SOLVING SYSTEMS OF EQUATIONS

WITH

APPLICATIONS TO OPTIMIZATION

Linda C. Collins

Consulting Professor: Dr. C. Bass

ACKNOWLEDGMENTS

The author wishes to thank her major Professor, Dr. Charles Bass for his kind encouragement, advice, and patience.

TABLE OF CONTENTS

CHAPTER I

AN INTRODUCTION AND OVERVIEW

Optimization is a topic of intense current interest to mathematicians and computer scientists with broad potential for application in government and industry. The importance of maximum efficency at minimum cost concerns all levels of operations and is just one of the many uses of optimization techniques. The translating of information into mathematical functions by applied mathematicians, with the assistance of modern computers, has made the analysis of various situations a much more manageable and useful instrument for the prosperity and enlightenment of our generation.

Nevertheless, the preponderance of research conducted on such a useful implement as optimization is impossible to cover within one prospectus. Therefore, the scope of this research has been focused on the investigation of iterative methods when applied to the solution of systems of equations. In general, all optimization techniques rely on iterations, for which convergence theory is incomplete and quite difficult to apply in multivariable cases.

The problem of optimization entails the analysis of a differentiable function of several variables. It is necessary to determine the values of the n unconstrained variables $x_1, x_2, \ldots, x_n$ which make the function f a maximum or a minimum. For a given function $f(x_1, x_2, \ldots, x_n)$, the maxima and minima can only occur where the n equations $\frac{\partial f}{\partial x_i} = 0$, $i=1,2,\ldots n$ are simultaneously satisfied, or at a boundary point of the domain of the variables. [5]

Thus, it does seem reasonable to consider the solution of systems of

equations. As a particular example, one can consider the following function:

$$f(x,y) = xy - \frac{x^3}{3} - \frac{y^3}{3} + 10. \qquad (1)$$

By taking partial derivatives, the following system is obtained:

$$f_x(x,y) = y - x^2 = 0, \qquad f_y(x,y) = x - y^2 = 0. \qquad (2)$$

Simple algebra will secure two solutions to the system, (0,0) and (1,1). It can be shown by analytical techniques that f(1,1) is a maximum and that (0,0) does not produce an extremum. [6]

Sometimes it is useful to transform a system such as (2) into an equilvalent fixed point problem. For example the point (1,1) is a fixed point for the function g, defined by the rule:

$$g(x,y) = (u,v)$$

$$u = x + \frac{x-2yx^2+y^2}{4xy-1} \qquad v = y + \frac{y-2xy^2+x^2}{4xy-1}. \qquad (3)$$

Thus an iterative scheme for approximating the solution might be devised by the use of g. One starts from a suitable initial point (x_0,y_0) and lets $(x_{n+1},y_{n+1}) = g(x_n,y_n)$. In general the choice of an initial point is quite important. Ideally, the point should be close to the actual solution. The source of the choice of g as well as the motivation leading to its definition, will be presented in Chapter 3.

Applying the fixed point method above beginning with the initial

guess $x_0=y_0=10$, the following results are obtained, rounded to four digits. (For actual calculations see Appendix I.)

n	x_n	y_n
1	5.263	5.263
2	2.908	2.908
3	1.756	1.756
4	1.227	1.227
5	1.036	1.036
6	1.001	1.001
7	1.000	1.000

Table 1

It is possible to explore some methods analogous to the above in considerable detail. In order for the author to construct a method, the theory for a restricted case, systems of linear equations, was consulted. Using geometric intuition, each equation was conceptualized as a plane, then the necessary procedure to compute the unique orthogonal projection from a point not on the plane to a point in the plane was determined. Subsequently, the method was rediscovered in Pizer's book. [8] Interestingly, the method is always convergent. (A proof can be found in the above source.)

This technique will be discussed in Chapter 2, and a generalization of it to certain non-linear systems is discussed in Chapter 3. Opportunities for further development and study of this method will also be discussed. Alternate approaches to the problem are treated in [4] and [7].

CHAPTER II

THE LINEAR CASE

Suppose that a system of equations in n variables is defined by n functions F_i, $i = 1,2,\ldots n$. $F_i(x_1,\ldots x_n) = 0$. The system is linear if each F_i is a function of the form:

$$F_i(x_1,\ldots,x_n) = a_i x_1 + \ldots + a_i x_n - b_i.$$

Alternatively, the system may be viewed as a matrix equation, $AX - B = 0$, and the vector $X = (x_1,\ldots,x_n)$ may be regarded as a solution of the system.

The typical approaches to the linear case entail both algebraic and iterative techniques. Systems such as $AX = b$, with coefficients that are mostly nonzero, are called dense, and primarily are solved by algorithms based on Gaussian elimination. When A is square, sparse (most coefficients are zero) and of large order, iterative methods are the preferred method of solution.[1]

A thorough discussion justifies an investigation of Gaussian elimination, the Jacobian method and the Gauss-Seidel method. The chapter concludes with a discussion of an iterative technique developed by the author and her advisor, which resulted after concentration primarily on the iterative approaches. They were chosen in expectation thay they might suggest generalization to non-linear problems.

It is important first to point out that, in general, methods for nonlinear simultaneous equations will extend to linear simultaneous equations, but the converse is not true. Therefore, the following

methods should be understood only to apply to the linear case.

Gaussian Elimination

Some readers will recall that Gaussian elimination involves the simplification of a system of equations using row reductions. These, applied to the augumented co-efficient matrix, result in reduction to a lower triangular matrix. The resulting system is then much easier to solve.

An example is the linear system

$$x_1 + 2x_2 + 4x_3 = 3$$
$$x_1 + 2x_3 = 0$$
$$2x_1 + 4x_2 + x_3 = 3$$

represented by the matrix

$$\left[\begin{array}{ccc|c} 1 & 2 & 4 & 3 \\ 1 & 0 & 2 & 0 \\ 2 & 4 & 0 & 3 \end{array}\right], \text{ which reduces to } \left[\begin{array}{ccc|c} 1 & 0 & 0 & -6/7 \\ 0 & 1 & 0 & 15/14 \\ 0 & 0 & 1 & 3/7 \end{array}\right],$$

The latter matrix represents a system equivalent to the original. It is certainly trivial to identify the vector $\begin{cases} x_1 = -6/7 \\ x_2 = 15/14 \\ x_3 + 3/7 \end{cases}$ as a solution to the system. [2] The method also works successfully on a system

with infinite solutions or with no solutions.

