ADAPTIVE PIECEWISE POLYNOMIAL
\[ L^1 \] APPROXIMATION

Paul G. Avila*

Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523

# TABLE OF CONTENTS

I. Introduction ............................................. 1
II. The Adaptive Curve Fitting Algorithm .................. 1
III. The |\_1\-Approximation Problem ...................... 7
IV. Numerical Results ..................................... 14
References ................................................. 23
Appendix .................................................. 24
ABSTRACT

In [1] and [2], algorithms were introduced for adaptively computing smooth piecewise polynomial approximations using uniform, least-squares ($\ell_2$), and restricted range uniform approximations. This paper deals with the introduction of $\ell_1$ adaptive curve fitting. Adopting the least-squares algorithm, a package has been developed that will compute smooth piecewise polynomial approximations to data and/or precise mathematical functions (in discrete form) allowing the user the option of using best $\ell_1$ or best $\ell_2$ approximations. (The code for this newly developed $\ell_1-\ell_2$ adaptive curve fitting package supercedes the code listed in [1] for the old least-squares package.) The adaptive curve fitting algorithm used in this package is described in detail in section II. Section III involves a discussion of the $\ell_1$-approximation problem as a linear programming problem, as well as a description of the $\ell_1$ algorithm used in the computing of the $\ell_1$ approximations. In section IV, the FORTRAN program that has been developed for this $\ell_1-\ell_2$ adaptive curve fitting algorithm is discussed, and numerical results are presented in an effort to illustrate how the $\ell_1$ version of this algorithm may be used most effectively.
I. Introduction

Let \( X \) be a finite set of real points and let \( f \) be a function defined on \( X \), or let data be given in tabular form. In the case of data given in tabular form, say \( \{(t_i, y_i)\}_{i=1}^{M} \), we shall let \( X = \{t_i\}_{i=1}^{M} \) and define \( f \) on \( X \) by \( f(t_i) = y_i, \; i = 1, 2, \ldots, M \), so that the functional notation may be used in what follows. Let \( a = \min\{x: x \in X\} \) and \( b = \max\{x: x \in X\} \) and for any function \( g \) defined on \( X \) denote \( \max(g(x): x \in X) \) by \( ||g||_X \). Finally, let \( N, \text{SMTH}, \) and \( \text{TOL} \) be parameters supplied by the user where \( N \) and \( \text{SMTH} \) are integer values with \( N > \text{SMTH} > -1 \) and \( \text{TOL} \) is a positive number. In this setting, the adaptive curve fitting algorithm presented here will calculate a piecewise polynomial approximation \( p \) to \( f \) and a set of points (knots) \( \{x_i\}_{i=0}^{k} \subset X \) with \( a = x_0 < x_1 < \ldots < x_k = b \) such that:

1. \( p \) restricted to \([x_{i-1}, x_i] \) is a polynomial \( p_i \in \Pi_{N-1} = \{q: q \) is a real algebraic polynomial of degree \( \leq N-1\} \),
2. \( p \) has \( \text{SMTH} \) continuous derivatives, and
3. \( ||f - p||_X \leq \text{TOL} \).

II. The Adaptive Curve Fitting Algorithm

We shall begin with a somewhat brief description of the algorithm, in an attempt to illustrate the basic logical flow. This shall be followed by an in-depth discussion of the algorithm's various components. Again, it should be mentioned that the input to the algorithm consists of the function to be approximated (in the discrete form as described above), and the values \( N, \text{SMTH}, \) and \( \text{TOL} \).

The algorithm begins by finding the largest point \( \hat{x}_1 \) in \( X \) such that:

1. \([a, \hat{x}_1] \cap X \) contains at least \( \max(2, N+1) \) points, and
(11) the best \( (\varepsilon_1 \text{ or } \varepsilon_2) \) approximation \( p_1 \in \Pi_{N-1} \) to \( f \) on \([a, x_1] \cap X\) meets the prescribed tolerance (i.e., \( \|f - p_1\|_{[a, x_1] \cap X} \leq \text{TOL} \)).

If \( x_1 = b \), then since \( p_1 \) is a piecewise polynomial satisfying (1)-(3), the algorithm is successfully terminated. If \( x_1 < b \), then the right endpoint of the first subinterval (i.e., the knot \( x_1 \)) is generally determined by "backing off" from \( x_1 \) to a point in \([a, x_1] \cap X\) selected in such a manner (to be explained later) so as to add to the stability of the algorithm.

Having thus determined the first subinterval \([a, x_1] \) and corresponding best \( (\varepsilon_1 \text{ or } \varepsilon_2) \) approximation \( p_1 \in \Pi_{N-1} \) to \( f \) on \([a, x_1] \cap X\), the algorithm goes on to determine the remainder of the piecewise polynomial approximation \( p \).

However, for the remainder of the approximation, the problem of meeting the smoothness constraint \( \text{SMTH} \) at the interior knots \( x_1, x_2, \ldots, x_{k-1} \) must be considered. Thus, the next step is to find the largest point \( x_2 \) in \( X \) such that:

(i) \([x_1, x_2] \cap X \) contains at least \( \max(2, N-\text{SMTH}) \) points, and

(ii) the best \( (\varepsilon_1 \text{ or } \varepsilon_2) \) approximation \( p_2 \in \Pi_{N-1} \) to \( f \) on \([x_1, x_2] \cap X\) subject to the constraint that \( p_2^{(j)}(x_1) = p_1^{(j)}(x_1), j = 0, 1, \ldots, \text{SMTH} \), satisfies \( \|f - p_2\|_{[x_1, x_2]} \leq \text{TOL} \).

Again, if \( x_2 = b \), \( p_1 \) and \( p_2 \) constitute a piecewise polynomial \( p \) satisfying (1)-(3) and the algorithm is successfully terminated. If \( x_2 < b \), then the knot \( x_2 \) is determined in the same manner as \( x_1 \), in order to establish the second subinterval \([x_1, x_2] \).

The algorithm continues in the same manner to its completion by finding successive subintervals \([x_2, x_3], [x_3, x_4], \ldots, [x_{k-1}, b] \), and corresponding polynomial approximations \( p_3, p_4, \ldots, p_k \in \Pi_{N-1} \) to \( f \) such that the TOL and SMTH constraints are met.

