APPROXIMATING SOLUTIONS TO NONLINEAR SYSTEMS OF EQUATIONS

by

Terence J. Blevins

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in the

Mathematics

Program

J. Douglas Faires — August 15, 1980

Adviser — Date

[illegible] — August 18, 1980

Dean of the Graduate School — Date

YOUNGSTOWN STATE UNIVERSITY

August 1980

ABSTRACT

APPROXIMATING SOLUTIONS TO NONLINEAR
SYSTEMS OF EQUATIONS

Terence J. Blevins
Master of Science
Youngstown State University

In this paper I will discuss methods used to approximate solutions to nonlinear systems of equations. I do not intend this to be an exhaustive report on the problem but rather to touch methods which I have found most interesting and most productive.

Chapter I introduces the reader to the general problem and gives ideas of the many methods available for solving this problem. The next chapter deals with the development of the methods that I use for this problem along with a brief discussion of alternate methods. Chapters III and IV contain problems that are used to test the methods and results of those tests respectively. The final chapter contains conclusions based on the test results.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. John Buoni for his helpful suggestions, Dr. Richard Burden for commenting on my programs and a special thanks to my adviser, Dr. Douglas Faires, for his direction and patience.

TABLE OF CONTENTS

LIST OF SYMBOLS

SYMBOL	DEFINITION
R^n	Real n dimensional space
$L(R^n)$, $L(R^n, R^m)$	Linear space of linear operators from R^n to R^n or from R^n to R^m
s, u, v, w, x, y, z	Vectors, unless otherwise indicated
a, b, c, α, ∂, δ	Scalars
A, B, U, V, H	Matrices
$F: R^n \to R^n$	A function, F, from R^n to R^n
$F(x) = 0$	A nonlinear system of n, equations in n unknowns
$F'(x)$ or $J(x)$	The Jacobian matrix evaluated at vector x
$\|\cdot\|_2$	The Euclidean norm
[a, b]	Closed interval with endpoints a and b
$\{x_i\}_{i=1}^{\infty}$	A sequence
$\|\cdot\|_F$	The Frobenius norm
$f_i(x)$ or $f(x)$	A linear or nonlinear function in one or more unknowns
$f_i'(x)$, $f'(x)$	The derivative of $f_i(x)$, $f(x)$
$(\cdot)^T$	The transpose of a vector or a matrix

CHAPTER I

INTRODUCTION

The General Problem

In many practical instances the simultaneous solutions to the following set of equations is sought.

$$
\begin{aligned}
f_1(x_1, \ldots, x_n) &= 0 \\
f_2(x_1, \ldots, x_n) &= 0 \\
&\vdots \\
f_n(x_1, \ldots, x_n) &= 0
\end{aligned}
$$

where $f_i\colon R^n \to R$ and $x_i \in R$ for $i = 1, \ldots, n$.

The frequent occurence of problems similar to the above has lead to the following notational simplification. Consider the following equation which is identical to the above,

$$F(x) = 0 \qquad (1)$$

where $F = (f_1, f_2, \ldots, f_n)\colon R^n \to R^n$ and x is the vector $(x_1, \ldots, x_n)^T$ with $x_i \in R$ for $i = 1, \ldots, n$. It will be assumed that at least one of the n functions will be nonlinear hence equation (1) represents a nonlinear system of n equations in n unknowns.

The classical approach to solving equation (1) has been to apply Newton's Method which takes an initial approximation to the solution and iteratively improves it. Newton's

Method guarantees that, under suitable conditions on F, quadratically convergence to the solution will be obtained. The above guarantee also assumes that the initial approximation is within some neighborhood of the solution. This local convergence property has a restrictive effect.

Some of the physical problems that require solving equation (1) do not yield initial approximations that are good enough to result in convergence by Newton's Method. Confronting this problem is the first goal of this paper, that is, to find a method that will yield convergence to the solution given any initial approximation. A method of this type is said to have the property of convergence.

The Problem of a Poor Initial Approximation

One approach to the problem of having a poor initial approximation to the solution of equation (1) dates back to 1944. Here K. Levenberg discussed the problem relative to nonlinear least squares analysis.[1] In 1952 M. Hestenes and E. Stiefel published a well-known paper which introduced the conjugate gradient method that confronts this problem relative to linear systems.[2]

[1]K. Levenberg, "A Method for the Solution of Certain Nonlinear Problems in Least Squares," Quarterly Journal of Applied Mathematics, 2 (1944):164.

[2]Magnus R. Hestenes and Edward Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," Journal of Research of the National Bureau of Standards, 49 (December 1952):409.

There is also a method described by Davidenko in a 1953 paper.[3] Finally in 1963 D. Marquardt published what has come to be known as the Levenberg-Marquardt Method.[4] The method that I chose to implement is based on the Levenberg-Marquardt Method. Reasons for this choice are described in Chapter II.

The Problem of Computational Efficiency

Another consideration that arises when solving equation (1) is that of computational efficiency. Do there exist methods to solve equation (1) which are more efficient than Newton's Method?

In Chapter II it will be seen that Newton's Method requires certain computations that can be quite costly. These computations in some instances are completely eliminated by considering a new approach and in other instances the computations are approximated based on prior approximations.

Brown's Method, Brent's Method[5] and Kizner's Method[6] are examples of methods which are new approaches. These

---

[3] D. Davidenko, "On the Approximate Solution of a System of Nonlinear Equations," Ukran. Mat. 5 (1953):196.

[4] Donald D. Marquardt, "An algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal of Applied Mathematics 11 (June 1963):431.

[5] Michel Y. Cosnard and Jorge J. More', "Numerical Solution of Nonlinear Equations," Transaction on Mathematical Software 5 (March 1979):64.

[6] W. Kizner, "A Numerical Method for Finding Solutions of Nonlinear Equations," SIAM Journal of Applied Mathematics 12 (1964):424.

methods have had poor initial success due to problems of implementation, but Brown's Method and Brent's Method have recently been revised and generalized so that programming is possible.[7]

Methods which eliminate certain computations by successive approximations are called quasi-newton methods. Some quasi-newton methods are; Powell's Method,[8] Broyden's A Method and Broyden's A^{-1} Method.[9] A survey article of these methods and others can be found in a paper by Dennis and More'.[10] I have chosen to study Broyden's A^{-1} Method and compare its efficiency to that of Newton's Method.

---

[7]D. M. Gay, "Implementing Brown's Method," _Report CNA-109_ (Austin:Center of Numerical Analysis, 1975), p. 1.

[8]Leon Cooper and David I. Steinberg, _Introduction to Methods of Optimization_ (Philadelphia: W. B. Saunders Company, 1970), p. 162.

[9]C. G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations," _Mathematics of Computation_ 19 (1965):577.

[10]J. E. Dennis and Jorge J. More', "Quasi-Newton Methods, Motivation and Theory," _SIAM Review_ 19 (January 1977):46.

CHAPTER II

METHODS

Newton's Method

Before deriving Newton's Method for a nonlinear system of equations I would like to look at Newton's Method for a single nonlinear equation in one unknown. Consider the following equation,

$$f(x) = 0 \tag{2}$$

where $f: R \rightarrow R$ and f is nonlinear and $x \varepsilon R$. The following is a derivation for Newton's Method for a single nonlinear equation.[11]

Let f be twice continously differentiable on [a, b]. Also let $x^* \varepsilon R$ be an initial approximation to x such that $|x - x^*|$ is small, i.e., we have a good initial approximation. Finally, let $f'(x) \neq 0$. Then we can approximate f(x) by

$$f(x) = f(x^*) + f'(x^*)(x - x^*) + f''[\xi(x)](x - x^*)^2/2 \tag{3}$$

using Taylor's expansion, where $\xi(x)$ lies between x and x^*. But since $|x - x^*|$ is small, $(x - x^*)^2$ is smaller yet. Hence if $f(x) = 0$, we have

$$f(x) = 0 \approx f(x^*) + f'(x^*)(x - x^*). \tag{4}$$

[11] Richard L. Burden, J. Douglas Faires, and Albert C. Reynolds, Numerical Analysis (Boston: Prindle, Weber and Schmidt, 1978), p. 39.

Equation (4) implies

$$x \approx x^* - f(x^*)/f'(x^*). \qquad (5)$$

This approximation provides the basis for Newton's Method for a single nonlinear equation in one unknown. The approximation described by equation (5) defines an iterative procedure where each successive iteration produces a value closer to the actual solution, x, i.e.,

$$x_{i+1} = x_i - f(x_i)/f'(x_i). \qquad (6)$$

Equation (6) defines the sequence $\{x_i\}_{i=0}^{\infty}$.

The following theorem shows that under suitable conditions the sequence defined by (6) converges to the actual solution, x.

<u>Theorem 1.</u>[12] Let f be twice continuously differon $[a, b]$. If $x \in [a, b]$ is such that $f(x) = 0$ and $f'(x) \neq 0$, then there exists a $\delta > 0$ such that Newton's Method generates $\{x_i\}_{i=1}^{\infty}$ where $x_i \to x$ as $i \to \infty$, when $x_o \in [x - \delta, x + \delta]$.