It should be noted that Gaussian elimination can become less practical for problems on a sufficiently large scale. When n is small, the method is unquestionably preferred. If n= 10, Gaussian elimination requires only 430 operations. Compare this to the number of operations required by Cramer's rule, another popular technique. For the case n = 10, the number of operations is 39,916,800. [1]

The Jacobian Method

The Jacobian method uses simultaneous displacements. At the end of each iteration, the components of an approximate solution x^i are replaced simultaneously with the corresponding components of a vector x^{i+1}. In general, for the equation AX=b

$$x_j^{i+1} = \frac{b_j}{A_{jj}} - \sum_{\substack{k=1 \\ k \neq j}}^{n} \frac{A_{jk}}{A_{jj}} x_k^i \quad \text{for } j = 1,2,\ldots n;$$

where the terms A_{jk} are entries of A; the terms b_j are components of B. That is, the first equation solves for x_1, the second equation for x_2, and so on, up through the n^{th} equation for x_n. The j^{th} equation produces a value for x_j^{i+1} by using the values of x_k^i on the right side of the equation.

The convergence of the method can sometimes be assured by first

making the matrix either row or column diagonally dominant. A matrix A is said to be row diagonally dominant if

$$|a_{ii}| > \sum_{j \neq i}^{n} |a_{ij}| \quad \text{for all } i=1,2,\ldots n,$$

and column diagonally dominant if

$$|a_{ii}| > \sum_{j \neq i}^{n} |a_{ji}| \quad \text{for all } i \quad ,2,\ldots n. \; [9]$$

It is important to point out that convergence is possible even if these conditions do not apply.

Consider,

$$\begin{aligned} x_1 + 6x_2 + 2x_3 &= 15 \\ x_1 + x_2 - 6x_3 &= -3 \\ 6x_1 + x_2 + x_3 &= 9. \end{aligned}$$

In accordance with the above, the equations are rearranged as follows:

$$\begin{aligned} 6x_1 + x_2 + x_3 &= 9 \\ x_1 + 6x_2 + 2x_3 &= 15 \\ x_1 + x_2 - 6x_3 &= -3 \end{aligned}$$

Solve equation i for x_1

$$\begin{aligned} x_1 &= 3/2 - (1/6)x_2 - (1/6)x_3 \\ x_2 &= 5/2 - (1/6)x_1 - (1/3)x_3 \\ x_3 &= 1/2 + (1/6)x_1 + (1/6)x_2. \end{aligned}$$

Let $x^0 = (0,0,0)$

$$x^1 = \begin{bmatrix} 3/2 \\ 5/2 \\ 1/2 \end{bmatrix} \qquad x^2 = \begin{bmatrix} 1 \\ 25/12 \\ 7/6 \end{bmatrix} \qquad x^3 = \begin{bmatrix} 23/24 \\ 35/18 \\ 73/72 \end{bmatrix}$$

The sequence $\{ x^i \}$ is seemingly converging. [8]

Gauss - Seidel Iteration

The Gauss - Seidel method applies essentially the same idea as the Jacobian. A key difference is that the computed values are employed as soon as possible to obtain each component x_j^i. In general, for AX = B the formal equation for a stage of the Gauss - Seidel method is as follows:

$$x_j^{i+1} = \frac{b_j}{A_{jj}} - \sum_{k=1}^{j-1} \frac{A_{jk}}{A_{jj}} x_k^{i+1} - \sum_{k=j+1}^{n} \frac{A_{jk}}{A_{jj}} x_k^i.$$

For example, application of this method to the above using initial vector (0,0,0) yields:

$$x_1^1 = 3/2$$
$$x_2^1 = 5/2 - 1/6\ (3/2) - 1/3\ (0) = 9/4$$
$$x_3^1 = 1/2 + 1/6\ (3/2) + 1/6\ (9/4) = 9/8.$$

Thus, $x^1 = (3/2, 9/4, 9/8)$. Returning to equation one, this estimate can be used to compute x_1^2, x_2^2, x_3^2, and so forth. [8]

