An Improved Algorithm for Discrete L1 Linear Approximation

Ian Barrodale and F.D.K. Roberts

MRC Technical Summary Report #1172

January 1972

THE UNIVERSITY OF WISCONSIN

MATHEMATICS RESEARCH CENTER

Contract No.: DA-31-124-ARO-D-462

AN IMPROVED ALGORITHM FOR DISCRETE ℓ_1 LINEAR APPROXIMATION

I. Barrodale and F. D. K. Roberts

MRC Technical Summary Report # 1172
January 1972

Received August 17, 1971

Madison, Wisconsin 53706

ABSTRACT

By modifying the simplex method of linear programming, we are able to present an algorithm for ℓ_1 approximation which appears to be superior computationally to any other known algorithm for this problem.

AN IMPROVED ALGORITHM FOR DISCRETE ℓ_1 LINEAR APPROXIMATION

I. Barrodale[*] and F. D. K. Roberts[**]

1. Introduction

This paper provides an algorithm which appears to be the most efficient yet devised for solving the general ℓ_1 linear approximation problem. The algorithm is a modification of the simplex method applied to the primal formulation of the ℓ_1 problem as a linear program. A Fortran IV program for the algorithm is supplied, and some numerical results are given comparing the computational behavior of this and other ℓ_1 algorithms.

The general ℓ_1 linear approximation problem can be stated as follows. Let $f(x)$ be a given real-valued function defined on a discrete subset $X = \{x_1, x_2, \ldots, x_m\}$ of Euclidean space E^N. Given $n\ (\leq m)$ real-valued functions $\phi_j(x)$ defined on X, we form a linear approximating function $L(A, x) = \sum_{j=1}^{n} a_j\phi_j(x)$ for any set $A = \{a_1, a_2, \ldots, a_n\}$ of real numbers. The ℓ_1 problem is to determine a best approximation $L(A^*, x)$ which minimizes

$$\sum_{i=1}^{m} |f(x_i) - L(A, x_i)| . \tag{1}$$

It is well known that at least one best approximation always exists, and there are now several algorithms for calculating a best approximation. However,

---

[*] Presently visiting the Mathematics Research Center, Madison, Wisconsin, U.S.A.
Work sponsored by the United States Army under Contract No.: DA-31-124-ARO-D-462.
[**] Department of Mathematics, University of Victoria, Victoria, B. C., Canada.

some of these algorithms require that the set of given functions $\{\phi_1(x), \phi_2(x), \ldots, \phi_n(x)\}$ be linearly independent on X, or that it satisfy the Haar condition on X. This latter condition, which is equivalent to requiring that the given functions be linearly independent on every subset of n distinct elements from X, can be quite restrictive in practice.

As yet, it seems that the only use made of "least-first-power" approximations has been in the analysis of experimental data. Even in this situation, many scientists are unwilling to use estimates of parameters computed in a norm for which few statistical tests are available. However, there is no doubt (see Barrodale [2]) that best ℓ_1 approximations are often superior to best ℓ_2 approximations when estimating the true form of data that contain some very inaccurate observations.

The approximating functions used by experimental scientists for interpreting data are almost never polynomials, and they rarely constitute Haar sets (indeed, continuous functions of more than one independent variable never form a Haar set). If the approximation problem is presented as the equivalent problem of solving an overdetermined system of linear equations, it may be quite difficult to decide on the rank of the given matrix. Consequently, we feel that a data-fitting algorithm should impose as few restrictions as possible on the user's choice of approximating function. Since our algorithm is based on the simplex method it can be used with any linear approximating function.

The connection between linear programming and ℓ_1 approximation was pointed out by Wagner [15] in 1959, although other more direct (but cumbersome) algorithms for the ℓ_1 problem had been proposed much earlier (see Barrodale [2] for references to these).

In 1966 Barrodale and Young [6] described a primal algorithm which takes advantage of the special structure of the linear programming formulation of the ℓ_1 problem. The primal algorithm of the present paper is an improved version of this earlier simplex algorithm; in short, we have been able to reduce significantly the total number of iterations required, by discovering how to pass through several neighboring simplex vertices in a single iteration.

In 1967 Usow [13] presented a descent method which attempts to calculate a best ℓ_1 approximation by locating the lowest vertex of a convex polytope representing the set of all possible ℓ_1 approximations. This iterative algorithm requires that the approximating function be formed from a Haar set, and it also stops prematurely if a certain degenerate situation arises. Usow [14] states that he has now corrected this latter deficiency and extended his algorithm to approximating functions formed from linearly independent sets of given functions.

In 1969 Robers and Ben-Israel [11] applied a new method for linear programming to the dual formulation of the ℓ_1 problem. Their new method (which they call interval linear programming) is capable of solving any bounded-variable linear programming problem, and so it is natural to apply it to the ℓ_1 problem in particular. Robers and Robers [12] have now supplied a special version of the general method of [11] which is designed specifically for the ℓ_1 problem.

Finally, early in 1971 Abdelmalek [1] described an algorithm which determines best ℓ_1 approximations as the limit of best ℓ_p approximations as $p \to 1^+$. His technique thus obtains a solution to a linear problem by solving a sequence of nonlinear problems.

Clearly, a comparison of the computational behavior of these various algorithms is both desirable and overdue. Consequently, we have conducted some numerical tests on (i) the latest descent method of Usow [14], (ii) the special adaptation by Robers and Robers [12] of the more general method of Robers and Ben-Israel [11], (iii) the bounded-variable simplex method of Dantzig [7] applied to the dual formulation of the ℓ_1 problem, and (iv) our modified standard form of the simplex method applied to the ℓ_1 problem in its primal form. (The numerical results given in Abdelmalek [1] demonstrate that his algorithm cannot possibly be as efficient as those selected above). These numerical tests confirm that our algorithm is the most efficient in general.

