SIAM J. MATRIX ANAL. APPL.
Vol. 11, No. 3, pp. 466-480, July 1990

ROBUST REGRESSION COMPUTATION USING ITERATIVELY REWEIGHTED LEAST SQUARES*

DIANNE P. O'LEARY†

Abstract. Several variants of Newton's method are used to obtain estimates of solution vectors and residual vectors for the linear model $Ax = b + e = b_{true}$ using an iteratively reweighted least squares criterion, which tends to diminish the influence of outliers compared with the standard least squares criterion. Algorithms appropriate for dense and sparse matrices are presented. Solving Newton's linear system using updated matrix factorizations or the (unpreconditioned) conjugate gradient iteration gives the most effective algorithms. Four weighting functions are compared, and results are given for sparse well-conditioned and ill-conditioned problems.

Key words. iteratively reweighted least squares, robust regression

AMS(MOS) subject classifications. 62J05, 65F20

1. Introduction.

Consider the linear model

$$Ax = b + e = b_{true},$$

where A, the model matrix, has dimension $m \times n$; b is the vector of observations; b_{true} is the unknown vector of true values; e is the unknown vector of observation errors; and x is the unknown vector of parameters. For a given vector x, we define the residual vector $r(x) = b - Ax$.

We discuss in this paper various algorithms for obtaining estimates of the solution vector $\hat{x}$, the residual vector $r(\hat{x})$, and the norm of the residual vector using the iteratively reweighted least squares criterion: i.e., we wish to solve the problem

$$\min_x \sum_{i=1}^{m} \rho(r_i(x)), \tag{1}$$

where ρ is a given function. For a discussion of the statistical properties of this type of regression, see, for example [19]. Taking $\rho(z) = z^2/2$ gives the ordinary linear least squares problem. In order to reduce the influence of outliers, other functions have been proposed, and we consider in this paper four such functions, each twice continuously differentiable almost everywhere, with nonnegative second derivative wherever it is defined. Huber [18] used

$$\rho(z) = \begin{cases} z^2/2, & |z| \le \beta, \\ \beta|z| - \beta^2/2, & |z| > \beta, \end{cases}$$

where β is a problem-dependent parameter. Dutter [11] gives a safeguarded algorithm that overcomes degenerate cases. Minimizing Huber's function leads to a quadratic programming problem, and it is possible to develop finitely terminating algorithms as in the work of Clark and Osborne [3]. The logistic function [4] is

$$\rho(z) = \beta^2 \log(\cosh(z/\beta)).$$

* Received by the editors August 9, 1989; accepted for publication (in revised form) December 8, 1989. This work was supported by the Air Force Office of Scientific Research under grant 87-0158.

† Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742 (oleary@cs.umd.edu).

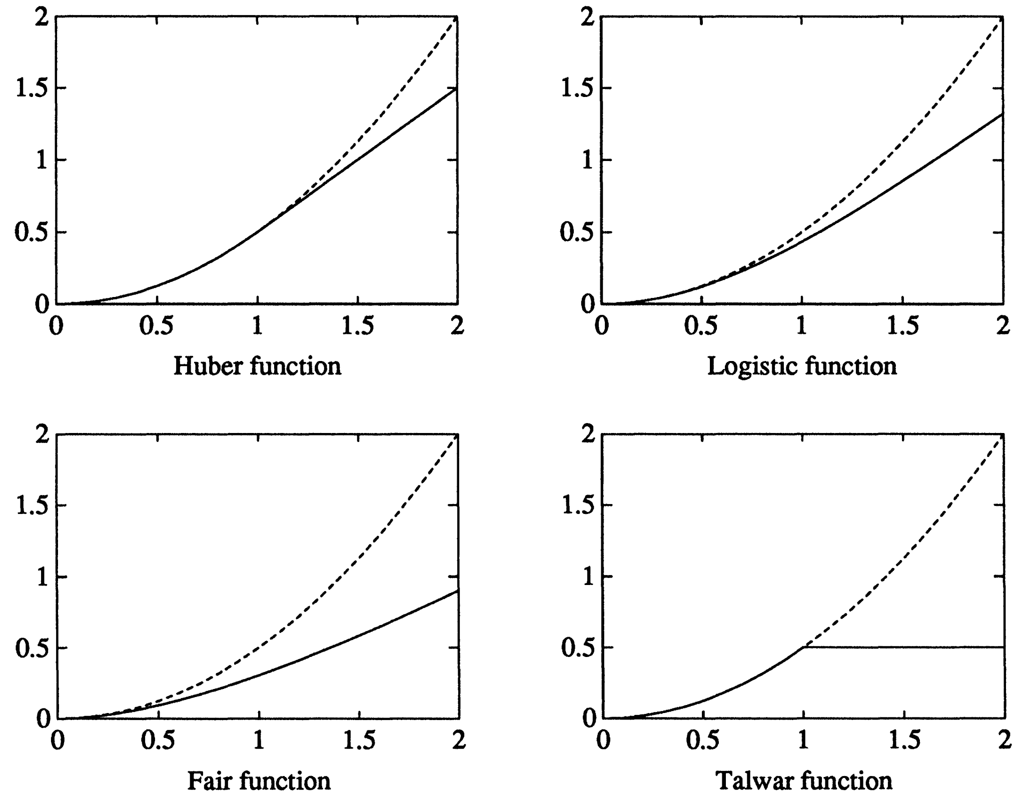

FIG. 1. *The weighting functions (solid lines) with the standard least squares function (dotted lines) as reference. The constant β is set to* 1.