The interval, $[x - \delta, x + \delta]$, is called the interval of convergence. One should note here that the interval of convergence could be very small. This is the first exposure of the need for a good initial approximation.

The above procedure for developing Newton's Method for equation (2) provides the basis for developing Newton's Method for equation (1). Recall equation (1),

$$F(x) = 0$$

[12]*Ibid.*, p. 43.

where $F: R^n \to R^n$, $x \varepsilon R^n$. The following equation is an n-dimensional analogue to equation (6):

$$x_{i+1} = x_i - J(x_i)^{-1}F(x_i) \qquad (7)$$

where $x_i \varepsilon R^n$, for i = 0, . . ., and $J(x_i)^{-1}$ is the inverse of the Jacobian Matrix evaluated at x_i. The Jacobian Matrix is the n-dimensional analogue of the derivative of f and multiplying by $J(x_i)^{-1}$ is analogous to dividing by $f'(x_i)$. The derivation of equation (7) is similar to, but much more complicated than, the above derivation for Newton's Method for a single equation.

Equation (7) defines a sequence which converges to the solution, x, under suitable conditions. The following theorem states those conditions.[13]

<u>Theorem 2</u>. Let F be twice Frechet differentiable on a convex set D. If $x \varepsilon D$. If $x \varepsilon D$ is such that $F(x) = 0$ and $F'(x) = 0$, then there exists an open neighborhood S_O D such that Newton's Method described by equation (7) generates a sequence.

In this n-dimensional case, S_o is the ball of convergence where x_o must be in order to obtain quardratic convergence to the solution, x. As in the one-dimensional case the ball of convergence can be very small.

[13]A detailed discussion of Newton's method along with a related theorem can be found in J. M. Ortega and W. C. Rheinbolt, <u>Iterative Solutions of Nonlinear Equations in Several Variables</u> (New York and London: Acedemic Press, 1970), p. 1.

Levenberg-Marquardt Method

The need for the initial approximation to the solution to be within some ball of convergence can greatly hinder solving equation (1). In many instances where the solution to equation (1) is sought there does not exist physical or analytic evidence to provide an initial approximation which is in the ball of convergence. Hence Newton's Method will not guarantee convergence to the solution.

As stated in Chapter I, there do exist alternate approaches to the problem. The Levenberg-Marquardt Method is one of those alternatives which is quite successful. The Levenberg-Marquardt Method is somewhat based on the method of steepest descent, or conjugate gradient method, which I will first describe.

The method of steepest descent was applied to linear systems of equations in a paper by Hestenes and Stiefel.[14] It is also described in a book by Johnson and Riess.[15] The following is a brief description of the concept as presented in Ralston and Rabinowitz.[16]

Finding a solution to equation (1) is equivalent to finding the minimum of the following function,

---

[14]Hestenes and Stiefel, "Solution of Nonlinear Problems," p. 164.

[15]Lee. W. Johnson and R. Dean Riess, Numerical Analysis, (Reading: Addision-Wesley Publishing Company, 1977), p. 341.

[16]Anthony Ralston and Phillip Rabinowitz, A First Course in Numerical Analysis (New York: McGraw Hill Book Company, 1978), p. 361.

$$G(x) = F^T(x)F(x) = \sum_{i=1}^{n} [f_i(x_1, \ldots, x_n]^2 \qquad (8)$$

$F: R^n \rightarrow R^n$ and $x_1 \varepsilon R$ for $i = 1, \ldots, n$. Finding the minimum to equation (8) can be done by searching along the path in the direction of the negative gradient, since it is well-known that the negative gradient points to the direction of steepest descent. Thus moving along the path in the direction of the negative gradient will be moving towards the minimum.

Noting that the gradient of G(x), denoted $\nabla G(x)$, is evaluated as follows,

$$\nabla G(x) = \left[\frac{\partial G(x)}{\partial x_1}, \ldots, \frac{\partial G(x)}{\partial x_n} \right], \qquad (9)$$

one can determine $\nabla G(x) = 2J^T(x)F(x)$. Hence the method of steepest descent becomes defined by

$$x_{i+1} = x_i - \alpha_i J^T(x)_i F(x)_i \qquad (10)$$

where $x_i \varepsilon R^n$ and $\alpha_i \varepsilon R$ for $i = 1, 2, \ldots$.

The problem that exists with this method is that a determination must be made at each step of how far to go along the path in the direction of steepest descent. In other words, determining α_i at each step. It is clear that the size of α_i could easily cause overshooting of the minimum if it is too large. Also, if α_i is chosen too small, convergence can greatly be impaired. This problem can be overcome by minimizing the following function,

$$h(\alpha) = F(x - \alpha \nabla G(x)^T). \qquad (11)$$

The α which minimizes this is the step size.

A method which incorporates the method of steepest descent in an elegant fashion is the Levenberg-Marquardt Method. This method was developed by D. Marquardt for application to nonlinear least squares analysis in a 1963 paper.[17] An adaptation for solving equation (1) by this method is also found in that paper. The concept of the method is to use the following equation as a generator for approximations to the solution to equation (1),

$$x_{i+1} = x_i - [J(x_i)^T J(x_i) + \partial_i I]^{-1} J(x_i)^T F(x_i), \qquad (12)$$

where $x_i \in R^n$, $\partial_i \in R$ for $i = 1, 2; \ldots$

It is quite obvious that $\partial_i=0$ implies that the above equation is Newton's Method. It is less obvious however, that if ∂_i is chosen sufficiently large, equation (12) is the method of steepest descent. Consider the following theorem.

<u>Theorem 3.</u>[18] Let $J(x)$ and $J(x_i)^T$ be defined, then:

(i) $J(x_i)^T J(x_i)$ is positive semidefinite,

(ii) $(J(x_i)^T J(x_i) + \partial_i I)^{-1}$ exists for $\partial_i > 0$ and is $R\left[D(\partial_i)^{-1}\right] R^T$, where $D = R^T\left[J(x_i)^T J(x_i)\right]$ and $D(\partial_i) = [d_1 + \partial_i, \ldots, d_n + \partial_i]$,

(iii) $(J(x_i)^T J(x\ ) + \partial_i I)^{-1}$ decreases as ∂_i increases.

---

[17] Marquardt, "Least-Squares Estimation," p. 164.

[18] Proof in Appendix D.

Hence when ∂_i is chosen sufficiently, equation (12) is the method of steepest descent defined by equation (10).

What is left to describe in the Levenberg-Marquardt Method is the method of choice for ∂_i. The Marquardt paper indicates a method of choice for ∂_i as follows. This is the method I have adapted.

Let $v > 1$.

Let ∂_{i-1} denote a previous value of ∂.

Let $\partial_o = 10^{-2}$ or some value chosen between 1 and 10^{-2}.

Compute $w = F[\partial_{i-1} x_{i-1}]$, $y = F[(\partial_{i-1}/v)\ x_{i-1}]$.

Then if

(i) $\|y\|_2 \leq \|F(x_{i-1})\|_2$, let $\partial_i = \partial_{i-1}/v$ to decrease as the solution is approached.

(ii) $\|y\|_2 > \|F(x_{i-1})\|_2$ and $\|w\|_2 \leq \|F(x_{i-1})\|_2$, let $\partial_i = \partial_{i-1}$ since decreasing ∂_{i-1} doesn't produce the desired result.

(iii) $\|y\|_2 > \|F(x_{i-1})\|_2$ and $\|w\|_2 > \|F(x_{i-1})\|_2$, increase ∂_{i-1} by multiples of v until for some smallest b, $\|F(\partial_{i-1}v^b)(x_{i-1})\|_2 \leq \|F(x_{i-1})\|_2$, then $\partial_i = \partial_{i-1}v^b$. This will put the iteration in the state of steepest descent.

As will be seen in the results of testing, this method of choice for ∂_i and the overall Levenberg-Marquardt Method is quite successful and the procedure produces the desired goal, global convergence.

Broyden's Methods

Before developing Broyden's methods I will briefly discuss the general form of all quasi-newton methods. Recall equation (7),

$$x_{i+1} = x_i - J(x_i)^{-1}F(x_i). \qquad (7)$$

It is quite easy to see that at each iteration Newton's Method requires n functional evaluations for computing $F(x_i)$. Also, inverting a general nxn matrix is known to take $0(n^3)$ arithmetic operations[19] whether done directly or done by solving a linear system. So, as was stated in the introduction, Newton's Method can be quite expensive. This expense has led to the development of the following approaches which try to eliminate some of the computations or functional evaluations.

Instead of equation (7) the quasi-newton methods follow equation (13),

$$x_{i+1} = x_i - A_i^{-1}\, F(x_i) \qquad (13)$$

where A_i is an approximation to $J(x_i)$ which is updated at each iteration.