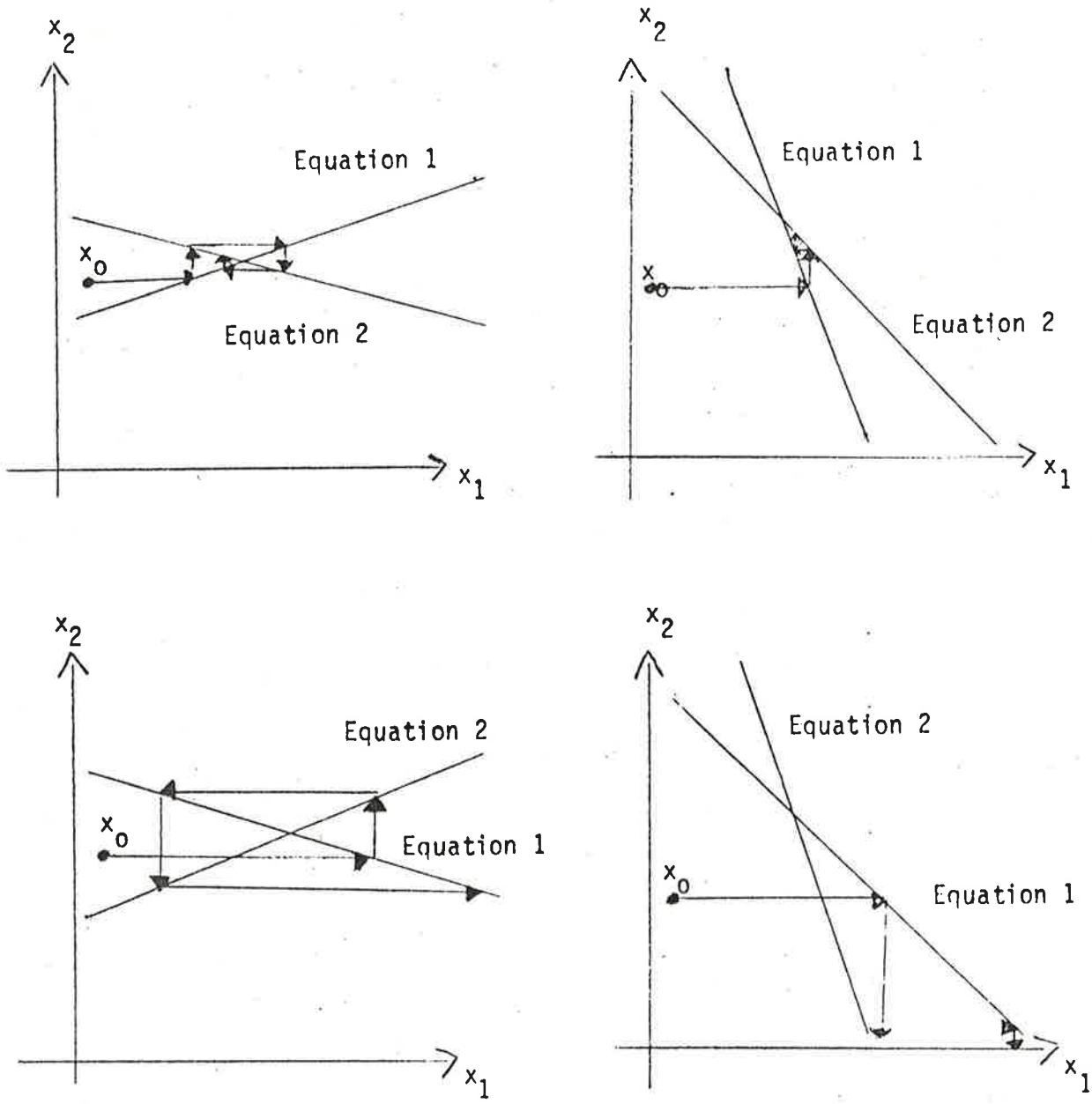

Figure 1

It might be noted at this point that convergence is guaranteed if both of the diagonal dominance conditions mentioned previously apply. Geometrically the convergence looks like the above. [8]

The Orthogonal Projection Method

The following method was developed by the author and her advisor. After initial development of the method, a thorough search of relevant literature resulted in the discovery that the method had previously been employed. A proof for convergence is found in Pizer's book. [8]

For clarity the discussion is confined to the case n = 3. The method extends easily to an arbitrarily large system.

In general, given a plane

$$a_1x_1 + a_2x_2 + a_3x_3 = b, \tag{1}$$

and a point $P = (u_1, u_2, u_3)$ not on the plane, the problem is to find the unique point $Q = (u_1', u_2', u_3')$ on the plane such that the vector P-Q is perpendicular to the plane.

$$P-Q = \langle u_1 - u_1', u_2 - u_2', u_3 - u_3' \rangle \quad . \tag{2}$$

Since P-Q is parallel to $\langle a_1, a_2, a_3 \rangle$, $\langle u_1-u_1', u_2-u_2', u_3-u_3' \rangle$ is equal to $C \langle a_1, a_2, a_3 \rangle$ for some constant scalar C.

$$\begin{aligned} u_1 - u_1' &= Ca_1 \\ u_2 - u_2' &= Ca_2 \\ u_3 - u_3' &= Ca_3 \end{aligned} \qquad \text{imply} \qquad \begin{aligned} u_1' &= u_1 - Ca_1 \\ u_2' &= u_2 - Ca_2 \\ u_3' &= u_3 - Ca_3. \end{aligned} \tag{3}$$

Replacing the variables in equation (1) by the coordinates of Q yields:

$$a_1u_1' + a_2u_2' + a_3u_3' = b,$$

and combining (1) and (3)

$$a_1(u_1 - ca_1) + a_2(u_2 - ca_2) + a_3(u_3 - ca_3) = b. \qquad (4)$$

Next solve for C.

$$\frac{a_1u_1 + a_2u_2 + a_3u_3 - b}{a_1^2 + a_2^2 + a_3^2} = C. \qquad (5)$$

Thus, the vector $\langle u_1', u_2', u_3' \rangle$ is obtained by using (5) in (3).

A FORTRAN Program (Appendix II) was used to implement the algorithm for various choices of n, giving the results in Table 2 for the system,

$$\begin{aligned} x_1 - x_2 + 2x_3 &= -3, \\ 5x_1 + 3x_3 &= 7, \\ 4x_1 - 3x_2 - x_3 &= 0. \end{aligned}$$

For brevity only one iterate from each cycle is included. The initial guess was (10, 10, 10). The iterates are rounded off to four places. The exact solution is (2, 3, -1).

The method described above was apparently discovered by Dr. Lester

n	x_1	x_2	x_3
0	10.	10.	10.
3	6.9529	9.843	-1.7174
6	5.0673	7.9523	-3.5876
9	3.8999	6.2708	-3.2129
12	3.1768	5.0839	-2.5443
15	2.729	4.3073	-2.006
18	2.4516	3.8145	-1.6372
21	2.2797	3.5059	-1.3988
24	2.1733	3.3138	-1.2482
27	2.1073	3.1945	-1.154
30	2.0665	3.1205	-1.0955

Table 2

Levy of Long Island Jewish Medical Center. [8] Based on the Pythogorean theorem, the iteration moves closer and closer to the intersection of the planes. As mentioned earlier, the orthogonal projection method can be shown to be always convergent, provided a solution exists and is unique.

CHAPTER III

THE NON-LINEAR CASE

The Newton-Raphson method seems to be the proper point of departure into the nonlinear investigation. Based on a Taylor Series approach, the method for one variable expands to a multiple variable application.

Newton-Raphson

If f is continous and twice differentiable, then

$$f(x_{i+1}) = f(x_i) = f'(x_i)(x_{i+1} - x_i) + f''(x_i)\frac{(x_{i+1} - x_i)^2}{2!} + \cdots$$

If $f(x_{i+1})$ is close to zero, and x_i is very close to x_{i+1} so that $(x_{i+1} - x_i)^2$ and all higher powers can be neglected then

$$0 \approx f(x_i) + f'(x_i)(x_{i+1} - x_i).$$

This suggests the equation:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},$$

which is commonly called the Newton-Raphson method.