At this point, a more detailed description of the various components of the algorithm is in order, beginning with how the \( x_i \) are determined. Suppose
some knot \( x_{i-1} \) has been established. (This may be the initial knot \( x_0 = a \) or some interior knot.) The process of finding \( \hat{x}_i \) is initiated by computing the best \( (\varepsilon_1 \) or \( \varepsilon_2 \) approximation (subject to the smoothness constraint SMTH if \( x_{i-1} \) is an interior knot) to \( f \) on \( [x_{i-1}, b] \cap X \). If this approximation satisfies the prescribed tolerance TOL, then \( \hat{x}_i = b \). Should this approximation not satisfy TOL, the algorithm sets \( \hat{b} = b \). In what follows, \( \hat{b} \) shall denote the current smallest point in X such that the appropriate best approximation to \( f \) on \( [x_{i-1}, \hat{b}] \cap X \) does not satisfy TOL. Next, the appropriate best approximation to \( f \) on \( [x_{i-1}, t] \cap X \) is computed, where \( t = \inf \{ x \in X : [x_{i-1}, x] \cap X \text{ contains at least } \max(2, N-\text{SMTH}) \text{ points} \} \). If this approximation fails to satisfy TOL, the algorithm cannot meet the desired accuracy and fails. Otherwise, the algorithm sets \( \hat{a} = t \). In what follows, \( \hat{a} \) shall denote the current largest point in X such that the appropriate best approximation to \( f \) on \( [x_{i-1}, \hat{a}] \cap X \) satisfies TOL. An iterative procedure then ensues to find the largest possible \( \hat{x}_i \). The algorithm sets \( t = \inf \{ x \in X : (\hat{b} - \hat{a})/2 \leq x < b \} \). If this set is empty, the algorithm sets \( t = \sup \{ x \in X : \hat{a} < x \leq (\hat{b} - \hat{a})/2 \} \). In effect, a "halfway point" between the current values of \( \hat{a} \) and \( \hat{b} \) is being sought. If \( t = \hat{a} \), then this procedure has converged and \( \hat{x}_i = \hat{a} \). Otherwise, the appropriate best approximation to \( f \) on \( [x_{i-1}, t] \cap X \) is computed. If this approximation satisfies TOL, we have a new value for \( \hat{a} \), namely \( \hat{a} = t \). If this approximation does not satisfy TOL, then we have a new value for \( \hat{b} \), namely \( \hat{b} = t \). This process is continued, essentially drawing the values of the largest "good" candidate for \( \hat{x}_i \) (the current \( \hat{a} \)) and the smallest "bad" candidate for \( \hat{x}_i \) (the current \( \hat{b} \)) closer together. When \( \hat{b} - \hat{a} \) is less than or equal to some user definable prescribed tolerance (ETA), or when \( \hat{a} \) and \( \hat{b} \) are adjacent elements of \( X \) (which ever occurs first), the process is completed and \( \hat{a} \) is accepted as a good approximation to \( \hat{x}_i \).
Remaining in the setting just outlined, i.e., having just established an \( \tilde{x}_i \), the method of "backing off" mentioned earlier used to determine the knot location \( x_i \) (and thereby establish the subinterval \([x_{i-1}, x_i]\)) will now be explained. (It should be noted here that if \( \text{SMTH} = -1 \) or 0, the following procedure is bypassed, and the knot location \( x_i \) is simply the point \( \tilde{x}_i \).)

First, the error function \( f(x) - p_i(x) \) is examined for those points \( \xi_1, \xi_2, \ldots, \xi_k \) in \([x_{i-1}, \tilde{x}_i] \cap X\) at which relative extrema occur. That is, \( \xi_{\nu} \) is such that \( |f(\xi_{\nu}) - p_i(\xi_{\nu})| \geq |f(x) - p_i(x)| \) for \( x = \max\{t \in X: t < \xi_{\nu}\} \) and \( x = \min\{t \in X: t > \xi_{\nu}\} \). (Note that \( p_i \) is the appropriate best approximation to \( f \) on \([x_{i-1}, \tilde{x}_i]\) \( \cap X \) computed in the previous process.) In a process to be subsequently described, one of the \( \xi_{\nu} \) is chosen for the knot location \( x_i \). The motivation for choosing \( x_i \) in this manner is as follows: If \( f \) were differentiable and \( X = [a, b] \), then choosing the knot \( x_i \) to be an interior extreme point of \( f - p_i \) on \([x_{i-1}, \tilde{x}_i]\) would ensure that \( f'(x_i) = p'_i(x_i) \). That is, the slope of the function \( f \) and its polynomial approximation on \([x_{i-1}, \tilde{x}_i]\), \( p_i \), would match at \( x_i \). This becomes advantageous when joining the next polynomial piece of the approximation to \( p_i \) at \( x_i \) when the approximation is required to be at least continuously differentiable. When the next polynomial piece, \( p_{i+1} \), is smoothly joined to \( p_i \) at \( x_i \), it follows that the function \( f \) and both of the polynomial pieces \( p_i \) and \( p_{i+1} \) shall have the same slope about the knot \( x_i \). If we simply joined the polynomial pieces \( p_i \) and \( p_{i+1} \) at \( \tilde{x}_i \), this is generally not the case and severe oscillatory problems tend to set in. This process of "backing off" from \( \tilde{x}_i \) to a smaller \( x_i \in \{\xi_{\nu}\}_{\nu=1}^2 \) contributes significantly to the stability of the algorithm.

In determining which \( \xi_{\nu} \) should be chosen for the knot location \( x_i \), steps must be taken to alleviate the fact that \( f \) (in the discrete form input to the algorithm) is in fact not differentiable. The values \( f'_i(\xi_1), \ldots, f'_k(\xi_2) \) are
calculated where $\hat{\xi}_v'$ is the derivative of the quadratic interpolation of $f$ centered at $\xi_v$. Then, $x_i$ is chosen to be the largest $\xi_v$ such that $
abla \hat{\xi}_v(\xi_v) - p_i'(\xi_v)$ is less than some user definable prescribed tolerance (EPS). If there does not exist such a $\xi_v$, then $x_i$ is chosen to be the largest $\xi_v$ at which $\nabla \hat{\xi}_v(\xi_v) - p_i'(\xi_v)$ is a minimum. (It should be noted here that in the implementation of the algorithm, it is generally not the case that all of the relative extreme points of $f - p_i$ on $(x_{i-1}, \hat{x}_i) \cap X$ are considered, but rather only the largest N-SMTH-1 of them.