As an example of quasi-newton methods let us refer bace to another method of solving equation (2). A discretized Newton Method for solving equation (2), sometimes called the secant method, is as follows,

---

[19]Burden, Faires, and Reynolds, Numerical Analysis, p. 325.

$$x_{i+1} = x_i - \left[\frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}\right]^{-1} f(x_i) \tag{14}$$

which implies

$$f'(x_i)(x_{i-1} - x_i) = f(x_{i-1)} - f(x_i). \tag{15}$$

For the n-dimensional case, equation (16) is asked to hold,

$$A_i(x_{i-1} - x_i) = F(x_{i-1}) - F(x_i). \tag{16}$$

The so-called secant equation where A_i is the approximation to the Jacobian matrix.

· Here one should note that A_i is not unique hence there are many quasi-newton methods. As pointed out in a paper by Dennis and Schnabel,[20] the most successful matrix updates are those which minimize the norm,

$$||A_i - A_{i-1}||_F,$$

where $||\cdot||$ is defined as $\left[\sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij})^2\right]^{\frac{1}{2}}$. Forcing A_i to minimize the above norm preserves previous information of the approximation matrix. Hence, in the case of the secant method described above, if one uses $A_o = J(x_o)$, then A_1 will change in the least way necessary to satisfy the secant equation. The information of the actual Jacobian is used to approximate the new Jacobian.

---

[20] J. E. Dennis, and R. B. Schnabel, "Least Change Secant Updates for Quasi-Newton Methods," <u>SIAM Review</u> 19 (January 1977):443.

To derive Broyden's A method the following theorem is necessary.[21]

Theorem 4. Let $A_{i-1} \varepsilon L(R^n)$; $S, Y \varepsilon R^n$ where $s \neq 0$ Let $Q(y, s) = \{M \varepsilon L(R^n) | Ms = y\}$. Then the unique solution to $\min\limits_{A_i \varepsilon Q(y, s)} ||A_i - A_{i-1}||_F$

$$\text{is } A_i = A_{i-1} + \frac{(y - A_{i-1}s)s^T}{||s||_2} . \tag{17}$$

Thus if $s = (x_i - x_{i-1})$ and $y = [F(x_i) - F(x_{i-1})]$, then

$$A_i = A_{i-1} + \frac{[F(x_i) - F(x_{i-1}) - A_{i-1}(x_i - x_{i-1})](x_i - x_{i-1})^T}{||x_i - x_{i-1}||_2} . \tag{18}$$

Equation (18) clearly satisfies the secant equation and so it is a quasi-newton method.

Equation (18) describes Broyden's A Method. This method approximates the Jacobian at each iteration, eliminating the n^2 functional computations necessary to compute $J(x_i)$. One must note that using A_i, as described by equation (18), in equation (13) is useful only if the Jacobian is very costly to evaluate. The sometimes overwhelming price of inversion still exists.

To deal with this second difficulty, Broyden developed a method to approximate A_i^{-1}, hence Broyden's A^{-1} Method. This method not only eliminates computation of the Jacobian, but

---

[21] Ibid., p. 445.

reduces the number of arithemetic operations at each step. Broyden's A^{-1} Method owes its existence to the following theorem.[22]

<u>Theorem 5</u>. Let $A \varepsilon L(R^n)$ such that A^{-1} exists and let $U, V \varepsilon L(R^m, R^n)$ $m \quad n$. Then $(A + UV^T)^{-1}$ exists if and only if $(I + V^TA^{-1}U)^{-1}$ exists. In that case

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}VTA^{-1}. \tag{19}$$

This formula for computing $(A + UV^T)^{-1}$ is called the Sherman-Morrison-Woodbury Formula. Its validity is readily seen if one multiplies the righthand side of equation (19) by $(A + UV^T)$.

If in equation (19) we let U and V be vectors x and $y \varepsilon R^n$, then equation (19) becomes

$$(A + xy^T)^{-1} = \frac{A^{-1} - A^{-1}xy^TA^{-1}}{1 + y^T A^{-1}x}, \tag{20}$$

where $y^TA^{-1}x \neq -1$.

Broyden used equation (2) to develop the A^{-1} approximation. Recall equation (17),

$$A_i = A_{i-1} + \frac{(y - A_{i-1}s)s^T}{||s||_2}, \tag{17}$$

which implies

$$A_i = \left[A_{i-1} + \frac{(y - A_{i-1}s)s^T}{||s||_2}\right]^{-1} \tag{21}$$

---

[22]Adapted from text in Ortega and Rheinbolt, "Iterative Solutions," p. 212.

but equation (21) is in the form of the left-hand side of equation (20) where $x - [y - A_{i-1}s]/||s||_2$ and $y = s$. Hence

$$A_i^{-1} = A_{i-1}^{-1} - \left[\frac{A_{i-1}^{-1}(y - A_{i-1}s)s^T A_{i-1}^{-1}}{||s||_2}\right]\left[1 + \frac{s^T A_{i-1}^{-1}(y - A_{i-1}s)}{||s||_2}\right]^{-1}.$$

which reduces to

$$A_i^{-1} = A_{i-1}^{-1} + \frac{(s - A_{i-1}^{-1}y)s^T A_{i-1}^{-1}}{s^T A_{i-1}^{-1}y}, \qquad (22)$$

where $s = x_i - x_{i-1}$, $y = F(x_i) - F(x_{i-1})$, and $s^t A_{i-1}^{-1} \, y \neq 0$.

Equation (22) allows equation (13) to be implemented without actually calculating the inverse, a savings that will be seen to be greatly justified.

Alternate Methods

There exists many other quasi-newton methods. Each of these methods has its special application and has its derivation based on special properties of the Jacobian Matrix. For example, there are special methods which handle symmetric Jacobians, e.g., Powell's Method. Another example is the Schubert-Broyden Method applied to problems where many entries of the Jacobian are zero, i.e., the Jacobian is sparse. When minimization techniques are used on systems of nonlinear equations the solution to equation (1) is sought, but special properties of the Hessian matrix lead to the use of variable metric methods.[23]

Approaches to solving equation (1) have gone beyond the classical iterative Newton Method. Of these, Kizner's Method uses integration techniques to yield new approximations to the solution. Another approach is the use of systems of differential equations to solve equation (1).[24] More significant are the methods of Brown and Brent. Different from quasi-newton methods, Brown's and Brent's methods retain the property of quadratic convergence while using about half the functional evaluations of Newton's Method. These last two methods, in particular, both succumb to the problem of local convergence.

The methods of Brown and Brent are quite similar and are both quite complicated to implement. As stated in Chapter I, D. Gay simplified this problem. The concept behind Brent's Method, therefore similarily Brown's Method, is to linearize the system $F(x) = 0$ by expansion about an approximation. This gives an approximation to the Jacobian Matrix that is reduced to lower triangular form by a procedure similar to the procedure used by the Q. R. algorithm in solving eigenvalue problems.[25] By this I mean that the lower triangular form is

---

[23]Each of these methods can be seen in Dennis and More', "Quasi-Newton Methods," p. 46.

[24]S. Incerti, V. Parisi, and F. Zirilli, "A New Method for Solving Nonlinear Simultaneous Equations," SIAM Journal of Numerical Analysis 16 (October 1979): 779.

[25]Burden, Faires, and Reynolds, Numerical Analysis, p. 425.

obtained by nultiplying the approximate Jacobian by a sequence of Given or Householder rotations. Then the following equation is solved for the new value of the solution,

$$F(x_{i-1}) + A^T(x_i - x_{i-1}) = 0, \tag{23}$$

where A^T is the approximation to the Jacobian. The savings occurs because only $n^2/2 + 3n/2$ functional evaluations are computed.

As was mentioned in Chapter I, I intend to test Newton's Method against Broyden's A^{-1} Method. Each of these methods will have the benefit of the Levenberg-Marquardt Method for starting approximations. It is expected that Newton's Method will converge faster than Broyden's A^{-1} Method relative to the number of iterations. This is expected since quasi-newton methods give up the property of guadratic convergence. However, it was seen that the number of functional evaluations in Newton's Method is $n^2 + n$ at each iteration, while in Broyden's A^{-1} the number is only n. Also, in Broyden's A^{-1} method there is the need for $0(n^2)$ arithemetic operations in computing $A_i^{-1}F(x)$, while $0(n^3)$ arithemetic operation are required to compute $J(x_i)^{-1}F(x_i)$ in Newton's Method. These two points seem to indicate that Broyden's Method might be faster in terms of time.

CHAPTER III

TEST PROBLEMS

The following problems have been selected as test problems. These test problems come from two sources.[26] These problems will test Newton's Method versus Broyden's A^{-1} Method and the Levenberg-Marquardt Method versus an approximate Levenberg-Marquardt Method, where the Jacobian is approximated by Broyden's A Method. The problems are listed with the solution, x*, and reasons for choosing the problem.

<u>Problem 1.</u> $f_1(x_1, x_2) = x_1^2 - x_2 - 1 = 0$

$f_2(x_1, x_2) = (x_1 - 2)^2 + (x_2 + .5)^2 - 1 = 0$

$x^* = (1.0673460858067, 0.1392276668869)^T$.

