1971

Starting approximations for rational fraction least-squares fitting

John Howard Crenshaw
Iowa State University

Starting approximations for rational fraction least-squares fitting

by

John Howard Crenshaw

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1971

TABLE OF CONTENTS

I. INTRODUCTION

A popular technique in the area of numerical approximation is the method of least-squares approximation. The classical least-squares problem seeks to determine a set of coefficients $\vec{a} = (a_1, a_2, \cdots, a_m)$ such that an objective function, $\Phi(\vec{a})$, is minimized. $\Phi(\vec{a})$ is defined by:

$$\Phi(\vec{a}) = \sum_{i=1}^{n} \left[y_i - f(x_i, \vec{a}) \right]^2 \tag{1.1}$$

where $\{(x_i, y_i)\}$, $(i=1,2,\cdots,n)$ are the coordinates of an observed data curve, and $f(x,\vec{a})$ is the approximating function for which the coefficient vector, $\vec{a}$, must be computed.

This paper will describe a method for computing the least-squares solution, $f(x,\vec{a})$, for a particular class of data curves and a particular class of functions, $f(x,\vec{a})$. The data curves must be characterized by the presence of one or more peaks. It is also assumed that all ordinate values are positive and asymptotically approach 0 from above at $\pm\infty$. The functions $f(x,\vec{a})$ to be considered are all rational functions of the form

$$f(x,\vec{a}) = \frac{p(x,\vec{a}^1)}{q(x,\vec{a}^2)}$$

where $p(x,\vec{a}^1)$ and $q(x,\vec{a}^2)$ are polynomials in x.

The use of rational functions introduces a non-linearity problem and requires an iterative procedure rather than a direct procedure as is the case in linear least-squares problems. Because an iterative procedure requires an initial guess to the solution or coefficient vector, $\vec{a}$, the method to be described computes a starting approximation to the vector $\vec{a}$ based on the data to be approximated. As will be seen, the efficiency of

the iterative procedure, measured by the number of iterations necessary for convergence, is partly dependent on the relative closeness of the initial guess to the correct solution.

The remainder of Chapter I discusses some of the pertinent theory concerning general least-squares techniques. The iterative method for solving the non-linear least-squares problem which is used in the present system is also discussed.

Chapter II describes the theory and methods used in obtaining the starting vector $\vec{a}$, which is entered into the iterative procedure of Chapter I. Chapter III describes the other elements of the complete approximation system and explains how a user might use it. The last chapter summarizes the results of various test cases using this system and discusses future research areas which should be explored.

In a linear least-squares problem, $\Phi(\vec{a})$ in equation (1.1) will be a minimum when the first partial derivatives of $\Phi(\vec{a})$ with respect to each a_j are equal to zero (3). That is, when

$$\frac{\partial \Phi(\vec{a})}{\partial a_j} = 0 \quad (j=1,2,\cdots,m) \tag{1.2}$$

Because every a_j appears linearly in $f(x,\vec{a})$, the equations of (1.2) can be computed and rearranged to form a linear system

$$C\vec{a} = \vec{b} \tag{1.3}$$

where C is a symmetric matrix with

$$c_{ij} = \sum_{k=1}^{n} \left\{ \frac{\partial f(x_k,\vec{a})}{\partial a_i} \cdot \frac{\partial f(x_k,\vec{a})}{\partial a_j} \right\} \tag{1.4}$$

and

$$b_j = \sum_{i=1}^{n} \left\{ \frac{\partial f(x_i, \vec{a})}{\partial a_j} \cdot y_i \right\} \tag{1.5}$$

Note from (1.4) above that the matrix $C = P^T P$ where

$$P = (p_{ij}) = \frac{\partial f(x_i, \vec{a})}{\partial a_j}$$

Here and in the non-linear case which follows, it will be assumed that the columns of the matrix P are linearly independent so that $C = P^T P$ is positive definite. The system (1.3) may be solved for $\vec{a}$ by a linear system solver to get the least-squares solution.

If $f(x,\vec{a})$ takes the form

$$f(x,\vec{a}) = \sum_{j=1}^{m} a_j x^{j-1} \tag{1.6}$$

then the matrix C becomes badly ill-conditioned as m increases. Turing's condition number, TC, defined by

$$TC(C) = \frac{1}{m} \cdot N(C)N(C^{-1}) \tag{1.7}$$

where

$$N(C) = \left\{ \sum_{i,j} (c_{ij})^2 \right\}^{\frac{1}{2}}$$

and m is the rank of C, measures the ill-conditioning of a matrix (3).

The second column of Table 1.1 is a tabulation of TC in (1.7) computed by using the C matrix of (1.4), the $f(x,\vec{a})$ of (1.6), and the 19 points, x = -0.9 (0.1) 0.9. Since large values of TC indicate ill-conditioning, the use of (1.6) as a choice of $f(x,\vec{a})$ should be avoided for large values of m.

Table 1.1. Matrix condition numbers

m	Condition Numbers TC. $f(x,\vec{a}) = \sum_j a_j x^{j-1}$	$f(x,\vec{a}) = \sum_j a_j T_{j-1}(x)$
1	1.00	1.00
3	5.97	2.30
5	1.1×10^2	4.56
7	3.0×10^3	12.4
9	1.1×10^5	54.7

The third column of Table 1.1 shows the value of TC when $f(x,\vec{a})$ is taken to be

$$f(x,\vec{a}) = \sum_{j-1}^{m} a_j T_{j-1}(x) \tag{1.8}$$

where $T_{j-1}(x)$ is the $(j-1)^{st}$ Chebyshev polynomial

$$T_{j-1}(x) = \cos\left[(j-1)\theta\right] \quad , \quad \cos(\theta) = x \tag{1.9}$$

The first few terms of these polynomials are:

$T_0(x) = 1$

$T_1(x) = x$

$T_2(x) = 2x^2-1$

$T_3(x) = 4x^3-3x$

Because of the better conditioning of C as shown in Table 1.1, the Chebyshev polynomials $T_{j-1}(x)$ will be used throughout this paper instead of the monic polynomials x^{j-1}. Chapter III will discuss the procedures necessary to convert polynomials of the form (1.6) to the Chebyshev form (1.8)

and vice versa. In this way, the user will not need to know anything about Chebyshev polynomial representation.

The data curves being examined in this paper exhibit one or more peaks and a 0 asymptote at $\pm \infty$. While polynomials can be constructed to approximate certain peak conditions, they are very poor in approximating asymptotes. For these reasons, the approximating function has been chosen to be of the form

$$f(x,\vec{a}) = \frac{\sum_{j=1}^{d} a_j T_{j-1}(x)}{\sum_{j=d+1}^{m} a_j T_{j-d-1}(x)} \tag{1.10}$$

When the first partial derivatives of $\underset{\sim}{\Phi}(\vec{a})$ are computed as in (1.2), a set of non-linear equations result so that it is not possible to set up a linear system similar to (1.3). The following section will trace the development of a general iterative method for solving this non-linear least-squares problem.

One of the simplest iterative procedures is the straight gradient method. In this method, an initial starting vector, $\vec{a}$, is chosen and is modified in successive iterations until the final converged value is attained (1). Each iteration consists of computing the negative gradient of $\underset{\sim}{\Phi}(\vec{a})$ and modifying $\vec{a}$ by some fractional length of the negative gradient vector. Thus, if $\vec{a}^{(k)} = (a_1^{(k)}, a_2^{(k)}, \cdots, a_m^{(k)})$ is the vector of coefficients in the k^{th} iteration, and

$$\vec{g}^{(k)} = -\left[\frac{\partial \underset{\sim}{\Phi}(a^{(k)})}{\partial a_1^{(k)}}, \frac{\partial \underset{\sim}{\Phi}(a^{(k)})}{\partial a_2^{(k)}}, \cdots, \frac{\partial \underset{\sim}{\Phi}(a^{(k)})}{\partial a_m^{(k)}} \right]$$

is the negative gradient at $\vec{a}^{(k)}$, then

$$\vec{a}^{(k+1)} = \vec{a}^{(k)} + \alpha \vec{g}^{(k)} \qquad 0 \leq \alpha \tag{1.11}$$

is the coefficient vector for the $(k+1)^{st}$ iteration. α is chosen to insure that

$$\Phi(\vec{a}^{(k+1)}) < \Phi(\vec{a}^{(k)}) \tag{1.12}$$

This is a necessary step because the length of $\vec{g}^{(k)}$ may be such that $\Phi(\vec{a}^{(k)} + \vec{g}^{(k)}) > \Phi(\vec{a}^{(k)})$. There are several possible procedures for choosing α. One procedure would be to seek to optimize the reduction of $\Phi(\vec{a})$ in each iteration. That is,

$$\sup_{0 \leq \alpha} \left\{ \Phi(\vec{a}^{(k)}) - \Phi(\vec{a}^{(k)} + \alpha \vec{g}^{(k)}) \right\}$$

A procedure which would take much less time would accept the first value of α which satisfies (1.12). Another procedure which might be considered as being midway between the previous two methods in the amount of time required involves searching along the direction of $\vec{g}^{(k)}$ until the function $\left\{ \Phi(\vec{a}^{(k)}) - \Phi(\vec{a}^{(k)} + \alpha \vec{g}^{(k)}) \right\}$ begins decreasing. This last method guarantees that a neighborhood of a relative maximum will be achieved, but it may not be the absolute maximum in the direction $\vec{g}^{(k)}$. Studies have shown that it is often faster in the long run to take short steps at each iteration than to seek the largest α which satisfies (1.12). One study has concluded that if $\vec{a}$ is not a very close approximation, then it is unwise to spend much time optimizing the step length (11).

In practice, the gradient method is so slow that it is seldom used in the form just described. The virtue of the gradient method is that it is one of the few methods for which convergence theorems can be proved (5).

Another approach to the non-linear problem is to attempt to convert it to a linear problem. The Gauss method (2) or Gauss-Newton method (6) uses this approach. Let an initial guess, $\vec{a}^{(0)}$, be given. Define the vector, $\vec{\delta}^{(0)}$, to be the vector joining $\vec{a}^{(0)}$ to $\vec{a}$, the converged solution vector. Then $f(x,\vec{a}) = f(x,\vec{a}^{(0)} + \vec{\delta}^{(0)})$ is the desired approximating function. Expanding in a Taylor series gives

$$f(x,\vec{a}^{(0)} + \vec{\delta}^{(0)}) = f(x,\vec{a}^{(0)}) + \sum_{j=1}^{m} \left\{ \left[\frac{\partial f(x,\vec{a}^{(0)})}{\partial a_j} \right] \cdot \left[\vec{\delta}_j^{(0)} \right] \right\} + R \quad (1.13)$$

where R includes all second-order and above terms. The Gauss-Newton method is based on the assumption that $\vec{a}^{(0)}$ and $\vec{a}$ are very close to each other so that $\vec{\delta}^{(0)}$ is very small. With this assumption, all second-order and above terms are disregarded as being negligible and we have

$$f(x,\vec{a}^{(0)} + \vec{\delta}^{(0)}) \approx f(x,\vec{a}^{(0)}) + \sum_{j=1}^{m} \left\{ \left[\frac{\partial f(x,\vec{a}^{(0)})}{\partial a_j} \right] \cdot \left[\vec{\delta}_j^{(0)} \right] \right\} \quad (1.14)$$

In (1.14), the unknown quantity is the vector $\vec{\delta}^{(0)} = (\delta_1^{(0)}, \delta_2^{(0)}, \cdots, \delta_m^{(0)})$ which occurs linearly. By substituting (1.14) for $f(x,\vec{a})$ in (1.1), we can solve for $\vec{\delta}^{(0)}$ in the usual linear least-squares sense. The linear system which results is

$$C\,\vec{\delta}^{(0)} = \vec{d} \quad (1.15)$$

with the matrix C being the same as (1.4) and

$$d_j = \sum_{i=1}^{n} \left\{ \left[y_i - f(x_i,\vec{a}) \right] \cdot \left[\frac{\partial f(x_i,\vec{a})}{\partial a_j} \right] \right\} \quad (1.16)$$

The solution vector, $\vec{\delta}^{(0)}$, is a change vector which would convert $\vec{a}^{(0)}$ to the true solution $\vec{a}$ if (1.14) were an exact equality. However, because R of (1.13) is non-zero, $\vec{\delta}^{(0)}$ will differ from the exact change

vector both in direction and length. Therefore, as with the gradient method, it is necessary to use a fraction of $\vec{\delta}^{(0)}$ in adjusting $\vec{a}^{(0)}$ to get the new coefficient vector, $\vec{a}^{(1)}$.

We have, then, the general iterative scheme

$$\vec{a}^{(k+1)} = \vec{a}^{(k)} + \alpha^{(k)} \cdot \vec{\delta}^{(k)} \qquad 0 \leq \alpha \tag{1.17}$$

where $\vec{a}^{(k+1)}$ will be substituted into (1.14) to compute the next change vector, $\vec{\delta}^{(k+1)}$. The computation of $\alpha^{(k)}$ for each iteration requires the same procedural decisions as were discussed for the gradient method. That is, we wish to satisfy (1.12) at each iteration, but the choice of a procedure for calculating an $\alpha^{(k)}$ will determine how much of a reduction of $\Phi(\vec{a}^{(k)})$ is achieved.

If $\vec{a}^{(k)}$ is not a very good approximation to $\vec{a}$, the true solution, then the change vector, $\vec{\delta}^{(k)}$, may be very poor in both direction and length. In this case, it may be difficult to find a value of $\alpha^{(k)}$ which will make a substantial reduction at (1.12). Therefore, the Gauss-Newton method is most efficient only when an initial vector $\vec{a}^{(k)}$ is relatively close to the true value, $\vec{a}$.

If the Gauss-Newton method is to be used when the initial estimate $\vec{a}^{(k)}$ may be quite poor, the correction vector can be limited in length so that $\vec{a}^{(k)} + \vec{\delta}^{(k)}$ lies in a predetermined neighborhood of $\vec{a}^{(k)}$. This will help insure that (1.14) is, in fact, a good approximation to $f(x,\vec{a}^{(k)} + \vec{\delta}^{(k)})$. One possible neighborhood is the hypersphere of radius 1. centered at $\vec{a}^{(k)}$(10). This requirement is the same as

$$\left\| \vec{\delta}^{(k)} \right\|^2 \leq 1. \tag{1.18}$$

Define

$$\hat{\Phi}(\vec{\delta}) = \sum_{i=1}^{n} \left[y_i - \left(f(x,\vec{a}) + \sum_{j=1}^{m} \left\{ \left[\frac{\partial f(x,\vec{a})}{\partial a_j} \right] \cdot \delta_j \right\} \right) \right]^2 \approx \Phi(\vec{\delta}) \quad (1.19)$$

We can then prove the following theorems:

Theorem 1.1:

Let $\lambda \geq 0$ be arbitrary and let $\vec{\delta}^{(0)}$ satisfy

$$(C + \lambda I)\, \vec{\delta}^{(0)} = \vec{d} \quad (1.20)$$

where C, $\vec{\delta}^{(0)}$, and $\vec{d}$ are defined as in (1.15) and I is the unit matrix. Then $\vec{\delta}^{(0)}$ minimizes $\hat{\Phi}(\vec{\delta})$ on the hypersphere $\| \vec{\delta} \|^2 = \| \vec{\delta}^{(0)} \|^2$. That is, for a given λ, the solution $\vec{\delta}^{(0)}$ of (1.20) is the best of all possible correction vectors whose squared length is equal to that of $\vec{\delta}^{(0)}$ (8).

Proof:

Define

$$\vec{d} = P^T(\vec{y} - \vec{f}) \quad (1.21)$$

where $\vec{f} = (f(x_1,\vec{a}), f(x_2,\vec{a}), \cdots, f(x_n,\vec{a}))^T$, $\vec{y}$ equals the vector of observed data values, and P is the matrix with

$$P_{ij} = \frac{\partial f(x_i,\vec{a})}{\partial a_j} \quad (1.22)$$

Then matrix C of (1.4) satisfies $C = P^T P$ and

$$\hat{\Phi}(\vec{\delta}) = \| \vec{y} - \vec{f} - P\vec{\delta} \|^2 \quad (1.23)$$

To minimize (1.23) subject to $\| \vec{\delta} \|^2 = \| \vec{\delta}^{(0)} \|^2$, we can use the method of Lagrange which states that a necessary condition for a stationary point is:

$$\frac{\partial u}{\partial \delta_1} = \frac{\partial u}{\partial \delta_2} = \cdots = \frac{\partial u}{\partial \delta_m} = 0 \quad , \quad \frac{\partial u}{\partial \lambda} = 0 \tag{1.24}$$

where

$$u(\vec{\delta}, \lambda) = \left\| \vec{y} - \vec{f} - P\vec{\delta} \right\|^2 + \lambda(\left\| \vec{\delta} \right\|^2 - \left\| \vec{\delta}^{(0)} \right\|^2) \tag{1.25}$$

Computing the derivatives indicated by (1.24) gives

$$0 = -\left[P^T(\vec{y} - \vec{f}) - P^T P \right] + \lambda \vec{\delta} \tag{1.26}$$

and

$$0 = \left\| \vec{\delta} \right\|^2 - \left\| \vec{\delta}(0) \right\|^2 \tag{1.27}$$

Since λ is arbitrary, we may solve for $\vec{\delta}$ in (1.26) to get

$$(P^T P + \lambda I)\vec{\delta} = P^T(\vec{y} - \vec{f}) \tag{1.28}$$

which is equivalent to (1.20) which proves that the solution $\vec{\delta}$ is a stationary point. It is, in fact, a minimum because $(P^T P + \lambda I)$ is positive definite for $\lambda \geq 0$. Q.E.D.