Fair [14] proposed the function

$$\rho(z) = \beta^2(|z|/\beta - \log(1 + |z|/\beta)).$$

Huber [18] also proposed the function

$$\rho(z) = \begin{cases} z^2/2, & |z| \leq \beta, \\ \beta^2/2, & |z| > \beta, \end{cases}$$

which is given the name Talwar [16] in [4]. Graphs of these functions are given in Fig. 1. There have been other proposals to use other convex and nonconvex weighting functions, but the methods we discuss may have stability problems for nonpositive second derivatives.

The subject of this paper is the comparison of some algorithms for solving the iteratively reweighted least squares problem, a comparison of the performance of various weighting functions, and a discussion of the interaction between the weighting and the conditioning in the matrix A.

Robust regression through the use of functions related to least squares has been the subject of intense research, and we note only selected references here. Dempster, Laird, and Rubin [6] discuss statistical properties of the estimates under the assumption that the observation errors are independent normal. Coleman et al. [4] developed a high quality set of routines to compute robust estimators for eight weight functions, including the four discussed in this paper.

We take the viewpoint that the variance of the observation errors is known, at least approximately, and thus the scale is fixed. This is appropriate in some but not all applications, and it is possible to obtain simultaneous estimates of the scale factors and the solution; see, for example, Shanno and Rocke [25] and Ekblom [13].

The algorithms are presented in §2, and results are discussed in §3 and summarized in §4.

A similar computational problem (see (2)) arises in the core step in algorithms like that of Karmarkar for solving linear programming problems [20], and the algorithms in this paper have application there as well.

2. The algorithms. We will use Newton-like methods to solve our problem. Our first observation is that although we are minimizing over x-space, it is easier to work in the appropriate subspace of r-space. To establish some notation, we express (1) as

$$\min_x \hat{f}(x) \equiv \min_x f(r(x)) \equiv \min_x \sum_{i=1}^{m} \rho(r_i(x)).$$

Let y be the gradient vector for $f(r)$:

$$y_i = \rho'(r_i),$$

and let $D(r)$ be a diagonal matrix with entries

$$d_{ii} = \rho''(r_i).$$

Now, the function $\hat{f}(x)$ has a gradient $\hat{g}$ and Hessian matrix of second derivatives $\hat{H}$ defined by

$$\hat{g} = -A^T y, \qquad \hat{H} = A^T D A,$$

and $\hat{H}$ is positive semidefinite if ρ'' is nonnegative.

The step direction for Newton's method for minimizing $\hat{f}(x)$ is $\hat{s} = -\hat{H}^{-1}\hat{g}$, and a change of $\hat{s}$ in the x variables will create a change in the residual of $s = -A\hat{s}$, or

$$s = -A(A^T D A)^{-1} A^T y. \tag{2}$$

Since we need to assess the progress of Newton's method by evaluating the function ρ at each element of the residual, the computation is more conveniently done without the x variables. Further, determining the search direction for the x variables involves a computation whose conditioning is related to that of A, but, as we demonstrate below (see Algorithm 2), the conditioning for the problem of determining the search direction in r depends on $Q^T D Q$, where the columns of Q form an orthonormal basis for the range of A.

The general method is as follows:

```
Given an initial $x$, compute an initial $r = b - Ax$.
Repeat until convergence:
    1. Compute the search direction $s = -A(A^T D(r) A)^{-1} A^T y$.
    2. Perform a linesearch to determine a value $\alpha$ for which
       $f(r + \alpha s)$ is sufficiently less than $f(r)$.
Upon convergence to a residual vector $r_{opt}$, compute the correspond-
ing $x_{opt}$ by solving the consistent linear system $Ax = b - r_{opt}$.
```

Due to round-off errors, the system $Ax = b - r_{opt}$ may fail to be consistent, and the norm of the residual from this system is a good diagnostic.

If fewer than n residuals are below the cut-off value β for the function ρ, then the Hessian matrix may be rank deficient. To prevent this from occurring at initial stages of the iteration, where we may be far from the optimal solution, we gradually decrease the cut-off value from a very large number to the desired value over the first four steps of the iteration. This has the effect of starting the iteration from the least squares solution.

Developing efficient and reliable linesearch algorithms is not an easy task, but one such algorithm, due to Jorge J. Moré and David J. Thuente, is CVSRCH in the MINPACK collection of routines. It uses function and gradient values. Since the value $\alpha = 1$ is almost always a good choice, we use that for the initial guess and use coarse tolerances (.1) for convergence in x, the function value, and the gradient value.

We now focus attention on the strategies for computing the search direction. Our basic tool is the QR factorization of an $m \times n$ matrix into the product of an $m \times n$ matrix Q with orthogonal columns, and an $n \times n$ upper triangular matrix R (see, for example, [7]).