The ℓ_1 problem (1) is restated as a linear program in §2, our algorithm is described in §3, and the numerical results are in §4.

2. Linear programming and ℓ_1 approximation

For the ℓ_1 problem (1) let us write $\phi_{j,i} \equiv \phi_j(x_i)$, $f_i \equiv f(x_i)$, and define nonnegative variables u_i, v_i, b_j, c_j by putting

$$f_i - \sum_{j=1}^{n} a_j \phi_{j,i} = u_i - v_i, \quad \text{for } i = 1, 2, \ldots, m,$$

and $a_j = b_j - c_j$ for $j = 1, 2, \ldots, n$. Then a best ℓ_1 approximation corresponds to an optimal solution to the (primal) linear programming problem:

$$\left.\begin{array}{l} \text{minimize } \sum_{i=1}^{m} (u_i + v_i) \\[2ex] \text{subject to } \quad f_i = \sum_{j=1}^{n} (b_j - c_j)\phi_{j,i} + u_i - v_i \\ \qquad\qquad\qquad\qquad\qquad (i = 1, 2, \ldots, m) \\[2ex] \text{and } b_j, c_j, u_i, v_i \geq 0 . \end{array}\right\} \qquad (2)$$

The formulation (2) appears in Barrodale and Roberts [4] along with the following result.

<u>Theorem</u>. If the column rank of the $m \times n$ matrix $\Phi = \{\phi_{j,i}\}^T$ is k ($\leq n$), then there exists a best ℓ_1 approximation which interpolates $f(x)$ in at least k points of X.

Several authors (Wagner [15], Rabinowitz [10], Barrodale [3], for example) have suggested that the dual of (2) should be solved instead when m is large. The dual of (2) is stated most conveniently (e.g. Rabinowitz [10]) as the following bounded-variable linear programming problem:

$$\left.\begin{array}{l} \text{maximize } \sum_{i=1}^{m} (w_i f_i - f_i) \\[2ex] \text{subject to } \sum_{i=1}^{m} w_i \phi_{j,i} = \sum_{i=1}^{m} \phi_{j,i} \\ \qquad\qquad\qquad\qquad (j = 1, 2, \ldots, n) \\[2ex] \text{and } 0 \leq w_i \leq 2 . \end{array}\right\} \qquad (3)$$

As the results of our numerical tests indicate, applying the bounded-variable simplex method of Dantzig [7] to (3) leads to a less efficient algorithm in general than solving (2) by our version of the standard form of the simplex method.

3. The algorithm

A direct application of the simplex method to (2) does not yield an efficient algorithm. (On the other hand, this might well be a reasonable strategy if a particular problem arises involving additional (inequality) constraints on the approximating function). Barrodale and Young [6] observed that an initial basic feasible solution to (2) is immediately available, and that most of the column vectors in the simplex tableau need not be stored explicitly. Barrodale and Roberts [4] recommended that for the first n iterations the choice of pivotal column be restricted to the columns associated with the variables b_j and c_j, and that thereafter a special pivotal column selection rule be employed. In spite of these improvements, the simplex method can take a large number of iterations to solve some problems. The reason for this can best be explained by considering the following particular problem.

Suppose that we wish to approximate $f(x) = e^x$ on 201 uniformly spaced points in the interval $[0, 2]$ by a straight line $L(A, x) = a_1 + a_2x$. The initial simplex basis is provided by the column vectors associated with $u_1, u_2, \ldots, u_{201}$. If the simplex method with the above modifications is applied to this problem, the first iteration brings the vector associated with b_1 into the basis and forces the vector associated with u_1 out. The second iteration brings the vector corresponding to b_2 into the basis, and that corresponding to u_2 goes out. Thus, after two iterations, the simplex tableau represents an approximation which interpolates the first and second data points (since the current values of u_1, v_1, u_2, and v_2, are all zero). Subsequent iterations correspond to approximations which interpolate the following pairs of data points: (1st., 3rd.), (2nd., 3rd.), (2nd., 4th.), (2nd., 5th.), (2nd., 6th.), (3rd., 6th.), (3rd., 7th.), ... After 201 iterations, the (unique) best approximation is obtained which interpolates the 51st. and 151st. data points. The difficulty arises because of the nonnegativity restrictions on u_i, v_i (and also b_j, c_j) which are somewhat artificial. What is required is an algorithm which at each iteration can bypass intermediate data points and thereby substantially reduce the number of iterations required to obtain a best approximation. We shall now describe such an algorithm; it requires just 7 iterations to solve the above problem.

Inspection of the linear programming problem (2) reveals that (i) an initial basic feasible solution is immediately available, and (ii) only n columns are needed to store the information contained in the right-hand sides of the equality constraints. Thus, denoting the columns of the simplex tableau corresponding to (2) by $\underline{R}$, $\underline{b}_j$, $\underline{c}_j$, $\underline{u}_i$, $\underline{v}_i$ (see Table 2, for example), an initial basis is provided by $\underline{u}_1, \underline{u}_2, \ldots, \underline{u}_m$ whenever each f_i is nonnegative. If an f_i is negative we change the sign of the corresponding row and replace $\underline{u}_i$ in the basis by $\underline{v}_i$. It is also clear that

$\underline{b}_j = -\underline{c}_j$ and $\underline{u}_i = -\underline{v}_i$, and that the sum of the marginal (or reduced) costs of $\underline{b}_j$ and $\underline{c}_j$ is zero and of $\underline{u}_i$ and $\underline{v}_i$ is -2. Thus the condensed form of the simplex method (in which the basis is suppressed) can be applied to just n columns, which initially contain $\underline{b}_1, \underline{b}_2, \ldots, \underline{b}_n$.