If we define the function $\vec{\delta}(\lambda)$ to be the solution of (1.20), we can prove the following

Theorem 1.2:

$\left\| \vec{\delta}(\lambda) \right\|^2$ is a continuous decreasing function of λ. As $\lambda \to +\infty$, $\left\| \vec{\delta}(\lambda) \right\|^2 \to 0$.

Proof:

Because $C = P^T P$ in (1.20) is positive definite, we may transform it into a diagonal matrix G such that $G_{ii} > 0$. Let G_i denote the diagonal element G_{ii}. Then (1.20) can be expressed as

$$\delta_i(\lambda) = \frac{\hat{d}_i}{(G_i + \lambda)} \tag{1.29}$$

and

$$\left|\left|\vec{\delta}(\lambda)\right|\right|^2 = \sum_{i=1}^{m} \frac{\hat{d}_i^2}{(G_i+\lambda)^2} \qquad (1.30)$$

It is clear from (1.30) that $\left|\left|\vec{\delta}(\lambda)\right|\right|^2$ is a continuous function of λ for $\lambda \geq 0$ since $G_i > 0$ for all i. (1.30) also shows that $\left|\left|\vec{\delta}(\lambda)\right|\right|^2$ is a decreasing function as λ increases and that as $\lambda \to +\infty$, $\left|\left|\vec{\delta}(\lambda)\right|\right|^2 \to 0$. Q.E.D.

These theorems suggest a strategy to be used in computing the best length and direction for a correction vector at each iteration. We first set $\lambda = 0$ and compute $\vec{\delta}(0)$ in (1.20). If $\Phi(\vec{a} + \vec{\delta}(0)) < \Phi(\vec{a})$ and $\left|\left|\vec{\delta}(0)\right|\right|^2 \leq 1$, then we accept $\vec{\delta}(0)$ as the correction vector to $\vec{a}$ because it reduces the squared norm $\Phi(\vec{a})$, it has the maximum permissible length according to Theorem 1.2, and it points in the best direction among all vectors of length $\left|\left|\vec{\delta}(0)\right|\right|^2$ according to Theorem 1.1. If $\Phi(\vec{a} + \vec{\delta}(0)) \geq \Phi(\vec{a})$ or $\left|\left|\vec{\delta}(0)\right|\right|^2 > 1$, then we must increase λ until both conditions are satisfied.

As will be seen in Chapter III, λ can also be used to try to improve the conditioning of C. If (1.20) cannot be solved for $\lambda = 0$ because C is too ill-conditioned, then λ may be increased as much as necessary to achieve a well-conditioned matrix $(C + \lambda I)$. Whenever $\lambda \neq 0$, the correction vector, $\vec{\delta}$, will differ from the desired Gauss-Newton solution vector in direct proportion to the magnitude of λ. Thus, for very large λ, the correction vector, $\vec{\delta}$, will be substantially different from the Gauss-Newton correction vector.

The following theorem, however, guarantees that a reduction in $\Phi(\vec{a})$ can be made regardless of how large λ becomes. A proof is given in (8).

Theorem 1.3:

Let θ be the angle between $\vec{g}$ and $\vec{\delta}$, where $\vec{g}$ is the negative gradient and $\vec{\delta}$ is the Gauss-Newton correction vector. Then θ is a continuous monotone decreasing function of λ such that as $\lambda \to +\infty$, $\theta \to 0$.

Thus (1.20) can be interpreted as a hybrid formula combining the gradient and Gauss-Newton methods. When $\lambda = 0$, the equation is exactly the Gauss-Newton method. As $\lambda \to +\infty$, the change vector, $\vec{\delta}$, becomes proportional to $\vec{d}$ which is the negative gradient. Figure 1.1 illustrates this relationship graphically.

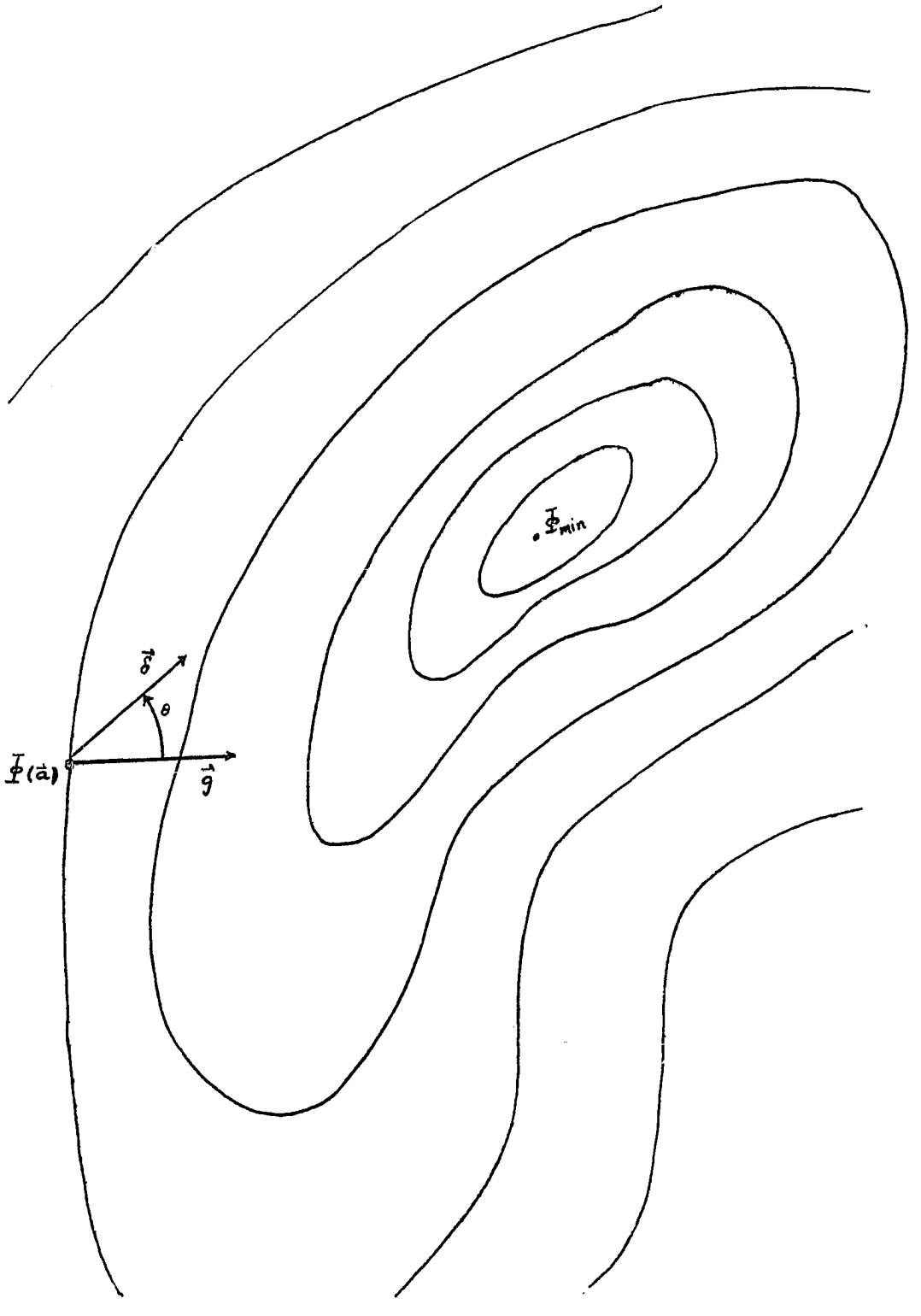


Figure 1.1. Relationship between gradient and Gauss-Newton correction vectors

II. STARTING VECTOR CALCULATION

The gradient-type methods discussed in Chapter I have good convergence properties as long as the starting coefficient vector lies in an acceptable neighborhood of the solution. Because this property is only a sufficiency condition and not a necessary condition, some starting vectors outside the acceptable neighborhood may still converge. For a given problem, however, it is difficult to tell a priori if a particular starting vector is close enough to ensure convergence.

The merit of a starting vector, $\vec{a}$, is usually measured by the norm or length of the resulting error vector whose square is given by:

$$\Phi(\vec{a}) = \left\| \vec{y} - \vec{f}(x;\vec{a}) \right\|^2 = \sum_{i=1}^{n} (y_i - f(x_i;\vec{a}))^2 \qquad (2.1)$$

Unfortunately, in most problems, the task of choosing a good starting vector is a hit-and-miss proposition.

Since the rational least-squares routine in the Scientific Subroutine Package (SSP) on the IBM S/360 is designed to handle all types of data curves, it uses the same starting vector for all data applications. This starting vector, $\vec{a}^0$, consists of a numerator of all zeroes and a denominator of all zeroes with the exception of the constant term which is a one. Thus, the square of the initial error norm is always

$$\tilde{\Phi} = \left\| \vec{y} - \vec{0} \right\|^2 = \sum_{i=1}^{n} y_i^2 \qquad (2.2)$$

The purpose of this chapter is to describe procedures for computing starting vectors for the class of data fitting problems under discussion.

The main objective is to be able to compute a starting vector, $\vec{a}$, whose squared error norm, $\Phi(\vec{a})$, is such that

$$\Phi(\vec{a}) < \tilde{\Phi} \tag{2.3}$$

This starting vector would then be used instead of $\vec{a}^0$ with the hope that either 1.) $\vec{a}$ would require fewer iterations for convergence than $\vec{a}^0$ or 2.) $\vec{a}$ would converge to a solution when $\vec{a}^0$ would not. The time and effort required in computing $\vec{a}$ must be considered when evaluating the increased efficiency gained in point 1.) above. The question of how much time should be spent in computing $\vec{a}$ is an important question and will be discussed later.

This section will be devoted to a study of one-peak data; multiple-peak data will be discussed in a later section. As stated in Chapter I, the data curves are assumed to be positive and approach zero asymptotically at $\pm\infty$. In order to ensure this latter property in the function to be determined, the degree of the denominator will always be set two greater than the numerator. Thus, we will be determining

$$f(x;\vec{a}) = \frac{p(x;\vec{a})}{q(x;\vec{a})r(x;\vec{a})}$$

where $p(x;\vec{a})$ and $q(x;\vec{a})$ are of the same degree and $r(x;\vec{a})$ is of second degree.

In examining many different types of single-peak data curves and the best rational functions fitting these curves, it was observed that the graph of each denominator appeared concave-upwards like a parabola. Thus, if we let $r(x;\vec{a}) = sx^2 + tx + u$, we would like to be able to determine the coefficients s, t, and u so that $r(x;s,t,u)$ might approximate the desired quadratic factor.

It was also observed that the abscissa of the vertex of the denominator parabolic curve was always in the immediate neighborhood of the abscissa of the peak in the original data. Because the original data was always positive, the denominator was likewise positive so that the ordinate of the vertex was always greater than zero. The ordinate of the vertex was observed to vary inversely with the ordinate of the original data peak. That is, as the height of the data peak was increased, the vertex of the denominator decreased towards zero.

These observations were used in designing a procedure to calculate the $r(x;\vec{a})$ factor of the rational function $f(x;\vec{a})$. A rational function has a free parameter which can be set arbitrarily. In this case, the coefficient of x^2 in $r(x;\vec{a})$ was chosen to equal 1. Given, then, $r(x;t,u) = x^2 + tx + u$ we transform it to the form $r(x;b,e) = (x-b)^2 + e$ so that we can make better use of the information discussed above. For example, we desire that the parabola be centered at the point where the peak of the original data curve is located. By a procedure to be described in Chapter III, the approximate abscissa value of the peak, x_p, can be determined. Setting $b = x_p$ ensures initially, at least, that the vertex is centered correctly.

The ordinate value of the parabola at its vertex, that is, $r(b) = e$, is supposed to vary inversely with the ordinate value of the original data curve. Good results were obtained for many data examples simply by setting $e = (y_p)^{-\frac{1}{2}}$ where y is the ordinate value of the peak. However, this inverse proportionality for e did not work as well in all data examples, so it seemed desirable to make use of more of the original data than just the peak location. Therefore, a process was devised to choose an e that would attempt to minimize the squared error norm

$$\Phi(e) = \sum_{i=1}^{n} \left\{ y_i - \frac{p(x_i;\vec{a})}{q(x_i;\vec{a}) \cdot \left[(x_i-b)^2 + e \right]} \right\}^2 \qquad (2.4)$$

Since the denominator function $q(x;\vec{a}) \cdot \left[(x-b)^2 + e\right]$ results in a strongly parabolic shape, it is assumed that $q(x;\vec{a})$ serves only to modify $\left[(x-b)^2 + e\right]$ slightly so that we may set $q(x;\vec{a}) = 1$ at this point. Likewise, $p(x;\vec{a})$ will be computed at a later stage but is set equal to a constant 1 during this phase. The function in e now looks like

$$\Phi(e) = \sum_{i=1}^{n} \left\{ y_i - \frac{1}{\left[(x_i-b)^2 + e \right]} \right\}^2 \qquad (2.5)$$

At this stage, a decision must be made concerning how much time should be spent in computing one coefficient of the starting vector. Equation (2.5) is a single non-linear equation in a single unknown. It should be possible to construct a numerical method to accurately compute the value of e satisfying (2.5), but the machine computation time must be weighed against the validity of the results of that computation time. In this situation, the best numerical solution to (2.5) will have inaccuracies built-in because of the inexact approximations used for $p(x;\vec{a})$, $q(x;\vec{a})$, and b. In all likelihood, all of these coefficients will be modified somewhat during the final convergence procedure. For this reason, it seems more advisable to determine an e in (2.5) which results in a reduction of $\Phi(e)$ with a reasonable computational cost.

To do this requires a closer analysis of (2.5). If $\Phi(e)$ of (2.5) is thought of as the composite or sum of n separate functions, we may get a feel for the properties of $\Phi(e)$ by looking at each separate function. Let $\Phi_i(e)$ be given as:

$$\Phi_i(e) = \left[y_i - \frac{1.}{(x_i-b)^2+e} \right]^2$$

Because of previously mentioned factors, we are interested only in $e \geq 0$, and we do not care about the behavior of $\Phi_i(e)$ for $e < 0$. $\Phi_i(e)$ goes to $+\infty$ at $e = -(x_i-b)^2$ which may occur at e=0 since b may be equal to x_i. By inspection, we can also see that as e tends to $+\infty$, $\Phi_i(e)$ tends to y_i^2 asymptotically from below.

We can discover other properties of $\Phi_i(e)$ by looking at its derivatives. Let

$$z_i = (x_i-b)^2$$

then

$$\Phi_i(e) = y_i^2 - \frac{2y_i}{z_i+e} + \frac{1}{(z_i+e)^2}$$

and

$$\frac{\partial \Phi_i}{\partial e} = \frac{2y_i}{(z_i+e)^2} - \frac{2(z_i+e)}{(z_i+e)^4} = 2\left[\frac{y_i(z_i+e)-1}{(z_i+e)^3} \right]$$

For notational simplicity, the subscript, i, will be deleted from y_i and z_i in the following discussion.

An extremum exists when $\frac{\partial \Phi_i}{\partial e} = 0$ and

$$\frac{\partial \Phi_i}{\partial e} = 0 = 2\left[\frac{y(z+e)-1}{(z+e)^3} \right]$$

when

$$e = \frac{1}{y} - z = \frac{1}{y} - (x-b)^2 \qquad (2.6)$$

Thus there is an extremum at $e = (1/y) - (x-b)^2$. The second derivative indicates whether the point is a maximum or minimum:

$$\frac{\partial^2 \Phi_i}{\partial e^2} = \frac{(z+e)^3 2y - [2y(z+e)-2]\cdot 3(z+e)^2}{(z+e)^6}$$

$$= 2\cdot\left[\frac{3-2y(z+e)}{(z+e)^4}\right]$$

And $\frac{\partial^2 \Phi_i}{\partial e^2}$ evaluated at the extremum (2.6) gives:

$$\frac{\partial^2 \Phi_i(\frac{1}{y}-z)}{\partial e^2} = 2\cdot\left[\frac{3-2y(z+\frac{1}{y}-z)}{(z+\frac{1}{y}-z)^4}\right]$$

$$= 2y^4 > 0$$

Thus, the extremum is a minimum, and the value of $\Phi_i(e)$ at the minimum is:

$$\Phi_i(\tfrac{1}{y}-z) = y - \frac{1}{z+\frac{1}{y}-z}$$

$$= 0$$

Setting the second derivative equal to zero locates the stationary point:

$$\frac{\partial^2 \Phi_i}{\partial e^2} = 2\left[\frac{3-2y(z+e)}{(z+e)^4}\right] = 0$$

when

$$e = \frac{3}{2y} - z \qquad (2.7)$$

Let s denote the stationary point and m denote the minimum point. The distance d between s and m is then:

$$d = s - m$$

$$= \left[\frac{3}{2y} - z\right] - \left[\frac{1}{y} - z\right]$$

$$= \frac{1}{2y}$$

so the stationary point is always to the right of the minimum. Finally, the value of $\Phi_i(e)$ at the stationary point is

$$\Phi_i\left(\frac{3}{2y}-z\right) = \left[y - \frac{1}{\left(z+\frac{3}{2y}-z\right)}\right]^2$$

$$= \frac{y^2}{9} \qquad (2.8)$$

We wish to utilize this information about the form of the $\Phi_i(e)$ curves to construct an algorithm for finding a value for e for which the composite function, $\Phi(e)$, is at or near a minimum. Let

$$E = \{e_1, e_2, \cdots, e_n\}$$

denote the set of values of e at which the functions $\Phi_1(e)$, $\Phi_2(e)$, $\cdots$, $\Phi_n(e)$ attain their minima. That is, $\Phi_1(e_1) = 0$, $\Phi_2(e_2) = 0$, etc. Let

$$e_{max} = \max_{e_i \in E} e_i$$

Because each $\Phi_i(e)$ is monotone increasing for all $e > e_i$, then $\Phi(e)$ itself is monotone increasing for all $e > e_{max}$. This is because $\Phi(e)$, for $e > e_{max}$, is the sum of n monotone increasing functions and, therefore, is monotone increasing also. If $e_{max} \leq 0$, then the monotonicity guarantees that $\Phi(0)$ is a minimum over the interval $[0, +\infty)$. The case where $e_{max} \leq 0$ is an unusual occurrence and will be discussed later in this section.