Given two continuous and twice differentiable functions, f_1 and f_2, definition of two simultaneous equations is possible:

$$f_1(x,y) = 0; \quad f_2(x,y) = 0. \qquad (1)$$

To generate a sequence $\{(x_i, y_i)\}$, which converges to a solution, one should use the two-dimensional Taylor series, expressing $f_1(x_{i+1}, y_{i+1})$ as

$$f_1(x_i, y_i) + \frac{\partial f_1}{\partial x}(x_{i+1} - x_i) + \frac{\partial f_1}{\partial y}(y_{i+1} - y_i) + \ldots$$

where the partial derivatives $\frac{\partial f_1}{\partial x}$ and $\frac{\partial f_1}{\partial y}$ are evaluated at $x = x_i$ and $y = y_i$. In the same way, f_2 is expanded as follows:

$$f_2(x_{i+1}, y_{i+1}) = f_2(x_i, y_i) + \frac{\partial f_2}{\partial x}(x_{i+1} - x_i) + \frac{\partial f_2}{\partial y}(y_{i+1} - y_i) + \ldots$$

If (x_{i+1}, y_{i+1}) and (x_i, y_i) are very close to the root $(\bar{x}, \bar{y})$, the equations are simplified to:

$$0 \approx f_1(x_i, y_i) + \frac{\partial f_1}{\partial x}(x_{i+1} - x_i) + \frac{\partial f_1}{\partial y}(y_{i+1} - y_i)$$

$$0 \approx f_2(x_i, y_i) + \frac{\partial f_2}{\partial x}(x_{i+1} - x_i) + \frac{\partial f_2}{\partial y}(y_{i+1} - y_i).$$

Rearranging,

$$\frac{\partial f_1}{\partial x}(x_{i+1}-x_i) + \frac{\partial f_1}{\partial y}(y_{i+1} - y_i) \approx -f_1(x_i,y_i)$$

$$\frac{\partial f_2}{\partial x}(x_{i+1}-x_i) + \frac{\partial f_2}{\partial y}(y_{i+1} - y_i) \approx -f_2(x_i,y_i). \qquad (2)$$

For convenience, let $h = (x_{i+1} - x_i)$ and $k = (y_{i+1} - y_i)$. These equations can now be solved for h and k by the use of determinants, provided the determinant of the coefficient matrix does not vanish at any time during the iteration process. Therefore,

$$h = \frac{\begin{vmatrix} -f_1(x_i,y_i) & \frac{\partial f_1}{\partial y} \\ -f_2(x_i,y_i) & \frac{\partial f_2}{\partial y} \end{vmatrix}}{\begin{vmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{vmatrix}}, \qquad k = \frac{\begin{vmatrix} \frac{\partial f_1}{\partial x} & -f_1(x_i,y_i) \\ \frac{\partial f_2}{\partial x} & -f_2(x_i,y_i) \end{vmatrix}}{\begin{vmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{vmatrix}}$$

The denominator is commonly called the Jacobian of the system.

After h and k are determined, x_{i+1} and y_{i+1} must be evaluated. To compute the next approximation set $x_i = x_{i+1}$, $y_i = y_{i+1}$ and repeat. [9]

The discerning reader might notice that the system (2) can be expressed in matrix notation.

$$J(x^I)(x^{I+1} - x^I) = -F(x^I), \tag{3}$$

where J is the Jacobian matrix for the vector valued function F, x^I is the vector (x_i, y_i), and x^{I+1} is the next iterate (x_{i+1}, y_{i+1}). If equation (3) is solved for x^{I+1}, the following is obtained:

$$x^{I+1} = x^I - J^{-1}(x^I)\, F(x^I). \tag{4}$$

The latter equation, which generalizes clearly to systems with an arbitrary number of variables, is often taken as a definition of Newton's method for iteratively solving systems of equations. Since the matrix J is difficult to invert, the usual procedure is to solve equation (3) for the vector $x^{I+1} - x^I$, as indicated earlier. The necessity of solving a matrix equation is one of the chief drawbacks of the method, especially as the number of unknowns increases.

Equation (4) also suggests the definition of a fixed point problem $G(x) = x$, which is equilvalent to the problem $F(x) = 0$:

$$G(x) = x + J^{-1}(x)\, F(x). \tag{5}$$

For the example discussed in Chapter 1, the function F is defined by two equations:

$$f_1(x,y) = y - x^2, \qquad f_2(x,y) = x - y^2.$$

The matrixes $J(x,y)$ and $J^{-1}(x,y)$ are given by:

$$J(x,y) = \begin{bmatrix} -2x & 1 \\ 1 & -2y \end{bmatrix}; \qquad J^{-1}(x,y) = \begin{bmatrix} 2y/(1-4xy) & 1/(1-4x4) \\ 1/(1-4xy) & 2x/(1-4xy) \end{bmatrix}$$

By performing the computations indicated in (5) one obtains the following representation of G used in Chapter 1 to solve $F(x,y) = (0,0)$.

$$g_1(x,y) = x - \frac{x - 2yx^2 + y^2}{1-4xy}, \quad g_2(x,y) = y - \frac{y - 2xy^2 + x^2}{1-4xy}. \qquad (7)$$

The Newton-Raphson method in practice is quite complicated. The requirement that the Jacobian is not allowed to be zero is difficult to predict without calculations and includes the risk that the process will be inoperative. In general, for n simultaneous equations in n unknowns, one must find n^2 partial derivatives, and must then solve for n increments h, k... by solving n simultaneous equations for n unknowns. Therefore, mathematicians have sought an easier method, which, while perhaps not converging as quickly, is easier to apply.[9]

One modification of the Newton-Raphson method consists of applying the single variable Newton-Raphson method n times, once for each variable,

assuming the other variables are fixed.

In general, for two equations and two unknowns:

$f_1(x,y) = 0$ and $f_2(x,y) = 0$, and

using x_0 and y_0 as initial approximations, one calculates a new value x_1, from the single-variable Newton-Raphson method:

$$x_1 = x_0 - \frac{f_1(x_0,y_0)}{\frac{\partial f_1}{x}}$$

where $\partial f_1 / \partial x$ is evaluated at x_0 and y_0. Using f_2 and the most recent values of x and y, in this case x_1 and y_0, a new value y_1 is calculated as follows:

$$y_1 = y_0 - \frac{f_2(x_1,y_0)}{\frac{\partial f_2}{\partial y}}.$$

After obtaining x_1 and y_1, one uses f_1 to calculate x_2, and so on.