Basis	$\underline{R}$	$\underline{b}_1$	$\underline{b}_2$	$\cdots$	$\underline{b}_n$
$\underline{u}_1$	f_1	$\phi_{1,1}$	$\phi_{2,1}$	$\cdots$	$\phi_{n,1}$
$\underline{u}_2$	f_2	$\phi_{1,2}$	$\phi_{2,2}$	$\cdots$	$\phi_{n,2}$
$\vdots$	$\vdots$				$\vdots$
$\underline{u}_m$	f_m	$\phi_{1,m}$	$\phi_{2,m}$	$\cdots$	$\phi_{n,m}$
Marginal costs $\rightarrow$	$\sum_{i=1}^{m} f_i$	$\sum_{i=1}^{m} \phi_{1,i}$	$\sum_{i=1}^{m} \phi_{2,i}$	$\cdots$	$\sum_{i=1}^{m} \phi_{n,i}$

TABLE 1: Initial condensed simplex tableau for the algorithm, assuming each f_i is nonnegative. (A full tableau is displayed in Table 2).

The entire computation is performed using an $m + 3$ by $n + 2$ matrix. The additional column and one additional row are required to label the vectors (see Table 1), and the remaining row is used whenever the initial tableau is scaled. We scale the initial data by dividing the column vectors $\underline{R}$, $\underline{b}_1$, $\underline{b}_2, \ldots, \underline{b}_n$ by their largest elements in absolute value. The use of this scaling option in our program is recommended when large problems are to be solved, although it will not always improve their condition.

The algorithm is implemented in two stages. Stage 1 restricts the choice of pivotal column during the first n iterations to the vectors $\underline{b}_j$ and $\underline{c}_j$. The vector to enter the basis is chosen as that with the most positive marginal cost. The vector leaving the basis is chosen from among the basic vectors $\underline{u}_i$ and $\underline{v}_i$ by selecting that which causes the maximum reduction in the objective function. At the end of Stage 1, the rank k $(\leq n)$ of the matrix Φ is determined by the total number of vectors $\underline{b}_j$, $\underline{c}_j$ in the basis. Since k of the vectors $\underline{u}_i$ (or $\underline{v}_i$) have been removed from the basis, the current simplex tableau represents an approximation which interpolates at least k data points. If the approximation interpolates more than k data points then the simplex tableau is degenerate; this does not cause any problems in practice.

Stage 2 involves interchanging nonbasic $\underline{u}_i$ or $\underline{v}_i$ with basic $\underline{u}_i$ or $\underline{v}_i$: the basic $\underline{b}_j$ and $\underline{c}_j$ vectors are not allowed to leave the basis during Stage 2. The vector entering the basis is that with the most positive marginal cost, and the vector leaving the basis is again chosen as that which causes the maximum reduction in the objective function. The algorithm terminates when all the marginal costs are nonpositive. In Stage 2 each simplex tableau corresponds to an approximation which interpolates k data points (assuming nondegeneracy). At each iteration, $k-1$ of these points remain fixed. The vector entering the basis determines which point is to be dropped from the interpolating set while the vector leaving the basis determines the new point of interpolation.

The final tableau at the end of Stage 2 may be infeasible since some of the basic vectors $\underline{b}_j$ and $\underline{c}_j$ can have negative values associated with them. The solution becomes feasible (and hence optimal) by interchanging such basic vectors $\underline{b}_j$ (or $\underline{c}_j$) with the corresponding nonbasic vectors $\underline{c}_j$ (or $\underline{b}_j$).

The main modification to the simplex method is in choosing the vector $\underline{u}_i$ or $\underline{v}_i$ to leave the basis. In both Stage 1 and Stage 2 this vector is chosen as that which causes the maximum reduction in the objective function. A direct search over all the basic vectors $\underline{u}_i$ and $\underline{v}_i$ is inefficient and a computationally more efficient technique can be used. This is best described by means of a simple worked example.

Example. Find the best ℓ_1 approximation to $\{(1, 1), (2, 1), (3, 2), (4, 3), (5, 2)\}$ by $L(A, x) = a_1 + a_2x$. Equivalently, find the ℓ_1 solution to the overdetermined system of equations:

$$a_1 + a_2 = 1$$

$$a_1 + 2a_2 = 1$$

$$a_1 + 3a_2 = 2$$

$$a_1 + 4a_2 = 3$$

$$a_1 + 5a_2 = 2$$

The full initial simplex tableau (without scaling) is given in Table 2.

Costs	→		0	0		0	1	1	1	1	1	1	1	1	1	1
↓	Basis	$\underline{R}$	$\underline{b}_1$	$\underline{b}_2$	$\underline{c}_1$	$\underline{c}_2$	$\underline{u}_1$	$\underline{u}_2$	$\underline{u}_3$	$\underline{u}_4$	$\underline{u}_5$	$\underline{v}_1$	$\underline{v}_2$	$\underline{v}_3$	$\underline{v}_4$	$\underline{v}_5$
1	$\underline{u}_1$	1	1	1	-1	-1	1					-1				
1	$\underline{u}_2$	1	1	2^{**}	-1	-2		1					-1			
1	$\underline{u}_3$	2	1	3^{***}	-1	-3			1					-1		
1	$\underline{u}_4$	3	1	4	-1	-4				1					-1	
1	$\underline{u}_5$	2	1	5^{*}	-1	-5					1					-1
Marginal costs	→	9	5	15	-5	-15	0	0	0	0	0	-2	-2	-2	-2	-2

TABLE 2: Full initial simplex tableau for the worked example.