Algorithm 1. If ρ'' is nonnegative, then the matrix D has nonnegative elements, and we may factor the matrix $D^{1/2}A$ as $\hat{Q}\hat{R}$. The definition (2) of s then becomes

$$s = -D^{-1/2}\hat{Q}\hat{R}(\hat{R}^T\hat{Q}^T\hat{Q}\hat{R})^{-1}\hat{R}^T\hat{Q}^T D^{-1/2}y = -D^{-1/2}\hat{Q}\hat{Q}^T D^{-1/2}y.$$

Dutter [10] uses this formulation in his "HV algorithm" for the Huber function.

Algorithm 2 (QR Newton). The first algorithm requires a QR factorization of an $m \times n$ matrix at each iteration. To avoid this, we could factor $A = QR$, which yields

$$s = -QR(R^TQ^TDQR)^{-1}R^TQ^Ty = -Q(Q^TDQ)^{-1}Q^Ty.$$

Each iteration is accomplished using the Cholesky factors of the symmetric $n \times n$ matrix $B = Q^TDQ$.

Algorithm 3 ($\bar{B}$ Newton). We can express the matrix B as

$$B = Q^TDQ = \sum_{i=1}^{m} d_{ii}\bar{q}_i\bar{q}_i^T,$$

where $\bar{q}_i^T$ is the ith row of Q. Only the elements d_{ii} change from iteration to iteration, and as the algorithm converges, we can expect many terms in the summation to remain relatively constant. Thus a reasonable way to reduce the computational work is to monitor B and perform rank-one updates to the Cholesky factors only when the change in some component $d_{ii}\bar{q}_i^T\bar{q}_i$ is large compared to the size of B. One way to measure this is to test whether $\bar{q}_i^T\bar{q}_i$ times the change in d_{ii} is greater than some tolerance times the norm of the matrix that we have factored. If so, a rank-one update (or downdate) to the factorization can be performed using standard algorithms implemented, for example, in LINPACK [7]. Ekblom [13] also used the update idea, but worked with A^TDA rather than with Q^TDQ.

Since B is not necessarily fully updated, the computed search direction is not necessarily the true Newton direction but is some approximation to it.

TABLE 1

Costs per iteration of the various algorithms. Not included in the table are costs common to all algorithms: the function evaluations in the line search (m ρ evaluations each) and the Hessian evaluation (m ρ'' evaluations). "Qmult." means multiplication of Q (or Q^T) times a vector. "Solve" means solution of a linear system using Cholesky factors.

Algorithm	Work per iteration	Operations counts (full matrix)
1. First Newton	QR fact. and 2 Qmults.	$mn^2 - 1/3n^3 + 2mn + O(n^2)$
2. QR Newton	Form and factor Q^TDQ, 1 solve, and 2 Qmults.	$m(n^2+n)/2 + n^3/6 + 2mn + n^2$
3. $\bar{B}$ Newton	k updates to B factors, 1 solve, and 2 Qmults.	$(1.75k+1)n^2 + 2mn$
4. PCG Newton	k updates to B factors, l pcg itns., and 2 Qmults.	$1.75kn^2 + 2(l+1)mn + ln^2 + 5nl$
5. CG Newton	2 Qmults. and l cg itns.	$2(l+1)mn + 5nl$

Algorithm 4 (PCG Newton). In Algorithm 3, we established a distinction between a matrix $\bar{B}$ for which we have Cholesky factors and the current true matrix B, and we settled for an approximation to the Newton search direction rather than fully updating $\bar{B}$. We can, however, compute the Newton direction quite efficiently by using the preconditioned conjugate gradient algorithm to solve the linear system $Bw = Q^Ty$ with $\bar{B}$ as the preconditioner. Decreasing the number of matrix updates increases the number of conjugate gradient iterations.

This algorithm is related to the truncated Newton method [5], [22].

Algorithm 5 (CG Newton). For the particular weighting functions we are using, the matrix D has a very special form. For the Huber and the Talwar functions, each diagonal entry is 1 for residuals with magnitude less than β, and 0 for the outlying residuals. The logistic and Fair functions have diagonal entries that fall quickly from 1 to 0 as the residual increases from 0. We notice that B is a multiple of the identity matrix whenever D is, and thus for practical problems B may differ from the identity by a matrix of small rank, where the rank is equal to the number of outliers, plus a matrix of small norm (for the logistic and Fair functions). Thus we also consider computing the Newton direction using the conjugate gradient algorithm with no preconditioning. This algorithm is particularly well suited for large sparse problems, since only the matrix Q and the diagonal matrix D are required. Conjugate gradients have been used by Scales, Gersztenkorn, and Treitel [24] with $\rho(z) = |z|^p$ (p less than one), but they solved linear systems involving A^TDA rather than Q^TDQ.

Table 1 presents the costs associated with an iteration of each of the five algorithms. The number of floating point additions and multiplications are tabulated, assuming that the matrix is dense and that updates to Cholesky factors result from increases in diagonal elements (at a cost of $1.5n^2$ operations) as often as decreases ($2n^2$ operations). From these numbers we see that function and Hessian evaluations have a negligible cost compared to the linear algebra overhead of an iteration.

For sparse matrices, working with the original matrix A rather than the factor Q would better preserve sparsity but, as we will see later, the linear system expressed in terms of Q requires no preconditioning in the conjugate gradient algorithm, and this is a substantial savings. There has been some work in reorderings of A that produce a sparse representation of Q (see, for example, Tewarson [27], Chen and Tewarson [2], and Duff [8]), but most of this work has been directed toward maintaining sparsity in Q and R simultaneously. The sparsity of R is not essential to the algorithms considered here.