When $e_{max} > 0$, we have reduced the domain of possible e's to the interval $(0, e_{max}]$. As we have shown,

$$\lim_{e \to +\infty} \Phi(e) = \sum_{i=1}^{n} (y_i^2) = \tilde{\Phi}$$

and since $\Phi(e)$ is monotone increasing for all $e \geq e_{max}$, then we can conclude that

$$\Phi(e) < \tilde{\Phi} \qquad \text{for } e \geq e_{max}$$

and

$$\Phi(e_{max}) < \Phi(e) < \tilde{\Phi} \qquad \text{for } e > e_{max}$$

The preceding discussion has proven the following theorem:

Theorem 2.1:

If C is any data curve defined by the points (x_i, y_i), $i = 1,2,\cdots,n$ such that $y_i > 0$ for all i, then there exists an $e \geq 0$ such that

$$\Phi(e) < \tilde{\Phi}$$

where $\Phi(e)$ is defined by (2.5) and $\tilde{\Phi}$ is defined by (2.2).

Thus we could compute e_{max} and, if it were positive, be assured that the starting vector composed of b and e_{max} would given an initial error norm satisfying (2.3).

The above procedure is based simply on the form of equation (2.5); it does not make use of the fact that the y_i values describe a peak curve. By utilizing this additional fact, we can hope to make a better choice for e. To do this, we shall look at two of the composite functions, $\Phi_j(e)$ and $\Phi_k(e)$, where:

$$y_j = \max_{i=1,n} y_i \quad , \quad y_k = \min_{i=1,n} y_i$$

Φ_j goes to infinity when $e = -(x_j-b)^2$. Because y is the peak value and b is chosen to approximate the abscissa at the peak value, $(x_j-b) \approx 0$. The minimum value of Φ_j will also be close to 0 since both $1/y_j$ and $(x_j-b)^2$ will be quite small. The stationary point of Φ_j lies to the right of the minimum point a distance of $1/2y_j$, which means that this distance is also small when y_j is the peak value. Finally, the value of Φ_j at the stationary point is $y_j^2/9$.

This analysis shows that the graph of Φ_j is a deep trench which drops down to the e-axis at $e=\left[1/y_j - (x_j-b)^2\right]$ and rebounds steeply before leveling off to approach y_j^2 at $+\infty$. The value of the second derivative of Φ_j at the minimum point verifies the steepness of Φ_j at the bottom of the trench as its value is $2y_j^4$.

In contrast, the graph of Φ_k shows a much more shallow trench whose sides are much less steep. The infinity point of $-(x_k-b)^2$ could be as negative as -4 since x_k records the position of the smallest y value while b approximates the position of the largest. Since y_k is much smaller than y_j, then

$$\frac{1}{y_k} > \frac{1}{y_j}$$

and the distance between the point of infinite Φ_k and the minimum point of Φ_k will be greater than it was for Φ_j. For the same reason, the distance to the stationary point will be greater for Φ_k, and the value of Φ_k at that point will also be much less since

$$\frac{y_k^2}{9} < \frac{y_j^2}{9}$$

The shallowness of Φ_k at the minimum is verified by the fact that $2y_k^4 < 2y_j^4$ where $2y_k^4$ is the value of the second derivative of Φ_k at the minimum.

If $\Phi(e)$, then, were composed of the sum of Φ_j and Φ_k only, it would be very reasonable to expect the minimum of $\Phi(e)$ to correspond much more closely to the minimum of Φ_j than to the minimum of Φ_k. This is the principle used in determining an approximation for the minimum of $\Phi(e)$. $\Phi(e)$ is the sum of n curves, each of which contains a trench-shaped section in which its minimum is found. Because of the high values of Φ_j at all points except in the bottom of its trench, the desired approximation to the minimum of $\Phi(e)$ is assumed to be the minimum of Φ_j. However, since the minimum of Φ_j could occur at negative e or since several values could be clustered near the peak, a systematic computation is made to ensure that a good positive approximation is found.

This computation involves finding the minimum of each $\Phi_i(e)$ and accepting the smallest value which does not lead to a negative e. If the number of data points, n, is very large, it would be more efficient to make a selection process so that only certain $\Phi_i(e)$ would be calculated. Even so, all points in the neighborhood of the peak should still be calculated. Figure 2.1 shows the plots of several $\Phi_i(e)$ curves and the composite curve, $\Phi(e)$, for curve 2 from Figure 4.2. In the upper part of Figure 2.1, the vertical scale has been compressed.

It is possible to construct examples of one-peak data curves such that

$$e_i = \left[\frac{1}{y_i} - (x_i - b)^2 \right] < 0 \quad \text{for all } i = 1,2,\cdots,n$$

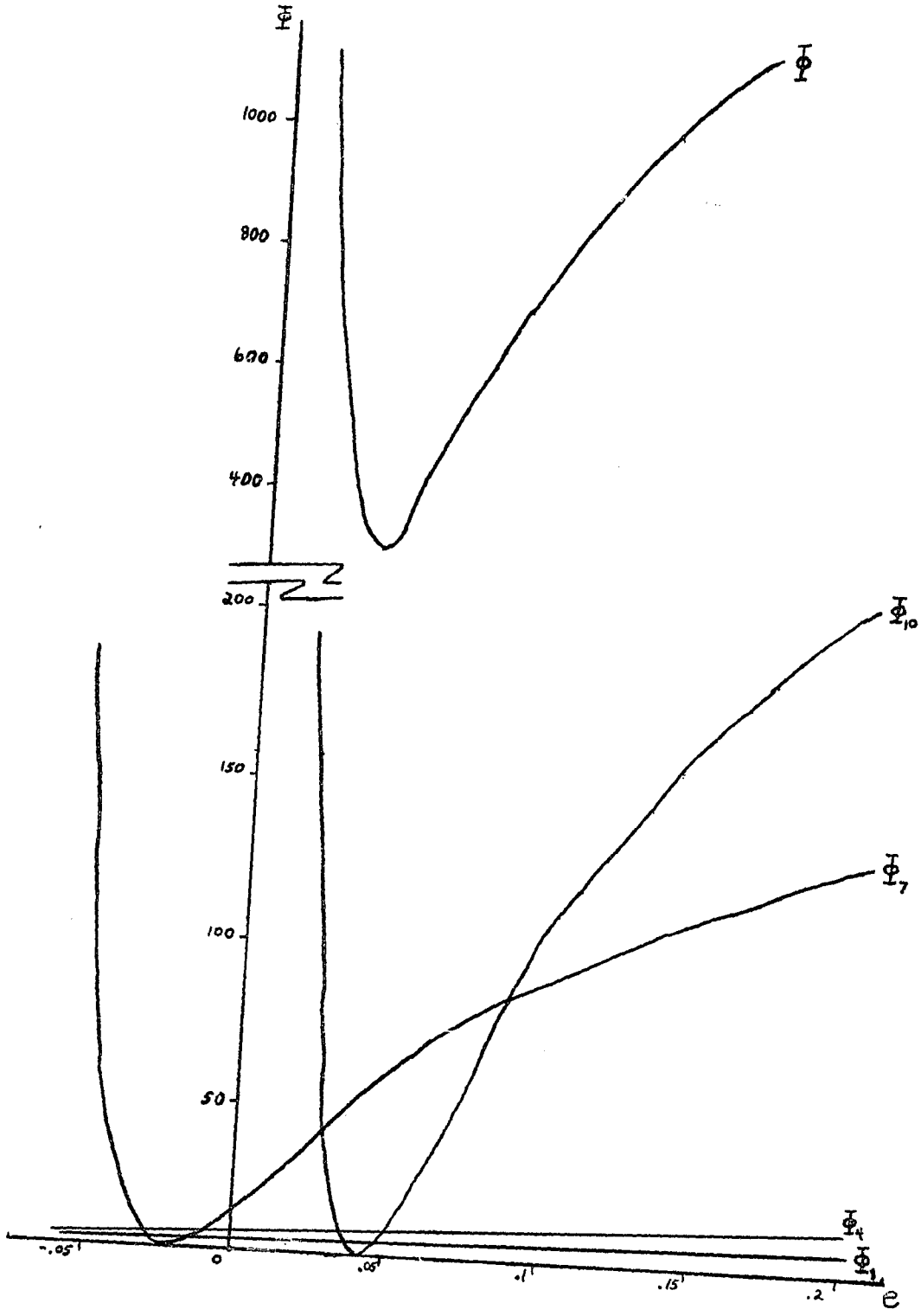


Figure 2.1. Graphs of $\Phi_i(e)$ and $\Phi(e)$ for data curve 2

If

$$y_j = \max_{i=1,n} y_i$$

and

$$e_j = \left[\frac{1}{y_j} - (x_j-b)^2\right] < 0$$

then this could mean that not enough data points near the peak are available. By taking more readings closer to b, the term $(x_j-b)^2$ will get smaller so that e_j may get larger and become positive. If this does not work, then the peak value may well be an infinite spike and e should be set equal to zero.

After the values of b and e have been calculated, the starting values for the numerator coefficients in $p(x;\vec{a})$ are calculated. This is accomplished by entering a linear least-squares system which has been modified to permit the inclusion of the denominator term, $(x_i-b)^2+e$, as a factor in each coefficient. For example, whereas the regular linear least-squares system would look at the function

$$p(x;\vec{a}) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$$

the modified linear system would look at

$$p(x;\vec{a}) = a_0\cdot\left[\frac{1}{(x-b)^2+e}\right] + a_1\cdot\left[\frac{x}{(x-b)^2+e}\right] + a_2\cdot\left[\frac{x^2}{(x-b)^2+e}\right] + \cdots + a_m\cdot\left[\frac{x^m}{(x-b)^2+e}\right]$$

The computation of the coefficients of $p(x;\vec{a})$ concludes the first phase of the convergence process. The second phase uses the IBM S/360 Rational Least-Squares System to complete the convergence. This phase is discussed in Chapter III.

This section discusses the computation of starting vectors for curves possessing two peaks. All other properties of the data curves are assumed to be the same as for one-peak curves. That is, all ordinate values are positive, and the curve approaches zero at $\pm\infty$. The approximating function to be determined is

$$f(x;\vec{a}) = \frac{p(x;\vec{a})}{q(x;\vec{a}) \cdot \left[(x-b_1)^2+e_1\right] \cdot \left[(x-b_2)^2+e_2\right]} \tag{2.9}$$

where the degree of $p(x;\vec{a})$ is two greater than that of $q(x;\vec{a})$. The two quadratic factors in the denominator are used in the two-peak problems for reasons analagous to those for using one quadratic factor in one-peak data. b_1 and b_2 are set equal to x_{p1} and x_{p2}, respectively, where x_{p1} and x_{p2} are the approximate abscissa values of the two peaks.

The approximation of e_1 and e_2 is somewhat more difficult than the approximation of e in one-peak problems. If we once again look at

$$\Phi_i = \left\{ y_i - \frac{1}{\left[(x_i-b_1)^2+e_1\right] \cdot \left[(x_i-b_2)^2+e_2\right]} \right\}^2$$

and attempt to choose e_1 and e_2 to minimize Φ_i by computing the partials of Φ_i, we get

$$\frac{\partial \Phi_i}{\partial e_1} = \frac{2y_i \cdot \left[(x_i-b_1)^2+e_1\right] \cdot \left[(x_i-b_2)^2+e_2\right] - 2}{\left[(x_i-b_1)^2+e_1\right]^3 \cdot \left[(x_i-b_2)^2+e_2\right]^2} \tag{2.10}$$

$$\frac{\partial \Phi_i}{\partial e_2} = \frac{2y_i \cdot \left[(x_i-b_1)^2+e_1\right] \cdot \left[(x_i-b_2)^2+e_2\right] - 2}{\left[(x_i-b_1)^2+e_1\right]^2 \cdot \left[(x_i-b_2)^2+e_2\right]^3} \tag{2.11}$$

Setting (2.10) and (2.11) equal to zero leads to the same equation in each case:

$$1 = y_i \cdot \left[(x_i-b_1)^2+e_1\right] \cdot \left[(x_i-b_2)^2+e_2\right] \qquad (2.12)$$

Even though we are not interested in negative e_1 or e_2, there are many positive solutions, (e_1, e_2), to (2.12) for any given x_i, y_i, b_1, and b_2. Thus we need more information about e_1 and e_2 before we can hope to make a wise choice for a solution pair. The next approach to be discussed provides a better means for choosing e_1 and e_2.

If the data describes two distinct peaks, we may think of the best approximating function as being the sum of two single-peak functions. That is,

$$f(x;\vec{a}) = g(x;\vec{a}) + h(x;\vec{a})$$

where

$$g(x;\vec{a}) = \frac{p_1(x;\vec{a})}{q_1(x;\vec{a}) \cdot \left[(x-b_1)^2+e_1\right]}$$

and

$$h(x;\vec{a}) = \frac{p_2(x;\vec{a})}{q_2(x;\vec{a}) \cdot \left[(x-b_2)^2+e_2\right]}$$

so that $f(x;\vec{a})$ may be rewritten as

$$f(x;\vec{a}) = \frac{p(x;\vec{a})}{q(x;\vec{a}) \cdot \left[(x-b_1)^2+e_1\right] \cdot \left[(x-b_2)^2+e_2\right]}$$

which is the same as (2.9).

$g(x;\vec{a})$ and $h(x;\vec{a})$ are then computed in the same way one-peak functions were approximated. b_1 and b_2 are the two respective peak locations as approximated by the scanning procedure. e_1 and e_2 are computed by attempting to minimize

$$\Phi_g = \sum_{i=1}^{k} \left\{ y_i - \frac{1}{(x_i - b_1)^2 + e_1} \right\}^2$$

and

$$\Phi_h = \sum_{i=k+1}^{n} \left\{ y_i - \frac{1}{(x_i - b_2)^2 + e_2} \right\}^2$$

by the same techniques used for one-peak data. The only difference is that all n points are not used in minimizing both Φ_g and Φ_h. Only those points on the data curve which describe the first peak are used in Φ_g, and only those points which describe the second peak are used in Φ_h. This is accomplished by having the scanning procedure determine the approximate location of the valley between the two peaks. If the abscissa of the valley, x_v, is such that

$$x_k \leq x_v < x_{k+1}$$

then $g(x;\vec{a})$ is assumed to be described by points x_1 to x_k and $h(x;\vec{a})$ by points x_{k+1} to x_n.

After e_1 and e_2 have been computed, starting values for the numerator, $p(x;\vec{a})$, of $f(x;\vec{a})$ are determined in the same manner that the numerator coefficients of the one-peak approximation were calculated. The resulting starting vector is then fed into the second phase to complete the convergence.

One major advantage of this method for two-peak data is its similarity to and compatibility with the method for one-peak data. With a small amount of additional programming logic, the one-peak approximating procedure can be converted to both a one- and two-peak procedure. Thus, if two

peaks are found in the scanning process, then a two-peak approximating function as in (2.9) will be constructed. If one peak is found, $f(x;\vec{a})$ will take the form

$$f(x;\vec{a}) = \frac{p(x;\vec{a})}{q(x;\vec{a}) \cdot \left[(x-b_1)^2+e_1\right]}$$

This same facility could easily be extended to process data with more than two peaks, but three-peak data has not been investigated in this paper.

III. COMPUTER SYSTEM DESCRIPTION

This chapter describes the implementation of the complete system for computing a rational least-squares approximation. Figure 3.1 shows the various steps and subprocedures which will be discussed and which are required in the system.

The preliminary input required includes the number of data points to be used, the desired degree of the numerator polynomial and denominator polynomial of the approximation, and a code denoting the presence or absence of weighting values for the data points. All preliminary input is read from one card with a format of (I3, I1, I1, I1) and is referenced internally by the names N, IP, IQ, and IW, respectively. IW=1 indicates that weights will be used; IW=0 indicates no weights.