At the first iteration in Stage 1, $\underline{b}_2$ is brought into the basis corresponding to the largest marginal cost (= 15). The normal simplex pivot (5^*) corresponds to an approximation in which $b_2 = 2/5$, $u_5 = v_5 = 0$, i.e. an approximation which interpolates the fifth data point. However, if we increase b_2 beyond the value 2/5 we can further reduce the objective function, but this makes u_5 negative. Hence we replace $\underline{u}_5$ in the basis by $\underline{v}_5$. This can be accomplished by subtracting twice the fifth row from the marginal cost row (thus making the marginal cost of $\underline{v}_5$ zero), changing the sign of the fifth row and replacing the label $\underline{u}_5$ by $\underline{v}_5$ in the basis. The marginal cost of $\underline{b}_2$ is now 5, and hence we increase b_2 further. The second pivot (2^{**}) corresponds to an approximation in which $b_2 = 1/2$, $u_2 = v_2 = 0$, i.e. an approximation which interpolates the second data point. Interchanging $\underline{u}_2$ and $\underline{v}_2$ in the basis reduces the marginal cost of $\underline{b}_2$ to 1. Thus we increase b_2 further. The third pivot (3^{***}) corresponds to an approximation in which $b_2 = 2/3$, $u_3 = v_3 = 0$, i.e. an approximation which interpolates the third data point. The objective function cannot be decreased further by increasing b_2, since if we interchange $\underline{u}_3$ and $\underline{v}_3$ in the basis, the marginal cost of $\underline{b}_2$ becomes -5. Hence we pivot on this element (3^{***}) and bring $\underline{b}_2$ into the basis in place of $\underline{u}_3$. Using the condensed tableau, the complete solution to this example is given in Table 3. After two iterations (at the end of Stage 1) the marginal costs of $\underline{u}_1$ and $\underline{u}_3$ are -1 and 0 respectively, and so the marginal costs of the (suppressed) vectors $\underline{v}_1$ and $\underline{v}_3$ are -1 and -2 respectively (recall that the sum of the marginal costs of $\underline{u}_i$ and $\underline{v}_i$ is -2). Since all of the nonbasic vectors have nonpositive marginal costs, there is no need for Stage 2 in this example. The final tableau represents a best approximation $L(A^*, x) = \frac{1}{2} + \frac{1}{2}x$ which interpolates the first and third data points. The deviations at the other points are given by $v_2 = \frac{1}{2}$, $u_4 = \frac{1}{2}$, $v_5 = 1$

Basis	$\underline{R}$	$\underline{b}_1$	$\underline{b}_2$
$\underline{u}_1$	1	1	1
$\underline{u}_2$	1	1	2^{**}
$\underline{u}_3$	2	1	3^{***}
$\underline{u}_4$	3	1	4
$\underline{u}_5$	2	1	5^{*}
Marginal costs	9	5	15

Basis	$\underline{R}$	$\underline{b}_1$	$\underline{u}_3$
$\underline{u}_1$	1/3	2/3	-1/3
$\underline{v}_2$	1/3	-1/3	2/3
$\underline{b}_2$	2/3	1/3	1/3
$\underline{u}_4$	1/3	-1/3	-4/3
$\underline{v}_5$	4/3	2/3	5/3
Marginal costs	7/3	2/3	-1/3

Basis	$\underline{R}$	$\underline{u}_1$	$\underline{u}_3$
$\underline{b}_1$	1/2	3/2	-1/2
$\underline{v}_2$	1/2	1/2	1/2
$\underline{b}_2$	1/2	-1/2	1/2
$\underline{u}_4$	1/2	1/2	-3/2
$\underline{v}_5$	1	-1	2
Marginal costs	2	-1	0

TABLE 3: Solution to the worked example using condensed tableaux.

and the error of approximation is 2. The solution is not unique since the final marginal cost of $\underline{u}_3$ is zero; this vector can be brought into the basis yielding a new best approximation which interpolates the first and fifth data points. Other best approximations can be generated as convex combinations of these two extreme point solutions.

This concludes the description of our primal algorithm. There are, of course, several variants of the standard form of the simplex method which can be used to solve a linear programming problem (two forms of the revised simplex method, the primal-daul algorithm, the dual simplex algorithm, etc.). However, the denseness of the condensed tableau corresponding to (2), the availability of an initial basic feasible solution, and the simplicity with which we can implement our main modification, combine together to make the standard form of the simplex method the most economical algorithm for the ℓ_1 problem.

4. Numerical results

The following test problems were designed both to illustrate the flexibility of the various algorithms and to compare their computational behavior.

Example 1. Approximate $f(x) = e^{-x}\sin x$ on $X = \{0(0.02)4\}$ by $L(A, x) = \sum_{j=1}^{n} a_j x^{j-1}$ for $n = 1, 2, \ldots, 7$.

Example 2. Approximate $f(x,y) = \mathrm{Re}\{\int_{x+iy}^{\infty} e^{-z}/z\, dz\}$ on 100 points defined by $x = 0.1(0.1)1.0$, $y = 0.1(0.1)1.0$, using the two-dimensional approximating function

$$L(A, x, y) = a_1 + a_2x + a_3y + a_4xy + a_5x^2 + a_6y^2 .$$

(The values of $f(x,y)$ are taken from Fox, Goldstein and Lastman [8] where they are recorded to six decimal places).