Note that the choice of using f_1 to calculate a new x and using f_2 to calculate a new y is not arbitrary. In fact, an improper choice of functions can lead to divergence. The function which has the steepest descent at the solution point should be used to find the next

x.[9]

There exists many other iterative methods designed to deal with single-variable, non-linear equations; the bisection method, the method of false position and various Picard iterations. However, methods like bisection and false position seem difficult to extend to many dimensions. The extension from single equations to systems of equations is a topic addressed briefly in most sources investigated by this researcher. One example, of course, is the modified Newton-Raphson method described above. Other techniques have been devised in various research papers. [4] and [10]

The following, an extension of the orthogonal vector method, is based partly on the underlying idea of the Gauss-Seidel iteration technique. The method apparently works with some effectiveness in several cases, though not in all of those attempted. Details of these results and potentially sufficient conditions for convergence will be discussed following the initial derivation.

Modified Orthogonal Vector Method

Suppose a system of equations is given,

$$F_i(x_1, \ldots, x_n) = 0. \qquad (8)$$

Suppose, in addition, that for each equation, there is a unique variable x_i, so that the particular equation can be solved for x_i in terms of the

other variables. Thus, one can write (8) in the form;

$$x_i = f_i\ (x_1, \ldots, \hat{x}_i, \ldots, x_n). \tag{9}$$

If $p_o = (x_1^o, x_2^o, \ldots, x_n^o)$, the initial point, and if q_1 is the point obtained in the surface F by varying only the first coordinate of p_o, then:

$$\bar{x}_1 = f_1(x_2^o, \ldots, x_n^o) \quad \text{and} \quad q_1 = (\bar{x}_1, x_2^o, \ldots, x_n^o).$$

The plane tangent to F_1 at q_1 is represented analytically by:

$$x_1 - \bar{x}_1 = \frac{\partial f_1}{\partial x_2}\ (x_2 - x_2^o) + \ldots + \frac{\partial f_n}{\partial x_n}\ (x_n - x_n^o). \tag{10}$$

In order to project orthogonally to a point p_1 in this plane, a normal vector is identifided as:

$$n = \left(-1, \frac{\partial f_1}{\partial x_2}, \ldots, \frac{\partial f_n}{\partial x_n}\right).$$

Then $p_1 - p_o = cn$ (for some constant c).

More specifically, if $p_1 = (x_1^1, \ldots, x_n^1)$ then:

$$x_1^1 - x_1^o = -c, \text{ and } x_i^1 - x_i^o = c\frac{\partial f_1}{\partial x_i} \quad \text{for } 2 \leq i \leq n. \tag{11}$$

Isolating the coordinates of p_1 yields:

$$x_1^1 = x_1^0 - c; \; x_i^1 + c\frac{\partial f_1}{\partial x_i} \quad \text{for } 2 \leq i \leq n. \tag{12}$$

To solve for c, one uses the equation of the tangent plane (10) and the system (11):

$$x_1^1 - \bar{x}_1 = x_1^0 - c - \bar{x}_1 = \frac{\partial f_1}{\partial x_2}(x_2^1 - x_2^0) + \ldots + \frac{\partial f_n}{\partial x_n}(x_n^1 - x_n^0).$$

$$x_i^0 - \bar{x}_1 = c + \frac{\partial f_1}{\partial x_2}\left(c\frac{\partial f_1}{\partial x_2}\right) + \ldots + \frac{\partial f_1}{\partial x_n}\left(c\frac{\partial f_1}{\partial x_n}\right)$$

$$= c\left\{1 + \left(\frac{\partial f_1}{\partial x_2}\right)^2 + \ldots + \left(\frac{\partial f_1}{\partial x_n}\right)^2\right\}$$

$$c = \frac{x_1^0 - \bar{x}_1}{1 + \left\{\left(\frac{\partial f_1}{\partial x_2}\right)^2 + \ldots + \left(\frac{\partial f_1}{\partial x_n}\right)^2\right\}} \tag{13}$$

Now, the iterate p_1 is computed from (12).

To obtain a general iterative step; $1 \leq i \leq n$, one follows an entirely similar procedure.

If $p_i = (x_1^i, \ldots, x_n^i)$

$\bar{x}_i^i = f_i(x_1^i, \ldots, \hat{x}_i^i, \ldots x_n^i)$

$$c = \frac{x_i^i - x_i^{-i}}{1 + \left(\frac{\partial f}{\partial x_1}\right)^2 + \ldots + \left(\frac{\partial f}{\partial x_{i-1}}\right)^2 + \left(\frac{\partial f}{\partial x_{i+1}}\right)^2 + \left(\frac{\partial f}{\partial x_n}\right)^2}$$ where

all partials are evaluated at $(x_1^i, \ldots, \hat{x}_i^i, \ldots, x_n^i)$.

$x_i^{i+1} = x_i^i - c$, and for $j \neq i$, $x_j^{i+1} + c\frac{\partial f_j}{\partial x_j}$.

If $i > n$, then $i = nq + r$ for some r, $0 \leq r < n$, $q \geq 1$. Thus p_{i+1} can be generated from p_{nq} in the same way that p_i is generated from p_o above when $i \leq n$.

The method was applied on several examples, producing the following results. The algorithm was terminated when $\sum_{i=qn+1}^{qn+n} \| p_i - p_{i-1} \|_\infty < \varepsilon$. The sum is taken over a cycle of n iterates. $\varepsilon = .1$.