This was chosen since it is a system of two polynomials in two unknowns, a relatively simple looking problem with a complicated solution.

<u>Problem 2.</u> $f_1(x_1, x_2) = 10^4 x_1 x_2 - 1 = 0$

$f_2(x_1, x_2) = e^{-x_1} + e^{-x_2} - 1.001 = 0$

$x^* = (.000010981593297, 9.1061467398)^T$.

This was chosen since one equation has an extremely large coefficient, 10^4, while the other equation has terms which could be very small.

---

[26]The two sources are (i) Burden, Faires and Reynolds, <u>Numerical Analysis</u>, p. 446-449 and (ii) Gay, "Brown's Method," p. 15.

<u>Problem 3.</u> $f_1(x_1, x_2) = -13 + x_1 + [(5 - x_2)x_2 - 2]x_2 = 0$

$f_2(x_1, x_2) = -29 + x_1 + [(x_2 + 1)x_2 - 14]x_2 = 0$

$x^* = (5, 4)^T$

This problem was chosen since it is a system of two third degree polynomials which has a simple solution.

<u>Problem 4.</u> $f_1(x_1, x_2, x_3) = 3x_1 - \cos(x_2x_3) - .5 = 0$

$f_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 1)^2 + \sin x_3 + 1.06 = 0$

$f_3 = (x_1, x_2, x_3) = e^{-x_1x_2} + 20x_3 + (10\pi - 3)/3 = 0$

$x^* = (.5, 0, -\pi/6)^T$

This was chosen since it contains exponential, trigonometric, and polynomial terms.

<u>Problem 5.</u> $f_i(x_1, \ldots, x_{10}) = 11 + 2x_i + \sum_{\substack{j=1 \\ j\neq 1}}^{n} x_j = 0;$

$i = 1, \ldots, 9$

$f_{10}(x_1, \ldots, x_{10}) = 1 + \sum_{j=1}^{10} x_j = 0$

$x^* = (1, \ldots, 1)^T$

This was chosen since it is an almost linear 10 x 10 system.

<u>Problem 6.</u> $f_i(x_1, \ldots, x_{20}) = \dfrac{\sin(x_{i+1}) - \sin(x_i)}{\sqrt{12.88}} = 0$

$i = 1, \ldots, 19$

$f_{20}(x_1, \ldots, x_{20}) = .2 \sum_{j=1}^{20} \tan(x_j) - 2 = 0$

$x^* = (.14, .20, .24, .28, .31, .35, .38, .41, .43, .46, .48, .51, .53, .55, .57, .60, .62, .64, .66, .68)^T$

This was chosen to test Broyden's A^{-1} versus Newton's Method only at initial approximation $(1, . . . , 1)^T$. Chosen since it is a large system of nonlinear equations.

Broyden's A^{-1} Method, Newton's Method and the Levenberg-Marquardt Method were each discussed in the previous chapters. Here I will discuss the approximate Levenberg-Marquardt Method. Recall equation (12) and equation (17),

$$x_{i+1} = x_i - \left[J(x_i)^T J(x_i) + \partial_i I\right]^{-1} J(x_i)^T F(x_i), \quad (12)$$

$$A_i = A_{i-1} + \left[(y - A_{i-1}s)s^T\right]/||s||_2. \quad (17)$$

The Levenberg-Marquardt Method requires that $J(x_i)$ be computed at each iteration. The approximate Levenberg-Marquardt Method will use Broyden's A_i approximation to $J(x_i)$, described by equation (17), in equation (12). This alleviates the work of computing $J(x_i)$, yielding,

$$x_{i+1} = x_i - (A_i^T A_i + \partial_i I)^{-1} A_i^T F(x_i). \quad (24)$$

Testing the approximate Levenberg-Marquardt Method against the Levenberg-Marquardt Method will provide insight to the savings, if any, of Broyden's A Method.

CHAPTER IV

RESULTS

Problems 1 through 5 listed in Chapter III were solved by: (i) the Levenberg-Marquardt Method, L-M, to generate good initial approximations; (ii) the approximate Levenberg-Marquardt Method, A-L-M; (iii) Newton's Method with a good initial approximation, N-W; (iv) Newton's Method without a good initial approximation, N; (v) Broyden's A^{-1} Method with a good initial approximation, B-W, and; (vi) Broyden's A^{-1} Method without a good initial approximation, B. Problem 6 was solved by B and by N, with initial approximation $(1, . . . , 1)^T$.

The L-M Method and the A-L-M Method were considered successful as starting procedures when there was no change in the approximate solutions in the first decimal place, i.e. $||x_i - x_{i-1}||_2 \leq .01$.

The methods of Broyden and Newton were considered successful as terminal procedures when there was no change in the approximate solutions in the tenth decimal place, i.e. $||x_i - x_{i-1}||_2 \leq 10^{-11}$.

Since the results must be correct to ten decimal places, double precision arithmetic in FORTRAN was used on an AMDAHL 470 V/5 computer to program the problems. There are sample programs for the L-M, N and B methods in the appendices.

The following two tables sum the results of the methods. Table I indicates the number of interations that the given method required to coverage to the above described tolerances. Table II indicates the time in seconds that each method required to coverage. Methods that did not coverage for a certain problem are marked "Failed".

TABLE I

ITERATIONS REQUIRED

PROBLEM	L-M	A-L-M	N	B	N-W	B-W
1	7	9	26	15	4	4
2	83	75	14	28	4	4
3	19	19	44	Failed	2	2
4	4	4	6	7	3	4
5	2	2	Failed	Failed	5	7
6	Not Tested	Not Tested	7	21	Not Tested	Not Tested

Table I indicates as expected that Newton's Method coverages, usually, in fewer iterations. Also the A-L-M Method coverages similar to the A-L Method.

TABLE II

TIME REQUIRED

PROBLEM	L-M	A-L-M	N	B	N-W	B-W
1	0.030119	0.044097	0.060533	0.029836	0.014483	0.013057
2	0.348452	0.364732	0.039304	0.053389	0.016311	0.013617
3	0.080099	0.109480	0.090887	Failed	0.010268	0.009911
4	0.032079	0.045856	0.046504	0.028814	0.025721	0.236429
5	0.383725	0.392817	Failed	Failed	0.453926	0.236429
6	Not Tested	Not Tested	4.531082	2.280336	Not Tested	Not Tested

Table II indicates, again as expected, that in the majority of the time Broyden's A^{-1} Method is quicker in terms of time to obtain convergence. There is only one instance when Newton's Method is quicker than Broyden's A^{-1} Method. The table also indicates that the A-L-M Method requires more time than the L-M Method.

Each of the terminal methods, N, B, N-W, B-W, yield results which are correct to the actual solution to at least 10 decimal places.

CHAPTER V

CONCLUSION

The purpose of this paper was to determine and implement a procedure that would allow the solution to equation (1) to be obtained given any initial approximation. Also it was desired to compare Newton's Method with a popular alternative. Broyden's A^{-1} Method was decided upon after a general search of methods to solve equation (1) was carried out.

The Levenberg-Marquardt Method has satisfied the first requirement. Test results have shown that this procedure works and works well. An attempt was made to improve the efficiency of the Levenberg-Marquardt Method, but this attempt did not reduce the use of time due to the fact that the matrix $(A_i^T A_i + \partial_i I)$ must still be inverted.

The comparison of Newton's and Broyden's methods can be done in two ways. One way is to compare the methods given any initial approximation and another way is to compare the methods given an initial approximation from the Levenberg-Marquardt Method.

The tables show that, whenever convergence is comparable, Broyden's Method is more efficient in every case except one. This is what was expected from the theory. Also, convergence can be obtained in Broyden's

Method in more interations than Newton's Method, but remain more efficient in terms of time. This demonstrates Broyden's superiority.

The one case where Broyden's Method was less efficient than Newton's Method brings up a drawback of any of the quasi-newton methods. This drawback is that the quasi-newton methods are not self corrective, as is Newton's Method. This is the most probable reason for Broyden's slower convergence in that one case.

One should not get disturbed by the fact that Broyden's Method failed in two cases. Those cases each had poor initial approximations. Note that the problem is cleared when an initial approximation is used from the Levenberg-Marquardt Method.

In conclusion I would suggest when solving equation (1) to apply the Levenberg-Marquardt Method then, once convergence is evident, apply Broyden's A^{-1} Method.

BIBLIOGRAPHY

BOOKS

1. Burden, Richard L.; Faires, J. Douglas; and Reynolds, Albert C. Numerical Analysis. Boston: Prindle, Weber and Schmidt, 1978.

2. Cooper, Leon; and Steinberg, David I. Introduction to Methods of Optimization. Philadelphia: W. B. Saunders Company, 1970.