This "backing off" procedure has proven to be invaluable for dampening oscillations for those approximation in which SMTH > 0. However, if no continuity is required of the approximation (SMTH = -1) or if the approximation is simply required to be continuous (SMTH = 0), this procedure serves no purpose and is overridden. (Again we note that in these instances $x_i = \hat{x}_i$, $i = 1, 2, \ldots$)

Another aspect of the algorithm that merits attention is the process of determining the last subinterval of the approximation. Suppose that the subintervals $[a, x_1], [x_1, x_2], \ldots, [x_{i-2}, x_{i-1}]$ and corresponding polynomial approximations $p_1, p_2, \ldots, p_{i-1}$ if $f$ have been determined. If $[x_{i-1}, b] \cap X$ contains at least max(2, N-SMTH) points, the algorithm goes on to determine $\hat{x}_i$ precisely as outlined earlier. However, if $[x_{i-1}, b] \cap X$ contains fewer than max(2, N-SMTH) points, a process is initiated that, if successful, will determine the last subinterval of the approximation. In this instance, the knot $x_{i-1}$ is replaced with some $\hat{x}_{i-1}$ in $(x_{i-2}, x_{i-1}) \cap X$ chosen such that $[\hat{x}_{i-1}, b] \cap X$ will contain at least max(2, N-SMTH) points. Specifically, the algorithm chooses $\hat{x}_{i-1}$ to be that point in $X$ closest to $(b - x_{i-2})/2$ such that:

1. $[\hat{x}_{i-1}, b] \cap X$ contains at least max(2, N-SMTH) points, and
2. the best ($\xi_1$ or $\xi_2$) approximation $p_i \in P_{N-1}$ to $f$ on $[\hat{x}_{i-1}, b] \cap X$ subject to the constraint that $p_j^{(j)}(\hat{x}_{i-1}) = p_j^{(j)}(\hat{x}_{i-1})$, $j=0,1,\ldots,\text{SMTH}$, satisfies $\|f - p_i\|_{[\hat{x}_{i-1}, b] \cap X} \leq \text{TOL}$. 
It is most often the case that such a point $\hat{x}_{i-1}$ is readily available, in which case the algorithm is successfully terminated. (Here, the last two subintervals and corresponding polynomial approximations to $f$ are, respectively, $[x_{i-2}, \hat{x}_{i-1}]$ and $p_{i-1}$.) If a satisfactory $\hat{x}_{i-1}$ cannot be found, the algorithm is terminated and an appropriate error message is generated.

The bulk of the discussion thus far has been centered around the manner in which the knots $a, x_0, x_1, \ldots, x_k = b$ of the piecewise polynomial approximation $p$ to $f$ are determined. This is due to the fact that the algorithm's adaptive nature, which allows it to calculate the total number and location of knots needed (as opposed to most data fitting spline techniques that require that the total number and location of the knots be specified in advance), is one of the main features of the algorithm. We shall now direct attention to the actual computation of the polynomial pieces $p_1, p_2, \ldots, p_k$ that comprise the approximation $p$ to $f$ on $X$, this having thus far been taken somewhat for granted.

Suppose that some knot $x_{i-1}$ has been established, and the procedure that determines $\hat{x}_i$ is currently in progress. Suppose further that some point $t \in [x_{i-1}, b] \cap X$ is being considered as a candidate for $x_i$. Thus, the next step is to determine the appropriate approximation to $f$ on $[x_{i-1}, t] \cap X$ (which we shall denote by $\hat{p}_i$). Let $[x_{i-1}, t] \cap X = \{t_1 \leq t_2 \leq \ldots \leq t_a\}$. If $t_1 = a$, or if $SMTH = -1$, there are no continuity constraints involved and the following over-determined system is constructed:

$$
\begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
1 & (t_2 - t_1) & (t_2 - t_1)^2 & \ldots & (t_2 - t_1)^{N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & (t_a - t_1) & (t_a - t_1) & \ldots & (t_a - t_1)^{N-1}
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
\vdots \\
c_N
\end{bmatrix} =
\begin{bmatrix}
f(t_1) \\
f(t_2) \\
\vdots \\
f(t_a)
\end{bmatrix}
$$
The solution of the above system (in the $\ell_1$ or $\ell_2$ sense) yields the coefficients \(\{c_v\}_{v=1}^N\) of a polynomial in the form \(\hat{p}_i(x) = \sum_{v=1}^N c_v(x - t_1)^{v-1}\). If \(t_1\) is an interior knot (in which case a previous polynomial \(p_{i-1}\) has been established) and \(\text{SMTH} = k > 1\), the smoothness constraint at \(t_1\) must be considered and the following over-determined system is constructed:

\[
\begin{bmatrix}
(t_2 - t_1)^k & \cdots & (t_2 - t_1)^{N-1} \\
\vdots & \ddots & \vdots \\
(t_a - t_1)^k & \cdots & (t_a - t_1)^{N-1}
\end{bmatrix}
\begin{bmatrix}
c_k + 2 \\
\vdots \\
c_N
\end{bmatrix}
= \begin{bmatrix}
f(t_2) = \sum_{j=0}^k \frac{p_{i-1}(j)(t_1)}{j!}(t_2 - t_1)^j \\
\vdots \\
f(t_a) = \sum_{j=0}^k \frac{p_{i-1}(j)(t_1)}{j!}(t_a - t_1)^j
\end{bmatrix}
\]

The solution of the above system (in the $\ell_1$ or $\ell_2$ sense) yields the coefficients \(\{c_v\}_{v=k+2}^N\) and a polynomial of the form

\[
\hat{p}_i(x) = \sum_{j=0}^k \frac{p_{i-1}(j)(t_1)}{j!}(x - t_1)^j + \sum_{v=k+2}^N c_v(x - t_1)^{v-1}.
\]

If the error of approximation of \(f\) by \(\hat{p}_i\) on \([x_{i-1}, t] \cap X\) is greater than TOL, a smaller candidate for \(\hat{\chi}_i\) (and corresponding polynomial approximation) is sought. If \(\hat{p}_i\) satisfies TOL, the next step depends on \(t\). If the process used for finding \(\hat{\chi}_i\) has not converged yet, a larger candidate for \(\hat{\chi}_i\) (and corresponding polynomial approximation) is sought. If the process has converged, \(t\) is accepted as \(\hat{\chi}_i\) and the polynomial approximation \(\hat{p}_i\) is accepted as \(p_i\).