The data points, values, and weights, if present, are then read in with one set per card. That is, the i^{th} data card will contain the i^{th} data point, the i^{th} data value, and the weight associated with the i^{th} data value. The format for the input is (3F15.8), and the internal references are X(I), X(I+N), and X(I+N+N), respectively. If IW equals 0, then X(1+N+N) thru X(N+N+N) are set equal to a vector of all ones after all data cards are read in, regardless of what might have been read in from the data cards.

There are two reasons for reading the input in this form rather than reading in all abscissa values followed by all ordinate values and then all weighting values as is often done. The first reason is that the possibility of an error occurring due to a different ordering of the abscissa values from the ordinate values is minimized. Secondly, this input scheme

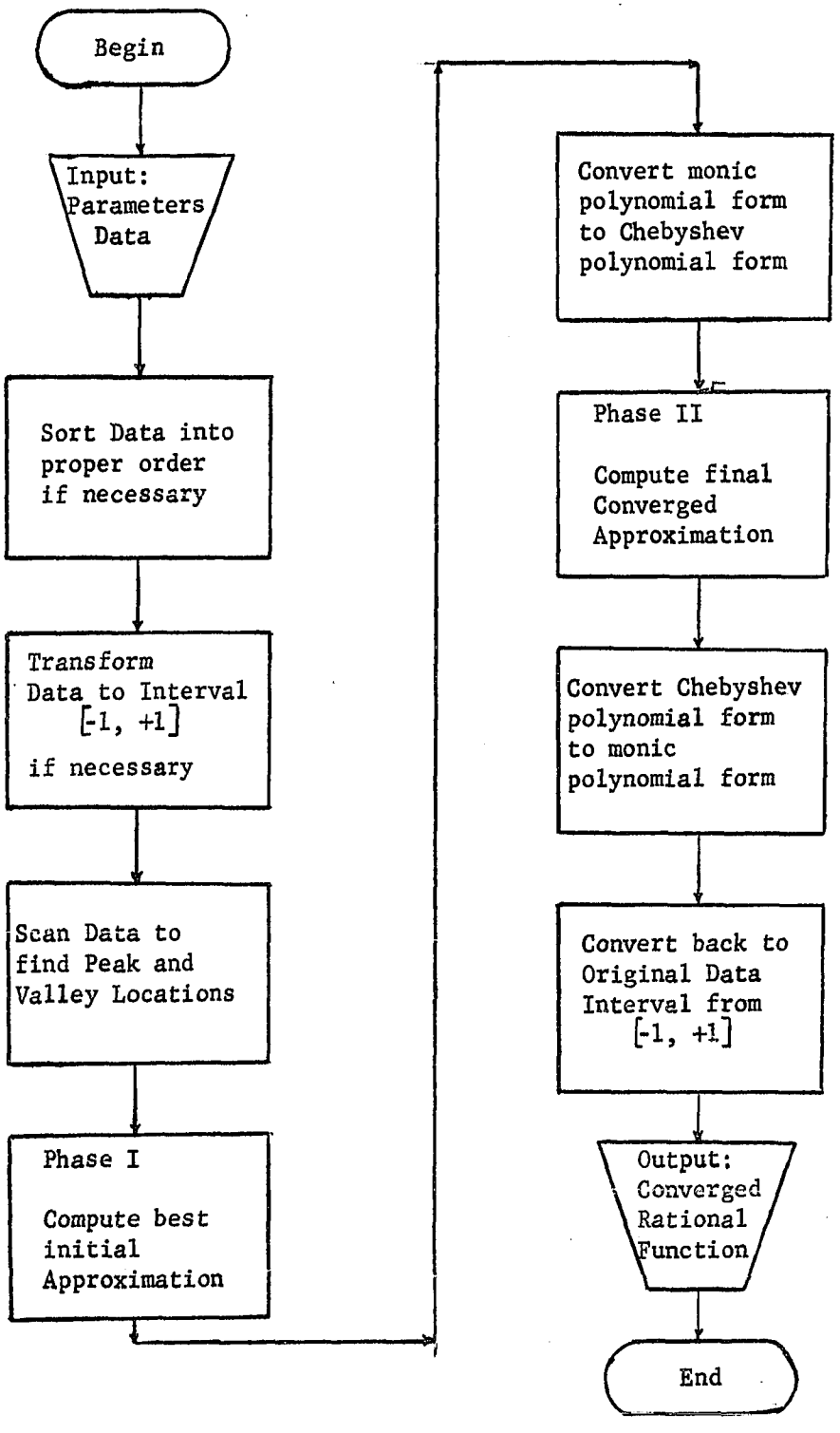


Figure 3.1. Block diagram for proposed rational approximation system

makes it much more convenient for the user to add or delete data points simply by adding or deleting data cards.

It is essential to this system that the data be in ascending order on the abscissa values. Without this order, it is not possible to determine the peak locations. Although the system would run faster if the user pre-sorted the data, it is a poor procedure to depend upon the user to do this. The system, therefore, has an internal sort routine included. If the data is already in order, only one pass is required to ensure that this is so. The method used is a reverse bubble sort in that the scan is made from bottom to top. A bubble sort is used because it requires a minimal amount of additional storage. The execution speed of a bubble sort is not exceptional, but the user can avoid this by pre-sorting the data.

A reverse scan is used because on randomly arranged data there is no difference in either speed or storage required between a forward and reverse scan routine. However, if a user has pre-sorted his data and then adds one more data point at the end of his data deck, forgetting to merge it into the correct place in the deck, a forward scan routine will require, on the average, n/2 passes to place the new card in its proper spot if there are n data points. A backward or reverse scan will correctly place the additional card in only one pass.

The abscissa values must all lie in the closed interval $[-1,1]$ because Chebyshev polynomials are used as fundamental units in the approximating rational function. Thus a transformation of the abscissa values is made according to the relation:

$$x_i' = \frac{2x-(x_{max} + x_{min})}{(x_{max} - x_{min})} \tag{3.1}$$

where $x_{max} = x_n$ which is the largest abscissa value and $x_{min} = x_1$ which is the smallest abscissa value. The ordinate values and weighting values are left unchanged.

After the interval transformation has been made, the data is scanned to determine where the peaks and valleys are located. Because the data is now in ascending order on the abscissa values, the location of the peaks and valleys can be determined by the algorithm indicated in Figure 3.2.

At the completion of this routine, the peak locations will be stored in the vector $\vec{p}$, and the valley locations will be in the vector $\vec{v}$. The variables jp and jv contain the number of peaks and valleys, respectively, which have been found. No attempt is made to determine the approximate height of the data curve at each peak and valley. d_1 is set equal to a positive value because it is assumed that the data curve is initially monotone increasing at the extreme left-hand end.

The equation

$$E = \left[\frac{(d_i + d_{i-1})}{2(d_i^2 + d_{i-1}^2)^{\frac{1}{2}}} \right] \cdot h + x_i \qquad (3.2)$$

attempts to interpolate, when necessary, to locate the position of a peak more accurately when it falls between two data points. If a situation exists where $y_i > y_{i-1} = y_{i+1}$ then (3.2) will compute $E = 0 + x_i = x_i$ which is the most reasonable estimate if the data points are equally spaced. If $y_i = y_{i+1}$, then (3.2) will compute $E = \frac{1}{2}h + x_i$ or half the distance between x_i and x_{i+1} which is also the reasonable choice.

Other values of y_{i-1}, y_i, and y_{i+1} will give a fractional increment to x_i based on the relative positions of y_{i-1} and y_{i+1}. If the data points

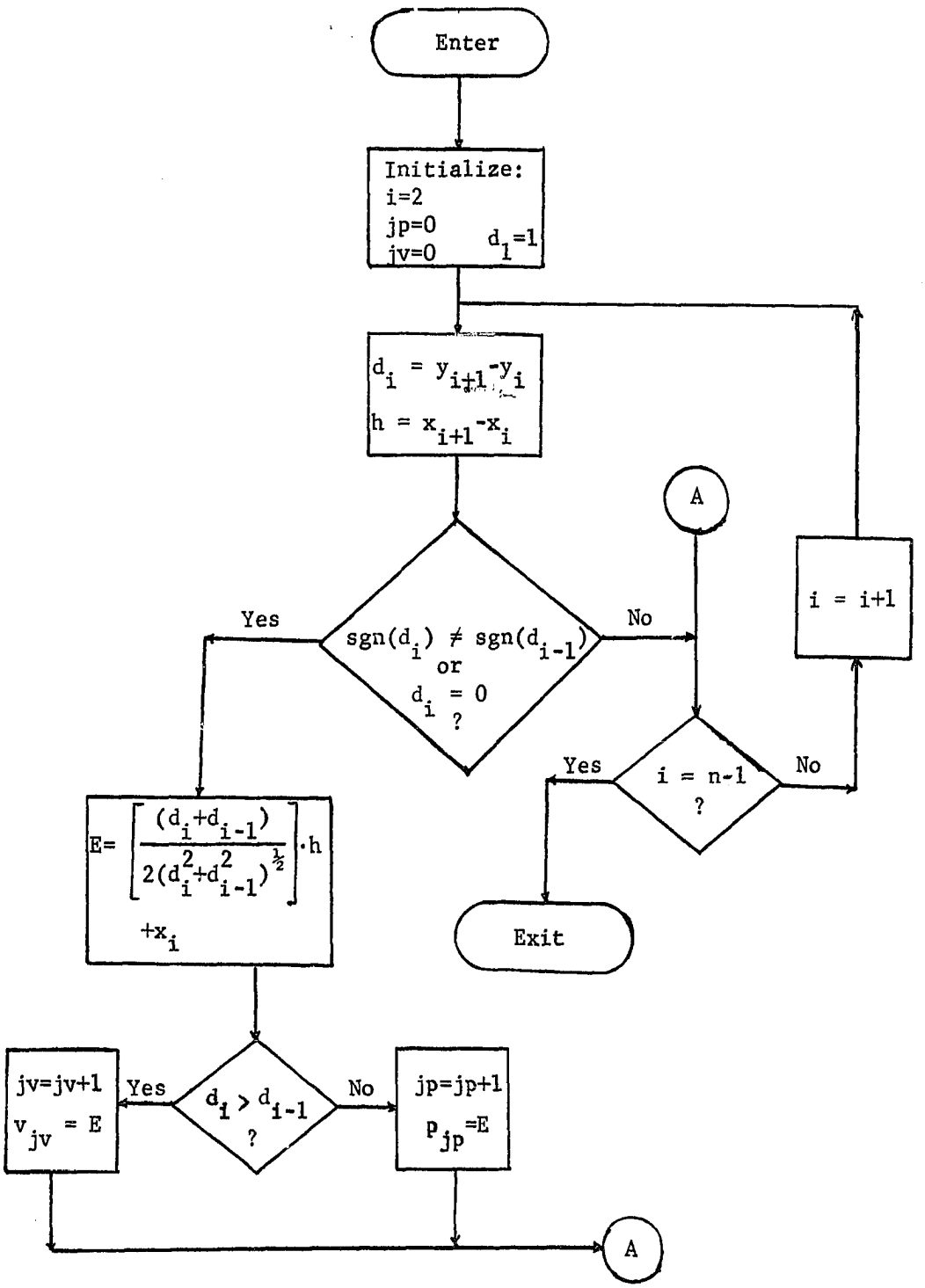


Figure 3.2. Scanning procedure for locating data peaks and valleys

are not equally spaced, the interpolated peak locations will differ somewhat from the most reasonable choices depending on how much of a spacing difference exists around the approximate location, y_i.

As was discussed in Chapter I, the use of monic polynomials

$$f(x;\vec{a}) = \sum_{i=1}^{m} a_i x^{i-1} \tag{3.3}$$

in a least-squares problem leads to a badly ill-conditioned linear system. The use of Chebyshev polynomials in the approximating function

$$f(x;\vec{a}) = \sum_{i=1}^{m} a_i T_{i-1}(x) \tag{3.4}$$

removes a major portion of the ill-conditioning. However, the form of (3.3) is much more understandable and is more efficiently evaluated in some computer applications, so a procedure is necessary to convert back and forth between (3.3) and (3.4).

In particular, between Phase I of the approximation procedure and Phase II, it is necessary to convert the initial approximation from a monic to a Chebyshev representation. After Phase II, the Chebyshev form is reconverted to monic form before it is output. The following discussion will explain the conversion method devised for this system.

The heart of the conversion procedure is the fact that we can construct linear systems

$$A\vec{m} = \vec{c} \tag{3.5}$$

and

$$B\vec{c} = \vec{m} \tag{3.6}$$

where $\vec{m}$ is a vector of coefficients from a monic representation (3.3), and $\vec{c}$ is the corresponding coefficient vector from a Chebyshev representation (3.4). A and B are upper triangular matrices and obviously, $A = B^{-1}$. Therefore, to convert from one form to the other, it is necessary only to perform a matrix-vector multiplication.

The speed advantage of this procedure is achieved at the expense of having to store two upper triangular matrices in core. In addition, the matrices must have an order equal to the largest order which might be required by a user. In this system, the number of coefficients in either the numerator or denominator must not be greater than eight, so A and B must be of order eight. However, since A and B are both triangular, a method has been constructed so that both matrices are stored together in a 8 x 9 matrix, D. A is stored as an upper triangular, and B^T is stored as a lower triangular matrix so there are no unused elements in the D matrix.

An additional amount of programming is necessary in order to allow for both computations (3.5) and (3.6). In order to compute (3.5), the calculation process is $D\vec{m} = \vec{c}$ such that

$$c_i = \sum_{j=1}^{k} \left[d_{i,j+i} \right] \cdot \left[m_j \right] \quad (i=1,2,\cdots,k)$$

(3.6) is calculated by $D\vec{c} = \vec{m}$ such that

$$m_i = \sum_{j=1}^{k} \left[d_{j,i} \right] \cdot \left[c_j \right] \quad (i=1,2,\cdots,k)$$

where $k \leq 8$ is the order of $\vec{c}$ and $\vec{m}$. In this manner, the single 8 x 9 matrix, D, can be used to convert polynomials of varying orders from monic form to Chebyshev form or vice versa.

Figure 3.3 shows the routines used in the second phase of the convergence procedure (7). ARAT is the major routine which sets up the calls to the other routines as needed. CNPS is required at several points to compute the numerical value of the Chebyshev expansion $a_1T_0(x)+a_2T_1(x)+\ldots+a_mT_{m-1}(x)$ for given arguments $\vec{a}$, m, and x. The single scalar value is returned to ARAT.

APLL is a standard routine for computing the normal equations for a linear least-squares problem. The output is the matrix C of equation (1.4) and the right-hand side vector $\vec{d}$ of equation (1.16). Because APLL is a general routine, another routine, FRAT, is necessary to supply the required values of $f(x_i;\vec{a})$ and y_i. Within FRAT, the routine CNP is used to compute the values of the Chebyshev polynomials $T_0(x)$, $T_1(x),\ldots,$ $T_m(x)$ for given arguments m and x. The values are returned to FRAT in a vector of m + 1 components.

APFS solves the linear system $C\vec{x} = \vec{d}$ which was set up by ARAT and APLL. The output of APFS is a vector of m components containing the least-squares coefficient solution. Since the matrix C in APLL and APFS is a symmetric matrix, a reduction in storage requirements is achieved by storing only the upper triangular portion of C columnwise.

Upon return of the solution vector from APFS, ARAT proceeds to compute the α factor of equation (1.17). The procedure is to set α equal to 1 and then keep reducing it as long as $\Phi(\alpha)$ keeps decreasing. As soon as $\Phi(\alpha)$ shows an increase, then the previous value of α is accepted, and the iteration is complete.