Example 1

This problem arose in Chapter 1 in connection with the problem of determining the critical points for the function:

$$f(x,y) = xy - \frac{x^3}{3} - \frac{y^3}{3} + 10.$$

$$\begin{cases} x = y^2 \\ y = x^2 \end{cases}$$

n	x_n	y_n	n	x_n	y_n
0	1.	2.	0	.6	.6
1	1.326	1.5318	1	.7348	.4856
2	1.1748	1.318	2	.731	.4704
3	1.1	1.1898	3	.7161	.4475
4	1.0598	1.1156	4	.6863	.4051
5	1.0365	1.0714	5	.6213	.3234
			6	.4575	.1691
			7	.1144	.0113
			8	1.928E-04	3.7126E-08
			9	0.0	0.0

Table 3

Example 2

$$\begin{cases} x = y - y^3 \\ y = x^3 - x \end{cases}$$

n	x_n	y_n
0	10.	10.
1	6.7039	6.6666
2	4.5248	4.4436
3	3.099	2.9595
4	2.1868	1.9823
5	1.6271	1.2709
6	1.2909	.7173
7	.5297	-.382
8	-.0532	.0544
9	1.5865E-04	-1.5866E-04
10	-4.02 E-12	4.02 E-12

Table 4

Example 2A

A modification of Example 2 to:

$$\begin{cases} x = 2(y-y^3) \\ y = 2(x^3-x), \end{cases}$$

does not converge from $p_o = (10,10)$, but does converge from $p_o = (.4,.4)$.

Example 3 This problem known as Rosenblock's parabolic valley is found in [4].

$$F(x,y) = 100(y-x^2)^2 + (1-x)^2$$

$$\begin{cases} x = y \\ y = x^2 - 1/200x + 1/200 \end{cases} \qquad \varepsilon = .001$$

n	x_n	y_n	n	x_n	y_n
0	1.2	1.0	0	0.86	1.14
1	1.0393	1.803	1	1.0297	1.0605
2	1.0393	1.803	2	1.0297	1.0605

Table 5

Example 4 [4]

$$F(x,y,z) = (x-y+z)^2 + (y+z-x)^2 + (x+y-z)^2$$

$$\begin{cases} x = (y+z)/3 \\ y = (x+z)/3 \\ z = (x+y)/3 \end{cases}$$

n	x_n	y_n	z_n
0	.5	1.	.5
1	.4038	.6574	.554
2	.3352	.5631	.4424
3	.2772	.4637	.3678
4	.2293	.3838	.304
5	.1897	.3175	.2516
6	.1569	.2627	.2081
7	.1298	.2173	.1722
8	.1074	.1798	.1424
9	.0888	.1487	.1178
10	.0735	.123	.0974
11	.0608	.1018	.0806
12	.0503	.0842	.0667
13	.0416	.0697	.0552
14	.0344	.0576	.0456
15	.0284	.0477	.0378
16	.0236	.0394	.0312
17	.0348	.0275	.0207

n	x_n	y_n	z_n
0	100.	-1.	2.5
1	26.5984	23.1687	16.589
2	23.2201	18.1613	13.7938
3	19.0713	15.1385	11.4033
4	15.7928	12.5107	9.4345
5	13.0632	10.3512	7.8048
6	10.807	8.5631	6.4567
7	8.9403	7.084	5.3415
8	7.396	5.8604	4.4188
9	6.1186	4.8482	3.6556
10	5.0617	4.0108	3.0242
11	4.1874	3.318	2.5018
12	3.4641	2.7449	2.0697
13	2.8658	2.2708	1.7122
14	2.3708	1.8786	1.4164
15	1.9613	1.5541	1.1718
16	1.6225	1.2856	.9694
17	1.3422	1.0636	.802
18	1.11	.8799	.6634
19	.9186	.7279	.5433
20	.75	.6022	.454
21	.6287	.4981	.3756
22	.52	.4121	.3107
23	.4304	.3409	.257
24	.3559	.282	.2127
25	.2945	.2333	.1759
26	.2436	.1930	.1455
27	.2015	.1597	.1204
28	.1667	.1321	.0996
29	.1379	.1093	.0824
30	.1141	.0904	.0682
31	.0944	.0748	.0564
32	.0781	.0619	.0467
33	.0646	.0512	.0386
34	.0534	.0423	.0319
35	.0442	.035	.0264
36	.0366	.029	.0219

Table 6

Convergence and Efficiency Considerations

At this writing, attempts to obtain sufficient conditions for convergence of the preceding iteration technique have been unsuccessful. It is strongly suspected that convergence can be guaranteed by the choice of an initial point P_o sufficiently close to the solution P, provided all of the first and second order partial derivatives of the functions f_i are continuous at P. The computational results above lend considerable support to this conjecture. Note, the method is a fixed point type of iteration. I.e., a function g so that $p_{n+1} = g(p_n)$ is obtainable for each iterate p_n. Thus, a well known convergence theorem might be useful in particular cases. The following theorem is taken from [8].

> If g_i denotes the coordinate functions defining g, the following can be assumed.
>
> Theorem Let K be a positive constant with $K<1$. Let p be a fixed point for the function g, and let R be a positive constant. If $\sum_{k=1}^{n} \left| \frac{\partial g_i}{\partial x_k}(x) \right| \leq K$ for all x satisfying $|p - x|_\infty \leq R$. Then the iteration given by $p_{n+1} = g(p_n)$ converges to p for any choice of initial point satisfying $|p_o - p|_\infty \leq R$.

For a slightly different version of this result see [3].

In order to make a rough comparison between the efficiency of the orthogonal projection method and Newton's method, the number of arithmetic operations required for the projection method was checked. To compute one

cycle of iterates, p^{mn}, p^{mn+1}, . . . , p^{mn+n} the number of operations required is $4n^2-n$. In addition $n^2 + n$ function evaluations are required. By comparison Newton's method requires the same number of function evaluations as well as the solution of a linear system of n equations in n variables. As n increases in size, the speed of machine operation might become a factor in favor of the projection algorithm, since the number of operations needed to solve the linear system of n equations is $O(n^3)$.

The computer implementation used by the author to test the projection method is indicated in Appendix III. The storage requirements for this program are roughly the same as would be needed for an implementation of Newton's method. Nevertheless , it now seems that the program is less effieient in this regard than it might be. For example, it does not seem necessarv to require space for all of the n^2 partial derivatives at any one time. At most n cells should be sufficient for each iterative step. Another economy might be obtained by storing each of the function values $f_i\ (x_1^m, \ldots \hat{x}_i^m, \ldots x_n^m)$ in the same cell. These considerations would become significant if the algorithm is applied to large-scale problems such as those commonly encountered in the numerical solutions of partial differential equations. It is acknowledged, of course, that the number of function evaluations limits this possibility significantly.