Example 3. Approximate $f(x) = \sqrt{x}$ on $X = \{0(0.02)1\}$ by the cubic spline $L(A, x) = \sum_{j=1}^{4} a_j x^{j-1} + \sum_{t=1}^{4} a_{4+t}(x-\alpha_t)_+^3$, where $\alpha_1 = 0.1$, $\alpha_2 = 0.2$, $\alpha_3 = 0.4$, $\alpha_4 = 0.7$.

Example 4. Approximate $f(x) = \sin x$ on $X = \{-\frac{\pi}{2}(\frac{\pi}{k})\frac{\pi}{2}\}$, for $k = 20$, 50, 100, 200, by $L(A, x) = P_5(x) = \sum_{j=1}^{6} a_j x^{j-1}$ with $P_5'(-\frac{\pi}{2}) = 0$, $P_5'(\frac{\pi}{2}) = 0$. The constraints on the derivative can be handled by regarding $(-\frac{\pi}{2}, 0)$ and $(\frac{\pi}{2}, 0)$ as two data points to be approximated by $P_5'(x)$. We then ensure that the derivative actually interpolates these points by suitably weighting the two extra rows corresponding to these constraints. (We simply multiplied both rows by 1000 in the initial tableau).

Example 5. Approximate $f(x) = \min(e^x, e^{1/2})$ on $X = \{0(\frac{1}{d})1\}$, for $d = 20, 50, 100, 200$, by $L(A, x) = a_1 + \sum_{j=1}^{5} (a_{2j} \sin jx + a_{2j+1} \cos jx)$.

Example 6. Calculate a best ℓ_1 solution to the following overdetermined system of linear equations:

$$5a_1 + 3a_2 + 4a_3 + 12a_4 + 4a_5 = 7$$

$$9a_1 + 7a_2 + 3a_3 + 19a_4 + 13a_5 = 4$$

$$6a_1 + 6a_2 + 0a_3 + 12a_4 + 12a_5 = 2$$

$$9a_1 + 9a_2 + 7a_3 + 25a_4 + 11a_5 = 7$$

$$3a_1 + 0a_2 + 1a_3 + 4a_4 + 2a_5 = 7$$

$$8a_1 + 1a_2 + 8a_3 + 17a_4 + 1a_5 = 7$$

$$1a_1 + 9a_2 + 8a_3 + 18a_4 + 2a_5 = 3$$

$$0a_1 + 9a_2 + 3a_3 + 12a_4 + 6a_5 = 3$$

$$3a_1 + 1a_2 + 1a_3 + 5a_4 + 3a_5 = 5$$

$$6a_1 + 7a_2 + 6a_3 + 19a_4 + 7a_3 = 1$$

$$6a_1 + 1a_2 + 9a_3 + 16a_4 + -2a_5 = 4$$

$$0a_1 + 4a_2 + 8a_3 + 12a_4 + -4a_5 = 1$$

$$0a_1 + 5a_2 + 7a_3 + 12a_4 + -2a_5 = 6$$

$$7a_1 + 3a_2 + 2a_3 + 12a_4 + 8a_5 = 6$$

$$5a_1 + 4a_2 + 9a_3 + 18a_4 + 0a_5 = 0$$

The entries in the first, second, and third columns were chosen from a table of random integers. The fourth column is the sum of the previous three columns, and the fifth column is the sum of the first two columns minus the third column. The matrix of coefficients Φ is thus a 15×5 matrix of column rank 3. The entries on the right-hand side, which correspond to values of $f(x)$ in the approximation problem, were also chosen from a table of random integers.

Example 7. Approximate $f(x) = 1 + x + x^2 + x^3 + x^4 + E(x)$, where $E(x) = 5$ for $0.94 \leq x \leq 1$ and $E(x) = 0$ otherwise, on $X = \{0(0.02)1\}$ by $L(A, x) = \sum_{j=1}^{5} a_j x^{j-1}$. This example was designed primarily to test the effectiveness of the new algorithm of Usow [14]. If an intermediate approximation is formed from just the uncontaminated portion of the data, corresponding to the range $0 \leq x \leq 0.92$, it will result in a large number of zero residuals (i.e. an $L(A, x)$ is formed which interpolates $f(x)$ in many more than n $(= 5)$ points of X). This phenomenon can cause the original algorithm of Usow [13] to stop prematurely. In a primal linear programming algorithm it causes degeneracy; in practice the simplex method handles degeneracy without any special provisions having to be made.

We attempted to solve all seven test examples using each of the four methods identified as:

(i) USOW: see Usow [14].

(ii) ROB2: see Robers and Robers [12].

(iii) DUAL: the dual formulation (3) solved by the bounded-variable simplex method of Dantzig [7].

(iv) PRIM: the primal formulation (2) solved by our algorithm

Methods (i) and (ii) were coded by their respective authors, and methods (iii) and (iv) were coded by us; all of the programs were written in Fortran IV. The program supplied by Dr. K. H. Usow is still essentially at the experimental stage, since there are some quite delicate computations involved in his method. Unfortunately, we were not able to solve some of our test problems using his program even though all seven problems were attempted.

The central processor times given in Table 4 are for a CDC 6600 computer, and they do not include time spent in compiling or setting up the data for each problem. (These CP times are <u>not</u> exact, for we have observed slight discrepancies when repeating some of the computations). The calculations were performed in single precision floating-point arithmetic using a 48-bit mantissa. The best ℓ_1 error of approximation given in the final column of Table 4 is from the output supplied by our algorithm PRIM. The algorithm DUAL produced essentially the same results as PRIM, but ROB2 sometimes differed from these two by somewhat more than the expected rounding error (particularly in Example 5). The error of approximation produced by Usow also differed in some cases from that shown in Table 4. In the linear programming algorithms we used a tolerance of 10^{-8} when deciding whether or not certain quantities are "zero", and the initial approximation required by Usow was chosen by interpolating the first n points of X.