A compromise between sparsity and ease of solution of systems Q^TDQ can be achieved by performing an LU factorization of A rather than a QR. Peters and Wilkinson suggested the use of this factorization for standard least squares problems, and Björck and Duff [1] studied its implementation for sparse matrices. All of the algorithms above can be rewritten for this factorization, substituting L for Q, and U for R. Since it has been observed that L is usually well conditioned, even for ill-conditioned A, there is hope that solving systems involving L^TDL will be substantially easier than solving those involving the original matrix A^TDA. Computational experience is reported in §3.3.

3. Results.

3.1. A note on perturbations. The solution x^* of an iteratively reweighted least squares problem is characterized by the gradient of $\hat{f}(x^*)$ being zero. The weight functions ρ are designed to diminish the effects on x^* of outliers in the observations b, but how is x^* affected by small perturbations in b? A simple first-order perturbation analysis will yield insight.

The gradient of $\hat{f}(x^*)$ is $-A^T\rho'(b - Ax^*) = 0$. If b is changed to $b + \Delta b$, then the solution will be changed to $x^* + \Delta x$, where

$$-A^T\rho'\,(b + \Delta b - A(x^* + \Delta x)) = 0.$$

We expand this to first-order terms as

$$A^T\rho'(b+\Delta b-A(x^*+\Delta x)) \approx A^T(\rho'(b-Ax^*)+\rho''(b-Ax^*)(\Delta b-A\Delta x)) = A^TD(\Delta b-A\Delta x),$$

and Δx is a vector that makes this equal to zero. Thus,

$$A^TDA\Delta x \approx A^TD\Delta b,$$

or, Δx is defined by $\Delta x \approx A_D^\dagger \Delta b$, where $A_D^\dagger = (A^TDA)^{-1}A^TD$ is a weighted pseudoinverse of A. We now have the conclusion that

$$||\Delta x||_2 \lesssim ||A_D^\dagger||_2||\Delta b||_2. \tag{3}$$

Unfortunately, the D in this expression is evaluated at the unknown solution r^*, but results in [26] and [21] guarantee that if D is positive semidefinite and if Q is a matrix whose columns form an orthonormal basis for the range of A, and if $\zeta \leq 1$ is the smallest of the nonzero singular values of all matrices formed from nonempty subsets of rows of Q, then

$$||A_D^\dagger||_2 \leq \zeta^{-1}||A_I^\dagger||_2.$$

Thus the change (3) in x can be bounded by the change in b magnified by a factor dependent only on the matrix A.

These expressions suggest that for ill-conditioned matrices A the weighting functions will not overcome the sensitivity of the solution to small perturbations in the observations, and this will be illustrated by the numerical results.

3.2. The test problems. There is a large number of small least squares test problems in the literature (see, for example, the previously cited references) but a very small number of large ones in the Harwell-Boeing test set [9]. This makes parametric studies difficult. Shanno and Rocke [25] and others use randomly generated problems, but such problems tend to be very well conditioned [12].

The following procedure was used to generate test problems with varying conditioning and varying number of outliers.

The $m \times n$ matrix A was constructed as the product of three matrices C, E, and F. The matrix C had the same dimensions as A and had $\mu m/2$ nonzeros in each column, each sampled from a normal probability distribution $N(0,1)$. The positions for the nonzeros were chosen randomly from a uniform distribution. F was a square matrix with diagonal entries chosen to be two times $N(0,1)$ samples and with one off-diagonal $N(0,1)$ entry (except in row n) in a random position. E was a diagonal matrix with entries between 1 and $1/\kappa$ (equally spaced on log scale). The product $A = CEF$ has approximately μm nonzeros per column (i.e., "density" μ) and its singular values usually have separations proportional to those of E.

The true solution vector was taken to be z, the vector of all ones, and the right-hand side was chosen to be $b = Az + \sigma N(0,1)$, except that outliers were generated by adding $100\sigma N(0,1)$ to n_{out} randomly chosen elements of b. In all cases, σ was taken to be .01.

Thus, the test problems have five parameters: m, n, μ, κ, and n_{out}.

Computations used double precision arithmetic on a Sun-3 machine. Convergence was declared when the change in the function value was less than 10^{-5}. This test is not suitable in general, but because of the uniform scaling of our problems it is sufficient for our purposes. See [4] for a better termination criterion.

The termination test for the conjugate gradient iterations was that the residual norm be less than 10^{-8} times the norm of the right-hand side, forcing a rather accurate solution to the linear systems.

We investigated several questions, some related to the algorithms and some related to the performance of the various weighting functions. Since Algorithm 1 is not as stable as Algorithm 2 and failed to find a full-rank Hessian matrix quite often in the experiments, we do not present data on its performance.