The test for convergence involves checking the following inequalities:

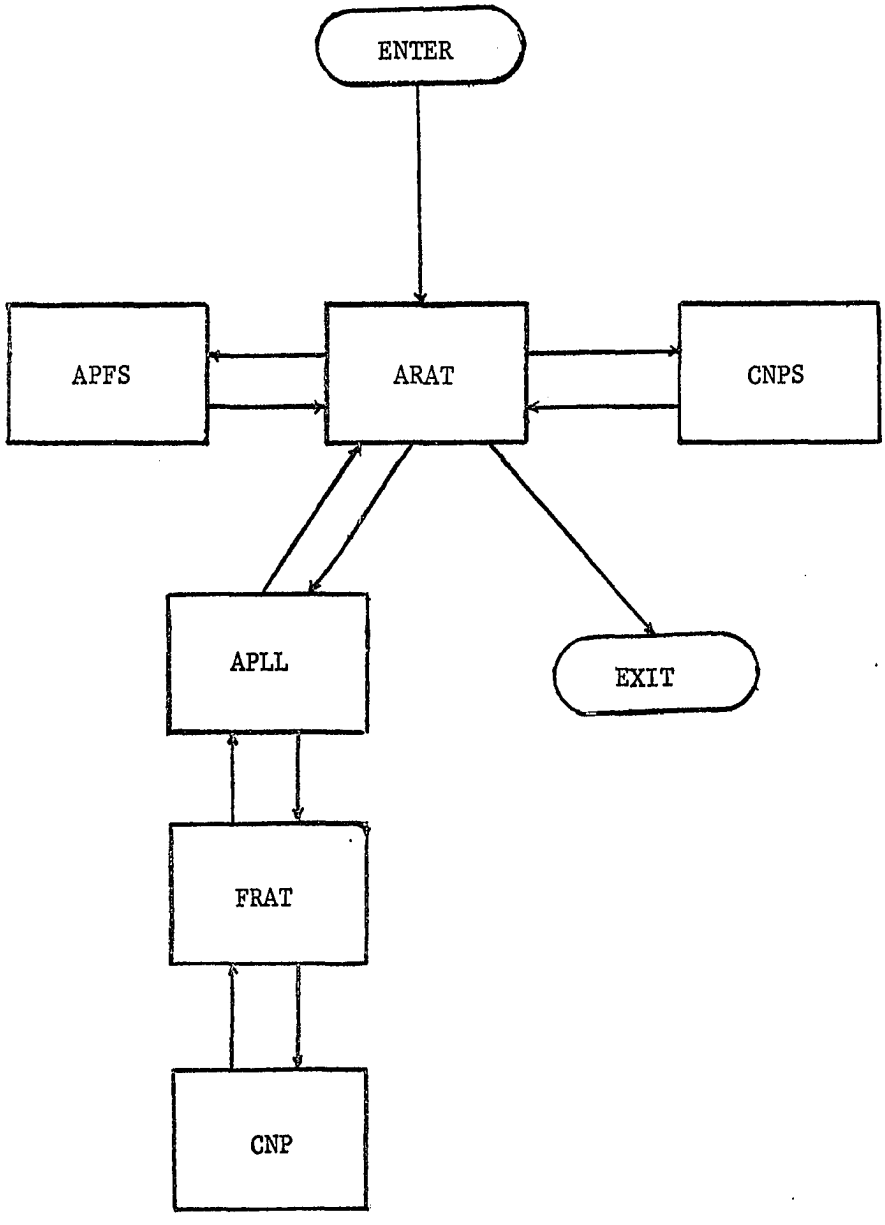


Figure 3.3. Subroutine linkage for phase 2 of the convergence procedure

$$\frac{\Phi^{(k)} - \Phi^{(k+1)}}{\Phi^{(k+1)}} \leq \alpha \cdot \varepsilon \qquad (\varepsilon = 1.E\text{-}5) \tag{3.7}$$

$$\frac{G_k - G_{k+1}}{G_{k+1}} \leq \alpha \cdot \varepsilon \qquad (\varepsilon = 1.E\text{-}5) \tag{3.8}$$

where

$$G_k = \sum_{i=1}^{m} \left| a_i^{(k)} \right|$$

If either of the inequalities above hold, then convergence is assumed, and an exit is made from ARAT to the routine for making the final conversion to monic polynomial form. If, after 20 complete iterations, the convergence criterion has not been met, then the process is terminated.

It is possible that the linear system set up in APLL will be ill-conditioned in spite of the use of Chebyshev polynomials. If the system is so badly ill-conditioned that it is impossible to obtain a solution, then APFS returns an error code to ARAT. A λ value is then added to each diagonal element of C as indicated in (1.20). The first such λ in the k^{th} iteration is computed to be a small fraction of the value of $\Phi^{(k-1)}$. If this modification still results in an ill-conditioned system, the value of λ is doubled. If after 20 attempts to modify C with λ the system is still unsolvable, the process is terminated.

At the completion of the second phase, either a solution has been found or an error code has been set. If a solution has been found, it is necessary to convert it into a form which is meaningful to the user. The coefficients at this stage correspond to the Chebyshev polynomial expansion. As discussed earlier, this form is an internal computational necessity and

usually is not very useful to the user. Therefore, the conversion indicated in (3.6) is performed to compute the coefficients corresponding to the expansion (3.3).

If an interval transformation was required earlier to transform the abscissa values to the domain, $\left[-1, +1\right]$, then the function just computed is not the approximating function for the original data but for the transformed data. Thus, if (3.1) represents the original transformation and x_i' denotes the transformed abscissa, then we have computed the approximation

$$f(x') = \frac{p(x')}{q(x')}$$

and we must compute the function

$$g(x) \equiv \frac{r(x)}{s(x)} = \frac{p(x')}{q(x')} \equiv f(x') \tag{3.9}$$

where x now denotes the original abscissa values.

Originally we were concerned only with transforming the abscissa values because there was no known function at that time which required a transformation. Now, however, we must transform the function f(x') given in (3.9). This is a more difficult task since the function f(x') contains terms of $(x')^k$ which must be expanded with (3.1) before the final coefficients corresponding to the new terms $(x)^k$ can be collected.

By inspection, it is determined that conversion (3.9) can be made by a matrix multiplication

$$U \vec{v} = \vec{w} \tag{3.10}$$

where $\vec{v}$ is a vector of coefficients corresponding to p(x'), and $\vec{w}$ is the desired coefficient vector corresponding to r(x). A second application is

also required in which $\vec{v}$ corresponds to $q(x')$, and $\vec{w}$ corresponds to $s(x)$. The matrix U is the same in both passes.

Let

$$a = 2. \qquad b = -(x_{max} + x_{min}) \qquad \text{and } c = (x_{max} - x_{min})$$

Then U takes the following form:

$$U = \begin{bmatrix} 1 & \frac{b}{c} & \frac{b^2}{c^2} & \frac{b^3}{c^3} & \cdots & \frac{b^n}{c^n} \\ 0 & \frac{a}{c} & \frac{2ab}{c^2} & \frac{3ab^2}{c^3} & \cdots & \frac{nab^n}{c^n} \\ 0 & 0 & \frac{a^2}{c^2} & \frac{3a^2b}{c^3} & & \cdot \\ & & & & & \cdot \\ 0 & 0 & 0 & \frac{a^3}{c^3} & & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdots & \frac{a^n}{c^n} \end{bmatrix}$$

and the general element u_{ij} is given by

$$u_{ij} = \binom{j-1}{i-1} \cdot \left[\frac{b}{c}\right]^{j-i} \cdot \left[\frac{a}{c}\right]^{i-1} \tag{3.11}$$

where $\binom{j-1}{i-1}$ is the binomial coefficient:

$$\binom{j-1}{i-1} = \frac{(j-1)!}{(i-1)! \cdot (j-i)!}$$

The conversion (3.9) is thus a well defined and straightforward task if U is available. The conversion comes at the very end of the system, so we may avoid having to allocate an additional array area for U by using an array area which was used earlier in the system and is no longer needed.

We can avoid having to set the lower triangular elements of U to zero by simply programming the multiplication procedure (3.10) so that it looks at upper triangular elements only.

Because (3.10) will be executed once for the numerator coefficients and once for the denominator coefficients, U must be of dimension n+1 where n is the largest power of x' appearing in either p(x') or q(x'). The constant terms of U form a Pascal triangle of order n+1 with the i^{th} column of U corresponding to the i^{th} row of a Pascal triangle. We use this fact and (3.11) to compute U with a minimum of computational time by the following algorithm:

1.) Set:

   a.) $d_0 = 1.$

   b.) $e_0 = 1.$

2.) For i equal 1 to n+1, set:

   a.) $u_{ii} = 1$

   b.) $u_{1i} = 1$

   c.) $d_i = (d_{i-1}) \cdot (b/c)$

   d.) $e_i = (e_{i-1}) \cdot (a/c)$

3.) For i equal 2 to n+1 and j equal 3 to n+1 when $j > i$, set:

   a.) $u_{ij} = (u_{i-1,j-1} + u_{i,j-1})$

4.) For i equal 1 to n+1 and j equal 1 to n+1 when $j \geq i$, set:

   a.) $u_{ij} = (u_{ij}) \cdot (d_{j-i}) \cdot (e_{i-1})$

The first application of (3.10) requires that the coefficients of p(x') be loaded into $\vec{v}$. If there are m coefficients in p(x') where m = n+1, then v_{m+1} to v_{n+1} will be set equal to zero. w_1 to w_m will then contain the

coefficients of r(x) after the matrix multiplication. In a similar manner, the denominator coefficients are computed with $\vec{v}$ being completed with zeros, if necessary.

The final procedure of the system is to print out the final coefficient values for the approximating function. These coefficients are stored internally in a vector $\vec{c}$ in case a user would like to attach a routine for further processing. The denominator coefficients are located in c_1 to c_i and the numerator coefficients in c_{i+1} to c_{i+j} where i and j represent the number of coefficients in $p(x')$ and $q(x')$, respectively.

IV. RESULTS AND CONCLUSIONS

In summarizing the results of this research, we wish to compare the proposed system for rational fraction least-squares fitting to the IBM-supplied system now in use. This discussion will cover the following areas of comparison: generality, storage requirements, user convenience, speed, and convergence reliability.

Generality, in this context, refers to the ability of a system to handle any type of data curve. Throughout this paper, it has been emphasized that the proposed system is designed to handle only a certain class of data problems, namely curves possessing positive ordinate values which exhibit one or more peaks and which approach zero at $\pm\infty$. Thus, the proposed system is not intended to be a universal system to replace the existing system but rather a supplemental system to be used when a data fitting problem of the above-mentioned class arises.

In Figure 3.1, the second phase of the proposed system is simply the existing system in its entirety. Therefore, if an installation installed the proposed system as a library routine, the existing IBM system would also be needed, and a user could call the existing system for those problems in which his data was not a peak data curve. Because of this, the proposed system does satisfy the condition of generality.

Since the existing system is a subsystem of the proposed package, the memory storage requirements are greater for the proposed system. The entire proposed system requires approximately 21,500 bytes of storage on the IBM 360/65, while the existing system requires approximately 17,000 bytes. However, this latter figure does not include any I/O interfacing

routines. The user must supply his own routines to read in the problem data and output the resulting approximating function.

It is in the discussion of these interfacing routines that one large advantage of the proposed system becomes clear. A user does not need to write any additional code and does not even have to know or understand anything about FORTRAN programming in order to use the proposed system. Because the system is a complete package, the user must add only his data cards as described in Chapter III.

The existing system, however, not only requires interfacing routines for input and output but also requires that the input conform to the interval, $[-1,1]$. This requirement stems from the use of Chebyshev polynomials in the approximating function. Thus, if a user has data taken from an interval $[a,b]$ where $a \neq -1$ or $b \neq +1$, then the user must either convert his data by hand or write another routine to accomplish this internally. The former method introduces a larger possibility of data error while the latter method requires more programming knowledge from the user. Because the proposed system allows the user to submit data taken over any interval, $[a,b]$, the user does not have to worry about data transformation at all.

The term "convergence reliability" is perhaps a somewhat misleading expression. The final phase of both systems is the gradient-type, Gauss-Newton method described in Chapter I. Because this method decreases the error norm at each step, convergence to a minimum is guaranteed. There is no guarantee, however, that the minimum will be reached in, say, k iterations, where k is some fixed upper limit independent of a particular data fitting problem. If one system converged on the solution within k itera-

tions while the other did not, then for that problem at least, the former system would possess more convergence reliability.

As discussed in Chapter II, the proposed system guarantees that the error norm of the calculated starting vector is smaller than the error norm of the starting vector of the existing system. While this measure is no guarantee as to the relative behavior of the two methods in later iterations, it does ensure that the proposed system is closer to the solution initially than the existing system.

Comparison of the speed of the two systems can be accomplished in two different ways. One way would be to measure the execution time for each system for several different representative problems. Another way would be to count the number of iterations required for convergence by each system. Both of these ways will be used in this section.

Each iteration of these two systems can involve several repeated attempts at solving an m x m linear system which computes the correction vector. After this correction vector has been successfully computed, the remaining time in each iteration is spent calculating the optimum length parameter corresponding to the change vector. Since the solution of the linear system takes up most of the execution time of each iteration, it is more accurate to count both the number of iterations required and the total number of times the linear system solver is entered.

Figures 4.1 to 4.7 show the graphs and the coordinate values for certain data curves used in comparing the two systems in the areas of speed and convergence reliability. Curves 1, 2, 3, and 7 are single-peak curves of varying steepness. Curve 1 has the exact function representation, x^2e^{-x}, and thus has a very slight peak. Curve 7 has the exact rational function

x	y
-1.	.009
- .9	.152
- .8	.368
- .7	.502
- .6	.541
- .5	.513
- .4	.448
- .3	.370
- .2	.293
- .1	.225
.1	.124
.2	.089
.3	.064
.4	.045
.5	.031
.6	.021
.7	.015
.8	.010
.9	.007
1.0	.005

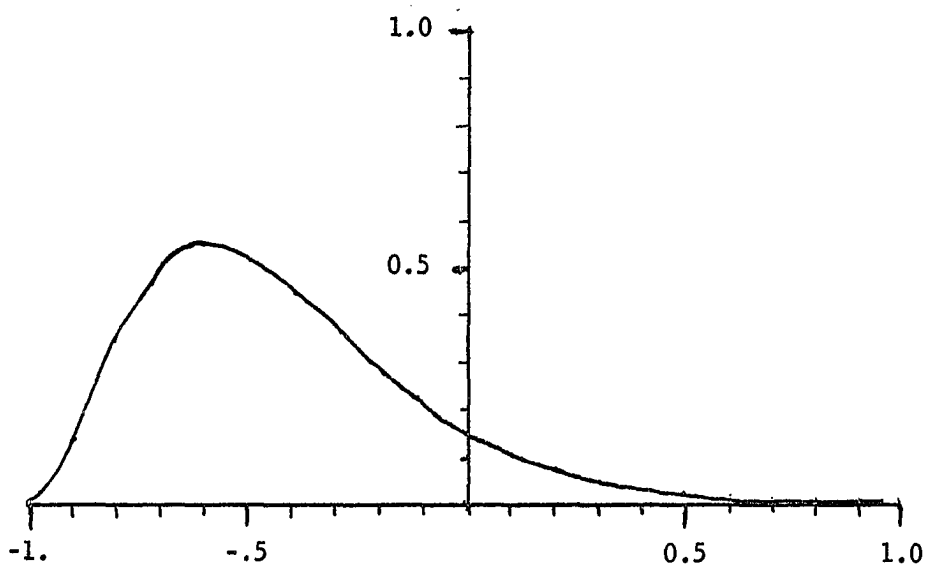


Figure 4.1. Curve 1 $f(x) = x^2 e^{-x}$

x	y
1.	2.0
2.	2.5
3.	2.9
4.	4.0
5.	6.0
6.	8.0
7.	11.0
8.	13.5
9.	16.5
10.	19.0
11.	19.0
12.	18.0
13.	14.6
14.	10.7
15.	9.0
16.	7.4
17.	6.0
18.	5.0
19.	4.2
20.	4.0