III. The $\ell_1$-Approximation Problem

The general $\ell_1$ (linear) approximation problem can be stated as follows:

Suppose we are given a real-valued function \(f\) (to be approximated) defined on a finite set of real points \(X = \{x_1, x_2, \ldots, x_m\}\). Let \(\{\phi_j\}_{j=1}^n\) (where \(n < m\)) be a set of real-valued functions defined on \(X\). Then, for any set of real numbers
For the general $\ell_1$-approximation problem outlined earlier, let
\[ \phi_{j,i} = \phi_j(x_i), \quad f_i = f(x_i), \]
and define non-negative variables $u_i, v_i, a_j$ and $b_j$ by putting
\[ f_i - \sum_{j=1}^{n} c_j \phi_{j,i} = u_i - v_i, \quad i = 1, 2, \ldots, m, \] and $c_j = a_j - b_j$ for
j = 1, 2, ..., n. Then a best $\ell_1$ approximation to $f$ on $X$ corresponds to an optimal solution to the linear programming problem:

\[ \text{Minimize } z = \sum_{i=1}^{m} (u_i + v_i) \]

subject to \( f_i = \sum_{j=1}^{n} (a_j - b_j)\phi_{j,i} + u_i - v_i \quad i = 1, 2, ..., m \)

and \( a_j, b_j, u_i, v_i \geq 0. \)

The formulation of the above correspondence appears in Barrodale and Roberts [6].

The main modifications to the standard form of the simplex method as presented in [3] result in an algorithm that exploits the above linear programming version of the $\ell_1$-approximation problem in such a manner that reduces the number of iterations required to arrive at an optimal solution (i.e., a best $\ell_1$ approximation). Basically, the algorithm allows for the passage through several neighboring simplex vertices (basic feasible solutions, extreme-point solutions) in a single iteration, thus reducing the number of time-consuming simplex transformations needed to arrive at an optimal solution.

To begin our description of the algorithm, we note that an initial basic feasible solution to the linear programming problem (2) is immediately available. That is, if we denote the columns of the simplex tableau corresponding to (2)-(4) by $A_j, B_j, j = 1, 2, ..., n$, and $U_i, V_i, i = 1, 2, ..., m$, an initial basis is provided by $U_1, U_2, ..., U_m$ (provided that each $f_i$ is non-negative). Below is the resulting initial simplex tableau:
Here, the column to the left of the tableau is used to indicate the basis, which initially consists of the vectors $U_1$. The upper portion of the tableau corresponds to the constraints (3), and the bottom (objective) row is a representation of the objective function (2). That is, if we consider (2) in form

$$ \sum_{i=1}^{m} (-u_i - v_i) + z = 0 $$

then it follows that

$$ \left( \sum_{i=1}^{m} \sum_{j=1}^{n} (a_j - b_j) \phi_{j,i} \right) + \sum_{i=1}^{m} v_i - \sum_{i=1}^{m} f_i \right) - \sum_{i=1}^{m} v_i + z = 0 $$

or

$$ \sum_{i=1}^{m} a_1 \phi_{1,i} + \sum_{i=1}^{m} a_2 \phi_{2,i} + ... + \sum_{i=1}^{m} a_n \phi_{n,i} - \sum_{i=1}^{m} b_1 \phi_{1,i} - \sum_{i=1}^{m} b_2 \phi_{2,i} - ... - \sum_{i=1}^{m} b_n \phi_{n,i} - 2v_1 - 2v_2 - ... - 2v_m + z = \sum_{i=1}^{m} f_i $$

which is equivalent to

$$ \left( \sum_{i=1}^{m} \phi_{1,i} \right) a_1 + \left( \sum_{i=1}^{m} \phi_{2,i} \right) a_2 + ... + \left( \sum_{i=1}^{m} \phi_{n,i} \right) a_n - \left( \sum_{i=1}^{m} \phi_{1,i} \right) b_1 - \left( \sum_{i=1}^{m} \phi_{2,i} \right) b_2 - ... $$

$$ \left( \sum_{i=1}^{m} \phi_{n,i} \right) b_n - 2v_1 - 2v_2 - ... - 2v_m + z = \sum_{i=1}^{m} f_i $$

Thus we see that the objective row of the initial simplex tableau is a representation of the objective function as shown in (5). If any of the $f_i$ are negative, the sign of the corresponding row is changed, and $U_i$ is replaced in the basis by $V_i$. Note also that $A_j = -B_j$, $j = 1, 2, ..., n$, $U_i = -V_i$, $i = 1, 2, ..., m$, the sum of the entries in the objective row corresponding to each pair of $A_j$ and $B_j$ is 0, and the sum of the entries in the objective row
corresponding to each pair of $U_i$ and $V_i$ is -2. This is utilized by the FORTRAN program of this algorithm which economizes on storage by using a condensed form of the simplex tableau (see [4]).

The algorithm consists of two distinct stages. Stage 1, which begins upon the establishment of the initial simplex tableau, restricts the choice of the pivotal columns during the first $n$ iterations to the vectors $A_j$ and $B_j$. The vector to enter the basis is chosen to be that which has the largest non-negative entry in the objective row. (We note two things here: Choosing a vector to enter the basis is equivalent to selecting a variable to enter the set of basic variables. Also, with regard to the choice of vectors to enter the basis (i.e., variables to enter the set of basic variables) in Stage 1, choosing from among those with non-negative entries in the objective row clearly serves to decrease the value of the objective function, given the formulation of the objective row.) The vector to leave the basis (or equivalently, the variable to enter the set of non-basic variables) in Stage 1 is chosen from among the basic vectors $U_i$ (and $V_i$) by selecting that vector which causes the maximum reduction in the objective function.

At the end of Stage 1, the resulting simplex tableau represents an approximation that interpolates at least $K$ of the data points, where $K$ is the number of vectors $A_j$ or $B_j$ in the basis. (This results from the fact that $K$ of the vectors $U_i$ (or $V_i$) have been removed from the basis.) If the approximation determined by the simplex tableau of the end of Stage 1 interpolates more than $K$ points, this is an indication that the tableau is degenerate. (This does not cause any problems in practice, however.)

During Stage 2, the non-basic $U_i$ and $V_i$ vectors are interchanged with the basic $U_i$ and $V_i$. The basic $A_j$ and $B_j$ vectors are not allowed to leave the basis during this stage. At each iteration, the vector chosen to enter the
basis is that (among the non-basic \(U_i\) and \(V_i\)) with the most positive entry in the objective row, and the vector to leave the basis is again chosen to be that (among the basic \(U_i\) and \(V_i\)) which causes the maximum reduction in the objective function. The algorithm terminates when all of the entries in the objective row are non-positive. (Although this normally occurs in Stage 2, should it occur in Stage 1, the second stage becomes unnecessary.)