3. Johnson, Lee W.; and Riess; R. Dean. Numerical Analysis. Reading: Addision-Wesley Publishing Company, 1977.

4. Ortega, J. M.; and Rheinbolt, W. C. Iterative Solutions of Nonlinear Equations in Several Variables. New York and London: Academic Press, 1970.

Articles

6. Broyden, C. G. "A Class of Methods for Solving Nonlinear Simultaneous Equations." Mathematics of Computation 19 (1965):577-93

7. Cosnard, Michel Y.; and More', Jorge J. "Numerical Solution of Nonlinear Equations." Transactions on Mathematical Software 5 (March 1979):64-85.

8. Davidenko, D. "On the Approximate Solution of a System of Nonlinear Equations." Ukran. Mat. 5 (1953): 196-206.

9. Dennis, J. E.; and More', Jorge J. "Quasi-Newton Methods, Motivation and Theory." SIAM Review 19 (January 1977):46-89.

10. Dennis, J. E.; and Schnabel, R. B. "Least Change Secant Updates for Quasi-Newton Methods." SIAM Review 21 (October 1979):443-59.

11. Gay, D. M. "Implementing Brown's Method." Report CNA-109. Austin: Center of Numercial Analysis, [1975].

12. Hestenes, Magnus R.; and Stiefel, Edward. "Methods of Conjugate Gradients for Solving Linear Systems." Journal of Research of the National Bureau of Standards 49 (December 1952): 409-36.

13. Incerti, S.; Parisi, V.; and Zirilli, F. "A New Method for Solving Nonlinear Simultaneous Equations." SIAM Journal of Numercial Analysis 16 (October 1979):779-789

14. Kinzner, W. "A Numercial Method for Finding Solutions of Nonlinear Equations." SIAM Journal of Applied Mathematics 12 (1964):424-28.

15. Levenberg, K. "A Method for the Solution of Certain Nonlinear Problems in Least Squares." Quarterly Journal of Applied Mathematics 2 (1944):164-68.

16. Marquardt, Donald W. "An Algorithm for Least-Squares Estimation of Nonlinear Parameters." SIAM Journal of Applied Mathematics 11 (June 1963): 431-41.

```
$JOB
C*****************************************************************
C        THIS PROGRAM SOLVES THE FOLLOWING NON-LINEAR             *
C        SYSTEM BY NEWTON'S METHOD.                               *
C        THE PROCEDURE IS CONSIDERED COMPLETE WHEN                *
C        THE EUCLIDEAN NORM OF THE OLD AND NEW VALUES             *
C        IS LESS THAN 0.00000000001.                              *
C        (NOTE THAT THIS VALUE CAN BE MADE AS LARGE               *
C        OR AS SMALL AS THE USER WISHES.)                         *
C                                                                 *
C        F(1)=-11+2*X1+X2+X3+...+X10                              *
C        .                                                        *
C        .                                                        *
C        .                                                        *
C        F(I)=-11+2*XI+SUM(XJ) J=1 TO 10 WHERE J¬=I               *
C        .                                                        *
C        .                                                        *
C        .                                                        *
C        F(10)=-1+PROD(XI)    I=1 TO 10                           *
C                                                                 *
C        THE FOLLOWING IS A LIST OF THE SUBROUTINES:              *
C                                                                 *
C        NAME       PURPOSE                                       *
C        -----------------------------------------------          *
C        FN         EVALUATES VECTOR FUNCTION F(X)                *
C                   AT PRESENT VALUE X                            *
C        XM         COMPUTES THE PRODUCT OF MATICES               *
C        AI         COMPUTES THE NEW VALUE OF THE                 *
C                   SOLUTION                                      *
C        XIN        INVERTS A MATRIX                              *
C        FJN        EVALUATES THE JACOBIAN MATRIX                 *
C                                                                 *
C        NEWTON'S METHOD WORKS AS FOLLOWS:                        *
C                                                                 *
C        LET XO BE AN INITIAL APPROXIMATION TO THE                *
C            SOLUTION OF THE SYSTEM.                              *
C        THEN GENERATE XN BY THE FOLLOWING NEWTON                 *
C          EQUATION:                                              *
C                                                                 *
C        XN=XO-(J**-1)*F(XO)          (1)                         *
C                                                                 *
C             J IS THE JACOBIAN MATRIX,                           *
C             F(X(I)) IS THE VALUE OF THE VECTOR                  *
C             FUNCTION.                                           *
C                                                                 *
C        XN BECOMES THE NEW VALUE OF THE SOLUTION.                *
C                                                                 *
C        THIS PROCEDURE CONTINUES AS PRESCRIBED BY                *
C        THE FOLLOWING ALGORITHM UNTIL THE VALUES                 *
C        OF THE SOLUTION CHANGE VERY LITTLE.                      *
C                                                                 *
C        STEP 1:LET XO BE AN INITIAL VALUE.                       *
C        STEP 2:COMPUTE XN.                                       *
C        STEP 3:IF ||XN-XO||<10**-11 GO TO STEP 5.                *
C        STEP 4:XO=XN, COMPUTE XN BY NEWTON EQUATION,             *
```

```
C                    GO TO STEP 3.                                 *
C          STEP 5:THE PROCEDURE IS COMPLETE AND XN                 *
C                    IS THE APPROXIMATE SOLUTION TO THE            *
C                    SYSTEM.                                       *
C                                                                  *
C*******************************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XO(10),AO(10,10),XN(10),FO(10),FNN(10),
     -AN(10,10),XX(10),F1(10),F2(10),YO(10),Y(10)
      EXTERNAL XM,FN,XIN,AI,FJN
C******************************************************************
C                   CALL TIME2 IS A PROCEDURE USED TO             *
C                   FIND VIRTUAL C.P.U. TIME.                     *
C******************************************************************
      CALL TIME2
C******************************************************************
C          STEP 1 IN ALGORITHM.                                   *
C******************************************************************
      DO 91 I=1,10
   91 XO(I)=.5
      N=10
C*******************************************************************
C          STEP 2 IN ALGORITHM.                                    *
C*******************************************************************
      CALL FJN(XO,AO)
      CALL AI(XO,AO,N,XN,FO)
      DO 5 I=1,200
      SUM=0.0
C*******************************************************************
C          STEP 3 IN ALGORITHM.                                    *
C*******************************************************************
      DO 6 J=1,N
    6 SUM=SUM+DABS(XN(J)-XO(J))**2
      SS=DSQRT(SUM)
      IF (SS.LT.0.00000000001)GO TO 99
C*******************************************************************
C          STEP 4 IN ALGORITHM.                                    *
C*******************************************************************
      DO 9 J=1,N
    9 XO(J)=XN(J)
      CALL FJN(XO,AO)
      CALL AI(XO,AO,N,XN,FO)
    5 CONTINUE
C*******************************************************************
C          THE PROCEDURE IS COMPLETE.                              *
C*******************************************************************
   99 CALL FN(XN,FNN,N)
      WRITE(6,55)I
      WRITE(6,56)(XN(J),J=1,N)
      WRITE(6,57)(FNN(J),J=1,N)
   55 FORMAT(/,15X,'THE NUMBER OF ITERATIONS
     -IN NEWTON'S METHOD IS',I5)
   56 FORMAT(/,1X,10(1X,F8.3))
   57 FORMAT(/,1X,10(1X,F8.3))
      CALL TIME2
      STOP
      END
C***************************************************************
C       XM COMPUTES A*B=C                                      *
C***************************************************************
      SUBROUTINE XM(A,B,N,C)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),B(10,10),C(10,10)
      DO 1 J=1,N
      DO 1 I=1,N
      C(J,I)=0.0
      DO 1 K=1,N
    1 C(J,I)=C(J,I)+A(J,K)*B(K,I)
      RETURN
      END
C***************************************************************
C         FN COMPUTES THE VECTOR FUNCTION F AT A               *
C***************************************************************
      SUBROUTINE FN(A,F,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(10),A(10)
      S=0.0
      P=1.0
      DO 1 I=1,10
      P=P*A(I)
    1 S=S+A(I)
      NN=9
      DO 2 J=1,NN
    2 F(J)=-11.+2.*A(J)+S-A(4)
      F(10)=-1.+P
      RETURN
      END
C***************************************************************
C         XIN COMPUTES A**-1=C                                 *
C***************************************************************
      SUBROUTINE XIN(A,C,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),B(10,20),C(10,10)
      DO 1 J=1,N
      DO 1 I=1,N
    1 B(J,I)=A(J,I)
      N1=N+1
      N2=N*2
      DO 2 J=1,N
      DO 2 I=N1,N2
    2 B(J,I)=0.0
      DO 3 J=1,N
    3 B(J,J+N)=1.0
      DO 4 J=1,N
      AA=B(J,J)
      DO 5 I=1,N2
    5 B(J,I)=B(J,I)/AA
      DO 6 K=1,N
      IF(K.EQ.J)GO TO 6
      AA=B(K,J)
      DO 7 L=1,N2
    7 B(K,L)=-1.*AA*B(J,L)+B(K,L)
    6 CONTINUE
    4 CONTINUE
      DO 9 J=1,N
      DO 9 I=1,N
    9 C(J,I)=B(J,I+N)
      RETURN
      END
C***********************************************************************
C          AI COMPUTES J**-1 AND THE NEW SOLUTION                      *
C          BY THE FOLLOWING EQUATION:                                  *
C          XN=XO-(J**-1)*F(XO)                                         *
C***********************************************************************
      SUBROUTINE AI(X1,A,N,XN,FO)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),HN(10,10),X1(10),XN10),FO(10)
      CALL FN(X1,FO,N)
      CALL XIN(A,HN,N)
      DO 3 J=1,N
      XN(J)=0.0
      DO 3 I=1,N
    3 XN(J)=XN(J)+HN(J,I)*FO(I)
      DO 4 I=1,N
    4 XN(I)=X1(I)-XN(I)
      RETURN
      END
C***********************************************************************
C          FJN COMPUTES THE JACOBIAN MATRIX                            *
C***********************************************************************
      SUBROUTINE FJN(A,B)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10),B(10,10)
      P=1.0
      DO 1 I=1,9
      DO 1 J=1,10
    1 B(I,J)=1.0
      DO 2 J=1,9
    2 B(J,J)=2.0
      DO 3 J=1,10
    3 P=P*A(J)
      DO 4 J=1,10
    4 B(10,J)=P/A(J)
      RETURN
      END
```