Conclusion

In conclusion, it is necessary to mention certain aspects of this work which are believed to warrant further study. Due to time restraints, the algorithm was not applied to a sufficiently wide variety of problems. Thus, further testing of the algorithm would provide useful insight into matters of efficiency, speed and storage requirements. In addition, this information would provide adequate data for a comparison between the method and other well known techniques, such as Newton's method.

A potential limitation of the orthogonal method is the necessity to solve for x_1 in each equation and compute derivatives for the explicit function obtained thereby. In some problems, neither of these steps is practicable. Perhaps an application of the one variable version of Newton's method for approximating x_1 in conjunction with the use of implicit differentiation for evaluation of partial derivatives would extend applicability of the algorithm to some of these more difficult problems.

There are some aspects of optimization, currently of great interest, have not been touched upon at all. This is perhaps, best illustrated by the fact that these solutions or results do not in any case help to determine if an extremum is located at the point in question. Nevertheless, the functions to be optimized usually exhibit behavior near a critical point which can help to identify the point as an extremum. For example, the directional derivatives may change in sign at the point, or the second order partial derivatives may be positive definite near the point. Such features

might be useful in the construction of types of algorithms which are totally different from the approach of this paper. For further information, see [4] and [5].

Needless to say, the facts are not all here. The author has left many open questions which require further research. Hopefully, this work will be extended beyond its present stance, either by the author and/or an intrigued reader.

REFERENCES

1 Atkinson, Kendall E. An Introduction to Numerical Analysis. New York: John Wiley and Sons, (1978) 435-436.

2 Block, Norman, and John Michaels. Linear Algebra. New York: McGraw-Hill Book Company, (1977) 14-15.

3 Burden, Richard, Faires, and Reynolds. Numerical Analysis. 2nd ed. Boston, Mass.: Prindle, Weber and Schmidt, (1981) 449.

4 Byrne, George, and Norman Steen. "The Problem of Minimizing Non-Linear Functionals." In Numerical Solution of Systems of Nonlinear Algebraic Equations, ed. George Byrne and Charles Hall. New York: Academic Press, (1973) 185-219.

5 Cohen, A. M., J. F. Cutts, R. Fielder, D. E. Jones, J. Ribbans, and E. Stuart. Numerical Analysis. New York: John Wiley and Sons, (1973) 269.

6 Courant, Richard, and Fritz John. Introduction to Calculus and Analysis. New York: John Wiley & Sons, (1974) 325-327.

7 Ortega, J., and W. Rheinboldt. Iterative Solution of Nonlinear Equations in Several Variables. New York: Academic Press, (1970) 34-58,181-278.

8 Pizer, Stephen. Numerical Computing and Mathematical Analysis. Chicago: Science Research Associates, Inc., (1975) 149-156, 163-168.

9 Stark, Peter. Introduction to Numerical Methods. London: The Macmillan Company, (1970) 133-6, 189.

10 Young, David. "On the Solution of Large Systems of Linear Algebraic Equations with Sparse, Positive Definite Matrices." In Numerical Solution of Systems of Nonlinear Algebraic Equations, ed. George Byrne and Charles Hall. New York: Academic Press, (1973) 101-155.

APPENDIX

APPENDIX I

$$x + \frac{x-2yx^2+y^2}{4xy-1} \qquad y + \frac{y-2xy^2+x^2}{4xy-1}$$

$x_0=y_0=10$

$$x_1=y_1=10 + \frac{10-2(10)^3+10^2}{4(10)^2-1}$$

$=10 - 4.7368$

$=5.2632$

$$x_2=y_2=5.2632 + \frac{5.2632-2(5.2632)^3+(5.2632)^2}{4(5.2632)^2-1}$$

$=5.2632 - 2.355$

$=2.9082$

$$x_3=y_3=2.9082 + \frac{2.9082-2(2.9082)^3+(2.29082)^2}{4(2.9082)^2-1}$$

$=2.9082 - 1.1521$

$=1.7559$

$x_4=y_4=1.7559 - .5284$

$=1.2274$

$x_5=y_5=1.2274 - .19186$

$=1.0355$

$x_6=y_6=1.0355 - .0343$

$=1.001$

$x_7=y_7=1.001 - .000998$

$=1.000$

APPENDIX II

```
C     THIS PROGRAM COMPUTES THE POINT OF INTERSECTION OF N PLANES IN R_N,
C     BY ORTHOGONAL PROJECTIONS INITIATED BY AN INITIAL GUESS

C     ESTABLISH SIZE OF ARRAYS
      DIMENSION P(11,10), A(10,10), B(10), D(10)

C     READ IN NUMBER OF PLANES BEING CONSIDERED
      READ (1,600) N

C     READ IN ERROR ALLOWANCE AND MAXIMUM NUMBER OF ALLOWED ITERATIONS
      READ (1,610) E, MAXIT
      M=0
      WRITE (3,620) N, E, MAXIT

C     READ IN AND WRITE INITIAL GUESS
      DO 10 J=1,N
      READ (1,630) P(1,J)
   10 CONTINUE
      WRITE (3,640) (P(1,J),J=1,N)

C     READ AND WRITE EQUATIONS
      DO 20 I=1,N
      DO 30 J=1,N
      READ (1,650) A(I,J)
   30 CONTINUE
      READ (1,660) B(I)
      WRITE (3,670) (A(I,J),J=1,N),B(I)
   20 CONTINUE

C     COMPUTE DENOMINATOR
      DO 45 I=1,N
      D(I)=0.0
      DO 40 J=1,N
      G=A(I,J)
      D(I)=(D(I)+A(I,J)**2)
   40 CONTINUE
   45 CONTINUE

C     COMPUTE C
      I=1
  100 F=0.0
      DO 50 J=1,N
      H=A(I,J)
      F=F+H*P(I,J)
   50 CONTINUE
      C=(F-B(I))/D(I)

C     COMPUTE ORTHOGONAL PROJECTION POINT AND REPLACE INITIAL GUESS WITH IT
      DO 60 J=1,N
   60 P(I+1,J)=P(I,J)-A(I,J)*C
      WRITE (3,680) (P(I+1,J),J=1,N)
```

```
C     CHECK TO SEE IF NEW PROJECTION POINT HAS BEEN COMPUTED FOR EACH PLANE
      IF (I .EQ. N) GO TO 200
C     IF NOT RETURN FOR MORE DATA
      I=I+1
      GO TO 100

C     FIRST CHECK TO SEE IF MAXIMUM NUMBER OF ITERATIONS HAS OCCURRED
  200 M=M+1
      IF (M .GE. MAXIT) STOP

C     CHECK TO SEE IF CONCLUSION REACHED
      S=0.0
      DO 300 I=1,N
      DO 300 J=1,N
      S=S+ ABS(P(I,J) - P(I+1,J))
  300 CONTINUE
      IF (S .GE. E) THEN DO
      DO 400 J=1,N
      I=1
      P(1,J)=P(N+1,J)
  400 CONTINUE
      GO TO 100
      ELSE DO
      WRITE (3,700) (P(N+1,J),J=1,N)
      END IF
      .
      .
      .
      .
      .
      .
      .
```