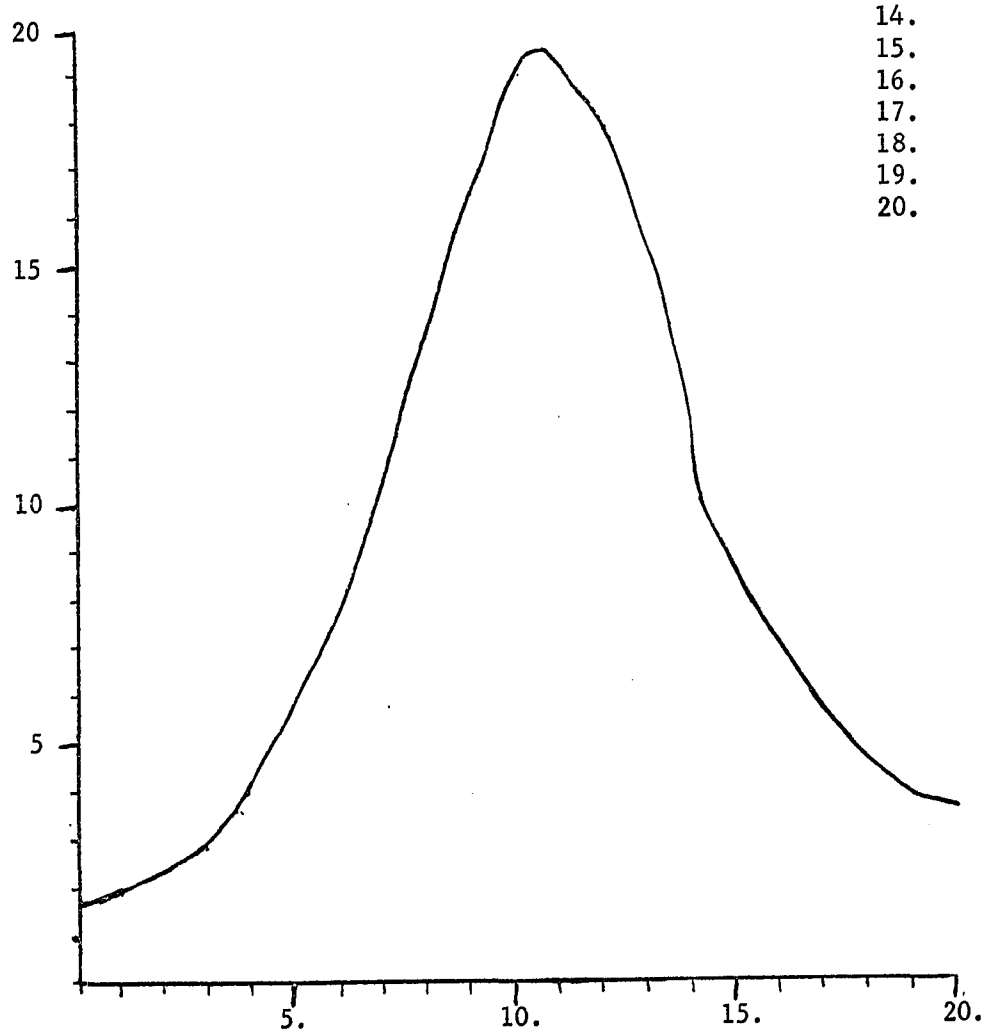
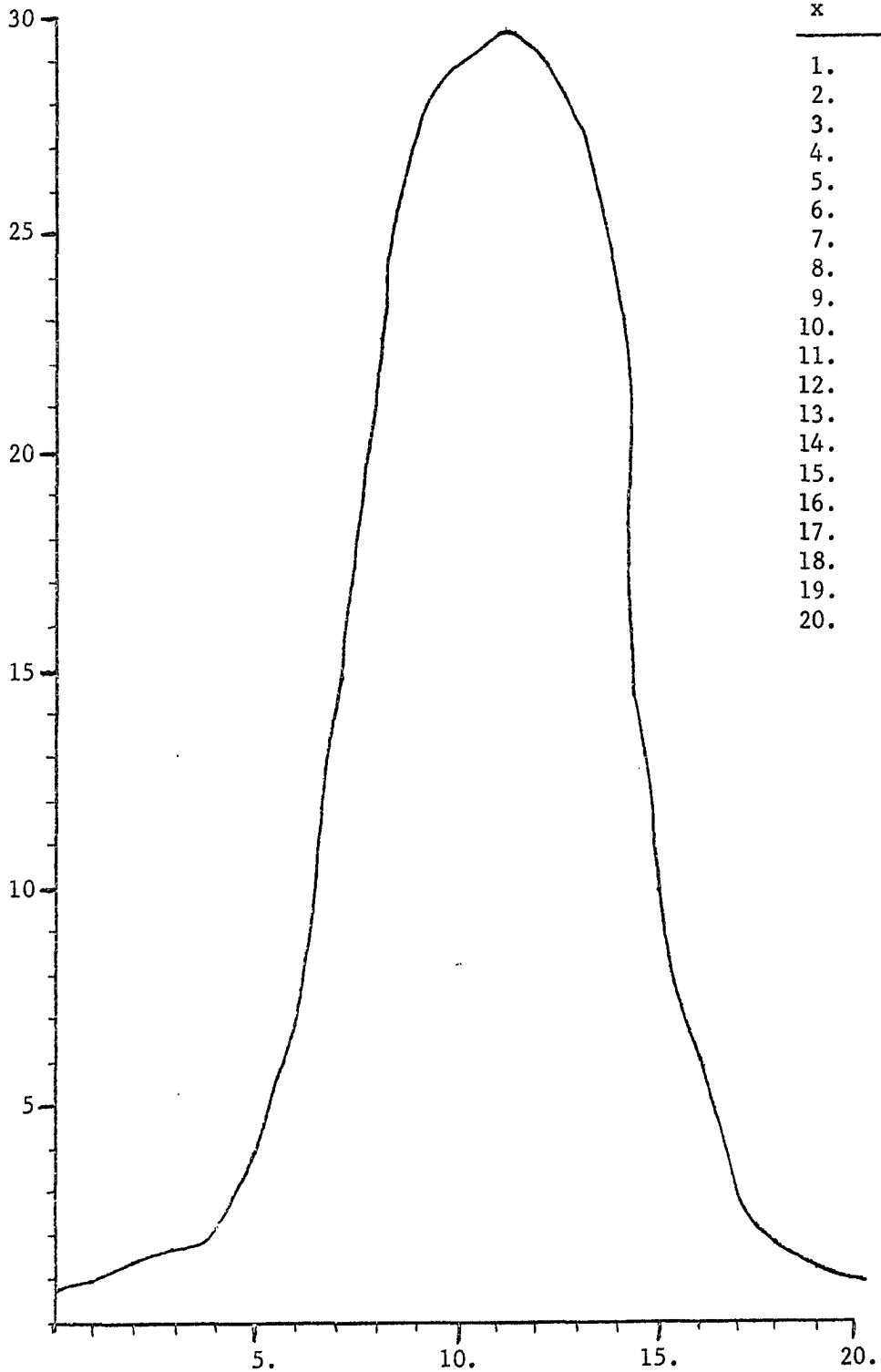


Figure 4.2. Curve 2



x	y
1.	1.0
2.	1.5
3.	1.8
4.	2.0
5.	4.0
6.	7.0
7.	15.0
8.	24.0
9.	28.0
10.	29.0
11.	30.0
12.	29.0
13.	27.0
14.	20.0
15.	10.0
16.	6.0
17.	3.0
18.	2.0
19.	1.5
20.	1.0

Figure 4.3. Curve 3

x	y
1.	1.0
2.	2.0
3.	4.0
4.	8.0
5.	12.0
6.	10.0
7.	8.0
8.	6.0
9.	5.0
10.	7.0
11.	9.0
12.	12.0
13.	15.0
14.	14.0
15.	11.0
16.	8.0
17.	6.0
18.	4.0
19.	2.0
20.	1.0

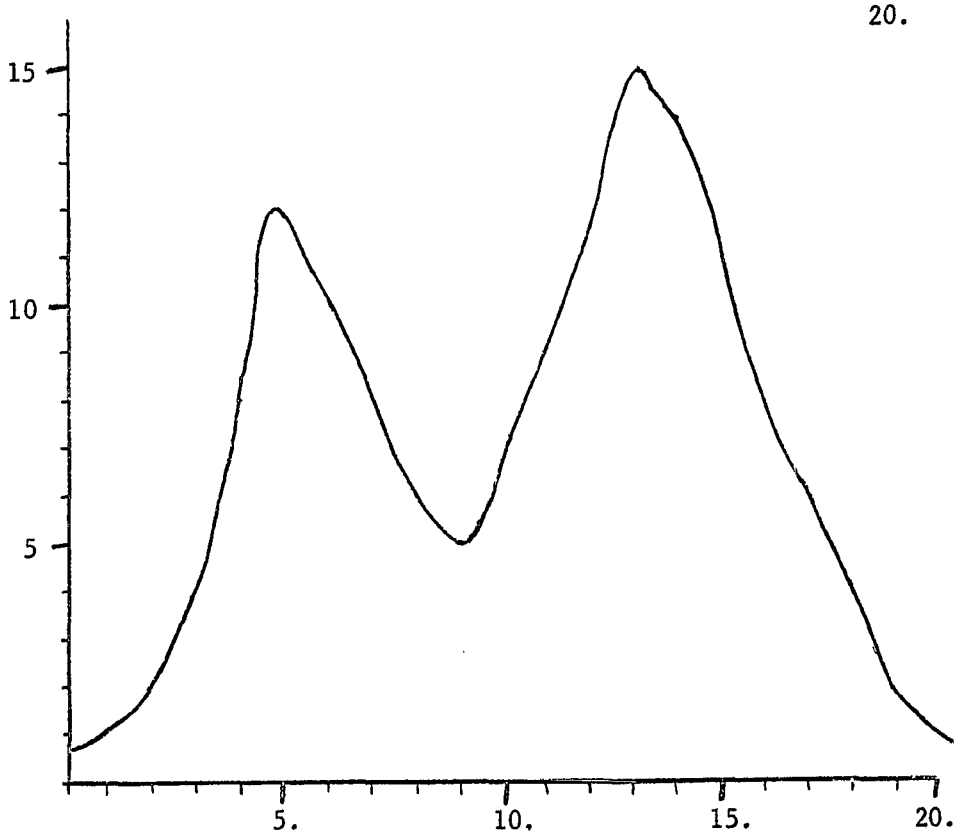
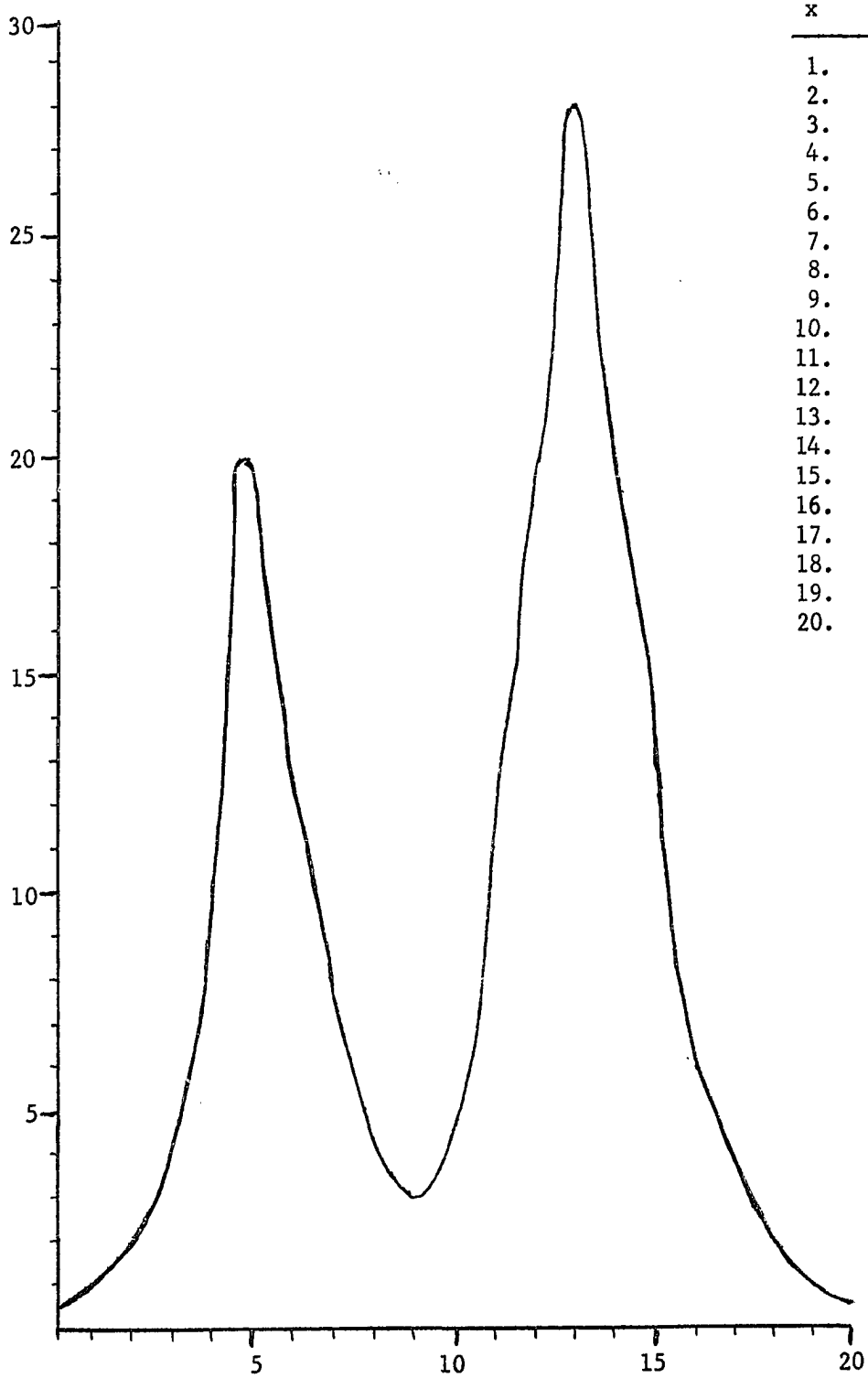


Figure 4.4. Curve 4



x	y
1.	1.0
2.	2.0
3.	5.0
4.	10.0
5.	20.0
6.	12.0
7.	8.0
8.	4.0
9.	3.0
10.	5.0
11.	13.0
12.	20.0
13.	28.0
14.	19.0
15.	12.0
16.	6.0
17.	4.0
18.	2.0
19.	1.0
20.	0.5

Figure 4.5. Curve 5

x	y
1.	1.0
2.	2.0
3.	4.0
4.	6.0
5.	7.0
6.	6.0
7.	5.5
8.	5.0
9.	6.0
10.	8.0
11.	12.0
12.	15.0
13.	16.0
14.	15.0
15.	13.0
16.	11.0
17.	8.0
18.	5.0
19.	4.0
20.	3.5

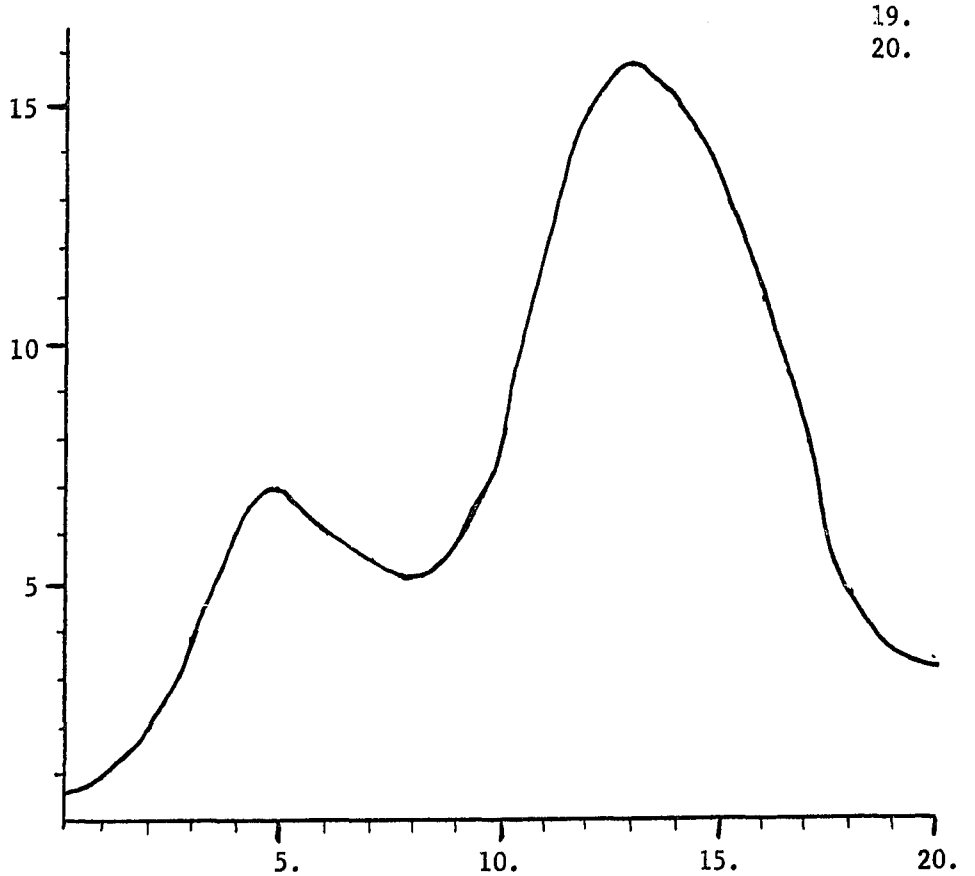


Figure 4.6. Curve 6

x	y
-10.	.051
- 9.0	.059
- 8.	.069
- 7.	.082
- 6.	.099
- 5.	.122
- 4.	.154
- 3.	.200
- 2.	.270
- 1.	.385
0.0	.588
1.	1.0
2.	2.0
3.	5.0
4.	10.0
5.	5.0
6.	2.0
7.	1.0
8.	.588
9.	.385

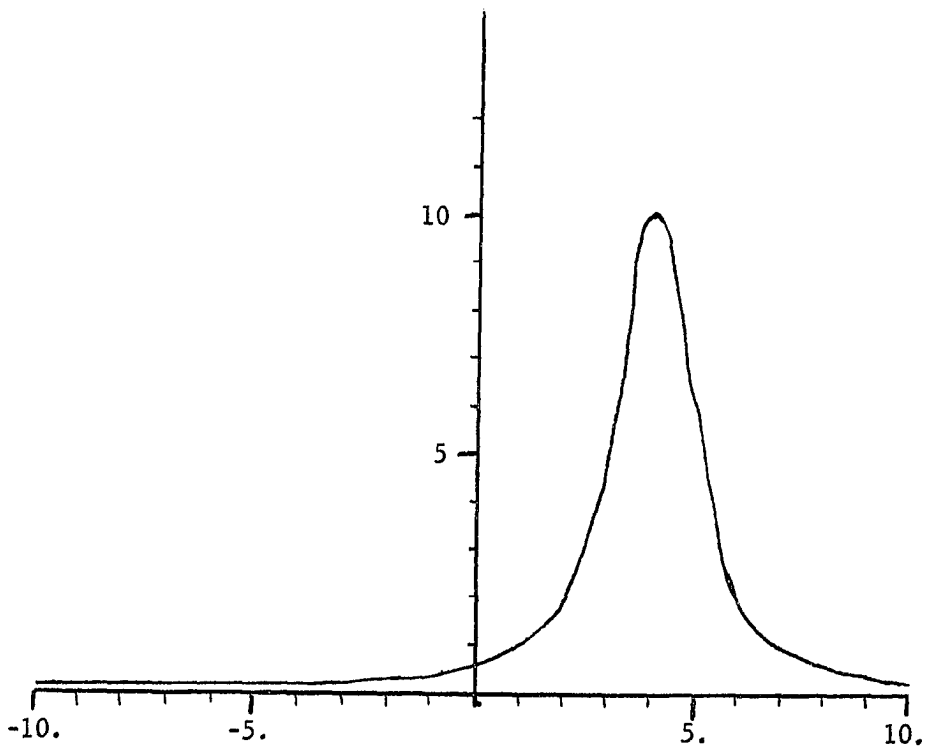


Figure 4.7. Curve 7 $f(x) = 10/((x-4)^2+1)$

representation, $10/((x-4)^2+1)$. Curves 2 and 3 were created by hand to describe larger and steeper peak situations.