Finally, we note that all of these test problems have unique best ℓ_1 approximations (except, of course, for the linearly dependent overdetermined system in Example 6).

Example	TIME (seconds)				Error of Approximation $\sum_{i=1}^{m} \lvert f(x_i)-L(A^*,x_i) \rvert$
	USOW	ROB2	DUAL	PRIM	
1(m=201)					
n=1	*	2.38	0.63	0.29	21.81556
n=2	2.04	13.95	2.79	0.85	9.27171
n=3	1.97	18.57	2.58	0.81	9.07895
n=4	2.61	30.00	4.21	1.69	4.64073
n=5	2.81	40.82	5.12	2.54	1.05559
n=6	7.46	54.76	6.09	3.55	0.06045
n=7	5.94	61.77	7.63	3.29	0.03471
2(m=100,n=6)	*	16.93	1.68	1.06	3.87741
3(m=51,n=8)	*	9.04	0.70	0.51	0.05312
4(n=6)					
m=23	0.18	0.39	0.12	0.05	0.00188
m=53	0.65	1.39	0.60	0.17	0.00436
m=103	1.57	6.28	2.10	0.51	0.00858
m=203	5.25	33.18	7.86	1.67	0.01694
5(n=11)					
m=21	0.85	*	0.28	0.15	0.06170
m=51	1.60	*	1.60	0.51	0.15642
m=101	4.42	*	4.24	1.33	0.30852
m=201	10.90	*	15.54	4.27	0.61230
6(m=15,n=5)	*	*	0.05	0.02	34.22449
7(m=51,n=5)	*	2.80	0.28	0.22	16.73658

* No results available

TABLE 4: A comparison of the calculation times required by each algorithm to solve the test problems on a CDC 6600 computer.

5. Remarks

It is quite clear that our new primal algorithm is both efficient and reliable, and we supply a Fortran IV program for the method in the Appendix. We have used a double precision version of this code to solve several other problems on an IBM 360/50 and on a Univac 1108 computer. (In particular, we have solved an overdetermined system of size 500×10 using double precision versions of PRIM and DUAL on a Univac 1108; PRIM took 4.56 seconds and DUAL required 59.68 seconds to solve this problem).

In practice, of course, it makes little difference whether (say) PRIM or DUAL is used to solve a single small problem in data analysis. However, it is important to have an efficient algorithm at hand if several tens or hundreds of problems are to be solved. An important instance of this occurs when nonlinear approximating functions are used to interpret data, for in this case some linearization of the problem will normally be employed which calls for the repeated application of a linear algorithm. Two recent methods along this line are described in Barrodale, Roberts and Hunt [5] and Osborne and Watson [9].

Acknowledgements

We are very grateful to Dr. Philip Robers and to Dr. Karl Usow for supplying us with programs for their algorithms. Mr. Fred Crary and Mr. Bruce Wilson provided us with excellent programming assistance. The financial support provided by NRC Grants A5251 and A7143 is also much appreciated.

REFERENCES

1. N. N. Abdelmalek: Linear L_1 Approximation for a Discrete Point Set and L_1 Solutions of Overdetermined Linear Equations. J. ACM, 18 (1971), 41-47.

2. I. Barrodale: L_1 Approximation and the Analysis of Data. Applied Stat., 17 (1968), 51-57.

3. I. Barrodale: On Computing Best L_1 Approximations. From Approximation Theory, A. Talbot (ed.), Academic Press (1970), 205-215.

4. I. Barrodale and F. D.K. Roberts: Applications of Mathematical Programming to ℓ_p Approximation. From Nonlinear Programming, J. B. Rosen, O. . L. Mangasarian and K. Ritter (eds.), Academic Press (1970), 447-464.

5. I. Barrodale, F. D. K. Roberts and C. R. Hunt: Computing Best ℓ_p Approximations by Functions Nonlinear in One Parameter. Comp. J., 13 (1970), 382-386.

6. I. Barrodale and A. Young: Algorithms for Best L_1 and L_∞ Linear Approximations on a Discrete Set. Numer. Math., 8 (1966), 295-306.

7. G. B. Dantzig: Linear Programming and Extensions. Princeton Univ. Press (1963).

8. P. Fox, A. A. Goldstein and G. Lastman: Rational Approximations on Finite Point Sets. From Approximation of Functions, H. L. Garabedian (ed.), Elsevier (1965), 57-67.

9. M. R. Osborne and G. A. Watson: On an Algorithm for Discrete Nonlinear L_1 Approximation. Comp. J., 14 (1971), 184-188.

10. P. Rabinowitz: Applications of Linear Programming to Numerical Analysis. SIAM Rev., 10 (1968), 121-159.

11. P. D. Robers and A. Ben-Israel: An Interval Programming Algorithm for Discrete Linear L_1 Approximation Problems. J. Approx. Th., 2 (1969), 323-336.

12. P. D. Robers and S. S. Robers: Discrete Linear L_1 Approximation by Interval Linear Programming. (To appear in an ACM publication).

13. K. H. Usow: On L_1 Approximation II; Computation for Discrete Functions and Discretization Effects. SIAM J. Numer. Anal., 4 (1967), 233-244.