APPENDIX III

```
 40  DIM R(10): DIM P(10,10): DIM F(10): DIM D(10,10): DIM R1(10)
 45  PRINT "SELECT ONE OF SUBROUTINES"
 46  INPUT Q
 50  PRINT "ENTER NUMBER OF EQUATIONS BEING CONSIDERED"
 60  INPUT N
 70  PRINT "ENTER ERROR ALLOWANCE "
 80  INPUT E
 90  PRINT "ENTER MAXIMUM NUMBER OF ITERATIONS ALLOWED"
100  INPUT M
110  PRINT
120  FOR J = 1 TO N
130  PRINT "ENTER INITIAL GUESS", J
140  INPUT R(J)
150  LET R1(J) = R(J)
160  NEXT J
170  PRINT

180  REM "SET UP COUNTER TO MONITOR NUMBER OF ITERATIONS"
185  LET K = 0
190  LET I = 0
200  I = I + 1
210  PRINT

220  REM "THE SUBROUTINE MUST HAVE BEEN CALCULATED AND ENTERED PREVIOUSLY
     FOR EACH NON-LINEAR EQUATION"
230  GOSUB 620
235  LET X = R(I)
240  LET Y = F(I)
250  PRINT

260  REM "COMPUTE SCALAR C"
270  LET S = 1
280  FOR J = 1 TO N
290  IF J = I THEN 300
292  LET S = S + D(I,J) **.2
300  NEXT J
310  LET C = (X - Y) / S
320  PRINT

330  REM "COMPUTE INDIVIDUAL CORDINATE OF PROJECTION POINT"
340  FOR J = 1 TO N
350  IF J = I THEN 355
352  LET P(I,J) = R(J) + C * D(I,J)
354  GO TO 360
355  LET P(I,J) = R(J) - C
360  NEXT J
370  FOR J = 1 TO N
380  PRINT P(I,J),
385  LET R(J) = P(I,J)
390  NEXT J
400  PRINT
```

APPENDIX III (Continued)

```
405  REM "CHECK IF NEW PROJECTION POINT HAS BEEN COMPUTED FOR EACH SURFACE"
410  IF I = N THEN 425
420  GO TO 200
425  LET K = K + 1

426  REM "IF MAXIMUM NUMBER OF ITERATIONS HAVE BEEN COMPUTED STOP"
430 IF K  M THEN 550

435  REM "CHECK TO SEE IF CONCLUSION REACHED"
437 LET T = 0.0
440  FOR J = 1 TO N
445  LET I = 1
447  LET T = T + ABS (R1(J) - P(1,J))
450  FOR I = 2 TO N
460  T = T + ABS (P(I,J) - P(I-1,J))
470  NEXT I
480  NEXT J
500 IF T  E THEN 560
510  PRINT "POINT OF INTERSECTION IS"
520  FOR J = 1 TO N
530  PRINT P(N,J)
540  NEXT J
543 PRINT
544  PRINT
545  PRINT "NUMBER OF CYCLES", K
550  STOP
560 FOR J = 1 TO N
570  LET R1(J) = R(J)
580  NEXT J
590  GO TO 190
620  IF Q = 1 THEN 1000
630  IF Q = 2 THEN 900
640  IF Q = 3 THEN 960
650  IF Q = 4 THEN 970

660  REM "NEXT SUBROUTINE GOES HERE"
900  GO SUB 2000
950  RETURN
960  GO SUB 3000
965  RETURN
970  GO SUB 4000
975  RETURN
1000  IF I = 1 THEN 1300
1100  LET F(I) = R(1) **2
1200  LET D(I,1) = 2 * R(1)
1205  RETURN
1300  LET F(I) = R(2) **2
1400  LET D(I,2) = 2 * R(2)
1405  RETURN
```

```
2000  If I = 1 THEN 2300
2100  LET F(I) = R(1) **3 - R(1)
2200  LET D(I,1) = 3 * R(1) **2 - 1
2250  RETURN

2300  LET F(I) = R(2) - R(2) **3
2400  LET D(I,2) = 1 - 3 * R(2) ** 2
2500  RETURN

3000  IF I = 1 THEN 3300
3100  LET F(I) = R(1) **2 - 1 (200 * R(1)) + 1 / 200
3200  LET D(I,1) = 2 * R(1) + 1 / (200 * R(1) **2)
3205  RETURN

3300  LET F(I) = SQR (R(2))
3400  LET D(I,2) = 1 / (2 * (SQR (R(2))))
3401  RETURN

4000  IF I = 1 THEN 4300
4001  IF I = 2 THEN 4500
4100  LET F(I) = (R(1) + R(2)) / 3
4200  LET D(I,1) = 1/3
4205  LET D(I,2) = 1/3
4210  RETURN

4300  LET F(I) = (R(2) + R(3))/ 3
4400  LET D(I,2) = 1/3
4405  LET D(I,3) = 1/3
4406  RETURN

4500  LET F(I) = R(1) + R(3)) / 3
4600  LET D(I,1) = 1/3
4700  LET D(I,3) = 1/3
4701  RETURN
4702  END
```