, but all too common. The literature from which economists typically learn nonlinear estimation presupposes that one package is good as another and gives no reason to doubt that any package gives an accurate answer (except in the case of the multiple optima). The growing literature on software reliability calls into question both these notions.

⁹As noted in Section 4, Newton-Raphson also permits second order conditions for a maximum to be verified.

The present article adds to this literature. It describes some of the reasons why a nonlinear algorithm may produce an incorrect answer, both in those cases that software packages fail to implement these algorithms properly and when the problem to which the package is applied has characteristics that make its solution challenging. Historically, software packages for economic applications have tended to be developed by individuals for their specific needs, and only subsequently made available generally to other economists. In part because of the time it takes to subject a package to intensive testing (itself a process that involves some degree of expert knowledge) and in part because of a general unawareness among economists of the possible pitfalls, many packages have been made available without sufficient prior testing of algorithms under a variety of different conditions. Moreover, the lack of general appreciation of the circumstances under which an implemented algorithm may fail has also contributed both to the undertesting of software and the lack of user aids that can assist in the field evaluation of these packages. One of the purposes of the present article is to raise general consciousness of these circumstances.

One experienced practitioner of the art [29] describes his general approach in practical terms: “[W]hen possible, I use analytic gradients and Hessians in the maximization procedure and finish off my ML’s by computing the Fisher Information matrix [in case the maximization procedure uses some other form of estimator for the covariance matrix.] Indeed, I don’t like any of the hands-off methods for ML and where possible I use a combination of graphical techniques, higher-order grid search and steepest ascent to find the maximum. It’s sort of like baking bread – you have to ‘feel’ the dough to get it right.”

References

- [1] A. Aleksandrov, A. Kolmogorov and M. Laurent’ev, *Mathematics: Its Content, Methods and Meaning*, (vol. 1), New York: Dorset Press, 1990.
- [2] M. Altman and M. McDonald, *The Robustness of Statistical Abstractions: A Look ‘Under the Hood’ of Statistical Models and Software*, manuscript, Harvard-MIT Data Center, 1999.
- [3] Y. Bard, *Nonlinear Parameter Estimation*, New York: Academic Press, 1974.
- [4] D.M. Bates and D.G. Watts, *Nonlinear Regression and Its Applications*, New York: J. Wiley and Sons, 1988.
- [5] M.S. Bazaraa, H.D. Sherali and C.M. Shetty, *Nonlinear Programming: Theory and Algorithms*, New York: Wiley, 1993.
- [6] D. Belsley, On the Efficient Computation of Full-Information Maximum-Likelihood Estimation, *Journal of Econometrics* **14** (1980), 203–224.
- [7] R. Courant, *Differential and Integral Calculus*, (vol. 2), New York: Wiley, 1968.
- [8] M.G. Dagenais, The Computation of FIML Estimates as Iterative Generalized Least Squares Estimates in Linear and Nonlinear Simultaneous Equations, *Econometrica* **46** (1978), 1351–1362.
- [9] R. Davidson and J.G. MacKinnon, *Estimation and Inference in Econometrics*, New York: Cambridge University Press, 1993.
- [10] J.E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Philadelphia: SIAM, 1996.
- [11] Dennis, Gay and Welsch, An Adaptive Nonlinear Least-squares Algorithm, *TOMS* **7**, 1981, pp. 348–368.

- [12] J.R. Donaldson and R.B. Schnabel, Computational Experience with Confidence Regions and Confidence Intervals for Nonlinear Least Squares, *Technometrics* **29** (1987), 67–82.
- [13] J.M. Dufour, M. Gaudry and T.C. Liem, The Cochrane-Orcutt Procedure: Numerical Examples of Multiple Admissible Minima, *Economics Letters* **6** (1980), 43–48.
- [14] C. Fraley, Software Performance on Nonlinear Least-Squares Problems, Report No. 89–1244, Stanford University Department of Computer Science, 1989.
- [15] D.M. Gay, Algorithm 611: Subroutines for Unconstrained Minimization Using a Model/Trust-Region Approach, *ACM TOMS* **9** (1983), 503–524.
- [16] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization*, New York: Academic Press, 1981.
- [17] W. Greene, *Econometric Analysis, 4e*, New York: MacMillan, 1999.
- [18] K.L. Judd, *Numerical Methods in Economics*, Cambridge, MA: MIT Press, 1998.
- [19] W. Kaplan, *Advanced Calculus*, Cambridge, MA: Addison-Wesley, 1952.
- [20] W. Kennedy and J. Gentle, *Statistical Computing*, New York: Marcel-Dekker, 1980.
- [21] D. Luenberger, *Introduction to Linear and Nonlinear Programming*, New York: Addison-Wesley, 1984.
- [22] B.D. McCullough, Assessing the Reliability of Statistical Software: Part II, *The American Statistician* **53** (1999), 149–159.
- [23] B.D. McCullough, Econometric Software Reliability: EViews, LIMDEP, SHAZAM and TSP, *Journal of Applied Econometrics* **14** (1999), 191–202, with comment and reply at **15**, 107–111.
- [24] B.D. McCullough, The Accuracy of *Mathematica v4.0* as a Statistical Package, *Computational Statistics* **15** (2000), 279–299.
- [25] B.D. McCullough and C. Renfro, Benchmarks and Software Standards: A Case Study of GARCH Procedures, *Journal of Economic and Social Measurement* **25**(2) (1999), 59–71.
- [26] B.D. McCullough and H.D. Vinod, The Numerical Reliability of Econometric Software, *Journal of Economic Literature* **37** (1999), 633–665.
- [27] B.D. McCullough and Berry Wilson, On the Accuracy of Statistical Procedures in Microsoft Excel 97, *Computational Statistics and Data Analysis* **31**(1) (1999), 27–37.
- [28] W. Murray, Failure, the Causes and Cures, in: *Numerical Methods for Unconstrained Optimization*, W. Murray, ed., New York: Academic Press, 1972, pp. 107–122.
- [29] M. Nerlove, personal communication, 1999.
- [30] P. Newbold, C. Agiakloglou and J. Miller, Adventures with ARIMA Software, *International Journal of Forecasting* **10** (1994), 573–581.
- [31] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.R. Flannery, *Numerical Recipes in Fortran, 2e*, New York: Cambridge University Press, 1994.
- [32] R.E. Quandt, Computational Problems and Methods, in: *Handbook of Econometrics*, (Vol. I), Z. Griliches and M.D. Intriligator, eds., Amsterdam: North-Holland, 1983.
- [33] C. Renfro, Normative Consideration in the Development of a Software Package for Econometric Estimation, *Journal of Economic and Social Measurement* **23** (1997), 277–330.
- [34] C. Rose and M. Smith, *Mathematical Statistics with Mathematica*, New York: Springer-Verlag, 2001.
- [35] C.W. Ueberhuber, *Numerical Computation*, (vol. 2), Berlin: Springer, 1997.
- [36] H.D. Vinod, Review of GAUSS for Windows, Including Its Numerical Accuracy, *Journal of Applied Econometrics* **15** (2000), 211–220.