APPENDIX B

```
$JOB
C*****************************************************************
C       THIS PROGRAM SOLVES THE FOLLOWING NON-LINEAR             *
C       SYSTEM BY THE LEVENBERG-MARQUARDT METHOD.                 *
C       THE PROCEDURE IS CONSIDERED COMPLETE WHEN                 *
C       THE EUCLIDEAN NORM OF THE OLD AND NEW VALUES              *
C       IS LESS THAN 0.01.                                        *
C       (NOTE THAT THIS VALUE CAN BE MADE AS LARGE                *
C       OR AS SMALL AS THE USER WISHES.)                          *
C                                                                 *
C       F(1)=-11+2*X1+X2+X3+...+X10                               *
C       .                                                         *
C       .                                                         *
C       .                                                         *
C       F(I)=-11+2*XI+SUM(XJ) J=1 TO 10 WHERE J¬=I                *
C       .                                                         *
C       .                                                         *
C       .                                                         *
C       F(10)=-1+PROD(XI)    I=1 TO 10                            *
C                                                                 *
C       THE FOLLOWING IS A LIST OF THE SUBROUTINES:               *
C                                                                 *
C       NAME       PURPOSE                                        *
C                                                                 *
C       ------------------------------------------------          *
C       FN         EVALUATES VECTOR FUNCTION F(X)                 *
C                  AT PRESENT VALUE X                             *
C       XM         COMPUTES THE PRODUCT OF MATICES                *
C       AI         COMPUTES THE NEW VALUE OF THE                  *
C                  SOLUTION                                       *
C       XIN        INVERTS A MATRIX                               *
C       FJN        EVALUATES THE JACOBIAN MATRIX                  *
C                                                                 *
C       THE LEVENBERG-MARQUARDT PROCEDURE WORKS AS                *
C       FOLLOWS:                                                  *
C       LET XO BE AN INITIAL APPROXIMATION TO THE                 *
C           SOLUTION OF THE SYSTEM.                               *
C       THEN GENERATE XN BY THE FOLLOWING L-M                     *
C           EQUATION:(DONE IN SUBROUTINE AI)                      *
C                                                                 *
C       XN=XO-((JT*J+RLAM*IM)**-1)*JT*F(XO)                       *
C                                                                 *
C             JT IS THE JACOBEAN MATRIX TRANSPOSE,                *
C             J IS THE JACOBEAN MATRIX,                           *
C             IM IS THE IDENTITY MATRIX,                          *
C             RLAM IS A VALUE CHOSEN BY A                         *
C             PROCESS DESCRIBED BELOW.                            *
C             F(XO) IS THE VALUE OF THE VECTOR                    *
C             FUNCTION.                                           *
C                                                                 *
C       XN BECOMES THE NEW VALUE OF THE SOLUTION.                 *
C                                                                 *
C       THIS PROCEDURE CONTINUES AS PRESCRIBED BY                 *
C       THE FOLLOWING ALGORITHM UNTIL THE                         *
C       SOLUTION CHANGES VERY LITTLE.                             *
C                                                                 *
```

```
C        STEP 1:LET XO BE AN INITIAL VALUE.                              *
C                LET RLAM BE A NUMBER BETWEEN .001                       *
C                AND 1.0, AND LET V=1.25.                                *
C        STEP 2:COMPUTE XN.                                              *
C        STEP 3:IF ||XN-XO||<0.01 GO TO STEP 6.                          *
C        STEP 4:XO=XN, COMPUTE NEW RLAM.                                 *
C        STEP 5: COMPUTE XN BY L-M EQUATION,                             *
C                GO TO STEP 3.                                           *
C        STEP 6:THE PROCEDURE IS COMPLETE AND XN                         *
C                IS THE APPROXIMATE SOLUTION TO THE                      *
C                SYSTEM.                                                 *
C                                                                        *
C        COMPUTATION OF NEW RLAM.                                        *
C                                                                        *
C        LET RLAM BE GIVEN INTITIALLY AND LET V=1.25.                    *
C        LET RLAMO DENOTE THE PREVIOUS RLAM.                             *
C        COMPUTE F(RLAMO*XO)=W                                           *
C                F((RLAMO/V)*XO)=Y                                       *
C                F(XO)=Z.                                                *
C        THEN (1) IF ||Y||<=||Z|| THEN                                   *
C                          RLAM=RLAMO/V.                                 *
C             (2) IF ||Y||>||Z|| AND ||W||<= ||Z|| THEN                  *
C                          RLAM=RLAMO.                                   *
C             (3) IF ||Y||>||Z|| AND ||W|| >||Z|| THEN                   *
C                   INCREASE RLAMO BY SUCCESSIVE                         *
C                   MULTIPLES OF V UNTIL                                 *
C                   ||F((RLAMO*(V**A))*XO)||<=||Z||                      *
C                   THEN LET RLAM=RLAMO*(V**A).                          *
C************************************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XO(10),AO(10,10),XN(10),FO(10),FNN(10),
     -AN(10,10),XX(10),F1(10),F2(10),YO(10),Y(10)
      EXTERNAL XM,FN,XIN,AI,FUN
C***********************************************************************
C               CALL TIME2 IS A PROCEDURE USED TO                     *
C               FIND VIRTUAL C.P.U. TIME.                             *
C***********************************************************************
      CALL TIME2
C************************************************************************
C        STEP 1 IN ALGORITHM.                                            *
C************************************************************************
      DO 91 I=1,10
   91 XO(I)=.5
      RLAM=.500
      V=1.25
      N=10
C************************************************************************
C        STEP 2 IN ALGORITHM.                                            *
C************************************************************************
      CALL FUN(XO,AO)
      CALL AI(XO,AO,RLAM,N,XN,FO)
      DO 5 I=1,200
      SUM=0.0
C************************************************************************
C        STEP 3 IN ALGORITHM.                                            *
```

```
C*****************************************************************
      DO 6 J=1,N
    6 SUM=SUM+DABS(XN(J)-XO(J))**2
      SS=DSQRT(SUM)
      IF (SS.LT.0.01)GO TO 99
C*****************************************************************
C          STEP 4 IN ALGORITHM.                                  *
C*****************************************************************
      DO 9 J=1,N
      XO(J)=XN(J)
      XN(J)=XC(J)*RLAM
    9 XX(J)=XC(J)*RLAM/V
      CALL FN(XN,F1,N)
      CALL FN(XX,F2,N)
      A=0.0
      B=0.0
      C=0.0
      DO 18 J=1,N
      A=A+FNN(J)
      B=B+F1(J)
   18 C=C+F2(J)
      IF(C.LE.A)GO TO 24
      IF(B.LE.A)GO TO 29
   21 RLAM=RLAM*V
      DO 19 J=1,N
   19 XN(J)=XN(J)*RLAM
      CALL FN(XN,F1,N)
      B=0.0
      DO 20 J=1,N
   20 B=B+F1(J)
      IF(B.LE.A)GO TO 29
      GO TO 21
   24 RLAM=RLAM/V
C*****************************************************************
C        . STEP 5 IN ALGORITHM.                                  *
C*****************************************************************
   29 CALL FJN(XO,AO)
      CALL AI(XO,AO,RLAM,N,XN,FO)
    5 CONTINUE
C*****************************************************************
C          THE PROCEDURE IS COMPLETE.                            *
C*****************************************************************
   99 CALL FN(XN,FNN,N)
      WRITE(6,55)I
      WRITE(6,56)(XN(J),J=1,N)
      WRITE(6,57)(FNN(J),J=1,N)
   55 FORMAT(/,15X,'THE NUMBER OF ITERATIONS
     -IN L-M IS',I5)
   56 FORMAT(/,1X,10(1X,F8.3))
   57 FORMAT(/,1X,10(1X,F8.3))
      CALL TIME2
      STOP
      END
C*****************************************************************
C          XM COMPUTES A*B=C                                     *
```

```
C*********************************************************************
      SUBROUTINE XM(A,B,N,C)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),B(10,10),C(10,10)
      DO 1 J=1,N
      DO 1 I=1,N
      C(J,I)=0.0
      DO 1 K=1,N
    1 C(J,I)=C(J,I)+A(J,K)*B(K,I)
      RETURN
      END
C*********************************************************************
C          FN COMPUTES THE VECTOR FUNCTION F AT A                     *
C*********************************************************************
      SUBROUTINE FN(A,F,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(10),A(10)
      S=0.0
      P=1.0
      DO 1 I=1,10
      P=P*A(I)
    1 S=S+A(I)
      NN=9
      DO 2 J=1,NN
    2 F(J)=-11.+2.*A(J)+S-A(4)
      F(10)=-1.+P
      RETURN
      END
C*********************************************************************
C          XIN COMPUTES A**-1=C                                       *
C*********************************************************************
      SUBROUTINE XIN(A,C,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),B(10,20),C(10,10)
      DO 1 J=1,N
      DO 1 I=1,N
    1 B(J,I)=A(J,I)
      N1=N+1
      N2=N*2
      DO 2 J=1,N
      DO 2 I=N1,N2
    2 B(J,I)=0.0
      DO 3 J=1,N
    3 B(J,J+N)=1.0
      DO 4 J=1,N
      AA=B(J,J)
      DO 5 I=1,N2
    5 B(J,I)=B(J,I)/AA
      DO 6 K=1,N
      IF(K.EQ.J)GO TO 6
      AA=B(K,J)
      DO 7 L=1,N2
    7 B(K,L)=-1.*AA*B(J,L)+B(K,L)
    6 CONTINUE
    4 CONTINUE
```

```
      DO 9 J=1,N
      DO 9 I=1,N
    9 C(J,I)=B(J,I+N)
      RETURN
      END
C*********************************************************************
C         AI COMPUTES THE L-M MATRIX AND THE NEW SOLUTION          *
C         BY THE FOLLOWING EQUATION:                               *
C         XN=XO-((JT*J+RLAM*IM)**-1)*JT*F(XO)                      *
C*********************************************************************
      SUBROUTINE AI(X1,A,R,N,XN,FO)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION AT(10,10),AP(10,10),A(10,10),
     -HN(10,10),AB(10,10),X1(10),XN(10),FO(10)
      CALL FN(X1,FO,N)
      DO 1 J=1,N
      DO 1 I=1,N
    1 AT(I,J)=A(J,I)
      CALL XM(AT,A,N,AP)
      DO 2 I=1,N
    2 AP(I,I)=R+AP(I,I)
      CALL XIN(AP,AB,N)
      CALL XM(AB,AT,N,HN)
      DO 3 J=1,N
      XN(J)=0.0
      DO 3 I=1,N
    3 XN(J)=XN(J)+HN(J,I)*FO(I)
      DO 4 I=1,N
    4 XN(I)=X1(I)-XN(I)
      RETURN
      END
C*********************************************************************
C         FJN COMPUTES THE JACOBIAN MATRIX                         *
C*********************************************************************
      SUBROUTINE FJN(A,B)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10),B(10,10)
      P=1.0
      DO 1 I=1,9
      DO 1 J=1,10
    1 B(I,J)=1.0
      DO 2 J=1,9
    2 B(J,J)=2.0
      DO 3 J=1,10
    3 P=P*A(J)
      DO 4 J=1,10
    4 B(10,J)=P/A(J)
      RETURN
      END
```