3.3. How well do the algorithms perform? How does the convergence rate depend on the test problem parameters? As shown in Tables 2 and 3, there seems to be no trend to increased work as the condition number of the problem increases or as the number of outliers increases. As the number of outliers increases, however, there is an increased tendency for the algorithms to fail to find a full rank Hessian matrix. Updating $\bar{B}$ less frequently usually increased the number of function and Hessian evaluations. But a factor of 10 fewer updates, costing $O(n^2)$ each, at worst

TABLE 2
Results of varying condition number. 500 × 100 *matrix, density* $\mu = .1$, $n_{out} = 10$ *outliers, constant* $\beta = 2.5\sigma$. *Table entries: number of function evaluations, number of Hessian evaluations, number of* cg *iterations, number of Cholesky updates for algorithms with few or frequent updates to the factors.*

	QR	$\bar{B}$	PCG	CG	CG–LU
$\kappa = 6$					
Fair, few updt.	16, 9	39,20,0, 100	16, 9, 76, 100	16, 9, 76	16, 9,362
Fair, freq. updt.		20,10,0,1184	16, 9, 32,1071		
Talwar, few updt.	19, 9	39,20,0, 100	24,11,186, 100	24,11,186	25,10,520
Talwar, freq. updt.		26,11,0, 788	18, 9, 50, 707		
$\kappa = 175$					
Fair, few updt.	16, 9	39,20,0, 100	16, 9, 76, 100	16, 9, 76	16, 9,414
Fair, freq. updt.		20,10,0,1184	16, 9, 32,1071		
Talwar, few updt.	67,10	75,20,0, 100	23,11,186, 100	23,11,186	19,10,553
Talwar, freq. updt.		62,11,0, 788	18,10, 53, 709		
$\kappa = 14576$					
Fair, few updt.	17, 9	40,20,0, 100	16, 9, 76, 100	16, 9, 76	16, 9,432
Fair, freq. updt.		21,10,0,1184	16, 9, 32,1071		
Talwar, few updt.	29,11	16, 5,0, 100	60,10,179, 100	60,10,179	23,10,568
Talwar, freq. updt.		31,11,0, 786	56,10, 55, 711		

TABLE 3
Results of varying number of outliers. 100 × 20 *matrices, density* $\mu = .1$*, well-conditioned problems* (E *=identity matrix), constant* $\beta = 2.5\sigma$*. Table gives number of function evaluations, number of Hessian evaluations, number of* cg *iterations, and number of Cholesky updates.*

Outliers						
True	Est.		QR	$\bar{B}$	PCG	CG
0	0	Huber	9,5	9,5,0, 20	13, 5, 0, 20	13, 5, 0
0	0	Logistic	9,5	9,5,0, 58	9, 5, 2, 58	9, 5, 3
0	0	Fair	8,5	8,5,0,106	13, 5, 3,106	13, 5, 5
0	0	Talwar	9,5	9,5,0, 20	13, 5, 0, 20	13, 5, 0
10	–	Huber	fail	fail	fail	fail
10	10	Logistic	19,9	17,9,0,295	19, 9, 14,291	19, 9, 54
10	13	Fair	16,8	16,8,0,314	16, 8, 12,309	16, 8, 39
10	10	Talwar	21,9	22,9,0,170	42, 9, 5,170	46,13,111
20	–	Huber	fail	fail	fail	fail
20	–	Logistic	fail	fail	fail	fail
20	28	Fair	15,8	16,8,0,330	15, 8, 11,335	15, 8, 45
20	10	Talwar	fail	40,9,0,120	47,20,110,190	25,11,104
30	–	Huber	fail	fail	fail	fail
30	–	Logistic	fail	fail	fail	fail
30	48	Fair	18,9	18,9,0,347	18, 9, 15,348	18, 9, 62
30	88-90	Talwar	fail	13,6,0,114	28, 7, 9,118	22, 7, 27

TABLE 4

Results of varying update parameter for factors. 100 × 20 *well-conditioned matrix, density* $\mu = .5$, $n_{out} = 10$ *outliers. Table entries: number of function evaluations, number of Hessian evaluations, number of* cg *iterations, and number of Cholesky updates.*

Update tolerance		$\bar{B}$	PCG
0.001	Huber	10, 5,0, 63	fail
	Logistic	23,10,0,222	22,10,59,215
	Fair	16, 8,0,227	15, 8,16,231
	Talwar	10, 5,0, 63	35, 8, 5, 92
0.010	Huber	10, 5,0, 41	fail
	Logistic	22, 9,0, 64	22,10,43, 66
	Fair	18, 9,0, 81	15, 8,37, 72
	Talwar	10, 5,0, 41	34, 8,23, 56
0.100	Huber	10, 5,0, 20	fail
	Logistic	26,13,0, 20	22,10,59, 20
	Fair	33,17,0, 20	15, 8,48, 20
	Talwar	10, 5,0, 20	33, 8,38, 20

TABLE 5

Variability of results over a set of 10 *well-conditioned problems.* 100 × 20, *density* $\mu = .25$, $n_{out} = 10$ *outliers.* $\beta = 10\sigma$, *update param* = .001. *Table entries: range and average number of function evaluations, number of Hessian evaluations, and number of Cholesky updates for Algorithm* 3, *the* $\bar{B}$ *Newton method.* (*Results for Huber exclude one problem that produced failure.*)

	Function evaluations		Hessian evaluations		Updates	
Huber	7-39	ave. 22	5-8	ave. 7	21- 45	ave. 35
Logistic	12-16	ave. 14	7-9	ave. 8	54-170	ave. 92
Fair	12-16	ave. 13	7-8	ave. 8	128-222	ave. 171
Talwar	4-39	ave. 20	5-8	ave. 7	21- 83	ave. 40

doubled the number of function and Hessian evaluations, that cost $O(m)$, resulting in a faster algorithm.