Each vector that enters the basis during Stage 2 determines a data point to be dropped from the interpolating set while the corresponding vector leaving the basis determines a new point of interpolation. Thus assuming nondegeneracy, the number of data points interpolated by the approximation resulting from Stage 1 is preserved. The final simplex tableau at the end of Stage 2 may contain basic vectors \(A_j\) or \(B_j\) that have negative values associated with them, thus resulting in an infeasible solution. This solution is made feasible (and hence optimal) by interchanging such basic vectors \(A_j\) (or \(B_j\)) with the corresponding non-basic vectors \(B_j\) (or \(A_j\)). Given the nature of the \(A_j\) and \(B_j\) vectors, this is a simple matter of changing the sign of the appropriate row.

It is important to note here the main modification to the standard simplex method that this algorithm introduces. For any given iteration (in both Stage 1 and Stage 2), upon having established the vector to enter the basis (i.e., the pivot column), the vector chosen to leave the basis (i.e., the pivot row) is not in general that vector which the standard simplex method dictates should be removed, but rather is that vector (from among the basic \(U_i\) and \(V_i\) vectors) which causes the maximum reduction in the objective function. Geometrically, the simplex transformation which then follows is equivalent to a passage through several neighboring extreme points of the feasible region. The standard procedure for determining the vector to leave the basis (see the Appendix or [7]) is modified as follows:
First, a tentative pivot row is determined using the standard procedure. If subtracting twice the value of the resulting pivot from the entry in the objective row of the pivot column yields a nonpositive result, then a normal simplex transformation is performed on the tableau. Otherwise, twice the pivot row is subtracted from the objective row, the pivot row is multiplied by -1, and the vector $U_i$ (or $V_i$) in the basis corresponding to the pivot row is replaced in the basis by the corresponding $V_i$ (or $U_i$). This serves to decrease the value of the objective function and change the sign of the pivot. The standard procedure for finding a pivot row is applied again, resulting in a new tentative pivot. The previous procedure is applied until a pivot is chosen which cannot be rejected (i.e., a pivot such that twice its value subtracted from the entry in the objective row of the pivot column is nonpositive). A normal simplex transformation is then performed with this pivot.

Remark

In practice, errors due to roundoff can result in a non-basic vector being chosen as a pivot column, even though there are no positive entries in the vector (and hence no candidate for a pivot). This is usually due to a loss of significance that can occur during simplex transformations on a tableau containing entries differing greatly in magnitude. In the FORTRAN version of this algorithm [4], a small positive tolerance (TOLER) is defined below which the magnitude of any quantity is considered to be zero. The algorithm is terminated prematurely if a pivot column is encountered that contains no candidates for a pivot that exceeds the value TOLER. This occurrence is rare, and generally indicates that the current solution is close to the actual optimal solution. Thus, the program outputs the current solution as a reasonable approximation to the actual optimal solution in these instances.
IV. Numerical Results

This $\varepsilon_1$-$\varepsilon_2$ adaptive curve fitting algorithm has been implemented as a FORTRAN program (standard ASCII code) and has been tested on Colorado State University's CDC 6400 and CDC Cyber 172. A fully documented listing of this package and complete discussion of the program's major components shall appear in a future paper. Following is a general discussion of input/output as well as some numerical results.

Aside from the function (or data) $f$ to be approximated (in discrete form), those values that must be supplied by the user include: IOPT, an integer value (1 or 2) specifying the choice of approximation (best $\varepsilon_1$ or best $\varepsilon_2$); $N$, the number of coefficients of each polynomial piece; and TOL, the desired error tolerance. In circumstances where relatively few data points are available, the program fills in the gaps between the data points by discretizing a piecewise linear interpolation of the original data. Should this be necessary, the number of additional "data points" to be inserted in this manner (NPTS) should also be supplied by the user.

The output consists of the coefficients of the polynomial pieces (where the polynomials are in standard form), the knot locations of the piecewise polynomial approximation, and the error of approximation for each polynomial piece. Also, all of the appropriate information about the piecewise polynomial approximation is stored so as to allow the user to evaluate the approximation at any point (in the interval of approximation) using an available function subprogram.

Two other values used in the program that are not input but are user definable are the values ETA and EPS. Recall (section II) that in the iterative procedure used to determine the location of a particular $\tilde{x}_i$, the values $\tilde{x}$ (the current largest "good" candidate for $\tilde{x}_i$) and $\bar{x}$ (the current smallest "bad"
candidate for \( x_i \) are drawn together, and when \( b - a' \) is less than or equal to the user prescribed tolerance ETA, or when \( b \) and \( a' \) become adjacent elements of the data set (whichever occurs first), \( a' \) is accepted as the location for \( x_i \).

In the examples that follow, ETA was set to be less than or equal to the mesh size of the data, thus ensuring that the iterative procedure will continue until \( a' \) and \( b' \) are in fact adjacent and the resulting \( x_i \) is as large as possible.

In the "backing off" procedure used to determine the actual knot locations \( x_i \) (see section II), \( x_i \) is chosen to be the largest \( \xi_v \) at which \( |\tilde{f}'(\xi_v) - p'_i(\xi_v)| \) is less than the user-prescribed tolerance EPS (where the \( \xi_v \) are the points in \( (x_{i-1}, x_i] \) at which the relative extrema of the error function \( f(x) - p_i(x) \) occur). Empirical evidence indicates that the value of .05 for EPS yields desirable results.

We now turn to some examples. Using the adaptive curve fitting program with best \( z_1 \) approximations (\( \text{IOPT} = 1 \)), the function \( \sqrt{x} \) on \([0, 2]\) was approximated on 201 equally spaced points with \( N = 6, \text{SMTH} = 2, \) and \( \text{TOL} = .01 \). Since this function is difficult to approximate by polynomials near \( x = 0 \), the algorithm's ability to automatically decrease the length of the subintervals near \( x = 0 \) and then recover by lengthening them away from the origin is illustrated. The results appear below:

\[
\begin{array}{cccccccc}
\text{Knot locations} & 0.0 & .06 & .18 & .41 & .84 & 1.49 & 2.0 \\
\text{Subintervals} & 7 pts. & 13 pts. & 24 pts. & 44 pts. & 66 pts. & 52 pts.
\end{array}
\]

(It should be noted here that the purpose of the above example is simply to illustrate the adaptive nature of the program and that in general, the \( z_1 \) version of this adaptive curve fitting algorithm is not particularly suited for approximating precise mathematical functions.)
We now turn our attention to some examples that better illustrate how the $L_1$ option of this adaptive curve fitting program is most effectively utilized. As one might suspect, given the nature of $L_1$ approximation, the piecewise polynomial $L_1$ approximations have shown to be very effective for approximating on data sets that contain points that are very inaccurate with respect to the overall accuracy of the data. For the example in figure 1, the function $|\sin(x)|$ on $[0, 2\pi]$ was discretized (101 points) and "noise" was generated using a random number generator. (The number of "bad" points and a bound on the deviation was set, but the location and magnitude [within the bound] of the noise was random.) The pertinent input values appear at the top of the graph and the execution time (in CPU seconds) appears below. (In all the examples that follow, the data points are denoted by X's.) Note that the noise has virtually no ill effect on the piecewise polynomial $L_1$ approximation. That is, the curve essentially ignores the "bad" points and passes through the "good" points.