APPENDIX C

```
$JOB
C***********************************************************************
C         THIS PROGRAM SOLVES THE FOLLOWING NON-LINEAR                 *
C         SYSTEM BY BROYDEN'S METHOD.                                  *
C         THE PROCEDURE IS CONSIDERED COMPLETE WHEN                    *
C         THE EUCLIDEAN NORM OF THE OLD AND NEW VALUES                 *
C         IS LESS THAN 0.0000000001.                                   *
C         (NOTE THAT THIS VALUE CAN BE MADE AS LARGE                   *
C         OR AS SMALL AS THE USER WISHES.)                             *
C                                                                      *
C         F(1)=-11+2*X1+X2+X3+...+X10                                  *
C         .                                                            *
C         .                                                            *
C         .                                                            *
C         F(I)=-11+2*XI+SUM(XJ) J=1 TO 10 WHERE J¬=I                   *
C         .                                                            *
C         .                                                            *
C         .                                                            *
C         F(10)=-1+PROD(XI)    I=1 TO 10                               *
C                                                                      *
C         THE FOLLOWING IS A LIST OF THE SUBROUTINES:                  *
C                                                                      *
C         NAME       PURPOSE                                           *
C                                                                      *
C         -------------------------------------------------            *
C         FN         EVALUATES VECTOR FUNCTION F(X(I))                 *
C                    AT PRESENT VALUE X(I)                             *
C         XIN        INVERTS A MATRIX                                  *
C         FJN        EVALUATES THE JACOBIAN MATRIX                     *
C                                                                      *
C         BROYDEN'S METHOD WORKS AS FOLLOWS:                           *
C                                                                      *
C         LET XO BE AN INITIAL APPROXIMATION TO THE                    *
C             SOLUTION OF THE SYSTEM.                                  *
C         THEN GENERATE XN BY THE FOLLOWING NEWTON                     *
C            EQUATION:                                                 *
C                                                                      *
C         XN=XO-(J**-1)*F(X(I))                    (1)                 *
C                                                                      *
C         ***** EQUATION (1) IS USED ONLY ONCE. *****                  *
C                                                                      *
C               J IS THE JACOBEAN MATRIX,                              *
C               F(X(I)) IS THE VALUE OF THE VECTOR                     *
C               FUNCTION.                                              *
C                                                                      *
C         XN   BECOMES THE NEW VALUE OF THE SOLUTION.                  *
C                                                                      *
C         NEXT AN APPROXIMATION TO THE JACOBIAN INVERSE                *
C         IS EVALUATED BY THE FOLLOWING EQUATION                       *
C                                                                      *
C                         (SS-AIO*Y)*SST*AIO                           *
C         AIN=AIO+ ------------------------------------      (2)       *
C                             SST*AIO*Y                                *
C                                                                      *
C        WHERE: AIN IS THE NEW APPROXIMATION TO THE                    *
C                     JACOBIAN INVERSE.                                *
```

```
C                 AIO IS THE OLD APPROXIMATION TO THE                  *
C                      JACOBIAN INVERSE.                               *
C                 SS  IS (XN-XO).                                      *
C                 Y   IS (F(XN)-F(XO)).                                *
C                                                                      *
C         FINALLY NEW VALUES TO THE SOLUTION ARE                       *
C         OBTAINED BY THE FOLLOWING EQUATION.                          *
C                                                                      *
C         XN=XO-AIN*F(XO)                       (3)                    *
C                                                                      *
C         THIS PROCEDURE IS CONTINUED AS PRESCRIBED BY                 *
C         THE FOLLOWING ALGORITHM UNTIL THERE IS                       *
C         LITILE CHANGE IN THE SOLUTION.                               *
C                                                                      *
C         STEP 1:LET XO BE AN INITIAL VALUE.                           *
C                LET AI BE ACTUAL JACOBIAN INVERSE.                    *
C         STEP 2:COMPUTE XN BY EQUATION 1.                             *
C         STEP 3:IF ||XN-XO||<10**-11 GO TO STEP 6.                    *
C         STEP 4:COMPUTE AIN BY EQUATION (2).                          *
C         STEP 5:XO=XN, GO TO 3 COMPUTE XN BY EQUATION (3).            *
C         STEP 6:THE PROCEDURE IS COMPLETE AND XN                      *
C                IS THE APPROXIMATE SOLUTION TO THE                    *
C                SYSTEM.                                               *
C                                                                      *
C***********************************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XO(10),AO(10,10),XN(10),SS(10),FNN(10),SN(10),
     -AI(10,10),C(10,10),FO(10),X(10),Y(10)
      EXTERNAL FN,XIN,FJN
C**********************************************************************
C                 CALL TIME2 IS A PROCEDURE USED TO                   *
C                 FIND VIRTUAL C.P.U. TIME.                           *
C**********************************************************************
      CALL TIME2
C**********************************************************************
C         STEP 1 IN ALGORITHM.                                        *
C**********************************************************************
      N=10
      DO 91 I=1,10
   91 XO(I)=1.004
      CALL FJN(XO,AO)
      CALL XIN(AO,AI,N)
C***********************************************************************
C         STEP 2 IN ALGORITHM.                                         *
C***********************************************************************
      CALL FN(XO,FO,N)
      DO 5 II=1,400
      DO 1 J=1,N
      XN(J)=0.0
      DO 1 JJ=1,N
    1 XN(J)=XN(J)+AI(J,JJ)*FO(JJ)
      DO 2 J=1,N
    2 XN(J)=XO(J)-XN(J)
      SUM=0.0
      S2=0.0
```

```
C                 AIO IS THE OLD APPROXIMATION TO THE                    *
C                     JACOBIAN INVERSE.                                  *
C                 SS  IS (XN-XO).                                        *
C                 Y   IS (F(XN)-F(XO)).                                  *
C                                                                        *
C         FINALLY NEW VALUES TO THE SOLUTION ARE                         *
C         OBTAINED BY THE FOLLOWING EQUATION.                            *
C                                                                        *
C         XN=XO-AIN*F(XO)                       (3)                      *
C                                                                        *
C         THIS PROCEDURE IS CONTINUED AS PRESCRIBED BY                   *
C         THE FOLLOWING ALGORITHM UNTIL THERE IS                         *
C         LITTLE CHANGE IN THE SOLUTION.                                 *
C                                                                        *
C         STEP 1:LET XO BE AN INITIAL VALUE.                             *
C                LET AI BE ACTUAL JACOBIAN INVERSE.                      *
C         STEP 2:COMPUTE XN BY EQUATION 1.                               *
C         STEP 3:IF ||XN-XO||<10**-11 GO TO STEP 6.                      *
C         STEP 4:COMPUTE AIN BY EQUATION (2).                            *
C         STEP 5:XO=XN, GO TO 3 COMPUTE XN BY EQUATION (3).              *
C         STEP 6:THE PROCEDURE IS COMPLETE AND XN                        *
C                IS THE APPROXIMATE SOLUTION TO THE                      *
C                SYSTEM.                                                 *
C                                                                        *
C************************************************************************
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XO(10),AO(10,10),XN(10),SS(10),FNN(10),SN(10),
     -AI(10,10),C(10,10),FO(10),X(10),Y(10)
      EXTERNAL FN,XIN,FJN
C***********************************************************************
C                 CALL TIME2 IS A PROCEDURE USED TO                     *
C                 FIND VIRTUAL C.P.U. TIME.                             *
C***********************************************************************
      CALL TIME2
C***********************************************************************
C         STEP 1 IN ALGORITHM.                                          *
C***********************************************************************
      N=10
      DO 91 I=1,10
   91 XO(I)=1.004
      CALL FJN(XO,AO)
      CALL XIN(AO,AI,N)
C************************************************************************
C         STEP 2 IN ALGORITHM.                                           *
C************************************************************************
      CALL FN(XO,FO,N)
      DO 5 II=1,400
      DO 1 J=1,N
      XN(J)=0.0
      DO 1 JJ=1,N
    1 XN(J)=XN(J)+AI(J,JJ)*FO(JJ)
      DO 2 J=1,N
    2 XN(J)=XO(J)-XN(J)
      SUM=0.0
      S2=0.0
      CALL FN(XN,FNN,N)
      DO 6 J=1,N
      SS(J)=XN(J)-XO(J)
      Y(J)=FNN(J)-FO(J)
      S2=S2+FNN(J)
    6 SUM=SUM+SS(J)*SS(J)
      SQ=DSQRT(SUM)
C*********************************************************************
C          STEP 3 IN ALGORITHM.                                      *
C*********************************************************************
      IF (SQ.LT.0.00000000001)GO TO 99
C*********************************************************************
C          STEP 4 IN ALGORITHM.                                      *
C*********************************************************************
      DO 8 J=1,N
      X(J)=0.0
      DO 8 JJ=1,N
    8 X(J)=X(J)+AI(J,JJ)*Y(JJ)
      DO 9 J=1,N
    9 SN(J)=SS(J)-X(J)
      Q=0.0
      DO 10 J=1,N
   10 Q=Q+X(J)*SS(J)
      DO 11 J=1,N
      X(J)=0.0
      DO 11 JJ=1,N
   11 X(J)=X(J)+SS(JJ)*AI(JJ,J)
      DO 14 J=1,N
      DO 14 JJ=1,N
   14 AO(J,JJ)=SN(J)*X(JJ)/Q
      DO 12 J=1,N
      DO 12 JJ=1,N
   12 AI(J,JJ)=AI(J,JJ)+AO(J,JJ)
C*********************************************************************
C          STEP 5 IN ALGORITHM.                                      *
C*********************************************************************
      DO 13 J=1,N
   13 XO(J)=XN(J)
      CALL FN(XO,FO,N)
    5 CONTINUE
C*********************************************************************
C          THE PROCEDURE IS COMPLETE.                                *
C*********************************************************************
   99 CALL FN(XN,FNN,N)
      WRITE(6,55)II
      WRITE(6,56)(XN(J),J=1,N)
      WRITE(6,57)(FNN(J),J=1,N)
   55 FORMAT(/,15X,'THE NUMBER OF ITERATIONS IN L-M IS',I5)
   56 FORMAT(/,1X,10(1X,F12.9))
   57 FORMAT(/,1X,10(1X,F12.9))
      CALL TIME2
      STOP
      END
C*********************************************************************
C          FN COMPUTES THE VECTOR FUNCTION F AT A                    *
```

```
C*****************************************************************
      SUBROUTINE FN(A,F,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(10),A(10)
      S=0.0
      P=1.0
      DO 1 I=1,10
      P=P*A(I)
    1 S=S+A(I)
      NN=9
      DO 2 J=1,NN
    2 F(J)=-11.+2.*A(J)+S-A(4)
      F(10)=-1.+P
      RETURN
      END
C*****************************************************************
C          XIN COMPUTES A**-1=C                                  *
C*****************************************************************
      SUBROUTINE XIN(A,C,N)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10,10),B(10,20),C(10,10)
      DO 1 J=1,N
      DO 1 I=1,N
    1 B(J,I)=A(J,I)
      N1=N+1
      N2=N*2
      DO 2 J=1,N
      DO 2 I=N1,N2
    2 B(J,I)=0.0
      DO 3 J=1,N
    3 B(J,J+N)=1.0
      DO 4 J=1,N
      AA=B(J,J)
      DO 5 I=1,N2
    5 B(J,I)=B(J,I)/AA
      DO 6 K=1,N
      IF(K.EQ.J)GO TO 6
      AA=B(K,J)
      DO 7 L=1,N2
    7 B(K,L)=-1.*AA*B(J,L)+B(K,L)
    6 CONTINUE
    4 CONTINUE
      DO 9 J=1,N
      DO 9 I=1,N
    9 C(J,I)=B(J,I+N)
      RETURN
      END
C*****************************************************************
C          FJN COMPUTES THE JACOBIAN MATRIX                      *
C*****************************************************************
      SUBROUTINE FJN(A,B)
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(10),B(10,10)
      P=1.0
      DO 1 I=1,9
      DO 1 J=1,10
    1 B(I,J)=1.0
      DO 2 J=1,9
    2 B(J,J)=2.0
      DO 3 J=1,10
    3 P=P*A(J)
      DO 4 J=1,10
    4 B(10,J)=P/A(J)
      RETURN
      END
```