The last two columns of Table 2 show the results of using the conjugate gradient algorithms with no preconditioning. The "CG" data results from use of the QR factors, while the "CG–LU" data involved the LU factors. The use of the LU factors required between 2.8 and 5.6 times as many conjugate gradient iterations, but there was no trend to increased work as the condition number of A was increased. Use of the original matrix A would have shown an increase in the number of iterations as the condition number grew. Each use of conjugate gradients for the LU factors took on average 40-50 iterations, while the theoretical maximum is $n = 100$. Using the LU factors with conjugate gradients, with or without a preconditioning matrix, seems to be a good approach for sparse matrices if the resulting Q would be too dense.

Table 4 shows further results of performing fewer updates to the approximate Hessian. On this problem of size 100×20, the matrix was never updated if the update tolerance was set greater than or equal to 0.100, and there was very little penalty in the number of function or gradient evaluations for either the $\bar{B}$ Newton (Algorithm 3) or the preconditioned conjugate gradient (Algorithm 4) methods.

Table 5 shows the variability of the computational work for a set of 10 random problems with the same test parameters.

3.4. Which functions perform better? How does ill-conditioning affect the performance of the functions? Figure 2 shows graphs of the solutions and residuals produced for well-conditioned problems by ordinary least squares and by the different ρ functions considered in this paper. A well-conditioned matrix of dimension 100×20 was generated, and 10 sample right-hand sides were generated by adding random noise to Az, using 10 different sets of outliers. The Huber and the Talwar functions each produced a solution vector bigger than 10^4 on one of the right-hand sides, and those runs were disregarded. Each of the weighting functions produces a solution vector closer to the unperturbed vector of ones than ordinary least squares, but the corresponding residual vectors are slightly larger.

Figure 3 shows the errors in the solution vector for a sequence of increasingly more ill-conditioned problems with 10 outliers. The residual norm for least squares was 5.00, whereas that for the Fair function was 5.74; neglecting the 10 largest components of the residual, the norm for least squares was 2.48 whereas that for the Fair function was 0.21. The norm of the error in the x vector was also at least ten times smaller in all cases using the Fair function. For a problem with condition number 175, least squares gave an error of 11.5, compared with the true solution of norm 10.0, so the computed solution vector had little resemblance to the true solution. Both least squares and the Fair function were unable to recover the x vector for the most ill-conditioned problem. This is predicted by the perturbation results in §3.1.

3.5. How do the algorithms perform on "real" problems? Experiments were also run using the housing price equation and the 506 observations of Boston census tracts discussed in [15]. This model expresses the median value of homes in each tract as a combination of 14 factors (crime rate, zoning statistics, average number of rooms in homes, accessibility to radial highways, etc.). The model was used without the scaling discussed in [15], and the integer parts of the singular values were 10,128, 672, 632, 272, 197, 156, 84, 74, 10, 8, 6, 5, 2, and 1. The right-hand side elements were around 10, and σ was estimated as 0.1. The constant β was taken to be 2.5σ. The four functions found between 59 and 61 outliers, using a solution vector of size approximately 10. The x vectors from the Huber, Fair, and Logistic