In figures 2 and 3, the function $e^x$ on $[0, 2]$ was discretized (101 points), noise was introduced in the manner just described, and the resulting data set was approximated on by the adaptive curve fitting program, exercising both the $L_1$ and $L_2$ (least-squares) option. In so much as least squares approximations have the tendency to dampen the effect of randomly distributed noise, we might expect (and in fact do achieve) desirable results in both instances. However, note that in figure 3 the approximation was visibly affected by the noise in the interval (.4, .8) and by the "bad" point near 1.0, whereas in figure 2, the $L_1$ approximation again is unaffected by the noise.

We have also had some success in fitting smaller data sets with piecewise polynomial $L_1$ approximations. The experimental data in figures 4 and 5 involves the bitumen yield and gas and oil yield (as a function of time) from oil shale heated to a constant temperature. Since relatively few data points were
available in each data sample (13-20 points), we filled in the gaps between the
data points with 200 equally spaced points by the manner previously described
(input NPTS = 200). In the example in figure 4, TOL was set to a rather
liberal 5.0 so as to allow the approximation the freedom of passing through the
scattering of data points, rather than forcing an interpolatory type fit that
would have resulted if the tolerance were set smaller. For the example in
figure 5, TOL was set to 2.0, which still gives the algorithm a great deal of
freedom but results in a fit that follows the data more closely.

Remark

It should be noted here that the $\ell_1$ version of this adaptive curve
fitting algorithm should be used with some discretion. For data containing
noise (particularly large data sets), it has proved to be a very effective
approximating scheme. In general, however, it should not be used for approxi-
mating precise mathematical functions, or for approximating "good" data sets
for which an interpolatory type fit would be desirable.

Also, a future paper is anticipated in which all of the adaptive curve
fitting programs that have been developed thus far (restricted range, $\ell_\infty$, $\ell_1$, and $\ell_2$) shall be compared in an effort to offer some insight into how they
may be used most efficiently.
**Figure 1**

L1 ABS(SIN(X)) (WITH NOISE)

\[ N = 6, \text{SWTH} = 1, TOL = .025 \text{ knots are indicated by X.} \]
L1

EXP(X) (WITH NOISE)

N = 6, SM/N = 2, TOL = 1.000 Knots are indicated by X.

PIECEWISE POLYNOMIAL APPROX. USING DISCRETE L1 APPROX. OPERATOR.

[cpu time: .857 secs.]

Figure 2
L2

$\exp(x)$ (WITH NOISE)

$N = 7$, $\delta YN = 2$, $\text{TOL} = 1.000$ Knots are indicated by X.

PIECEWISE POLYNOMIAL APPROX. USING (DISCRETE) L2 APPROX. OPERATOR.

[cpu time: .388 secs.]

Figure 3
75.0 GAL/TON TEMP = 425 GAS + OIL
N = 6, SIGN = 2, TL = 5,000 Knots are indicated by X.

PIECEWISE POLYNOMIAL APPROX. USING (DISCRETE) L1 APPROX. OPERATOR.

[cpu time: 2.126 secs.]

Figure 4
Figure 5

PIECEWISE POLYNOMIAL APPROX. USING DISCRETE L1 APPROX. OPERATOR.

[cpu time: 10.12 secs.]

26.7 GAL/TON TEMP = 475 BITUMEN
N = 6, SPIN = 2, TOL = 2.000 Knots are indicated by X.
REFERENCES


APPENDIX

A. The General Linear Programming Problem

The general linear programming problem may be stated as follows:

Optimize (i.e., maximize or minimize) the linear form

\[ z = c_1x_1 + c_2x_2 + \ldots + c_nx_n \]  

subject to the constraints

\[ x_j = 0, \ j = 1, 2, \ldots, n \]

and

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq (\geq) \ b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq (\geq) \ b_2 \]
\[ \vdots \]
\[ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq (\geq) \ b_m \]

where the \( a_{ij}, b_i \) and \( c_j \) are given constants.

The linear form (A.1) to be optimized is called the objective function.

Here we assume that at least one coefficient \( a_{ij} \) is nonzero in each row and that every variable appears in some nontrivial inequality with a nonzero coefficient. We make no special assumptions about the \( b_i \), they may be positive, negative, or zero. It should also be pointed out here that the minimum of an objective function occurs at the same set of values as the maximum of the negative of that objective function. Thus, regardless of the constraints, the problem of minimizing \( z = k_1x_1 + k_2x_2 + \ldots + k_nx_n \) is equivalent to maximizing \( -z = -k_1x_1 - k_2x_2 - \ldots - k_nx_n \). Also, any inequality

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \]

is equivalent to

\[ -a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n \leq -b_1. \]

A feasible solution to the linear programming problem is a vector \( X = (x_1, x_2, \ldots, x_n) \) that satisfies conditions (A.2) and (A.3). The set of
all feasible solutions to the linear programming problem is a convex region in n-dimensional Euclidean space and is referred to as the feasible region, or region of feasibility. The feasible region can either be void (in which case no solution to the problem exists), a convex polyhedron, or a convex region which is unbounded in some direction.