APPENDIX D

Proof of Theorem 3

Theorem 3 follows immediately from the more general result listed below.

Theorem - For any real matrix A, A^TA is positive semi definite, $(A^TA + \lambda I)^{-1}$ exists when $\lambda > 0$ and lim $||(A^TA + \lambda I)^{-1}A^Tx|| = 0$, $\lambda \to \infty$, for each x in R^n.

Proof - The symmetric matrix A^TA is semi positive definite since $(x, A^TAx) = ||Ax||^2 \geq 0$ for each x. Consequently, matrix R exists with $R^TR = I$ and $R^TA^TAR = D$, when D is a diagonal matrix with diagonal entries $d_1, d_2, \ldots d_n$. Thus, for $\lambda > 0$, $(A^TA + \lambda I)^{-1} = R[D(\lambda)]^{-1}RT$, where $D(\lambda)$ is the diagonal matrix with diagonal entries $d_1 + \lambda, d_2 + \lambda, \ldots d_n + \lambda$.

Since
$$\begin{aligned} ||(A^TA + \lambda I)^{-1}A^Tx|| &= ||R[D(\lambda)]^{-1}R^TA^Tx|| \\ &\leq ||R||\ ||[D(\lambda)]^{-1}|| \\ &\quad ||R^T||\ ||A^T||\ ||x|| \end{aligned}$$

and $[D(\ )]^{-1}$ is the diagonal matrix with diagonal entries $(d_i + \lambda)^{-1}, (d_2 + \lambda)^{-1}, \ldots (d_n + \lambda)^{-1}$ $\lim_{n \to \infty} ||[D(\lambda)]^{-1}|| = 0$ and $\lim_{\lambda \to \infty} ||(A^Tx + \ I)^{-1}A^Tx\ || = 0$.

The L-M method clearly gives Newton's method when $\lambda = 0$, that it approaches the method of steepest descent as λ increases is an immediate consequence of the above result.