Curves 4, 5, and 6 are each double-peak curves. Each curve describes a different type of double-peak data curve. Curve 5 has two peaks of approximately the same height with a very deep valley between them. Therefore, it looks like two distinct single-peak curves. Curve 4 is similar except that the valley is higher so that the curve looks more like twin peaks on top of the same trunk. Curve 6 looks like a single-peak curve with a smaller secondary peak arising from the middle of one side.

Tables 4.1 and 4.2 summarize the results obtained by testing both systems with each of the curves. System A is the existing system, and System B is the proposed system which computes a starting vector. The "Degrees of f" column indicates the degrees of the numerator and denominator in each approximating function, f.

The results for single-peak data in Table 4.1 show that in every test case except one, the proposed system required fewer total iterations for convergence. The one case in which the proposed system required as many iterations as the existing system occurred in curve 1. This is not unexpected because curve 1 contains only a slight hump and is not, therefore, a good test example for the proposed system. As curves with successively sharper and steeper peaks were tested, the proposed system became more reliable and faster in the number of iterations saved over the existing system.

The double-peak data curves shown in Figures 4.4 to 4.6 do not have the variation in size that the single-peak curves do. This is because the objective was to test the proposed system against several examples from the

Table 4.1. Results: single-peak curves

Curve no.	Method	Degree of f	Starting norm	Final norm	Entries into linear solver	No. of iterations
1	A	0/2	1.472	.0466	8	8
	B	0/2	.579	.0459	7	7
	A	1/3	1.472	.0461	8	8
	B	1/3	.370	.0456	8	8
	A	2/4	1.472	.00082	14	14
	B	2/4	.276	.00013	10	10
2	A	0/2	2314.2	9.21	6	6
	B	0/2	206.4	7.24	5	5
	A	1/3	2314.2	3.30	16	11
	B	1/3	200.1	2.62	11	11
	A	2/4	2314.2	Did not converge to solution		
	B	2/4	57.2	2.19	19	7
3	A	0/2	5523.7	194.0	9	9
	B	0/2	1023.6	194.0	6	6
	A	1/3	5523.7	170.2	20	12
	B	1/3	1013.7	170.4	10	10
	A	2/4	5523.7	4.81	11	11
	B	2/4	739.5	4.87	8	8
	A	3/5	5523.7	Did not converge to solution		
	B	3/5	735.8	3.93	28	10
7	A	0/2	161.2	.5 E-5	10	10
	B	0/2	57.8	.8 E-6	7	7
	A	1/3	161.2	Did not converge to solution		
	B	1/3	56.5	.7 E-6	28	11
	A	2/4	161.2	.5 E-5	19	12
	B	2/4	45.2	.7 E-6	19	5
	A	3/5	161.2	Did not converge to solution		
	B	3/5	31.5	.7 E-6	15	5

Table 4.2. Results: double-peak curves

Curve no.	Method	Degree of f	Starting norm	Final norm	Entries into linear solver	No. of iterations
4	A	2/4	1391.0	4.74	9	9
	B	2/4	29.3	4.74	6	6
	A	3/5	1391.0	3.72	10	9
	B	3/5	22.0	4.1	7	7
	A	4/6	1391.0	1.36	13	12
	B	4/6	11.0	1.34	9	9
5	A	2/4	2703.3	19.2	10	10
	B	2/4	144.4	19.1	6	6
	A	3/5	2703.3	17.7	12	11
	B	3/5	144.3	17.7	22	8
	A	4/6	2703.3	Did not converge to solution		
	B	4/6	65.1	7.8	18	18
6	A	2/4	1554.5	5.60	15	15
	B	2/4	50.76	5.61	12	12
	A	3/5	1554.5	Did not converge to solution		
	B	3/5	23.3	3.82	9	9
	A	4/6	1554.5	1.18	13	12
	B	4/6	21.87	1.18	20	10

different classes of double-peak curves which are possible. As Table 4.2 indicates, the proposed system did considerably better on each curve than the existing system did.

Tables 4.1 and 4.2 also record the initial error norm calculated by each system for each test. The theory of Chapter II proved that the proposed system would compute a starting vector whose error norm was always less than that of the existing system. Tables 4.1 and 4.2 show that this norm reduction did occur. As can be seen, the norm for the proposed system

was never more than 0.4 of the norm of the existing system. The best reduction was achieved in curve 4 where the norm of the proposed starting vector was 1/125 of the norm of the standard starting vector. A comparison of starting vectors for the 23 test cases showed that, on the average, the norm for the existing system was 26 times greater than the norm for the proposed system.

The execution time for each test case was about one second or less. Several curves were tested in the same machine run in order to get a larger execution time for comparison purposes. Curves 1, 2, and 3 were tested together under the existing system and required 6.8 seconds. The same curves tested under the proposed system required 5.8 seconds. This latter figure includes all the interval transformations and scanning procedures. Curves 4, 5, and 6 tested with the existing system required 11.7 seconds while the same curves took 10.0 seconds to execute under the proposed system.

To summarize, a computer system to compute the best least-squares rational function approximation for a particular class of data has been designed and implemented. The uniqueness of the system lies in its ability to create a starting vector based on the characteristics of the data being fitted. Theoretical considerations have been given to prove that the starting vector thus created has an initial error norm smaller than that of the standard starting vector in the currently available system.

Experimental results from representative data curves demonstrated the effectiveness of the proposed system. In certain examples, the proposed system converged to the correct solution while the existing system did not. In the other examples, both systems converged, and the proposed system

required fewer iterations in all but one instance in which the same number of iterations were required. Execution timings favored the proposed system as being the faster method.

The proposed system has been designed to be as complete as possible so that a minimum of additional effort is required by a user. Some of the subsystems included for this purpose include input/output routines, automatic interval transformations, and conversions from internal Chebyshev polynomial forms to standard monic polynomial forms. None of these features are included in the currently available system.

There are several areas in which further work might be attempted. One area involves a deeper study of the class of data curves for which this system is efficient. The results for single-peak data indicate that the sharper and steeper the data curve is, the better the results will be in using the proposed system. It would be helpful to have a set of criteria established by which it would be possible to determine if a particular curve belonged to the class of data curves for which the proposed system was designed.

No attempt was made to approximate data containing more than two peaks. Techniques similar to those used in double-peak curves could be investigated for multiple-peak curves. A study of multiple-peak curves might result in a better procedure for computing the starting vectors for double-peak curves.

A final area for study would be to convert the system into an interactive system. The input routine is currently designed to facilitate the addition or deletion of data points if the user would want to do so on successive test runs. An interactive system would also enable the user to

observe the norm values from iteration to iteration if he wished and allow him to switch to a higher degree approximating function if necessary.

If a user in an interactive environment desired to look at the best solution for several functions of different degrees, then it might prove more efficient to choose the solution of the first or lowest degree function as a starting vector for the next higher degree function.

These are a few of the possible directions that future research could follow. Further study of these techniques may suggest other areas for investigation.

V. BIBLIOGRAPHY

1. ACTON, F. S. Numerical Methods that Work. Harper and Row, New York, 1970.

2. BARD, Y. Comparison of gradient methods for the solution of nonlinear parameter estimation problems. SIAM J. Numer. Anal. 7, 1 (March 1970), 157-187.

3. BERZTISS, A. T. Least squares fitting of polynomials to irregularly spaced data. SIAM Review 6, 3 (July 1964), 203-227.

4. FLETCHER, R. and POWELL, M. J. D. A rapidly covergent method for minimization. Comput. J. 6 (1963), 163-168.

5. GOLDSTEIN, A. A. Cauchy's method of minimization. Num. Math. 4 (1962), 146-150.

6. HARTLEY, H. O. The modified Gauss-Newton method. Technometrics 3, 2 (May 1961), 269-280.

7. IBM CORPORATION. System/360 scientific subroutine package (360A-CM-03X) Version III programmer's manual. White Plains, N.Y., 1968.

8. MARQUARDT, D. W. An algorithm for least-squares estimation of nonlinear parameters. SIAM J. 2 (1963), 431-441.

9. MEETER, D. A. On a theorem used in nonlinear least squares. SIAM J. Appl. Math. 14, 5 (September 1966), 1176-1179.

10. MORRISON, D. D. Methods for nonlinear least squares problems and convergence proofs. Proc. Jet Propulsion Lab. Seminar: Tracking Problems and Orbin Determination, 1960, 1-9.

11. POWELL, M. J. D. A survey of numerical methods for unconstrained optimization. SIAM Review 12, 1 (January 1970), 79-97.

12. SHANNO, D. F. An accelerated gradient projection method for linearly constrained nonlinear estimation. SIAM J. Appl. Math. 18, 2 (March 1970), 322-334.

13. WALSH, J. (Ed.). Numerical Analysis: An Introduction. Thompson Book Company, Inc., Washington, D.C., 1966.

VI. ACKNOWLEDGMENTS

I wish to express my sincere thanks and appreciation to Dr. R. J. Lambert for his guidance and assistance during this research work as well as during my course of study at Iowa State University.

Appreciation is also extended to Mrs. Carla Jacobson for her typing of the manuscript.