**Example 1**

Maximize \( z = 3x + 2y \)

Subject to

\[
\begin{align*}
&x \geq 0, y \geq 0 \\
&x + y \leq 7 \\
&x + 2y \leq 10 \\
&2x + y \leq 12
\end{align*}
\]

**Example 2**

Maximize \( z = 3x + 5y \)

Subject to

\[
\begin{align*}
&x \geq 0, y \geq 0 \\
&x + 3y \geq 9 \\
&x + y \geq 5 \\
&2x + y \geq 6
\end{align*}
\]

By introducing non-negative slack variables \( x_{n+1}, x_{n+2}, \ldots, x_{n+m} \), we can express the problem (A.1)-(A.3) in the equivalent form:

Optimize (i.e., maximize or minimize) the linear form

\[
(A.4) \quad z = c_1x_1 + c_2x_2 + \ldots + c_nx_n + 0x_{n+1} + \ldots + 0x_{n+m}
\]

subject to the constraints

\[
(A.5) \quad x_j = 0, \quad j = 1, 2, \ldots, n + m
\]
and

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + (-) x_{n+1} = b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + (-) x_{n+2} = b_2 \]
\[ \vdots \]
\[ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + (-) x_{n+m} = b_m \]  

(A.6)

A basic solution to (A.6) is a solution obtained by setting \( n \) variables equal to zero and solving for the remaining \( m \) variables (provided the solution is unique). The \( m \) variables are called basic variables; the \( n \) that are pre-specified as zero are called non-basic variables. A basic feasible solution is a basic solution which also satisfies (A.5); i.e., all the basic variables are non-negative. An optimal feasible solution is a feasible solution which also maximizes or minimizes (A.4).

With the assumption that a particular linear programming problem possesses an optimal solution, the following holds:

1. There is an extreme point (corner point) of the feasible region at which the objective function takes on its maximum or minimum.
2. Every basic feasible solution corresponds to an extreme point of the feasible region.

From the above, we see that it is only necessary to examine extreme-point solutions (i.e., basic feasible solutions). Since there are at most \( \binom{n}{m} \) of them, we have an upper bound to the number of possible solutions to the problem (A.4)-(A.6). However, for large \( n \) and \( m \), evaluating all the possible solutions to find one that maximizes or minimizes the objective function is an unreasonable way to proceed. The simplex method, devised by G. B. Dantzig, is a computational scheme that selects, in an orderly fashion, a small subset of the possible solutions that converges to an optimal solution.

This algorithm is a method for moving from one extreme point to another in such a way that the objective function is always improved, or at the least,
the same. In a finite number of steps, (usually between \( m \) and \( 2m \)) an optimal feasible solution (if one exists) is found. The simplex method also indicates the existence of alternate optimal solutions, empty feasible regions, and unbounded solutions. It is an extremely effective tool for solving any linear programming problem. It should be noted, however, that the proof of its effectiveness is a result of empirical evidence, rather than underlying theory.

B. The Simplex Method

It should be noted at the outset that the manners in which the standard simplex method can be described are as diverse as the forms that linear programming problems can take on. In so much as most elementary treatises on the simplex method deal with the "general" linear programming problem:

Maximize
\[
z = c_1x_1 + c_2x_2 + \ldots + c_nx_n \tag{B.1}
\]
Subject to
\[
x_j \geq 0, \ j = 1, 2, \ldots, n \tag{B.2}
\]
and
\[
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\
\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \tag{B.3}
\]
or, introducing non-negative slack variables, the equality form of the general problem:

Maximize
\[
z = c_1x_1 + c_2x_2 + \ldots + c_nx_n + 0x_{n+1} + \ldots + 0x_m \tag{B.4}
\]
Subject to
\[
x_j \geq 0, \ j = 1, 2, \ldots, m+n \tag{B.5}
\]
and
\[
a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_{n+1} = b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + x_{n+2} = b_2 \tag{B.6}
\]
\[
\vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + x_{n+m} = b_m
\]
we shall proceed along these lines. (Note that since any linear programming
problem can be expressed in the above form, it is in some sense a general
form.)

We begin by illustrating the simplex method with a specific linear
programming problem, this to be followed by a general description applicable
to any problem of the form (B.4)-(B.6). Consider the problem in example 1 of
Appendix A. By introducing slack variables, r, s, and t, this problem can be
expressed as:

Maximize \( z = 3x + 2y + 0r + 0s + 0t \) \( (B.7) \)
Subject to \( x \geq 0, y \geq 0, r \geq 0, s \geq 0, t \geq 0 \) \( (B.8) \)
and \( x + y + r = 7 \) \( (B.9) \)
\( x + 2y + s = 10 \)
\( 2x + y + t = 12 \).

Note that a basic feasible solution is immediately available. That is, if
we let \( x = y = 0 \), then it follows that \( r = 7, s = 10 \) and \( t = 12 \). This can be
represented in tableau form as

\[
\begin{array}{cccccc}
& x & y & r & s & t \\
1 & 1 & 1 & 0 & 0 & 0 & 7 \\
1 & 2 & 0 & 1 & 0 & 0 & 10 \\
2 & 1 & 0 & 0 & 1 & 0 & 12 \\
-3 & -2 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

\( (B.10) \)

Here, the column to the left of the tableau is used to indicate the set of
basic variables (or basis) which in this instance consists of the variables
\( r, s \) and \( t \). The upper portion of the tableau represents the constraints \( (B.9) \)
and the lower portion of the tableau, known as the objective row, is a repre-
sentation of the objective function \( (B.7) \) expressed in the form: \( -3x - 2y + z = 0 \)
(where we need not keep a column for the \( z \)).
Hence, at this stage we have our first basic feasible solution: \( x = 0, y = 0, r = 7, s = 10, \) and \( t = 12. \) The entry in the lower right-hand corner of (B.10) indicates that \( z = 0. \) Geometrically, this basic feasible solution corresponds to the extreme point of the feasible region (see example 1, Appendix A) occurring at the origin. The simplex method consists of employing Gauss-Jordan elimination to proceed to another basic feasible solution (i.e., to another corner point of the feasible region) in such a way that the value of the objective function \( (z) \) shall be increased (or at worst remain the same.)

We see that given the way in which the objective function is expressed in the objective row, \( z \) can be increased by increasing any variable with a negative coefficient in the objective row. Those variables having negative coefficients in the objective row are \( x \) and \( y. \) Since the greatest increase in \( z \) will clearly result from increasing that variable with the most negative coefficient, we choose to increase the variable \( x. \) The corresponding column of the tableau, in this instance the first, is called the pivot column.

We now consider to what extent can the variable \( x \) be increased (we wish to increase \( x \) as much as possible without violating any of the constraints). If we consider the equations in (B.9), or equivalently the tableau given by (B.10):

\[
\begin{align*}
    r &= 7 - x - y = 0 \\
    s &= 10 - x - 2y = 0 \\
    t &= 12 - 2x - 2y = 0.
\end{align*}
\]

We see that (keeping in mind that \( y \) is presently equal to 0) if we wish to increase \( x, \) we can not allow it to be greater than 6. This is the smallest of the ratios \( \eta_1 = \frac{7}{1}, \eta_2 = \frac{10}{1}, \eta_3 = \frac{12}{2}. \) If we increase \( x \) beyond 6, then the last of the above equations dictates that \( t \) will become negative, which is not permitted by (B.8). We find the \( \eta \)-ratios by dividing the entries in the
right-hand column of the tableau by the corresponding positive entries in
the pivot column. The pivot row is that which yields the smallest non-
negative ratio; in this instance, the third row of tableau (B.10).