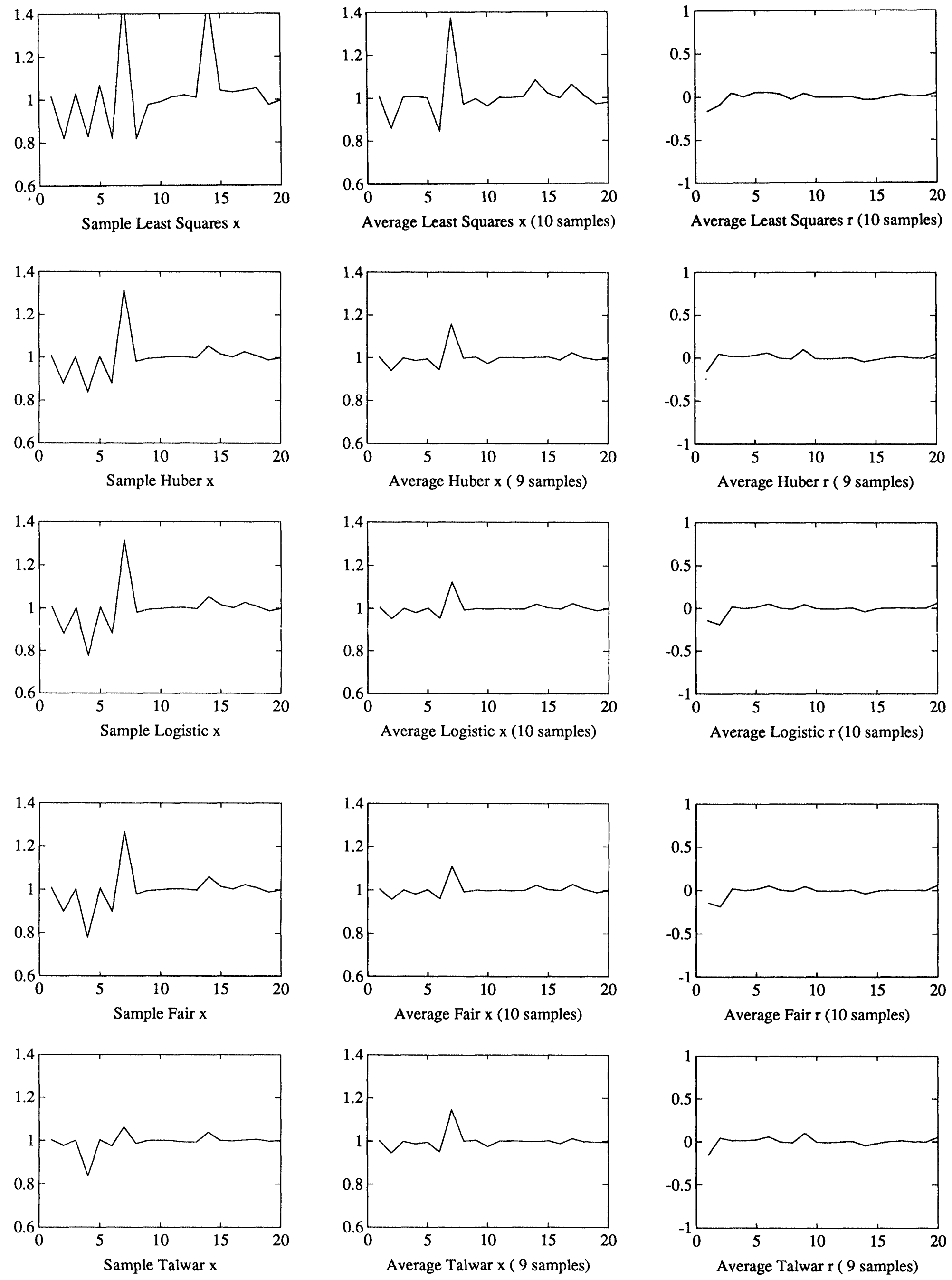

FIG. 2. *Solution vectors for one problem and average solution and residual vectors for* 10 *problems,* 100×20, *density* $\mu = .1$, *well conditioned,* 10 *outliers.*

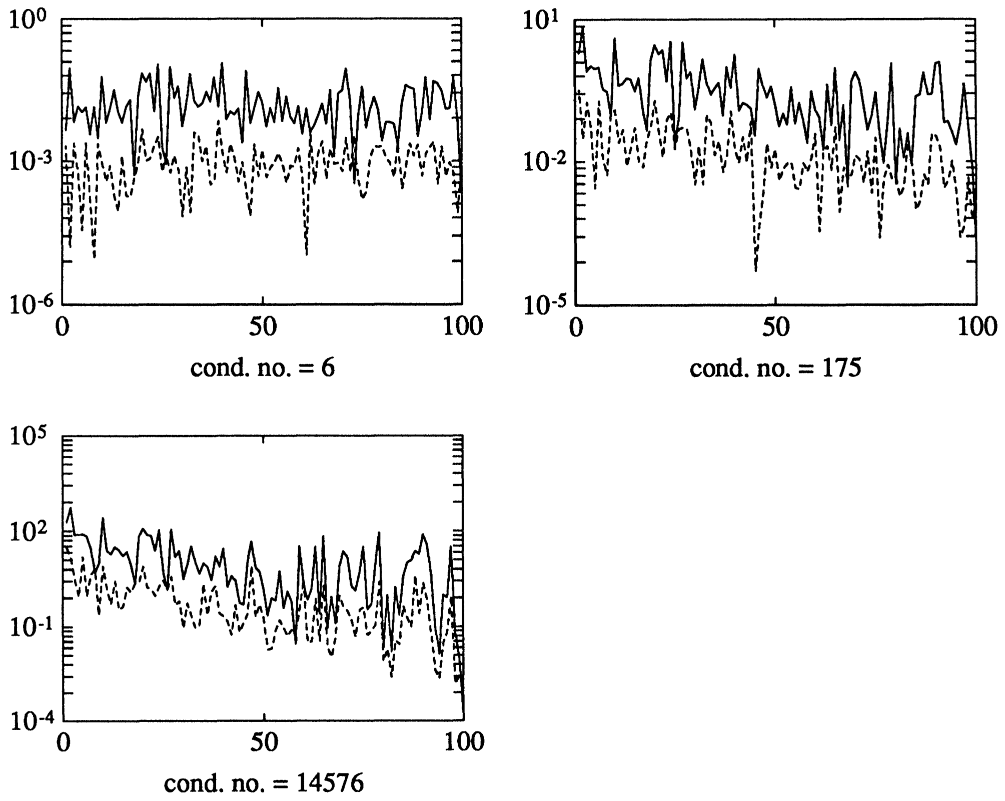

FIG. 3. *The absolute error in each component of the x vector for ordinary least squares (solid) vs. the Fair function (dashed) ($\beta = 2.5\sigma$) for three problems of dimension* 500×100 *with density* $\mu = .1$ *and* 10 *outliers. The residual norm for each problem was* 5.00 *for least squares and* 5.74 *for the Fair function. Neglecting the* 10 *largest components of the residual, the norm was* 2.48 *for least squares and* 0.21 *for the Fair function.*

functions had infinity norm differences of at most .04; the Talwar vector differed from the Fair function by .25. The residual norms were 4.096, 4.086, and 4.088 for the first three functions and 4.650 for Talwar. The CG Newton algorithm took 27 function evaluations, 7 Hessian evaluations, and 17 cg iterations for the Huber function, and 10 function evaluations, 7 Hessian evaluations, and 16-18 cg iterations for the Logistic and the Fair functions.