VII. APPENDIX: PROGRAM LISTING

```
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION PEAK(10),VALLEY(10),X(300),WORK(600),C(20),B(8),U(8,8)
      DIMENSION DD(8),EU(8)
      EQUIVALENCE (U(1,1),WORK(1))
      EXTERNAL FRAT2
      P(A) = 1.
      Q(A) = 1.
      EPZ=1.D-5
      ETA = 1.D-11
      LR=5
      LW=6

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         RUUTINE TO INPUT DATA CURVE FOR A LEAST-SQUARES FIT
C         N=NUMBER OF DATA POINTS
C         IP = DESIRED NUMBER OF NUMERATOR PARAMETERS
C         IQ = DESIRED NUMBER OF DENUMINATOR PARAMETERS
C         IW = 1 DENOTES WEIGHTING VALUES WILL BE USED
C         IW = 0 DENOTES NO WEIGHTING VALUES
C         X = VECTOR OF LENGTH 3*N CONTAINING:
C                                ABSCISSA VALUES IN POS. 1 TO N
C                                ORDINATE VALUES IN POS. N+1 TO 2*N
C                                WEIGHTING VALUES IN POS. 2*N+1 TO 3*N
C         IF IW = 0 THEN X(2*N+1) TO X(3*N) IS SET TO 1'S
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      READ (LR,900)  N,IP,IQ,IW
  900 FORMAT (I3,3I1)
      WRITE (LW,901) N,IP,IQ
```

```
  901 FORMAT ('1NO. OF POINTS =',I3,' NO. OF NUMERATOR COEFF. = ',I1,
     1    '  NO. OF DENOM. COEFF. = ',I1)
      IF (IW.EQ.0) WRITE (LW,902)
      IF (IW.EQ.1) WRITE (LW,903)
  902 FORMAT (' WEIGHTING VALUES ARE NOT PRESENT')
  903 FORMAT (' WEIGHTING VALUES ARE PRESENT')
      NN=N+1
      N2=N+N
      NNN=NN+N
      N2N=N2+N
      DO 30 I=1,N
      READ (LR,910) X(I),X(I+N),X(I+N+N)
  910 FORMAT (3F15.7)
   30 CONTINUE
      IF (IW.NE.0) GO TO 50
      DO 40 I=1,N
      X(I+N+N)=1.
   40 CONTINUE
   50 CONTINUE

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO SORT THE ABSCISSA VALUES IN ASCENDING ORDER
C         METHOD USED IS A REVERSE BUBBLE SORT
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      NM=N-1
   60 IEX=0
      DO 80 I=1,NM
      IF (X(N-I+1).GE.X(N-I)) GO TO 80
      IEX=1
      DO 70 J=1,3
      TEMP=X(J*N-I+1)
      X(J*N-I+1)=X(J*N-I)
   70 X(J*N-I)=TEMP
```

```
   80 CONTINUE
      NM=NM-1
      IF (IEX.EQ.1) GO TO 60
      WRITE (LW,918)
  918 FORMAT ('0ABSCISSA VALUES:')
      WRITE (LW,920) (X(I),I=1,N)
      WRITE (LW,919)
  919 FORMAT ('0ORDINATE VALUES:')
      WRITE (LW,920) (X(I+N),I=1,N)
  920 FORMAT (5(F15.7,3X))

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO TRANSFORM ABSCISSA TO INTERVAL -1 TO 1 INCLUSIVE
C         X(I) = (2*X(I)-(XMAX+XMIN))/(XMAX-XMIN) WHERE
C                    XMAX = X(N) = LARGEST ABSCISSA VALUE    AND
C                    XMIN = X(1) = SMALLEST ABSCISSA VALUE
C         IF X(1) = -1 AND X(N) = 1 THEN ROUTINE IS BYPASSED
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      XNP=X(N)+X(1)
      XDM=X(N)-X(1)
      IF (X(1).EQ.-1..AND.X(N).EQ.1.) GO TO 149
      DO 120 I=1,N
      X(I) = (2.*X(I)-XNP)/XDM
  120 CONTINUE
  149  CONTINUE
      WRITE (LW,921)
  921 FORMAT ('0TRANSFORMED ABSCISSA VALUES')
      WRITE (LW,920) (X(I),I=1,N)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO FIND THE PEAKS AND VALLEYS OF THE DATA CURVE
C         JP = NUMBER OF PEAKS FOUND
```

```
C         JV = NUMBER OF VALLEYS FOUND
C         PEAK = VECTOR CONTAINING ABSCISSA VALUES OF THE PEAK LOCATIONS
C         VALLEY = VECTOR CONTAINING ABSCISSA VALUES OF VALLEY LOCATIONS
C         AN INTERPOLATION FORMULA IS USED TO MORE ACCURATELY LOCATE
C              PEAKS AND VALLEYS WHICH FALL BETWEEN TWO GIVEN ABSCISSAS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      D1=1.
      JP=0
      JV=0
      DO 190 L=2,N
      D=D1
      D1 = X(L+N)-X(L+N-1)
      H = X(L)-X(L-1)
      IF (D1.NE.0..AND.D1*D.GE.0.) GO TO 190
      E = ((D+D1)/(2.*DSQRT(D**2+D1**2)))*H+X(L-1)
      IF (D1.GT.D) GO TO 180
      JP=JP+1
      PEAK(JP)=E
      GO TO 190
  180 JV=JV+1
      VALLEY(JV)=E
      VALLEY(JV+5) = L-1
  190 CONTINUE
      IF (JP.NE.0) WRITE (LW,924) (PEAK(I),I=1,JP)
      IF (JV.NE.0) WRITE (LW,925) (VALLEY(I),I=1,JV)
  924 FORMAT ('0PEAK LOCATIONS ',5(F15.7,3X))
  925 FORMAT ('0VALLEY LOCATIONS ',5(F15.7,3X))

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO COMPUTE THE BEST EPSILON FOR THE FACTOR
C                (X-B)**2 + EPSILON IN THE DENOMINATOR TERM
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
      IB=1
      IN=N
      IF (JP.EQ.2) IN=VALLEY(6)
      K=1
  415 B1=PEAK(K)
      T1=1.E+6
      DO 425 I=IB,IN
      XX=X(I)
      E1=P(XX)/(Q(XX)*X(I+N))-(XX-B1)**2
      IF (E1.LT.0) GO TO 425
      P1=0.
      DO 420 J=IB,IN
      XX=X(J)
  420 P1=P1+X(J+N2)*(X(J+N)-P(XX)/(Q(XX)*((XX-B1)**2+E1)))**2
      IF (P1.GE.T1) GO TO 425
      T1=P1
      TE1=E1
      IT1=I
  425 CONTINUE
      WRITE (LW,960) TE1,IT1
      E1=TE1
      IF (IN.EQ.N) GO TO 1451
      K=2
      B0=B1
      E0=TE1
      IT0=IT1
      IB=IN+1
      IN=N
      GO TO 415
  960 FORMAT ('0BEST EPS =',E12.5,3X,'AT DATA POINT NUMBER',I3)
 1451 CONTINUE
 1920 FORMAT (1X,2E15.7)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO COMPUTE BEST LINEAR APPROXIMATION WITH
```

```
C                       THE GIVEN EPS AND BO
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      IQQ=IQ+1
      IPQ = IP+IQ
      DO 452 I=1,N
      X(N2+I)=((X(I)-B1)**2+E1)
      IF (JP.EQ.2) X(N2+I)=X(N2+I)*((X(I)-BO)**2+EO)
  452 CONTINUE
      INUMF = 1
      CALL DAPLL (FRAT2,N,IP,WORK,WORK(N2),X,IERV)
      CALL DAPFS (WORK(N2),IP,IRES,1,EPZ,ETA,IER)
      IF (IRES.EQ.IP) GO TO 453
      INUMF = 0
      WRITE (LW,961) IER,IRES
  961 FORMAT ('OIER=',I2,' IRES=',I2)

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         PREPARE INPUT FOR MONICH CONVERSION
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

  453 NY = IP*(IP-1)/2 + N2
      DO 454 I=1,IP
      C(IQ+I)=WORK(NY+I-1)
  454 CONTINUE
      DO 470 I=1,IQ
  470 C(I)=0.
      IF (JP.EQ.2) GO TO 1470
      C(1)= E1+B1*B1
      C(2)=-2.*B1
      C(3)=1.
      GO TO 1471
 1470 B12=BO*BO+EO
```

```
      B22 = B1*B1+E1
      C(1)=B12*B22
      C(2)=-2.*(B0*B22+B1*B12)
      C(3) = B12+B22+4.*B0*B1
      C(4)=-2.*(B0+B1)
      C(5) = 1.
 1471 CONTINUE
      CALL MONICH (IQ,C,C,1,IER)
      IF (INUMF.EQ.1) GO TO 472
      C(IQ+1)=1.
      DO 460 I=2,IP
  460 C(IQ+I)=0.
      CALL MONICH (IP,C(IQ+1),C(IQ+1),1,IER)
  472 CONTINUE
      WRITE (LW,809)
  809 FORMAT ('0STARTING VECTOR, NUMERATOR')
      WRITE (LW,920) (C(I),I=IQQ,IPQ)
      WRITE (LW,808)
  808 FORMAT ('0STARTING VECTOR, DENOMINATOR')
      WRITE (LW,920) (C(I),I=1,IQ)
      IER=1

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ENTER PHASE II FOR FINAL CONVERGENCE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      IF (IW.EQ.0) X(N+N+1)=0
      CALL DARAT (X,N,WORK,C,IP,IQ,IER)
      IF (IER.NE.0) WRITE (LW,810) IER
  810 FORMAT ('  IER= ',I2)
  805 FORMAT (10X,9F9.4)
      CALL MONICH (IQ,C,C,0,IER)
      CALL MONICH (IP,C(IQQ),C(IQQ),0,IER)
      IF (XNP.EQ.0..AND.XDM.EQ.2.) GO TO 790
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO TRANSFORM F(X') TO G(X) WHERE X' IS IN THE
C         INTERVAL (-1,+1) AND X IS IN THE INTERVAL OF THE ORIGINAL
C         ABSCISSA VALUES.
C         TRANSFORMATION IS DONE BY U*V = W WHERE U IS THE TRANS-
C         FORMATION MATRIX, V IS A VECTOR OF NUMERATOR OR DENOM-
C         INATOR COEFFICIENTS OF F(X') AND W IS THE RESULTING
C         VECTOR OF NUMERATOR OR DENOMINATOR COEFFICIENTS FOR G(X).
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      BU=-XNP/XDM
      AU=2./XDM
      DU(1)=1.
      EU(1)=1.
      DO 1100 I=1,IQQ
      U(I,I)=1.
      U(1,I)=1.
      DU(I+1)=DU(I)*BU
 1100 EU(I+1)=EU(I)*AU
      DO 1110 J=3,IQQ
      JJ=J-1
      DO 1110 I=2,JJ
 1110 U(I,J)=U(I-1,JJ)+U(I,JJ)
      DO 1120 J=1,IQQ
      DO 1120 I=1,J
 1120 U(I,J)=U(I,J)*DU(J-I+1)*EU(I)
      DO 1150 I=1,IQ
      XC = 0.
      DO 1150 J=I,IQ
      XC=XC+U(I,J)*C(J)
 1150 C(I)=XC
      DO 1160 I=1,IP
      XC=0.
      DO 1160 J=I,IP
```

```
      XC=XC+U(I,J)*C(J+IQ)
 1160 C(I+IQ)=XC
  790 WRITE (LW,820)
  820 FORMAT (' FINAL NUMERATOR COEFFICIENTS')
      WRITE (LW,920) (C(I),I=IQQ,IPQ)
      WRITE (LW,821)
  821 FORMAT (' FINAL DENOMINATOR COEFFICIENTS')
      WRITE (LW,920) (C(I),I=1,IQ)
  799 STOP
      END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO CONVERT MONIC FORM TO CHEBYSHEV FORM OR VICE VERSA
C         D IS A FIXED CONVERSION MATRIX
C         C IS THE INPUT VECTOR OF COEFFICIENTS
C         B IS THE OUTPUT VECTOR OF COEFFICIENTS
C         IC = 1 MEANS CONVERT MONIC TO CHEBYSHEV FORM
C         IC = 0 MEANS CONVERT CHEBYSHEV TO MONIC FORM
C         K IS THE ORDER OF THE INPUT VECTOR C AND MUST BE < 9
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      SUBROUTINE MONICH  (K,C,B,IC,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION D(8,9),C(K),B(K)
      DATA D/1.,0.,-1.,0.,1.,0.,-1.,0.,1.,1.,0.,-3.,0.,5.,0.,-7.,0.,
     1  1.,2.,0.,-8.,0.,18.,0.,.5,0.,.5,4.,0.,-20.,0.,56.,0.,.75,
     3  0.,.25,8.,0.,-48.,0.,.375,0.,.5,0.,.125,16.,0.,-112.,0.,
     4  .625,0.,.3125,0.,.0625,32.,0.,.3125,0.,.46875,0.,.1875,
     5  0.,.03125,64.,0.,.546875,0.,.328125,0.,.109375,0.,.015625/
      IF (K.GT.8) GO TO 600
  200 IF (IC.EQ.1) GO TO 300
      DO 260 I=1,K
      BB=0.
      DO 250 J=1,K
```

```
  250 BB=BB+D(J,I)*C(J)
  260 B(I)=BB
      GO TO 360
  300 DO 355 I=1,K
      BB=0.
      DO 350 J=I,K
  350 BB=BB+D(I,J+1)*C(J)
  355 B(I)=BB
  360 IER=0
      RETURN
  600 IER = 1
      RETURN
      END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         ROUTINE TO COMPUTE VALUES OF FUNDAMENTAL FUNCTIONS P(J) FOR
C         USE IN SETTING UP NORMAL EQUATIONS
C              P(J) = T(J)(X(I))     J = 1,2,...,IP
C                                    I = 1,2,...,N
C           T(J)(X(I)) IS THE JTH CHEBYSHEV POLYNOMIAL EVALUATED AT X(I)
C         THE CHEBYSHEV POLYNOMIALS ARE COMPUTED BY THE RECURSIVE FUNC.:
C              T(N+1)(X) = 2.*X*T(N)(X)-T(N-1)(X)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      SUBROUTINE FRAT2 (I,N,IP,P,DATI,WGT,IER)

      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION P(1),DATI(1),IER(1)
      IER(1)=0
      WGT=1.
      DE = DATI(2*N+I)
      DX = DATI(I)
      P(1)=1./DE
      P(2)=DX/DE
      IF (IP.LT.3) GO TO 210
```

```
      DO 200 J=3,IP
      P(J) = (2.*DX*P(J-1)-P(J-2))
  200 CONTINUE
  210 P(IP+1)=DATI(N+I)
      RETURN
      END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C         THE FOLLOWING ROUTINES COMPRISE PHASE 2 OF THE ITERATION
C         PROCESS.  FOR A DETAILED DESCRIPTION OF THE OPERATION OF
C         EACH, CONSULT THE IBM/360 SCIENTIFIC SUBROUTINE PACKAGE
C         VERSION III PROGRAMMER'S MANUAL: H 20-0205-3
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

      SUBROUTINE DARAT (DATI,N,WORK,P,IP,IQ,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      EXTERNAL DFRAT
      DIMENSION IERV(3)
      DIMENSION DATI(1),WORK(1),P(1)
      LIMIT = 20
      KSS=0
      KSG=0
      ETA=1.E-6
      EPS=1.E-5
      IF (N)4,4,1
    1 IF (IP)4,4,2
    2 IF(IQ)4,4,3
    3 IPQ=IP+IQ
      IF (N-IPQ)4,5,5
    4 IER=-1
      RETURN
    5 KOUNT=0
      IERV(2)=IP
      IERV(3)=IQ
```

```
      NDP=N+N+1
      NNE=NDP+NDP
      IX=IPQ-1
      IQP1=IQ+1
      IRHS=NNE+IPQ*IX/2
      IEND=IRHS+IX
      IF(IER)8,6,8
    6 DO 7 I=2,IPQ
    7 P(I)=0.
      P(1)=1.
    8 DO 9 J=1,N
      T=DATI(J)
      I=J+N
      CALL DCNPS(WORK(I),T,P(IQP1),IP)
      K=I+N
    9 CALL DCNPS(WORK(K),T,P,IQ)
   10 CALL DAPLL(DERAT,N,IX,WORK,WORK(IEND+1),DATI,IERV)
      IF(IERV(1))4,11,4
   11 INCR=0
      RELAX=2.
   12 J=IEND
      DO 13 I=NNE,IEND
      J=J+1
   13 WORK(I)=WORK(J)
      IF (KOUNT+INCR) 14,14,15
   14 OSUM=WORK(IEND)
      DIAG=OSUM*EPS
      K=IQ
      IF(WORK(NNE))17,17,19
   15 IF(INCR)19,19,16
   16 K=IPQ
   17 J=NNE-1
      DO 18 I=1,K
      WORK(J)=WORK(J)+DIAG
   18 J=J+I
   19 CONTINUE
```

```
      CALL DAPFS(WORK(NNE),IX,IRES,1,EPS,ETA,IER)
      KSS=KSS+1
      IF(IRES)4,4,20
   20 IF(IRES-IX)21,24,24
   21 IF(INCR)22,22,23
   22 DIAG=DIAG*0.125
   23 DIAG=DIAG+DIAG
      INCR=INCR+1
      RELAX=8.
      IF(INCR-LIMIT)12,45,45
   24 L=NDP
      KSG=KSG+1
      J=NNE+IRES*(IRES-1)/2-1
      K=J+IQ
      WORK(J)=0.
      IRQ=IQ
      IRP=IRES-IQ+1
      IF(IRP)25,26,26
   25 IRQ=IRES+1
   26 DO 29 I=1,N
      T=DATI(I)
      WORK(I)=0.
      CALL DCNPS(WORK(I),T,WORK(K),IRP)
      M=L+N
      CALL DCNPS (WORK(M),T,WORK(J),IRQ)
      IF(WORK(M)*WORK(L))27,29,29
   27 SUM=WORK(L)/WORK(M)
      IF (RELAX+SUM)29,29,28
   28 RELAX=-SUM
   29 L=L+1
      SSOE=OSUM
      ITER=LIMIT
   30 SUM=0.
      RELAX=RELAX*0.5
      DO 32 I=1,N
      M=I+N
```

```
      K=M+N
      L=K+N
      SAVE=DATI(M)-(WORK(M)+RELAX*WORK(I))/(WORK(K)+RELAX*WORK(L))
      SAVE=SAVE*SAVE
      IF(DATI(NDP))32,32,31
   31 SAVE=SAVE*DATI(K)
   32 SUM=SUM+SAVE
      IF(ITER)45,33,33
   33 ITER=ITER-1
      IF(SUM-USUM)34,37,35
   34 USUM=SUM
      GO TO 30
   35 IF(USUM-SSUE)36,30,30
   36 RELAX=RELAX+RELAX
   37 T=0.
      SAVE=0.
      K=IRES+1
      DO 38 I=2,K
      J=J+1
      T=T+DABS(P(I))
      P(I)=P(I)+RELAX*WORK(J)
   38 SAVE = SAVE+DABS(P(I))
      DO 39 I=1,N
      J=I+N
      K=J+N
      L=K+N
      WORK(J)=WORK(J)+RELAX*WORK(I)
   39 WORK(K)=WORK(K)+RELAX*WORK(L)
      IF(INCR)40,40,42
   40 IF(SSUE-USUM-RELAX*EPS*USUM)46,46,41
   41 IF (DABS(T-SAVE)-RELAX*EPS*SAVE) 46,46,42
   42 IF(USUM-ETA*SAVE)46,46,43
   43 KOUNT=KOUNT+1
      IF(KOUNT-LIMIT)10,44,44
   44 IER=2
      RETURN
```

```
 45 IER=1
    RETURN
 46 IER=0
    WRITE (6,452) KSS
452 FORMAT (' APFS ENTERED ',I3,' TIMES')
    WRITE (6,453) KSG
453 FORMAT (' APFS SUCCESSFUL ',I3,' TIMES')
    WRITE (6,454) USUM
454 FORMAT ('UFINAL NORM=',E15.7)
    RETURN
    END
    SUBROUTINE DAPLL(FFCT,N,IP,P,WORK,DATI,IER)
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION P(1),WORK(1),DATI(1),IER(1)
    IF(N)10,10,1
  1 IF (IP)10,10,2
  2 IF(N-IP)10,3,3
  3 IPP1=IP+1
    M=IPP1*(IP+2)/2
    IER(1)=0
    DO 4 I=1,M
  4 WORK(I)=0.
    DO 8 I=1,N
    CALL FFCT(I,N,IP,P,DATI,WGT,IER)
    IF(IER(1))9,5,9
  5 J=0
    DO 7 K=1,IPP1
    AUX=P(K)*WGT
    DO 6 L=1,K
    J=J+1
  6 WORK(J)=WORK(J)+P(L)*AUX
  7 CONTINUE
  8 CONTINUE
  9 RETURN
 10 IER(1)=-1
    RETURN
```

```
      END
      SUBROUTINE DAPFS(WORK,IP,IRES,IOP,EPS,ETA,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION WORK(1)
      IRES=0
      IF(IP)1,1,2
    1 IER=-1
      RETURN
    2 IPIV=0
      IPP1=IP+1
      IER=1
      ITE=IP*IPP1/2
      IEND=ITE+IPP1
      TOL=DABS(EPS*WORK(1))
      TEST=DABS(ETA*WORK(IEND))
      DO 11 I=1,IP
      IPIV=IPIV+I
      JA=IPIV-IRES
      JE=IPIV-1
      JK=IPIV
      DO 9 K=I,IPP1
      SUM=0.
      IF(IRES)5,5,3
    3 JK=JK-IRES
      DO 4 J=JA,JE
      SUM=SUM+WORK(J)*WORK(JK)
    4 JK=JK+1
    5 IF(JK-IPIV)6,6,8
    6 SUM=WORK(IPIV)-SUM
      IF(SUM-TOL)12,12,7
    7 SUM=DSQRT(SUM)
      WORK(IPIV)=SUM
      PIV=1./SUM
      GO TO 9
    8 SUM=(WORK(JK)-SUM)*PIV
      WORK(JK)=SUM
```

```
 9 JK=JK+K
   WORK(IEND)=WORK(IEND)-SUM*SUM
   IRES=IRES+1
   IADR=IPIV
   IF(IOP)10,11,11
10 IF(WORK(IEND)-TEST)13,13,11
11 CONTINUE
   IF(IOP)12,22,12
12 IF(IOP)14,23,14
13 IER=0
14 IPIV=IRES
15 IF(IPIV)23,23,16
16 SUM=0.
   JA=ITE+IPIV
   JJ=IADR
   JK=IADR
   K=IPIV
   DO 19 I=1,IPIV
   WORK(JK)=(WORK(JA)-SUM)/WORK(JJ)
   IF(K-1)20,20,17
17 JE=JJ-1
   SUM=0.
   DO 18 J=K,IPIV
   SUM=SUM+WORK(JK)*WORK(JE)
   JK=JK+1
18 JE=JE+J
   JK=JE-IPIV
   JA=JA-1
   JJ=JJ-K
19 K=K-1
20 IF(IOP/2)21,23,21
21 IADR=IADR-IPIV
   IPIV=IPIV-1
   GO TO 15
22 IER=0
23 RETURN
```

```
      END
      SUBROUTINE DCNPS(Y,X,C,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION C(1)
      IF(N)1,1,2
    1 RETURN
    2 IF(N-2) 3,4,4
    3 Y=C(1)
      RETURN
    4 ARG=X+X
      H1=0.
      H0=0.
      DO 5 I=1,N
      K=N-I
      H2=H1
      H1=H0
    5 H0=ARG*H1-H2+C(K+1)
      Y=0.5*(C(1)-H2+H0)
      RETURN
      END
      SUBROUTINE DFRAT(I,N,M,P,DATI,WGT,IER)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION P(1),DATI(1),IER(1)
      IP=IER(2)
      IQ=IER(3)
      IQM1=IQ-1
      IPQ=IP+IQ
      T=DATI(I)
      J=I+N
      F=DATI(J)
      FNUM=P(J)
      J=J+N
      WGT=1.
      IF(DATI(2*N+1))2,2,1
    1 WGT=DATI(J)
    2 FDEN=P(J)
```

```
      F=F*FDEN-FNUM
      IF(FDEN)4,3,4
    3 IER(1)=1
      RETURN
    4 WGT=WGT/(FDEN*FDEN)
      FNUM=-FNUM/FDEN
      J=IQM1
      IF(IP-IQ)6,6,5
    5 J=IP-1
    6 CALL DCNP (P(IQ),T,J)
    7 IF(IQM1)10,10,8
    8 DO 9 II=1,IQM1
      J=II+IQ
    9 P(II)=P(J)*FNUM
   10 P(IPQ)=F
      IER(1)=0
      RETURN
      END
      SUBROUTINE DCNP(Y,X,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(1)
      Y(1)=1.
      IF(N)1,1,2
    1 RETURN
    2 Y(2)=X
      IF(N-1)1,1,3
    3 F=X+X
      DO 4 I=2,N
    4 Y(I+1)=F*Y(I)-Y(I-1)
      RETURN
      END
```