14. K. H. Usow: An Algorithm for Discrete Linear L_1 Approximation with Application to Spline Functions. (Manuscript).

15. H. M. Wagner: Linear Programming Techniques for Regression Analysis. J. Amer. Statist. Assoc., 54 (1959), 206-212.

APPENDIX

The Fortran IV SUBROUTINE L_1 (Q, M3, N2, TOLER, SCALE)

This subroutine calculates (for $n \leq m$) a best ℓ_1 approximation to $f(x)$ by $\sum_{j=1}^{n} a_j\phi_j(x)$ on a set $X = \{x_1, x_2, \ldots, x_m\}$, i.e. parameters $a_1^*, a_2^*, \ldots, a_n^*$ are determined which minimize $\sum_{i=1}^{m} |f(x_i) - \sum_{j=1}^{n} a_j\phi_j(x_i)|$. The algorithm is a modification of the standard form of the simplex method applied to the primal formulation of the ℓ_1 problem as a linear program.

INPUT

Q: A real array of dimensions $(m + 3, n + 2)$ which initially contains the $m \times n$ matrix

$$\Phi = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_n(x_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \cdots & \phi_n(x_m) \end{pmatrix}$$

in the first m rows and n columns, and the vector of function values $[f(x_1), f(x_2), \ldots, f(x_m)]^T$ in the first m rows of the (n+1)st. column. No other input is required.

M3: A positive integer set equal to $m + 3$.

N2: A positive integer set equal to $n + 2$.

TOLER: A positive real number set to approximately the desired order of relative accuracy of the calculated parameters a_j^*, e.g. 10^{-5} or 10^{-8}. This number prevents pivoting on elements of Q which are too small, and a decrease in its value will usually cause more iterations of the simplex method.

SCALE: An integer that should be set to 1 only if the user wishes the initial data to be scaled columnwise. In this event, the final tableau is rescaled automatically. (Scaling is usually beneficial).

No other input to the subroutine is required. If a best <u>weighted</u> ℓ_1 approximation is desired, the user should multiply each row of data by its chosen (positive) weight prior to entering the subroutine.

OUTPUT

All output parameters are real numbers contained in Q.

Q(m+2,n+1): Has the value 1 if a best approximation has been calculated, and the value 2 if no suitable pivot is available and the iterations have been terminated prematurely (this latter condition can occur only with badly conditioned problems).

Q(m+1,n+1): Error of approximation $\sum_{i=1}^{m} |f(x_i) - \sum_{j=1}^{n} a_j^* \phi_j(x_i)|$.

Q(m+1,n+2): The rank of the matrix Φ. (If the rank of Φ is less than n, the surplus columns of Φ are stored in the first columns of Q).

Q(m+2,n+2): The number of iterations required by the simplex method.

Q(m+3,n+2): Has the value 1 if the best approximation is unique and the value 0 if there is more than one best approximation (which may be due to the rank of Φ being less than n).

The absolute values of the parameters $a_1^*, a_2^*, \ldots, a_n^*$ are contained in the first n rows of the (n+1)st. column, and their signed labels are contained in the corresponding rows of the (n+2)nd. column (a_1^* has the label ± 1, a_2^* has the label ± 2, etc.). The absolute values of (m-n) of the deviations $f(x_i) - \sum_{j=1}^{n} a_j^* \phi_j(x_i) = u_i - v_i = d_i$ are contained in the remaining (m-n) rows of the (n+1)st. column, and their signed labels are contained in the corresponding rows of the (n+2)nd. column (d_1 has the label $\pm (10^6 + 1)$, d_2 has the label $\pm (10^6 + 2)$, etc.). Thus, for the worked example given in Table 3 of this paper, the first m rows of the (n+1)st. and (n+2)nd. columns of Q finally appear as:

1/2	1
1/2	2
1/2	$-(10^6 + 2)$
1/2	$10^6 + 4$
1	$-(10^6 + 5)$

If the labels $\pm (10^6 + i)$ do not appear in the (n+2)nd. column, then $d_i = 0$ and x_i is the abscissa of a point of interpolation.

```
      SUBROUTINE L1 (Q,M3,N2,TOLER,SCALE)                                     L1*00010
      REAL MAX,MIN                                                            L1*00020
      INTEGER OUT,SCALE                                                       L1*00030
      DIMENSION Q(M3,N2)                                                      L1*00040
      LOGICAL STAGE,TEST                                                      L1*00050
C   ***BIG SHOULD BE SET TO SOME LARGE POSITIVE REAL NUMBER.  (HERE IT        L1*00060
C   ***IS SET FOR THE UNIVAC 1108.)  THIS IS THE ONLY MACHINE DEPENDENT       L1*00070
C   ***FEATURE OF THIS SUBROUTINE.                                            L1*00080
      DATA BIG /1E37/                                                         L1*00090
                                                                              L1*00100
      M=M3-3                                                                  L1*00110
      M1=M+1                                                                  L1*00120
      M2=M+2                                                                  L1*00130
      N=N2-2                                                                  L1*00140
      N1=N+1                                                                  L1*00150
C   ASSIGN LABELS TO ROW M2 AND COLUMN N1                                     L1*00160
      DO 1 J=1,N                                                              L1*00170
    1 Q(M2,J)=FLOAT(J)                                                        L1*00180
      DO 2 I=1,M                                                              L1*00190
    2 Q(I,N2)=1.E6+FLOAT(I)                                                   L1*00200
C     CHECK FOR POSITIVE R.H.S. (COLUMN N1)                                   L1*00210
      DO 3 I=1,M                                                              L1*00220
      IF(Q(I,N1).GE.0.) GO TO 3                                               L1*00230
      DO 4 J=1,N2                                                             L1*00240
    4 Q(I,J)=-Q(I,J)                                                          L1*00250
    3 CONTINUE                                                                L1*00260
C     CHECK FOR SCALING OPTION (SCALE=1)                                      L1*00270
      IF(SCALE.EQ.0) GO TO 5                                                  L1*00280
      DO 6 J=1,N1                                                             L1*00290
      MAX=0.                                                                  L1*00300
      DO 7 I=1,M                                                              L1*00310
      B=ABS(Q(I,J))                                                           L1*00320
      IF(B.LE.MAX) GO TO 7                                                    L1*00330
      MAX=B                                                                   L1*00340
    7 CONTINUE                                                                L1*00350
```

```
      Q(M3,J)=MAX
      DO 8 I=1,M
    8 Q(I,J)=Q(I,J)/MAX
    6 CONTINUE
C   COMPUTE MARGINAL COSTS
    5 DO 9 J=1,N1
      SUM=0.
      DO 10 I=1,M
   10 SUM=SUM+Q(I,J)
    9 Q(M1,J)=SUM
C   STAGE I
      STAGE=.TRUE.
      KOUNT=0
      KR=1
      KL=1
   11 MAX=-1.
      DO 12 J=KR,N
      IF(ABS(Q(M2,J)).GT.1.E6) GO TO 12
      B=ABS(Q(M1,J))
      IF(B.LE.MAX) GO TO 12
      MAX=B
      IN=J
   12 CONTINUE
      IF(Q(M1,IN).GE.0.) GO TO 13
      DO 14 I=1,M2
   14 Q(I,IN)=-Q(I,IN)
C   DETERMINE VECTOR TO LEAVE THE BASIS
   13 MIN = BIG
      TEST=.FALSE.
      DO 15 I=KL,M
      B=Q(I,IN)
      IF(B.LE.TOLER) GO TO 15
      TEST=.TRUE.
      B=Q(I,N1)/B
      IF(B.GE.MIN) GO TO 15
      MIN=3
      OUT=I
   15 CONTINUE
C   CHECK FOR LINEAR DEPENDENCE IN STAGE I
      IF(TEST.OR..NOT.STAGE) GO TO 16
      DO 17 I=1,M2
      B=Q(I,KR)
      Q(I,KR)=Q(I,IN)
   17 Q(I,IN)=B
      KR=KR+1
      GO TO 36
   16 IF(TEST) GO TO 18
      Q(M2,N1)=2.
      RETURN
   18 PIVOT=Q(OUT,IN)
      IF(Q(M1,IN)-PIVOT-PIVOT.LT.TOLER) GO TO 19
      DO 20 J=KR,N1
      B=Q(OUT,J)
      Q(M1,J)=Q(M1,J)-B-B
   20 Q(OUT,J)=-B
      Q(OUT,N2)=-Q(OUT,N2)
      GO TO 13
C   PIVOT ON Q(OUT,IN)
   19 DO 21 J=KR,N1
      IF(J.EQ.IN) GO TO 21
      Q(OUT,J)=Q(OUT,J)/PIVOT
   21 CONTINUE
      DO 22 I=1,M1
      IF(I.EQ.OUT) GO TO 22
      B=Q(I,IN)
      DO 23 J=KR,N1
      IF(J.EQ.IN) GO TO 23
      Q(I,J)=Q(I,J)-B*Q(OUT,J)
   23 CONTINUE
   22 CONTINUE
```

```
      DO 24 I=1,MI
      IF(I.EQ.OUT) GO TO 24
      Q(I,IN)=-Q(I,IN)/PIVOT
   24 CONTINUE
      Q(OUT,IN)=1./PIVOT
      B=Q(OUT,N2)
      Q(OUT,N2)=Q(M2,IN)
      Q(M2,IN)=B
      KOUNT=KOUNT+1
      IF(.NOT.STAGE) GO TO 25
C   INTERCHANGE ROWS IN STAGE I
      KL=KL+1
      DO 26 J=KR,N2
      B=Q(OUT,J)
      Q(OUT,J)=Q(KOUNT,J)
   26 Q(KOUNT,J)=B
   36 IF(KOUNT+KR.NE.N1) GO TO 11
C   STAGE II
      STAGE=.FALSE.
   25 MAX = -BIG
      DO 27 J=KR,N
      B=Q(M1,J)
      IF(B.GE.O.) GO TO 28
      IF(B.GT.-2.) GO TO 27
      B=-B-2.
   28 IF(B.LE.MAX) GO TO 27
      MAX=B
      IN=J
   27 CONTINUE
      IF(MAX.LE.TOLER) GO TO 29
      IF(Q(M1,IN).GT.O.) GO TO 13
      DO 30 I=1,M2
   30 Q(I,IN)=-Q(I,IN)
      Q(M1,IN)=Q(M1,IN)-2.
      GO TO 13
```

```
C   PREPARE OUTPUT
 29 L=KL-1
    DO 31 I=1,L
    IF(Q(I,N1).GE.0.) GO TO 31
    DO 32 J=KR,N2
 32 Q(I,J)=-Q(I,J)
 31 CONTINUE
    Q(M2,N1)=1.
    Q(M2,N2)=FLOAT(KOUNT)
    Q(M1,N2)=FLOAT(N1-KR)
    IF(SCALE.EQ.0) GO TO 100
    DO 33 I=1,L
    J=ABS(Q(I,N2))+0.5
    B=Q(M3,J)
    DO 34 J=KR,N1
 34 Q(I,J)=Q(I,J)/B
 33 CONTINUE
    B=Q(M3,N1)
    DO 35 I=1,M1
 35 Q(I,N1)=Q(I,N1)*B
100 DO 110 J=1,N
    IF(ABS(Q(M1,J)).GT.TOLER) GO TO 110
    Q(M3,N2)=0.
    RETURN
110 CONTINUE
    Q(M3,N2)=1.
    RETURN
    END
```