4. Conclusions. (1) Quadratic programming algorithms should be used for functions such as those of Huber and Talwar, but the best algorithms for the other functions are the $\bar{B}$ Newton algorithm if the problem is not too large and the CG Newton algorithm (with QR or LU factorization) for larger problems.

(2) The functions considered here give better solution vectors than ordinary least squares, but even so, the elements of the solution vector are often heavily contaminated with error if the product of the matrix condition number and the standard deviation of the errors in the data is greater than one.

(3) The number of iterations for the Newton-type algorithms seems insensitive to the conditioning of the matrix and to the number of outliers in the data.

(4) The algorithm for generating sparse test problems with varying conditioning may be useful elsewhere.

(5) The development of parallel algorithms for this class of problems is the subject

of current research. For these Newton-like algorithms, we need a parallel algorithm for determining the search direction and a parallel linesearch algorithm. Parallel versions of the conjugate gradient algorithm [23] are promising candidates for computing the direction.

Acknowledgments. Virginia Klema and Beth Ducot kindly provided the data for the housing model, and Gene Golub provided several of the references. Bob Plemmons made helpful comments on the manuscript.

REFERENCES

[1] Å. BJÖRCK AND I. S. DUFF, *A direct method for the solution of sparse linear least squares problems*, Linear Algebra Appl., 34 (1980), pp. 43–67.

[2] Y. T. CHEN AND R. P. TEWARSON, *On the fill-in when sparse vectors are orthonormalized*, Computing, 9 (1972), pp. 53–56.

[3] D. I. CLARK AND M. R. OSBORNE, *Finite algorithms for Huber's M-estimator*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 72–85.

[4] D. COLEMAN, P. HOLLAND, N. KADEN, AND V. KLEMA, *A system of subroutines for iteratively reweighted least squares computations*, ACM Trans. Math. Software, 6 (1980), pp. 327–336.

[5] R. S. DEMBO AND T. STEIHAUG, *Truncated-Newton algorithms for large-scale unconstrained optimization*, Math. Programming, 26 (1983), pp. 190–212.

[6] A. P. DEMPSTER, N. M. LAIRD, AND D. B. RUBIN, *Iteratively reweighted least squares for linear regression when errors are normal/independent distributed*, in Multivariate Analysis V, P. R. Krishnaiah, ed., North-Holland, New York, 1980, pp. 35–57.

[7] J. J. DONGARRA, C. B. MOLER, J. R. BUNCH, AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[8] I. S. DUFF, *Pivot selection and row ordering in Givens reduction on sparse matrices*, Computing, 13 (1974), pp. 239–248.

[9] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[10] R. DUTTER, *Robust regression: Different approaches to numerical solutions and algorithms*, Tech. Report Research Report No. 6, Fachgruppe fuer Statistik, ETH, Zurich, 1975.

[11] ———, *Algorithms for the Huber estimator in multiple regression*, Computing, 18 (1977), pp. 167–176.

[12] A. EDELMAN, *Eigenvalues and condition numbers of random matrices*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 543–560.

[13] H. EKBLOM, *A new algorithm for the Huber estimator in linear models*, BIT, 28 (1988), pp. 123–132.

[14] R. C. FAIR, *On the robust estimation of econometric models* (ref. by [17]), Ann. Econ. Social Measurement, 3 (1974), pp. 667–678.

[15] G. GOLUB, V. KLEMA, AND S. C. PETERS, *Rules and software for detecting rank degeneracy*, J. Econometrics, 12 (1980), pp. 41–48.

[16] M. J. HINICH AND P. P. TALWAR, *A simple method for robust regression*, J. Amer. Statist. Assoc., 70 (1975), pp. 113–119.

[17] P. W. HOLLAND AND R. E. WELSCH, *Robust regression using iteratively reweighted least-squares*, Commun. Statist. - Theor. Meth., A6 (1977), pp. 813–827.

[18] P. J. HUBER, *Robust estimation of a location parameter*, Annals Math. Statist., 35 (1964), pp. 73–101.

[19] ———, *Robust Statistics*, John Wiley, New York, 1981.

[20] N. KARMARKAR, *A new polynomial algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[21] D. P. O'LEARY, *On bounds for scaled projections and pseudo-inverses*, Linear Algebra Appl., 1990, to appear.

[22] ———, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Programming, 23 (1982), pp. 20–33.

[23] ———, *Parallel implementation of the block conjugate gradient algorithm*, Parallel Comput., 5 (1987), pp. 127–139.

[24] J. A. SCALES, A. GERSZTENKORN, AND S. TREITEL, *Fast l_p solution of large, sparse, linear systems: application to seismic travel time tomography*, J. Comput. Phys., 75 (1988), pp. 314–333.

[25] D. F. SHANNO AND D. M. ROCKE, *Numerical methods for robust regression: Linear models*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 86–97.

[26] G. W. STEWART, *On scaled projections and pseudo-inverses*, Linear Algebra Appl., 112 (1989), pp. 189–194.

[27] R. P. TEWARSON, *On the orthonormalization of sparse vectors*, Computing, 3 (1968), pp. 268–279.