The entry in the tableau located in both the pivot column and the pivot
row (circled in (B.10)) is called the pivot. The first "simplex transformation"
consists of applying Gauss-Jordan elimination to the tableau in (B.10) using
the given pivot. The resulting tableau appears below:

\[
\begin{array}{cccccc}
 x & y & r & s & t \\
 r & 0 & 1/2 & 1 & 0 & -1/2 & 1 \\
s & 0 & 3/2 & 0 & 1 & -1/2 & 4 \\
x & 1 & 1/2 & 0 & 0 & 1/2 & 6 \\
 0 & -1/2 & 0 & 0 & 3/2 & 18 \\
\end{array}
\]

(B.11)

Let us reflect upon what has transpired. The variable \( x \), previously equal
to 0, has been increased to 6. Thus \( x \) has become a basic variable, or
equivalently, has "entered the basis". The variable \( t \), previously equal to
12, has been decreased to 0. Thus \( t \) has become a non-basic variable, or
equivalently, has "left the basis". The new tableau represents the basic
feasible solution: \( x = 6, y = 0, r = 1, s = 4 \) and \( t = 0 \). The entry in the
lower right-hand corner of (B.11) indicates that \( z \) has been increased to 18.
Geometrically, the above simplex transformation corresponds to a transfer to
an adjacent extreme point of the feasible region, and the basic feasible
solution yielded by the tableau (B.11) corresponds to that extreme point of
the feasible region located at the point (6, 0).

Since one of the entries in the object row of (B.11) is negative, it
follows that \( z \) can be increased further. The pivot column for the next
simplex transformation shall be the second column. (This indicates that \( y \)
will enter the basis.) Upon examining the \( e \)-ratios \( \theta_1 = \frac{1}{2} = 2, \theta_2 = \frac{4}{3} = \frac{8}{3} \).
and \( \theta_3 = \frac{6}{2} = 12 \) we see that the pivot row is the first, corresponding to the smallest \( \theta \)-ratio. (This indicates that \( r \) will leave the basis.) Performing a second simplex transformation (using the pivot circled in (8.11)) yields:

\[
\begin{array}{ccccccc}
 & x & y & r & s & t & \\
y & 0 & 1 & 2 & 0 & -1 & 2 \\
s & 0 & 0 & -3 & 1 & 1 & 2 \\
x & 1 & 0 & -1 & 0 & 1 & 5 \\
0 & 0 & 1 & 0 & 1 & 19 \\
\end{array}
\]  

(B.12)

Since there are no negative entries in the objective row, we have arrived at the optimal solution. The final tableau represents the optimal feasible solution: \( x = 5, y = 2, r = 0, s = 2, \) and \( t = 0 \). The maximal value of the objective function is 19. This solution corresponds to the extreme point of feasible region located at the point (5, 2).

At this point, we summarize the simplex method as illustrated in the previous example. If we consider the general linear programming problem as posed in (B.4)-(B.6), we see that an initial simplex tableau is given by:

\[
\begin{array}{ccccccccccc}
 & x_1 & x_2 & \cdots & x_j & \cdots & x_n & x_{n+1} & x_{n+2} & \cdots & x_{n+m} \\
x_{n+1} & a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\
x_{n+2} & a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
x_{n+j} & a_{j1} & a_{j2} & \cdots & a_{jj} & \cdots & a_{jn} & 0 & 0 & \cdots & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
x_{n+m} & a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} & 0 & 0 & \cdots & 1 \\
-c_1 & -c_2 & \cdots & -c_j & \cdots & -c_n & 0 & 0 & \cdots & 0 \\
\end{array}
\]  

(B.13)

We assume that \( x_j \geq 0 \) for \( j = 1, 2, \ldots, x_{n+m} \) and that \( b_j \geq 0 \) for \( i = 1, 2, \ldots, m \). Then a basic feasible solution is immediately available, namely, \( x_j = 0 \) for \( j = 1, 2, \ldots, n \) and \( x_{n+i} = b_i \) for \( i = 1, 2, \ldots, m \). We then proceed as follows:
1. Determine whether the current basic feasible solution is optimal, i.e., whether all the entries in the objective row are greater than or equal to zero.

2. If there is at least one negative entry in the objective row, choose the pivot column to be that with the most negative entry in the objective row. (Assume this to be the \( j^{th} \) column.) In the case of a tie, choose arbitrarily from among those tied.

3. Determine the \( \theta \)-ratios \( \theta_i = \frac{b_i}{a_{ij}}, i = 1, 2, \ldots, m \) for the positive entries in the pivot column.

4. Select the pivot row to be that with the smallest non-negative \( \theta \)-ratio. (Assume this to be the \( i^{th} \) row.) In the case of a tie, choose arbitrarily from among those tied. (Theoretically, a tie in this step may cause problems. See the remarks that follow.)

5. Carry out the simplex transformation by employing Gauss-Jordan elimination with the pivot \( a_{ij} \).

6. Return to step 1.

Each iteration produces a new basic feasible solution. The procedure terminates when no suitable pivot column or pivot row can be found. If there is no pivot column (i.e., no negative entry in the objective row), then the current solution is optimal. If there is a suitable pivot column, but all of its entries are either zero or negative, this is an indication that the solution is unbounded.

Remark

Note that performing a simplex transformation when the choice of pivot row is made arbitrarily as a result of a tie among potential pivot rows (see (4) above) results in a basic variable (that corresponding to the row not
chosen to be the pivot row) turning to zero. The resulting basic feasible solution (which may or may not be optimal) is said to be degenerate. As degeneracy is almost never a computational problem (except in rare instances when endless cycling around a loop of non-optimal degenerate basic feasible solutions occurs), degenerate solutions are treated as any other while computing an optimal solution via the simplex method. For a more complete discussion of degeneracy and cycling, see [7] or any other comparable text on linear programming.
Previously, programs had been introduced for adaptively computing smooth piecewise polynomial approximations using uniform, least-squares ($L_2$), and restricted range uniform approximations. In this paper, a program for computing smooth piecewise polynomial $L_1$ approximations is introduced. The adaptive curve fitting scheme employed is discussed in detail as is the algorithm used for determining the $L_1$ approximations. Finally, numerical results are presented in an attempt to illustrate the effective use of this newly developed $L_1$ curve fitting package.